MULTIAGENT SYSTEMS

MIT Press, 2011

Contents

List of Figures

List of Tables

Chapter 12

Distributed Constraint Handling and Optimization

A. Farinelli, M. Vinyals, A. Rogers and N.R. Jennings

1 Introduction

Constraints pervade our everyday lives and are usually perceived as elements that limit solutions to the problems that we face (e.g., the choices we make everyday are typically constrained by limited money or time). However, from a computational point of view, constraints are key components for efficiently solving hard problems. In fact, constraints encode *knowledge* about the problem at hand, and so restrict the space of possible solutions that must be considered. By doing so, they greatly reduce the computational effort required to solve a problem.

Against this background, constraint processing can be viewed as a wide and diverse research area unifying techniques and algorithms that span across many different disciplines including Planning and Scheduling, Operation Research, Computer Vision, Automated Reasoning and Decision Theory. All these areas deal with hard computational problems that can be made more tractable by carefully considering the constraints that define the structure of the problem.

Here we will focus on how constraint processing can be used to address optimization problems in Multi-Agent Systems. Specifically, we will consider Distributed Constraint Optimization Problems (DCOPs) where a set of agents must come to some agreement, typically via some form of negotiation, about which action each agent should take in order to jointly obtain the best solution for the whole system. This framework has been frequently used in the MAS literature to address problems such as meeting scheduling in human organizations, where agents must agree on a valid meeting schedule while maximizing the sum of individual preferences about when each meeting should be held, or target tracking in sensor networks, where sensors must agree on which target they should focus on to obtain the most accurate estimation of the targets' positions. A key common aspect of DCOPs for MAS is that each agent negotiates *locally* with just a subset of other agents (usually called neighbors) that are those that can directly influence his/her behaviors. Depending on the problem setting and on the solution technique used, this aspect can greatly reduce the computational effort that each agent faces, making hard problems tractable even for large-scale systems. For example in the meeting scheduling problem, an agent will directly negotiate with and care about people that he/she must meet, which is usually a small subset of the agents involved in the whole problem.

In more detail, this chapter aims to provide the reader with a broad knowledge of the main DCOP solution approaches describing both exact algorithms, approximate approaches and quality guarantees that can be provided in the DCOP framework. After reading it, you will understand:

- The mathematical formulation of a DCOP, being able to model distributed decision-making problems using the DCOP framework.

- The main exact solution techniques for DCOPs and the key differences between them in terms of benefits and limitations.

- Why and when approximate solution techniques should be used for DCOPs and the main algorithms in this space.

- Why quality guarantees are important, how to differentiate between various types of quality guarantees and which techniques are currently available in the literature to achieve them.

The chapter is organized as follows, first Section 2 presents the mathematical background for constraint networks and distributed constraint processing; next Section 3 provides examples of practical problems that can be addressed using DCOPs and a description of the abstract benchmarking problems commonly used to empirically evaluate the different solution techniques. Section 4 introduces a selection of exact solution algorithms for DCOPs, focusing on two representative examples: (i) ADOPT, as an example of a search based technique and (ii) DPOP, as an example of a dynamic programming based approach. Section 5 discusses the need of approximate solutions for DCOPs describing some representative techniques in this area, and Section 6 discusses quality guarantees for approximate DCOP solution algorithms, focusing on two representatives: the k-optimality framework and the bounded max-sum approach. Finally, Section 7 concludes the chapter.

2 Distributed Constraint Handling

A key element for distributed constraint handling is the concept of the constraint network. Here we provide standard formal definitions relating to constraint networks and then we focus on the distributed constraint processing paradigm itself.

2.1 Constraint Networks

A constraint network $\mathcal{N}$ is formally defined as a tuple $< X, D, C >$, where $X = \{x_1, \cdots, x_n\}$ is a set of *discrete* variables, $D = \{D_1, \cdots, D_n\}$ is a set of variable domains, which enumerate all possible values of the corresponding variables, and $C = \{C_1, \cdots, C_m\}$ is a set of *constraints*. A constraint C_i can be of two types: hard or soft.

A hard constraint C_i^h is a *relation* R_i defined on a subset of variables $S_i \subseteq X$. Variables in S_i are the *scope* of the constraint and the relation R_i enumerates all the *valid* joint assignments of all variables in the scope of the constraint. Therefore R_i is a subset of the Cartesian product of variable domains that are in the constraint's scope: $R_i \subseteq D_{i_1} \times \cdots \times D_{i_r}$ and $r = |S_i|$ is the arity of the relation. A soft constraint C_i^s is a *function* F_i defined again on a subset of variables $S_i \subseteq X$ which comprise the scope of the function. Each function F_i maps every possible joint assignment of all variables in the scope to a real value, therefore $F_i : D_{i1} \times \cdots \times D_{ir} \Rightarrow \Re$ and, as above, $r = |S_i|$ is the arity of the function.

Notice that constraints can be defined over any subset of the variables, however most of the work in constraint networks (both solution algorithms and theoretical analysis) focus on binary constraint networks, where each constraint (soft or hard) is defined over two variables. Actually, it is possible to show that every constraint network can be mapped to a binary constraint network[1], nonetheless here we use

[1]In general this requires the combination or addition of both variables and constraints [4].

the general formalization and specify when the analysis is restricted to the binary case.

When the constraint set involves only hard constraints, the problem we face is known as a constraint satisfaction problem (CSP). In this context, we aim to find an assignment for all the variables in the network that satisfies all constraints. An assignment satisfies a constraint if the tuple of values of the variables in the constraint's scope belongs to the constraint's relation, if $(a_{i_1}, \cdots, a_{i_r}) \in R_i$ where $a_j \in D_j$ and $S_i = \{x_{i_1}, \cdots, x_{i_r}\}$ such an assignment is referred to as a *solution* for the network.

If the constraint set involves soft constraints, then we face a constraint optimization problem (COP) and our aim is to find the *best* solution. That is, an assignment for all variables that satisfies all constraints and that optimizes a global function $F(\bar{a})$. The global function $F(\bar{a})$ is an aggregation of the functions representing the soft constraints (local functions): $F(\bar{a}) = \sum_i F_i(\bar{a}_i)$, where $\bar{a} = (a_1, \cdots, a_n)$ with $a_j \in D_j$, and $\bar{a}_i$ is a restriction of $\bar{a}$ to S_i. A COP can be a maximization or a minimization problem. Without loss of generality, we focus here on maximization problems, therefore our aim is to find the assignment $\bar{a}^*$ that satisfies all hard constraints and such that:

$$\bar{a}^* = \arg\max_{\bar{a}} \sum_i F_i(\bar{a}_i) \tag{12.1}$$

In general, every CSP can be viewed as an optimization task, where we aim to find the assignment that violates the least number of constraints. This is particularly useful for over constrained problems where a solution for the CSP might not exist. Specifically, we can assign a constant fixed cost to every violated constraint and search for an assignment that minimizes the sum of the costs. This problem is known as the *Max-CSP* problem.

Moreover, we can always encode hard constraints using only soft constraints by using infinite values to penalize assignments that do not satisfy hard constraints. For example, assume, without loss of generality, that we are solving a maximization problem, and let R_i be a relation that corresponds to a hard constraint C_i. We can construct a function $F_i(\bar{a}) = -\infty$ if $\bar{a} \notin R_i$ and $F(\bar{a}) = 0$ otherwise[2].

[2]Notice that this may result in inferior solution techniques, as explicit hard constraints might be exploited to reduce the solution search space [9].

2.2 Distributed Constraint Processing

Distributed constraint processing extends the standard constraint processing framework by considering a set of agents that have control over variables and interact to find a solution to the constraint network. As before, we can have satisfaction or optimization tasks resulting in two types of problems: distributed CSP (DCSP) and distributed COP (DCOP). The DCSP paradigm was originally proposed to address coordination problems in a multi-agent setting [41], however in recent years the DCOP framework has received increasing attention as it can better represent many practical application scenarios.

Formally a DCOP can be represented as a constraint network $\mathcal{N} =< X, D, C >$ containing soft constraints, plus a set of agents $A = \{A_1, \cdots, A_k\}$. Each agent can control only a subset of the variables $X_i \subseteq X$, and each variable is assigned to exactly one agent. In other words the assignment of variables to agents must be a partition of the set of variables. Agents can *control* only the variables assigned to them, meaning that they can observe and change the values of their assigned variables only. Moreover, agents are only aware of constraints that involve variables that they can control. Such constraints are usually termed local functions and the sum of these local functions is the local utility of the agent. Finally, two agents are considered neighbors if there is at least one constraint that depends on variables that each controls. Only neighboring agents can directly communicate with each other.

Within this context, the goal for the agents is to find the optimal solution to the constraint network, i.e. to find the assignment for all the variables in the system that optimizes the global function. Thus, in a standard DCOP setting, agents are assumed not to be self-interested, i.e. their goal is to optimize the *global* function and not their local utilities.

Finding an optimal solution for a DCOP is an NP-Hard problem, which can be seen by reducing a DCOP to the problem of deciding on the 3-colorability of a graph; a problem known to be NP-Complete [26].

In the next section we will present a number of practical problems that can be addressed using the DCOP framework, as well as some exemplar and benchmarking DCOP instances.

3 Applications and Benchmarking Problems

To provide concrete examples of how the DCOP framework can be applied to real world scenarios we report here on a number of practical problems that can be successfully addressed using the DCOP framework discussed above. Following this we go on to show a set of abstract benchmarking problems that are commonly used to evaluate and compare DCOP solution techniques.

3.1 Real World Applications

Many real world applications can be modeled using the DCOP framework, ranging from human-agent organizations to sensor networks and robotics. Here we focus on two such applications that have been frequently used as motivating scenarios for work in the MAS literature.

3.1.1 Meeting Scheduling

The problem of scheduling a set of tasks over a set of resources (e.g., schedule a set of lectures over a set of lecture halls or a set of jobs to a set of processors) is a very common and important problem that can be conveniently formalized using constraint networks.

A typical example of this is the meeting scheduling problem, which is a very relevant problem for large organizations (e.g., public administration, private companies, research institutes etc.) where many people, possibly working in different departments, are involved in a number of work meetings. In more detail, people involved in a meeting scheduling problem might have various private preferences on meeting start times, for example a given employee might prefer his/her meetings to start in the afternoon rather than in the morning (to happily conjugate a late night social life with work!). Given this, the aim is to agree on a *valid* schedule for the meeting while maximizing the sum of the individuals' private preferences. To be valid, a schedule must meet obvious hard constraints, for example two meetings that share a participant cannot overlap.

A possible DCOP formalization for the meeting scheduling domain involves a set of agents representing the people participating in the meeting and a set of variables that represent the possible starting time of a given meeting according to a participant. Constraints force equality on variables that represent the starting time of the same meeting across different agents and ensure that variables that represent the starting times of different meetings for the same agent are non-overlapping. Finally, preferences can be represented as soft constraints on meeting starting times and the overall aim is to optimize the sum of all the soft constraints. Notice that, while in this setting we do have private preferences, we are maximizing the sum of preferences of all the agents, and thus we are still considering a scenario where agents are fully cooperative, i.e. they are willing to diminish their own local utility if it will maximize the global utility.

While this problem could be easily formalized as a centralized COP, in this case a distributed approach not only provides a more robust and scalable solution, but it can also minimize the amount of information agents must reveal to each other (thus preserving their privacy). This is because, as mentioned above, in a DCOP, agents are required to be aware only of constraints that they are involved in. For example, consider a situation where Alice must meet Bob and Charles in two separate meetings. In a centralized approach Alice would have to reveal the list of people she has to meet with. On the other hand, in a DCOP only people involved in any particular meeting will be aware that the meeting is taking place. Thus in our example, Bob does not need to know that Alice will also meet with Charles.

3.1.2 Target Tracking

Target tracking is a crucial and widely studied problem for surveillance and monitoring applications. It involves a set of sensors tracking a set of targets in order to provide an accurate estimate of their positions. Sensors can have different sensing modalities that impact on the accuracy of the estimation of the targets' positions. For example a pan, tilt and zoom (PTZ) camera could move to focus on a specific area of the environment, reducing observation uncertainty for targets in that area. Moreover, collaboration among sensors is crucial to improve the performance of the system. For example two cameras could decide to track different targets to maximize coverage of the environment, or to both focus on a potentially dangerous target, thus providing a more accurate estimate of its position.

There are several ways to formalize this scenario using the DCOP framework. A widely accepted formulation is one where sensors are represented by agents and variables encode the different sensing modalities of each sensor. Constraints are usually defined among sensors that have an overlapping sensing range. Each constraint relates to a specific target and represents how the joint choice of sensor modalities impacts on the tracking performance for that target. Constraints can specify the minimum number, or particular number, of sensors that are required to correctly identify a target, or provide a measure of position tracking accuracy for each possible combination of agents' sensing modalities. The global function for the DCOP is the sum of constraints' values. For example, the system could aim to maximize the number of targets correctly identified or to maximize the sum of tracking accuracy over all targets.

Within this context, the main reasons for a using distributed approach for the optimization problem are robustness and scalability, which are both crucial issues in surveillance and monitoring applications. Specifically, a distributed solution improves scalability as it exploits the decomposition of the problem to reason locally. Thus reducing the communication and computation that each agent must perform. This is very important as this type of application typically involves the use of hardware devices that have inherent constraints on communication and computation. Furthermore, robustness is enhanced as each agent decides on its own sensing modalities, thus avoiding a central point of failure.

3.2 Exemplar and Benchmarking Problems

As previously remarked, finding an optimal solution for a DCOP is known to be an NP-Hard problem. Therefore empirical evaluation of DCOP solution techniques is a crucial point in order to evaluate their likely practical impact. In particular, to have a meaningful comparison between the different solution techniques it is essential to be able to run the various programs on shared test bed. This problem has been addressed by the DCOP community using benchmarking problem instances inspired by practical applications, such as meeting scheduling and target tracking[3].

In addition, there are also a number of exemplar NP problems that are frequently used to test solution techniques such as propositional satisfaction (SAT) or graph coloring. Here we focus on the latter problem as it has been widely used to evaluate the techniques that will be presented later in this chapter and is particularly useful for illustrative purposes.

The graph coloring problem is an extremely simple problem to formulate and is attractive since the computational effort associated with finding the solution can be easily controlled using few parameters (e.g., number of available colors, and the ratio of number of constraints to the number of nodes). The constraint satisfaction version of a graph coloring problem can be described as follows: given

[3]For example see the repository of shared DCOP benchmarking problems created and maintained by the Teamcore research group, available at `http://teamcore.usc.edu/dcop/`.

a graph of any size and k possible colors, decide whether the nodes of the graphs can be colored with no more than k colors, so that any two adjacent nodes have different colors.

In the CSP formulation of the graph coloring problem, nodes are variables, the set of k colors is the variable domain (which is the same for all the variables) and constraints are *not-equal* constraints that hold between any adjacent nodes. An assignment is a map from nodes to colors without constraint violations. The optimization version is a max-CSP problem where the aim is to minimize the number of constraint violations. The optimization version of the graph coloring problem can be generalized in many ways, for example by assigning different weights to violated constraints or by giving different values to conflict violations based on the color that causes the conflict (e.g., a penalty of 1 if both nodes are blue and a penalty of 10 if both nodes are red).

4 Solution Techniques: Complete Algorithms

Given the previous description of a DCOP, we now focus on complete solution techniques, i.e. those that always find a solution that optimizes the global objective function. These techniques are particularly interesting and elegant from a theoretical point of view, but since we are dealing with an NP-Hard problem they also exhibit an exponentially increasing coordination overhead (either in the size and/or number of messages exchanged, or in the computation required by each agent) as the number of agents in the system increases.

Broadly, these complete approaches can be divided in two classes: those that are search based [25, 7, 15, 10, 17], and those that exploit dynamic programming [30]. Moreover, search based approaches can be further divided between synchronous ones, such as SyncBB [17] and AND/OR search [10][4], and asynchronous ones, such as ADOPT [25], NCBB [7] and AFB [15]. In the synchronous execution model, agents wait for messages from other agents before computing and sending out new messages themselves. In contrast, in the asynchronous execution model agents perform computation and send out messages without waiting for messages from their neighbors. Asynchronous operation is desirable in a multi-agent context as it allows agents to take decisions without waiting for others agent to complete their computation, thus fully exploiting parallel computation. On the other hand, the synchronous model ensures that agents

[4]AO search was originally developed for centralized optimization problems but can easily be extended to work in decentralized settings, see [28].

always have the most relevant and up to date information before executing their computation, thus minimizing redundancy in both computation and communication.

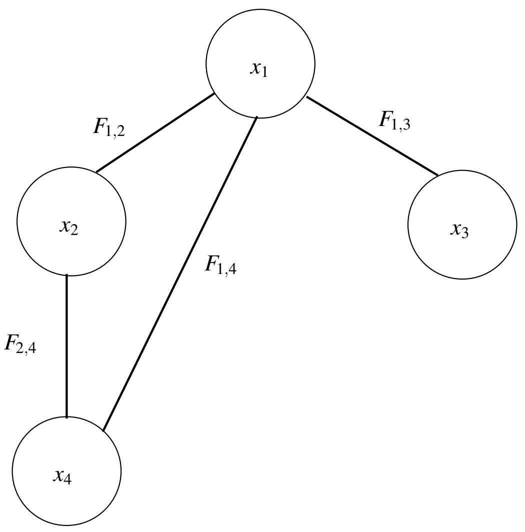
All the above techniques are completely decentralized, in the sense that each agent has complete control over its variables and is aware of only local constraints. However, centralizing part of the problem can sometimes reduce the effort required to find a globally optimal solution. This is the key concept behind the Optimal Asynchronous Partial Overlay (optAPO) approach [23]. In more detail, optAPO aims to discover parts of the problem that are particularly hard to solve in a decentralized fashion (parts that are strongly interconnected) and centralizes them into sub problems that are delegated to mediator agents (which act as centralized solvers). OptAPO has been shown to consistently reduce the communication overhead with respect to other decentralized techniques such as ADOPT. However, it is very hard to control how much of the problem will be centralized, and thus, it is difficult to predict the computational effort that mediator agents must be able to sustain.

Here we focus on two decentralized approaches that are good representatives of the two general solution classes: ADOPT for search based techniques and DPOP for dynamic programming.

4.1 Search Based: ADOPT

ADOPT (Asynchronous Distributed OPTimization) was proposed by Modi et al. and both guarantees solution optimality and allows agents to asynchronously change the values of the variables that they control [24, 25]. As it is common in the DCOP literature, the original ADOPT formulation assumes that each agent controls only one variable and that the constraints are binary. While there are ways to remove these assumptions without significantly changing the algorithm, most of the work related to the ADOPT technique falls within this setting, therefore in what follows we embrace these assumptions. Moreover, to maintain a close correspondence with the original ADOPT description we assume that our task here is a minimization problem. Hence constraints represent costs and agents wish to find an assignment that minimizes the sum of these costs.

ADOPT is a search based technique that performs a distributed backtrack search using a best-first strategy; each agent always assigns to its variable the best value based on local information. The key components of the ADOPT algorithm are: (i) local lower bound estimates, (ii) backtrack thresholds and (iii) termination conditions. In particular, each agent maintains a lower bound estimate for each



$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

Figure 12.1: Exemplar constraint network.

possible values of its variable. This lower bound is initially computed based only on the local cost function, and is then refined as more information is passed between the agents. Each agent will choose the value of its variable that minimizes this lower bound, and this decision is made asynchronously as soon as the local lower bound is updated.

Backtrack thresholds are used to speed up the search of previously explored solutions. This can happen because the search strategy is based on local lower bounds, and thus, agents can abandon values before they are proven to be suboptimal. Backtrack thresholds are lower bounds that have been previously determined and can prevent agents from exploring useless branches of the search tree.

Finally, ADOPT uses a bound interval to evaluate the search progress. Specifically, each agent maintains not only a lower bound but also an upper bound on the optimal solution. Therefore, when these two values agree, the search process can terminate and the current solution can be returned. In addition, this feature can be used to look for solutions that are sub-optimal but within a given predefined bound of the optimal solution. The user can specify a valid error bound (i.e. the distance between the optimal solution value and an acceptable suboptimal solution) and as soon as the bound interval becomes less than this value the search process can be stopped.

Before executing the ADOPT algorithm, agents must be arranged in a depth

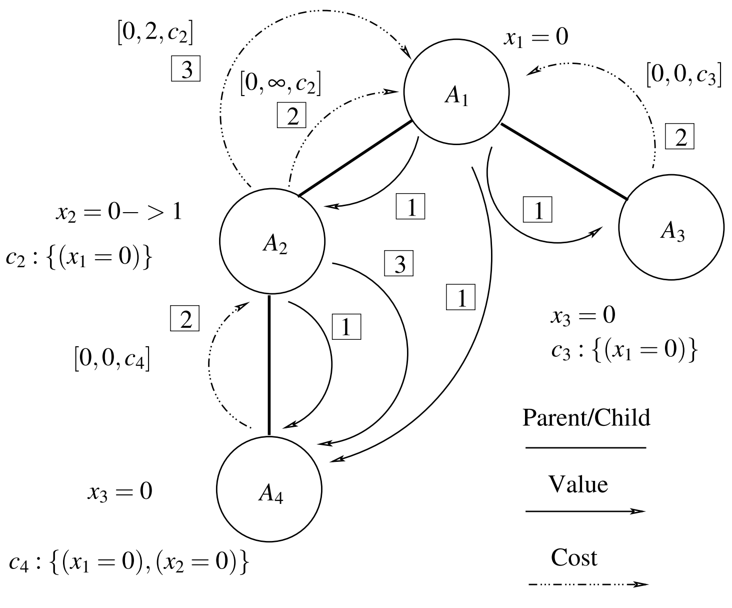


Figure 12.2: Message exchange in the ADOPT algorithm: *Value* and *Cost* messages for one possible trace of execution. Numbers within squares indicate the (partial) order of the messages.

first search (DFS) tree. A DFS tree order is defined by considering direct parent-child relationship between agents. DFS tree orderings have been frequently used in optimization (see for example [30]) because they have two interesting properties: (i) agents in different branches of the tree do not share any constraints, (ii) every constraint network can be ordered in a DFS tree and this can be done in polynomial-time with a distributed procedure [28]. The fact that agents in different branches do not share constraints is an important property as it ensures that they can search for solutions independently of each other. Figure 12.1 shows an exemplar constraint network and Figure 12.2 reports a possible DFS order, where solid lines show parent child relationships and constraints are not represented. Given a constraint network, the DFS ordering is not unique and ADOPT's performance (in terms of coordination overhead) depends on the actual DFS ordering used. Finding the optimal DFS tree is a challenging problem that the ADOPT technique does not address.

Given a DFS ordering of the agents, the algorithm proceeds by exchanging three types of messages: *Value*, *Cost* and *Threshold*. When the algorithm starts, all agents choose a random value for their variables, and initialize the lower bound

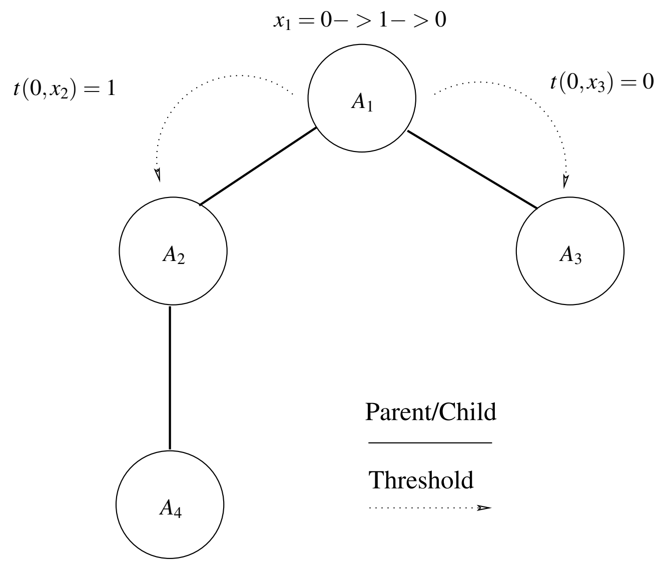


Figure 12.3: Message exchange in the ADOPT algorithm: *Threshold* messages for a revisited context.

and upper bound of their variables' possible values to zero and infinity respectively. These bounds are then iteratively refined as more information is transmitted across the network. Figures 12.2 reports messages exchanged among the agents during the first stages of the algorithm. Since the algorithm is asynchronous, we report here one possible trace of execution and numbers within squares indicate the (partial) order of the messages.

In more detail, *Value* messages are sent by an agent to all its neighbors that are lower in the DFS tree order than itself, reporting the value that the agent has assigned to its variable. For example, in Figure 12.2 agent A_1 sends three value messages to A_2, A_3 and A_4 informing them that its current value is 0. Notice that this message is sent to A_4 even though there is no parent/child relationship between A_1 and A_4 because A_4 is a neighbor of A_1 that is lower in the DFS order.

Cost messages are sent by an agent to its parent, reporting the minimum lower and upper bound across all the agent's variable values, and the current context. The current context is a partial variable assignment, and in particular, it records the assignment of all higher neighbors. For example, in Figure 12.2 the current context for A_4, c_4, is $\{(x_1,0),(x_2,0)\}$. The minimum lower bound and minimum upper bound are computed with respect to the current context. To compute the minimum lower bound each agent evaluates its own local cost for each possible value of its variable, adding all the lower bound messages received from children that are compatible with the current variable value. The local cost for an agent is the sum of the values of local cost function for all the higher neighbors. For example, consider the cost message sent by A_4. The minimum lower bound (which is 0) is computed by finding the minimum between $\delta(x_4 = 0) = 4$ and $\delta(x_4 = 1) = 0$ where $\delta(a)$ is the local cost function when the variable assumes the value a. The local cost function is computed by summing up the values of the cost functions for all neighbors higher in the DFS order and by assigning their values according to the current context. A similar computation is performed for the upper bound.

Cost messages for agents that are not leaves of the DFS tree (e.g., A_2) also include the lower and upper bound for each child. For example consider the cost message sent by A_2 to A_1 and let **LB** represent the minimum lower bound across all variable's values. **LB** is then computed by finding the minimum between $LB(x_2 = 0) = \delta(x_2 = 0) + lb(x_2 = 0, x_4) = 2$ and $LB(x_2 = 1) = \delta(x_2 = 1) + lb(x_2 = 1, x_4) = 0$, resulting in **LB** = 0. Here $lb(a, x_l)$ is the lower bound for the child variable x_l when the current variable is assigned to a in the current context.

Threshold messages are sent from parents to children to update the agent's backtrack thresholds. Backtrack thresholds are particularly useful when a previously visited context is re-visited. Each agent stores cost information (e.g., upper and lower bounds) only for the current context and deletes previously stored information as soon as the context changes. In fact, if an agent did maintain such information for every visited context it will need an exponential space in memory. However, since a context might be visited multiple times during the search process, whenever this happens the agent starts computing cost information from scratch. Now, since this context was visited before, the agent reported the sum of cost information to its parent and since the parent has that information stored, it can now send it back to the agent via a threshold message. The threshold message is used to set the agent's threshold to a previous valid lower bound, and propagate cost information down the tree, avoiding needless computation. Notice that the information that the parent stores is the accumulated cost information. Therefore, to propagate information down the tree, the agent must subdivide this accumulated cost across its children using some heuristic, as the original cost subdivision is lost, and then correct this subdivision over time as cost feedback is received from the children. For example, assume that during the search process, x_1 changes its value and then the context with $x_1 = 0$ is visited again. Agent A_1 will then send threshold messages to A_2 and A_3 as shown in Figure 12.3. Notice that the value of these messages is the lower bound value sent by the corresponding child agent for that context, e.g. the message $t(x_1 = 0, x_2)$ equals the lower bound sent by A_2 to A_1 with context $\{(x_1, 0)\}$.

Finally, agents asynchronously update a variable's value whenever the stored lower bound for the current value exceeds the backtrack threshold and the new variable's value is the one that minimizes the stored lower bounds. For example, consider agent A_2 in Figure 12.2, when receiving the cost message from A_1. In this case, the lower bound for the current value ($LB(x_2 = 0) = 2$) exceeds the threshold (initially set to 0). Therefore, the agent updates its variable value to the one that minimizes the lower bound, which in our case is $x_2 = 1$. It then sends cost and value messages accordingly. When the minimum lower bound for a variable value is also an upper bound for that value, the agent can stop propagating messages as that value will be optimal given the current context. When this condition is true at the root agent, a terminate message is sent to all the children. Agents propagate the termination message if the termination condition is true for them as well. When the terminate message has propagated to all the agents, the algorithm stops, and the optimal solution has been found.

ADOPT is particularly interesting because it is asynchronous and because the memory usage of each agent is polynomial in the number of variables. Moreover, messages are all of a fixed size. However, the number of messages that agents need to exchange is, in the worst case, exponential in the number of variables. This impacts on the time required to find the optimal solution. In particular, the number of message synchronization cycles, defined as all agents receiving incoming messages and sending outgoing messages simultaneously, is exponential in the number of variables. This is a frequently used measure to evaluate DCOPs solution techniques as it is less sensitive to variations in agents' computation speed and communication delays than the wall clock. As previously remarked, such exponential elements are unavoidable in complete approaches and they can severely restrict the scalability of the approach.

Several works build on ADOPT attempting to reduce computation time. For example, Yeoh et al. propose BnB-ADOPT, which is an extension of ADOPT that consistently reduces computation time by using a different search strategy; depth first search with branch and bound instead of best first search [39]. Moreover, Ali et al. suggest the use of preprocessing techniques for guiding ADOPT search and show that this can result in a consistent increase in performance [3]. Finally, Gutierrez and Meseguer show that many messages that are sent by BnB-ADOPT are in fact redundant and most of them can be removed resulting in significant reduction in communication costs [16].

In the next section we describe a completely different approach based on dynamic programming.

4.2 Dynamic Programming: DPOP

DPOP (Dynamic Programming Optimization Protocol) was proposed by Petcu and Falting [30], and is based on the dynamic programming paradigm, and more specifically, on the Bucket Elimination (BE) algorithm [9].

In more detail, BE is a dynamic programming approach for solving both constraint networks and also more general graphical models such as Bayesian networks, Markov random fields and influence diagrams. BE takes as input a constraint network and an ordering of the network variables. It then associates a bucket to each variable and partitions the constraints; assigning them to the bucket following the given ordering. The optimal assignment is obtained by running two phases. First, the buckets are processed (from last to first), essentially running a variable elimination algorithm. Specifically, when processing a bucket, all constraints in the bucket are summed together and the variable that corresponds to the bucket is eliminated by maximization. This results in a new constraint that is placed in the first bucket, following the specified order, that contains one of the variables that are in the constraint scope. When the first phase is completed, the optimal value for the variable associated with the first bucket can be computed. This optimal value can be fixed, and then, given this value, the optimal value for the next variable in the ordering can be found. Proceeding in this way the entire optimal assignment is generated.

Now, while BE is normally defined over a linear ordering of the variables, it can be extended to work on a tree ordering via message passing between the nodes; resulting in the Bucket Tree Elimination Algorithm (BTE) [9]. Against this background, DPOP can essentially be seen as a special case of BTE that operates on a DFS tree ordering of the constraint network. This specific arrangement is important because it ensures that during the optimization process, agents have knowledge of, and can control, only their own variables, and that they communicate only with other agents that share at least one constraint.

The DPOP algorithm can be divided in to three phases: (i) arrangement of the variables into a DFS tree; (ii) propagation of *Util* messages bottom-up along the DFS tree (from leaves to root); and (iii) propagation of *Value* messages top-down (from root to leaves). We will briefly discuss these three phases in the following. As with the ADOPT example, for ease of presentation, we assume that each agent controls only one variable and that constraints are binary. However, relaxing this assumption does not result in significant changes to the algorithm. In contrast to

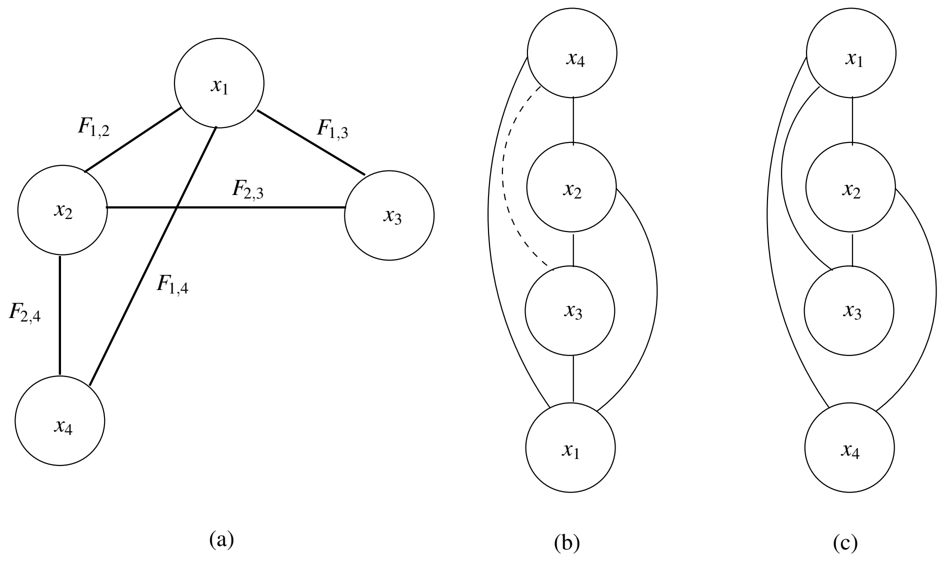


Figure 12.4: (a) Exemplar constraint network, with (b) induced graph with DFS order $\{x_4, x_2, x_3, x_1\}$, and (c) induced graph with DFS order $\{x_1, x_2, x_3, x_4\}$

the description of ADOPT, and in line with the original description of DPOP, we deal here with maximization problems.

As with ADOPT, DPOP first pre-processes the constraint network to create a DFS tree. In contrast to ADOPT, however, DPOP guarantees that the optimal solution can be obtained with a linear number of messages; resulting in messages whose size is exponential in the induced width of the DFS tree ordering.

More specifically, DPOP can operate on a pseudo-tree ordering of the constraint network. A pseudo-tree ordering is one where nodes that share a constraint fall in the same branch of the tree. DFS tree ordering is thus a special case of a pseudo-tree that can be easily obtained with a DFS traversal of the original graph. Now, the complexity of the DPOP algorithm is strongly related to the DFS arrangement on which the algorithm is run, and in particular, it is exponential in the *induced width* of the DFS tree ordering. Given a graph and an ordering of its nodes, the width induced by the ordering is the maximum induced width of a node, which is simply given by how many parents it has in the induced graph. The induced graph can be computed by processing the nodes from last to first, and for each node, adding edges to connect all the parents of that node (i.e. neighbors that precede the node in the order).

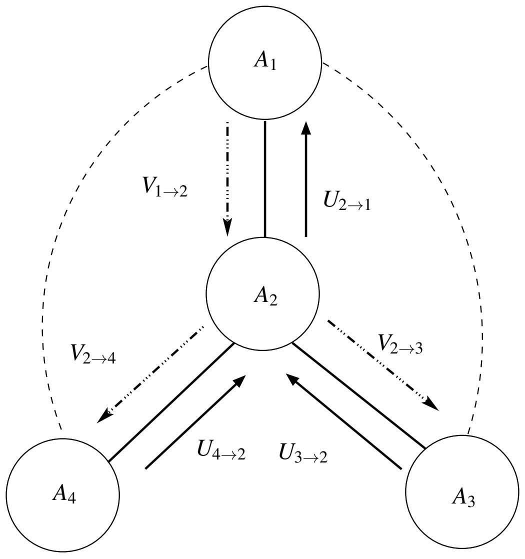


Figure 12.5: Message exchange in DPOP.

In particular, Figure 12.4 shows a constraint network and two induced graphs given by different orderings. The induced width for the graph in Figure 12.4(b) is 3, while the induced width for the graph in Figure 12.4(c) is 2. Notice the dashed edge between x_3 and x_4 in Figure 12.4(b) that was added when building the induced graph. While there are various heuristics to generate DFS orderings with small induced width, finding the one with minimal induced width is an NP-Hard problem. Figure 12.5 report a DFS arrangement for the constraint network shown in Figure 12.4(a) along with messages that will be exchanged during the following phases. Dashed edges represent constraints that are part of the constraint network but are not part of the DFS tree. These are usually called back-edges.

Once the variables have been arranged in a DFS tree structure, the *Util* propagation phase starts. *Util* propagation goes from leaves, up the tree, to the root node. Each agent computes messages for its parent considering both the messages received from its children and the constraints that the agent is involved in. In general, the *Util* message $U_{i\to j}$ that agent A_i sends to its parent A_j can be computed according to the following equation:

$$U_{i\to j}(Sep_i) = \max_{x_i}\left(\bigoplus_{A_k\in C_i} U_{k\to i} \oplus \bigoplus_{A_p\in P_i\cup PP_i} F_{i,p}\right) \tag{12.2}$$

where C_i is the set of children for agent A_i, P_i is the parent of A_i, PP_i is the set of agents preceding A_i in the pseudo-tree order that are connected to A_i through a back-edge (pseudo parents), and Sep_i is the set of agents preceding A_i in the pseudo-pstree order that are connected with A_i or with a descendant of A_i[5]. The $\oplus$ operator is a join operator that sums up functions with different but overlapping scopes consistently, i.e. summing the values of the functions for assignments that agree on the shared variables. For example, considering again Figure 12.5, agent A_3 sends the message $U_{3\rightarrow 2}(x_1,x_2) = max_{x_3}(F_{1,3}(x_1,x_3) \oplus F_{2,3}(x_2,x_3))$ because there are no messages from its children, while agent A_2 sends the message $U_{2\rightarrow 1}(x_1) = max_{x_2}(U_{3\rightarrow 2}(x_1,x_2) \oplus U_{4\rightarrow 2}(x_1,x_2) \oplus F_{1,2}(x_1,x_2))$. It is possible to show that the size of the largest separator in a DFS tree equals the induced width of the tree, which clarifies the exponential dependence on the induced width of message size.

Finally, the *Value* message propagation phase builds the optimal assignment proceeding from root to leaves. Root agent A_r computes x_r^* which is the argument that maximizes the sum of the messages received by all its children (plus all unary relations it is involved in) and sends a message $V_{r\rightarrow c} = \{x_r = x_r^*\}$ containing this value to all its children $A_c \in C_r$. The generic agent A_i computes $x_i^* = \arg\max_{x_i}(\sum_{A_j \in C_i} U_{j\rightarrow i}(\mathbf{x}_p^*) + \sum_{A_j \in P_i \cup PP_i} F_{i,j}(x_i, x_j^*))$, where $\mathbf{x}_p^* = \bigcup_{A_j \in P_i \cup PP_i}\{x_j^*\}$ is the set of optimal values for A_i's parent and pseudoparents received from A_i's parent. Finally, the generic agent A_i sends a message to each child A_j with value $V_{i\rightarrow j} = \{x_i = x_i^*\} \cup \bigcup_{x_s \in Sep_i \cap Sep_j}\{x_s = x_s^*\}$. For example, assume agent A_1's optimal value is $x_1^* = 1$, then agent A_2 computes $x_2^* = \arg\max_{x_2}(U_{3\rightarrow 2}(1,x_2) \oplus U_{4\rightarrow 2}(1,x_2) \oplus F_{1,2}(1,x_2))$ and propagates the message $\{(x_1 = 1),(x_2 = x_2^*)\}$ to agents A_3 and A_4. Notice that the maximization performed by agent A_4 in the value propagation phase is the same as the one previously done to compute the *Util* messages, but now with the aim to find the value that maximizes the equation. Hence computation can be reduced by storing the appropriate values during the *Util* propagation phase.

As discussed above, DPOP message size, and hence the computation that agents need to compute them, is exponential. However, it is only exponential in the induced width of the DFS tree ordering used, that, in general, is much less than the total number of variables. Furthermore, there are many extensions of DPOP that address various possible trade-offs in the approach. In particular, MB-DPOP exploits the cycle-cut set idea to address the trade-off between the number

[5] This set is called the separator because it is precisely the set of agents that should be removed to completely separate the subtree rooted at A_i from the rest of the network.

of messages used and the amount of memory that each message requires [31]. On the other hand, A-DPOP addresses the trade-off between message size and solution quality [29]. Specifically, A-DPOP attempts to reduce message size by optimally computing only a part of the messages and approximating the rest (with upper and lower bounds). Given a fixed approximation ratio, A-DPOP can then reduce message size to meet this ratio, or alternatively, given a fixed maximum message size, it propagates only those messages that do not exceed that size.

As a final remark, note that there is a close relationship between DPOP and the Generalized Distributive Law (GDL) framework which we shall discuss further in Section 5.2 [2]. GDL represents a family of techniques frequently used in information theory for decoding error correcting codes[6] [21], and solving graphical models (e.g., to find the maximum *a posteriori* assignment in Markov random fields [37], or the posterior probabilities [38]).

5 Solution Techniques: Approximate Algorithms

As discussed earlier, solving a constraint network is an NP-Hard problem. Therefore the worst case complexity of complete methods are often prohibitive for practical applications. This is particularly the case for applications involving physical devices, such as sensor networks or mobile robots, which have severe constraints on memory and computation.

In these settings, approximate algorithms are often preferred, as they require very little local computation and communication, and are, as such, well suited for large scale practical distributed applications in which the optimality of the solution can be sacrificed in favor of computational and communication efficiency (see [6] for a review of such algorithms). Furthermore, such approximate techniques, have been shown to provide solutions which are very close to optimality in several problem instances [12, 22]. However, these approaches do not provide guarantees on the solution quality in general settings. This is particularly troublesome because the quality of solution to which most approximate algorithms converge is highly dependent on many factors which cannot always be properly assessed before deploying the system. Therefore there is no guarantee against particularly adverse behavior on specific pathological instances.

Thus, we next describe two classes of approximate algorithms for addressing DCOPs: local greedy methods and GDL based approaches.

[6]Decoding turbo codes is probably the most important representative application for which GDL techniques are used. See [21], chapter 48.4 for details.

5.1 Local Greedy Approximate Algorithms

A local greedy search starts with a random assignment for all the variables and then performs a series of local moves trying to greedily optimize the objective function. A local move usually involves changing the value of a small set of variables (in most cases just one) so that the difference between the value of the objective function with the new assignment and the previous value is maximized. This difference is usually called the *gain*. The search stops when there is no local move that provides a positive gain, i.e., when the process reaches a *local* maximum. Local greedy search is a very popular approximate optimization technique, as it requires very little memory and computation, and can obtain extremely good solutions in many settings. The main problem for this type of approach is the presence of *local* maxima that can, in general, be arbitrarily far from the global optimal solution. Many heuristics can be used to avoid local maxima such as using random restart (sometimes called Stochastic Local Search [9]) or introducing stochastic steps in the search process (resulting in algorithms such as walkSAT and Simulated Annealing [9]).

Greedy local search methods have been widely used for DCOPs resulting in many successful approaches [12, 22]. When operating in a decentralized context an important issue for these techniques is that to execute a greedy local move agents need some type of coordination. In fact, the gain for a local move involving a variable x_i is computed assuming that all other variables $X \setminus \{x_i\}$ do not change their values. If all agents are allowed to execute in parallel, a potentially greedy move can become harmful (i.e., result in negative gain) because each agent has out-of-date knowledge about the choices of other agents. Such incoherence may also compromise the convergence of the approach leading to thrashing behaviors.

5.1.1 The Distributed Stochastic Algorithm

A simple and effective way to reduce such incoherence is to introduce a stochastic decision on whether agents should actually perform a move when they see the opportunity to optimize the gain [12]. This approach is usually called the Distributed Stochastic Algorithm (DSA) and has been widely studied and applied in many domains. In more detail, assuming a synchronous execution model (each agent waits for the messages from all its neighbors), DSA has an initialization phase, where each agent chooses a random value for its variable, and then an infinite loop is executed by each agent. At each execution step, each agent A_i executes the following operations:

- Choose an activation probability $p_i \in [0,1]$.
- Generate a random number $r_i \in [0,1)$.
- If $r_i < p_i$ choose a value a_i such that the local gain is maximized.
- If $r_i \geq p_i$ do not change its value.
- If the variable value changed, send information to all neighbors notifying them of the change.
- Receive messages from neighbors and update information accordingly.

DSA can also be used in an asynchronous context and empirical results show that the algorithm is still effective if the rate of variable change is low with respect to the communication latency, thus allowing information to be propagated coherently in the system. Moreover, in most work, the activation probability is not decided at each optimization step, but is fixed at the beginning of the execution and is the same for all agents. The main strength of the DSA algorithm is its extremely low overhead in terms of memory, computation and communication. In fact, each agent needs to store and reason only about its direct neighbors which, in general, are far fewer than the total number of agents in the system. Moreover, there is no exponential increase in computation and communication, as the optimization step considers only the current values of neighbors, and the communication step involves a message to communicate just the new value of the agent's variable. Finally, empirical results show that the algorithm typically monotonically increases the solution quality with each execution step, resulting in anytime behavior that is very well suited for practical applications; a solution is always available and the longer an agent waits before acting, the better the solution will be. Note, however, that there is no theoretical guarantee of such anytime behavior, but this is often obtained in practice with a suitable tuning of the activation probability.

The main drawback of DSA is that the solution quality can be strongly dependent on the activation probability, and there is no way to compute its value from an analysis of the problem instance. Moreover, the sensitivity of the algorithm's performance, with respect to the activation probability, is domain dependent, and it is hard to generalize the behavior of the algorithm across different domains.

5.1.2 The Maximum Gain Message Algorithm

An alternative approach to address the possible out-of-date knowledge about other agents' variable values, is for neighboring agents to agree on who is the agent that can perform a move, while the others wait without changing their values. This approach is the basic idea behind the Maximum Gain Message algorithm (MGM) [22]. MGM is based on the well known Distributed Breakout Algorithm (DBA), but is adapted to avoid outdated knowledge of the agent about its neighbours. Assuming again a synchronous execution model, at each execution step each agent A_i executes the following operations:

- Send its current value a_i to neighbors and receive values from neighbors.
- Choose a value a_i^* such that the local gain g_i^* is maximised (assuming neighbors do not change value).
- Send the gain g_i^* to neighbors and receive gain from neighbors.
- If the gain for the agent is the highest in the neighborhood, update the value of x_i to a_i^*.

MGM is guaranteed to be anytime and several empirical evaluations show that it has comparable performance with respect to DSA in various domains [22]. Moreover, unlike DSA, MGM does not require any parameter tuning.

A common characteristic of both DSA and MGM, is that decisions are made considering local information only, i.e., when deciding the next value for its variable each agent optimizes only with respect to the current assignments of its neighbors. This provides for extremely low cost and scalable techniques, however, solution quality is strongly compromised by local maxima that can, in general, be arbitrarily far from the optimal solution. In the next section, we present an algorithm for solving DCOP, based on the Generalised Distributive Law (GDL) framework, that overcomes this limitation.

5.2 GDL Based Approximate Algorithms

As previously mentioned, the Generalised Distributive Law (GDL) is a unifying framework to perform inference in graphical models. Specifically, the GDL framework operates on commutative semi-rings, and depending on the specific semi-ring used, we obtain different algorithms such as the max-sum, max-product, or sum-product. Such algorithms are widely used to perform inference tasks such as finding the maximum *a posteriori* assignment in Markov random fields (max-product) [37], or computing marginal distributions in Bayesian networks (sum-product or belief propagation) [21]. In particular, the max-sum algorithm can be

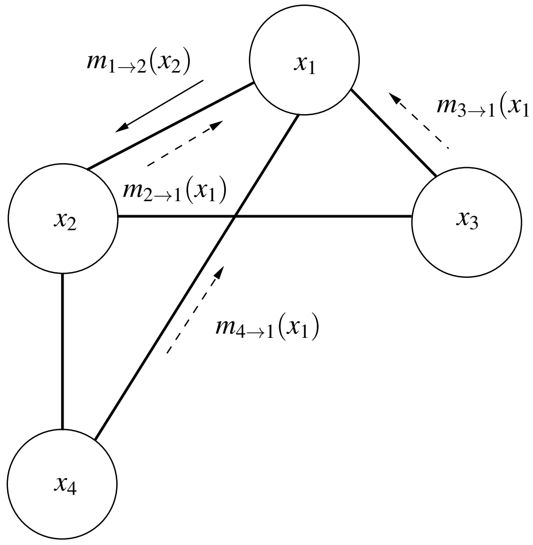


Figure 12.6: Message exchange in max-sum.

used to solve constraint networks as it can find the argument that maximizes a global optimization function expressed as the sum of local functions.

5.2.1 The Max-Sum Algorithm

In more detail, the max-sum algorithm is an iterative message passing algorithm, where agents continuously exchange messages to build a local function that depends only on the variables they control. This function represents the dependence of the global function on the agents' values and is used to find the optimal assignment. The max-sum algorithm can directly handle n-ary constraints and more variables per agent, however for ease of presentation, we report here a description of the algorithm in line with the earlier assumptions that each agent controls one variable and all constraints are binary. The interested reader can find the description of the algorithm in more general settings in [32]. Finally, we again assume a synchronous execution model.

Given this, at each execution step each each agent A_i updates and sends to each of its neighbors A_j the message, $m_{i\to j}(x_j)$, given by:

$$m_{i\to j}(x_j) = \alpha_{ij} + \max_{x_i}\left(F_{ij}(x_i,x_j) + \sum_{k\in N(i)\setminus j} m_{k\to i}(x_i)\right) \tag{12.3}$$

where α_{ij} is a normalization constant, $N(i)$ is the set of indices for variables that are connected to x_i, and F_{ij} is the constraint defined over the variables controlled by A_i and A_j. The normalization constant α_{ij} is added to all the components of the message so that $\sum_{x_j} m_{i\to j}(x_j) = 0$. This is necessary on graphs with loops because otherwise message values might grow indefinitely, possibly leading to numerical errors.

At the first iteration all messages are initialized to constant functions, and at each subsequent iteration, each agent A_i aggregates all incoming messages and computes the local function, $z_i(x_i)$, which is given by:

$$z_i(x_i) = \sum_{k\in N(i)} m_{k\to i}(x_i) \tag{12.4}$$

This is then used to obtain the max-sum assignment, $\tilde{x}$, which, for every variable $x_i \in X$ is given by:

$$\tilde{x}_i = arg\max_{x_i} z_i(x_i) \tag{12.5}$$

Figure 12.6 shows input and output messages for agent A_1. In this example, the message to agent A_2 is computed as $m_{1\to 2}(x_2) = \max_{x_1}(F_{1,2}(x_1,x_2) + m_{3\to 1}(x_1) + m_{4\to 1}(x_1))$ and $z_1(x_1) = m_{2\to 1}(x_1) + m_{3\to 1}(x_1) + m_{4\to 1}(x_1)$.

The max-sum technique is guaranteed to solve the problem optimally on acyclic structures, but when applied to general graphs which contain loops, only limited theoretical results hold for solution quality and convergence. Nonetheless, extensive empirical evidence demonstrates that, despite the lack of convergence guarantees, the max-sum algorithm does in fact generate good approximate solutions when applied to cyclic graphs in various domains [13, 11, 19]. When the algorithm does converge, it does not converge to a simple local maximum, but rather, to a neighborhood maximum that is guaranteed to be greater than all other maxima within a particular large region of the search space [37]. Characterizing this region is an ongoing area of research and to date has only considered small graphs with specific topologies (e.g., several researchers have focused on the analysis of the algorithm's convergence in graphs containing just a single loop [37, 1]).

The max-sum algorithm is attractive for decentralized coordination of computationally and communication constrained devices since the messages are small (they scale with the domain of the variables), and the number of messages exchanged typically varies linearly with the number of agents within the system. Moreover, when constraints are binary, the computational complexity to update the messages and perform the optimization is polynomial. In the more general case of n-ary constraints, this complexity scales exponentially with just the number of variables on which each function depends (which is typically much less than the total number of variables in the system). However, as with the previously discussed approximate algorithms, the lack of guaranteed convergence and guaranteed solution quality in general cases, limits the use of the standard max-sum algorithm in many application domains.

A possible solution to address this problem is to remove cycles from the constraint graph by arranging it into tree-like structures such as junction trees [20] or pseudo-trees [30]. However, such arrangements result in an exponential element in the computation of the solution or in the communication overhead. For example, DPOP is functionally equivalent to performing max-sum over a pseudo-tree formed by depth-first search of the constraint graph, and the resulting maximum message size is exponential with respect to the width of the pseudo-tree. This exponential element is unavoidable in order to guarantee optimality of the solution and is tied to the combinatorial nature of the optimization problem. However, as discussed in the introduction of this chapter, such exponential behavior is undesirable in systems composed of devices with constrained computational resources.

To address these issues, low overhead approximation algorithms that can provide quality guarantees on the solution are a key area of research, and we discuss the most prominent approaches in this area in the next session.

6 Solution Techniques with Quality Guarantees

Developing approximate algorithms that can provide guarantees on solution quality is a growing area of research that is gaining increasing attention. Such approaches are particularly promising as they can conveniently address the unavoidable trade off between guarantees on solution quality and computation effort. Addressing this trade-off is particularly important in dynamic settings and when the agents have severe constraints on computational power, memory or communication (which is usually the case for applications involving embedded devices, such as mobile robots or sensor networks). Moreover, having a bound on the quality of the provided solutions is particularly important for safety critical application (such as disaster response, surveillance, etc.) because pathological behavior of the system is, in this case, simply unacceptable.

Guarantees that can be provided by approximate algorithms can be broadly divided in two main categories: *off-line* and *on-line*. The former can provide a characterization of the solution quality without running any algorithm on the specific problem instances. In contrast, the latter can only provide quality guarantees for a solution after processing a specific problem instance. Off-line guarantees represent the classical definition of approximation algorithms [8], and they provide very general results which are not tied to specific problem instances. In this sense they are generally preferred to on-line guarantees. However, on-line guarantees are usually much tighter than the off-line, precisely because they can exploit knowledge on the specific problem instance, and thus, better characterize the bound on solution quality.

Here we present two representative approaches for these two classes: the k-optimality framework and the bounded max-sum approach.

6.1 Off-line Guarantees

A widely used approach to provide off-line guarantees for solution quality in DCOPs is based on the *k-size* optimality framework. The main idea behind k-size optimality is to consider optimal solutions for sub-groups of k agents, and then provide a bound on the globally optimal solution. More specifically, a solution is k-optimal if the corresponding value of the objective function cannot be improved by changing the assignment of k or fewer agents. For example, consider again the constraint network in Figure 12.1. The solution $\hat{\mathbf{x}} = \{x_1 = 1,\ x_2 = 1,\ x_3 = 1,\ x_4 = 1\}$ is a 2-optimal solution but not a 3-optimal solution. In fact, $F(\hat{\mathbf{x}}) = F_{1,2} + F_{1,3} + F_{1,4} + F_{2,4} = 1 + 1 + 1 + 1 = 4$, and thus, clearly if only two agents can change their variables' values, there is no solution that obtains a value higher than four. However, if we allow three agents to change their values, then we can obtain a better solution. Consider for example the assignment $\hat{\mathbf{x}}' = \{x_1 = 0,\ x_2 = 0,\ x_3 = 1,\ x_4 = 0\}$ we can see that $F(\hat{\mathbf{x}}') = F_{1,2} + F_{1,3} + F_{1,4} + F_{2,4} = 2 + 0 + 2 + 2 = 6 \geq 4$. Moreover, notice that $\hat{\mathbf{x}}'$ is not the optimal solution, as the optimal solution is $\mathbf{x}^* = \{x_1 = 0,\ x_2 = 0,\ x_3 = 0,\ x_4 = 0\}$ and $F(\mathbf{x}^*) = 8$.

Building on the k-optimality concept, Pearce and Tambe were able to provide an approximation ratio (i.e., the ratio between the unknown optimal solution and the approximate solution [8]) for k-optimal algorithms which is valid for any DCOP with non-negative reward structure[7] [27]. The accuracy of such an approximation ratio, in any particular setting, depends on the number of agents, on the arity of the constraints and on the value of k. Specifically, for any DCOP with

[7]Reward structure here makes reference to the particular values that return the functions of the DCOP.

non-negative rewards, the following equation holds:

$$F(\hat{\mathbf{x}}) \geq \frac{\binom{n-m}{k-m}}{\binom{n}{k} - \binom{n-m}{k}} F(\mathbf{x}^*) \tag{12.6}$$

where $\hat{\mathbf{x}}$ is a k-size optimal solution, $\mathbf{x}^*$ is the optimal solution, n is the number of agents and m is the maximum constraint arity and $m \leq k < n$. Notice that every DCOP that does not have infinite negative costs can be normalized to one with all non-negative rewards. However the analysis is not applicable to DCOPs that include hard constraints.

For the usual case of a binary network (i.e. $m = 2$) the above equation simplifies to:

$$F(\hat{\mathbf{x}}) \geq \frac{k-1}{2n-k-1} F(\mathbf{x}^*) \tag{12.7}$$

Thus, for the constraint network in Figure 12.1, we can immediately conclude that for any 2-optimal solution, $\hat{\mathbf{x}}$, we will have that $F(\hat{\mathbf{x}}) \geq 1/5 \cdot F(\mathbf{x}^*)$ simply because $\frac{k-1}{2n-k-1} = 1/5$ when $k = 2$ and $n = 4$. In fact it is easy to see that this inequality holds for the 2-optimal solution considered above.

Notice that, the above inequalities hold for every possible constraint network with every possible reward structure (as long all rewards are non-negative). For example, if we add a constraint between x_3 and x_4 in our exemplar constraint network, no matter what function we define for that constraint, we are still guaranteed that the value of any 2-optimal solution will be greater than $1/5$ of the value of the optimal solution. This is clearly a very strong and general result, but unfortunately, the accuracy of this bound depends on the number of agents, on the arity of the constraints, and on the value of k. Specifically, the bound is more accurate when k is higher, but less accurate when the number of agents in the system grows, and the maximum arity of constraints is high. Clearly, by increasing k it is possible to achieve better bounds, however this would result in an exponential increase in computation and communication required to obtain a k-size optimal solution.

From an algorithmic point of view, the k-size optimal framework assumes that we are able to find a k-size optimal solution. This basically requires that a group of k agents coordinate their choice to find a solution that is optimal for the group. Now, any local hill climbing algorithm is k-size optimal for $k = 1$. Hence, approaches such as DSA [12] and MGM [22] are able to find a 1-optimal solution. However, for $k = 1$ we are unable to provide any guarantees as the k-optimal analysis is valid only if $m \leq k$. Therefore, $k = 2$ variants of MGM and

DSA, termed MGM-2 and DSA-2, have been devised [22]. Moreover, there are also algorithms to find k-size optimal solutions with arbitrary k [18].

The k-size optimality framework has been recently extended introducing a different criterion for local optimality; in particular, t-distance optimality. This is based on the distance between nodes on the constraint graphs, rather than on the size of the groups. Furthermore, Vinyals et al. recently introduced the $\mathcal{C}$-optimality framework that generalizes both k-optimality and t-optimality by providing quality guarantees for local optima in regions that can be defined by arbitrary criteria [36]. Specifically, they propose a new criterion to define regions (i.e., the size-bounded-distance criterion); showing that this criterion outperforms both k-size and t-distance, yielding more precise control of the computational effort required to provide such local optimal solutions. Finally, the $\mathcal{C}$-optimality framework has been recently used to provide quality guarantees for fixed-point assignments of the max-sum algorithm (i.e., assignments to which the algorithm converged). Specifically, building on the results obtained by Weiss that characterized any fixed point max-product assignment as a local optima for a specific region (named Single Loops and Trees) [37], Vinyals et al. were able to characterize the quality guarantees for the max-sum algorithm in that region [33]. Thus, if the max-sum algorithm converges, it is possible to provide a worst case quality guarantee equivalent to 3-optimal solutions of the k-optimality framework.

As mentioned above, inequality 12.6 is valid for every possible constraint network. This is because the bound is the result of a worst case analysis on a completely connected graph. If we restrict our attention to specific constraint network topologies it is possible to obtain better bounds. For example, for a network with a ring topology, where each agent has only two constraints, we have that $F(\hat{\mathbf{x}}) \geq \frac{k-1}{k+1} F(\mathbf{x}^*)$. This is a much better bound as it does not depend on the number of agents in the system, but applies only to a very specific topology. Similar considerations hold for assumptions on the reward structure. Specifically, better guarantees can be provided assuming some *a priori* knowledge on the reward structure. For example, Bowring et al. show that the approximation bounds can be improved by knowing the ratio between the minimum reward to the maximum reward [5]. In addition, they also extend the k-optimality analysis to DCOPs that include hard constraints. However, while they are able to significantly improve on the accuracy of the bound by exploiting *a priori* knowledge on the reward, the bound is still dependent on the number of agents, and decreases as the number of agents grows; thus, being of little help for large scale applications.

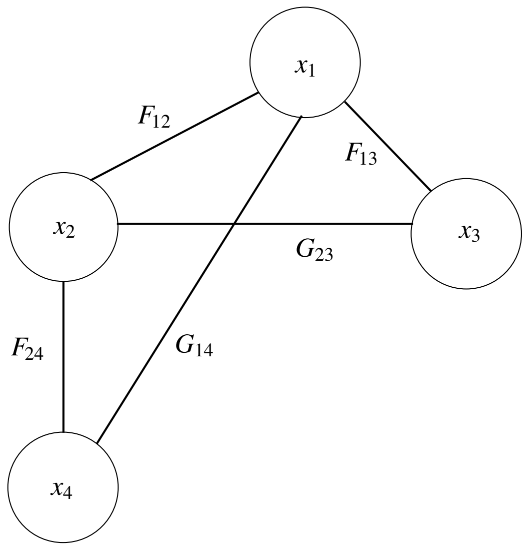
6.2 On-line Guarantees

On-line approaches for providing quality guarantees are complementary to off-line ones, as they usually give accurate bounds but only for specific problem instances [32, 34]. In this respect, the Bounded Max-Sum (BMS) approach is a good representative for this kind of technique [32]. The main idea behind BMS is to remove cycles in the original constraint network by simply ignoring some of the dependencies between agents. It is then possible to optimally solve the resulting tree structured constraint network, whilst simultaneously computing the approximation ratio for the original problem instance. The BMS approach uses the max-sum algorithm to provide an optimal solution for the tree-structured problem, hence the name, but any distributed constraint optimization technique that is guaranteed to provide optimal solutions on a tree-structured constraint network could be used. The choice of the max-sum approach is driven by its efficiency in terms of low communication overhead (specifically in the number of messages), low computational requirement and ease of decentralization. The main issue with this approach is to choose which dependencies should be removed to form the tree-structured constraint network, and to somehow relate the solution quality of the new problem to that of the original loopy one.

To do this, BMS assigns weights to constraints, where the weights quantify the maximum impact that removing any constraint may have on the optimal solution. In other words, the weight of a constraint specifies the worst case outcome if we were to solve the problem ignoring that constraint. BMS will then remove constraints that have the least impact on the solution quality, (i.e. constraints that have smaller weights). This can be done by computing a maximum spanning tree of the original constraint network.

For ease of presentation, we again provide a description of the algorithm under the usual assumptions that each agent controls exactly one variable, and that all constraints are binary. The interested reader can find the description of the algorithm in more general settings in [32]. In this context, the steps of the algorithm are the following:

1. Define the weight of each constraint as: $w_{ij} = \min\{w'_{ij}, w''_{ij}\}$ where $w'_{ij} = \max_{x_j}\left[\max_{x_i} F_{ij} - \min_{x_i} F_{ij}\right]$ and $w''_{ij} = \max_{x_i}\left[\max_{x_j} F_{ij} - \min_{x_j} F_{ij}\right]$. For example, Figure 12.7 reports a constraint network and the weights that the BMS algorithm would compute. Specifically, for the constraint between x_1 and x_4 we have that $w'_{14} = max_{x_4}[max_{x_1} G_{14} - min_{x_1} G_{14}] = 3$ and $w''_{14} = max_{x_1}[max_{x_4} G_{14} - min_{x_4} G_{14}] = 1$ therefore $w_{14} = 1$.



$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

$G_{i,j}$	x_i	x_j
4	0	0
3	0	1
1	1	0
2	1	1

Figure 12.7: Loopy constraint network with two types of constraints.

2. Remove constraints from the original cyclic constraint network by building a maximum weight spanning tree. In this way it is possible to minimize the sum of the weights of the removed edges, pursuing the objective of removing constraints that have least impact on the solution.

   Moreover, define the sum of the weights of the removed constraints as:

   $$W = \sum_{c_{ij} \in C^r} w_{ij} \tag{12.8}$$

   where C^r is the set of constraints removed from the constraint network.

   For example, in Figure 12.7 we have that $C^r = \{G_{14}, G_{23}\}$ therefore $W = w_{14} + w_{23} = 2$.

3. Run the max-sum algorithm on the remaining tree structured constraint network. For constraints which have been removed, add unary constraints obtained by minimizing out one of the two variables. The removed variable is the one that yields the minimum weight for that constraint. For example, in our case the assignment we obtain after running max-sum on the spanning tree solves the constraint network shown in Figure 12.8, thus optimizing the global function $G_1^m + F_{12} + F_{13} + G_2^m + F_{24}$ where $G_1^m = \min_{x_4} G_{14}$ and $G_2^m = \min_{x_3} G_{23}$.

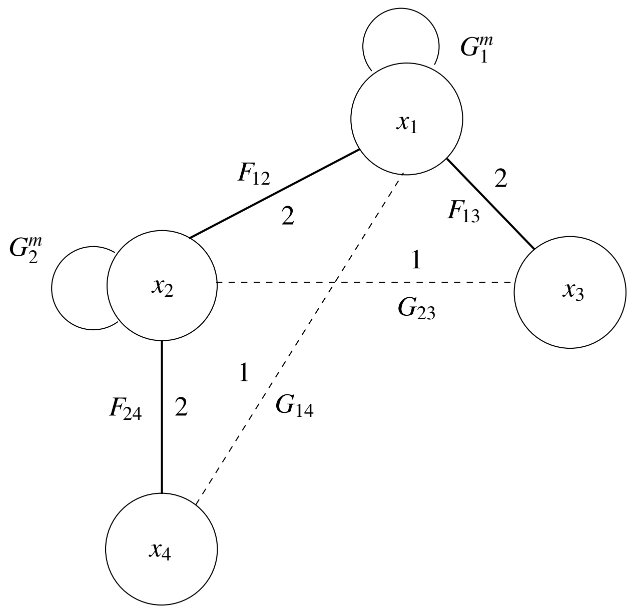


Figure 12.8: Tree-like constraint network formed by the BMS algorithm when run on the loopy constraint network of Figure 12.7. Numbers represent weights for binary constraints.

4. The resulting variable assignment, $\tilde{\mathbf{x}}$, represents the approximate solution to the original optimization problem, and it is possible to prove that this approximate solution is within a calculated bound from the optimum solution. More precisely:

$$V^* \leq \rho \tilde{V} \tag{12.9}$$

where the approximation ratio $\rho = (\tilde{V}^m + W)/\tilde{V}$, and $\tilde{V}^m$ represents the optimal solution to the tree structured constraint network. Here V^* represents the value of the unknown optimal solution to the original cyclic constraint network and $\tilde{V}$ is the approximate solution, found using the tree structured constraint network, but evaluated on the original cyclic constraint network.

All the above steps can be performed using a decentralized approach. In particular, the computation of the maximum spanning tree can be performed in a distributed fashion using various message passing algorithms, such as, for example, the minimum spanning tree algorithm by Gallager, Humblet and Spira (GHS), modified to find the maximum spanning tree [14].

As previously mentioned, this approach is able to provide guarantees on solution quality that are instance based. Therefore the algorithm must be run on the specific problem instance in order to obtain the bound. By exploiting knowledge of the particular problem instance, BMS is able to provide bounds which are very accurate. For example, the BMS approach has been empirically evaluated in a mobile sensor domain, where mobile agents must coordinate their movements to sample and predict the state of spatial phenomena (e.g., temperature or gas concentration). When applied in this domain, BMS is able to provide solutions which are guaranteed to be within 2% of the optimal solution.

Other data dependent approximation approaches with guarantees have also been investigated. For example, Petcu and Faltings propose an approximate version of DPOP [29], and Yeoh et al. provide a mechanism to trade-off solution quality for computation time for the ADOPT and BnB-ADOPT algorithms [40]. Such mechanisms work by fixing an approximation ratio and reducing computation or communication overhead as much as possible to meet that ratio.

More specifically, BnB-ADOPT fixes a predetermined error bound for the optimal solution, and stops when a solution that meets this error bound is found. In this approach, the error bound is fixed and pre-determined off-line, but the number of cycles required by the algorithm to converge is dependent on the particular problem instance, and, in the worst case, remains exponential. The BMS approach discussed above, in contrast, is guaranteed to converge after a polynomial number of cycles, but the approximation ratio is dependent on the particular problem instance.

Similar considerations hold with respect to A-DPOP [29]. A-DPOP attempts to reduce message size (which is exponential in the original DPOP algorithm in the width of the pseudo tree) by optimally computing only a part of the messages, and approximating the rest (with upper and lower bounds). In this case, given a fixed pre-determined approximation ratio, A-DPOP reduces message size to meet this ratio. Alternatively, given a fixed maximum message size, A-DPOP propagates only those messages that do not exceed that size. As a result of this, the computed solution is not optimal, but approximate. If the algorithm is used by fixing a desired approximation ratio, the message size remains exponential. In contrast, if we fix the maximum message size, the approximation ratio is dependent on the specific problem instance.

7 Conclusions

The constraint processing research area comprises powerful techniques and algorithms which are able to exploit problem structure, and thus, solve hard problems efficiently. In this chapter we focused on the DCOP framework where constraint processing techniques are used to solve decision making problems in MAS.

This chapter provides an overview of how DCOPs have been used to address decentralized decision making problems by first presenting the mathematical formulation of DCOPs and then describing some of the practical problems that DCOPs can successfully address. We detailed exact solution techniques for DCOPs presenting two of the most representative algorithms in the literature: ADOPT and DPOP. We then discuss approximate algorithms, including DSA and MGM, before presenting GDL and the max-sum algorithm. Finally, we presented recent ongoing work that is attempting to provide quality guarantees for these approaches.

Overall, the DCOP framework, and the algorithms being developed to solve such problems, represent an active area of research within the MAS community, and one that is increasingly being applied within real world contexts.

8 Exercises

1. Level 1 Consider the coordination problem faced by intervention forces in a rescue scenario. In particular, consider a set of fire fighting units that must be assigned to a set of fires in order to minimize losses to buildings, infrastructure and civilians. Each fire fighting unit can be assigned to just one fire. However, if more than one unit work on the same fire at the same time, they can extinguish it faster (collaboration has a positive synergy). Furthermore, a fire fighting unit can only be assigned to fires which are within a given distance from its initial position (due to the travel time required to reach the fire). As a result, any particular fire fighting unit can only be assigned to a subset of the fires that exist.

   - Formalize this task assignment problem as a DCOP specifying (i) what the variables represent, (ii) the domain of the variables, and (iii) the constraints.
   - Present an example involving about five fire fighting units and three fires, instantiating each of the features above.

2. $\boxed{\text{Level 2}}$ Consider the coordination problem described in Exercise 1, but now extended to include two types of intervention forces: fire fighting units and ambulance units. Assume that instead of minimizing loses, we must now assign exactly one fire fighting unit and one ambulance unit to each fire. As before, any particular fire fighting or ambulance unit can only attend a subset of the existing fires. Provide a CSP formulation of this problem.

3. $\boxed{\text{Level 4}}$ Consider the coordination problem described in Exercise 1, and specifically, a situation with two fire fighting units and two fires, where both units can be assigned both fires. Depending on the units' travel times, the severity of the fires, and the function that defines the losses that result, the best solution could result in both units being assigned to the same fire, leaving the other one uncontrolled. Assume now that we want to have a fair assignment of fire fighting units to tasks. Can we formalize this as a DCOP? Which approach could be used to tackle this problem?

4. $\boxed{\text{Level 2}}$ Consider the following constraint network representing a graph coloring problem:

   - $X = \{x_1, .., x_6\}$
   - $D = \{d_1 = d_3 = d_4 = d_6 = \{Red, Blue\}, d_2 = d_5 = \{Red, Blue, Green\}\}$
   - $C = \{< x_1, x_2 >, < x_1, x_3 >, < x_1, x_6 >, < x_2, x_3 >, < x_3, x_4 >, < x_4, x_5 >, < x_4, x_6 >, < x_5, x_6 >\}$

   Assume that every node in the graph is controlled by one agent. Find the computational complexity of DPOP with the following pseudo-tree ordering: $o_1 =< x_1, x_2, x_3, x_4, x_5, x_6 >$. State whether there is a pseudo-tree ordering that would result in less computational complexity and, if so, present an instance of one.

5. $\boxed{\text{Level 1}}$ Show a complete execution of DPOP that solve the MaxCSP formulation of the constraint network provided in Exercise 4 (use either o_1 or a pseudo-tree ordering of your choice).

6. $\boxed{\text{Level 1}}$ Give an execution example of MGM where the algorithm finds the global optimum, and another where it gets trapped in a local minimum.

7. $\boxed{\text{Level 3}}$ The max-sum algorithm is guaranteed to converge to the optimal solution on graphs that do not contain cycles. However, if there are several optimal assignments, the distributed maximization performed in Equation

12.5 is problematic, and may lead to sub-optimal solutions. Discuss whether a value propagation phase would solve this problem. Provide an execution example.

8. Level 4 The DPOP algorithm is based on a pseudo-tree arrangement of the agents. In particular, DFS trees which are a specific class of pseudo-tree are typically used. On the other hand, many constraint optimization approaches [9], including GDL-based techniques, are based on the concept of a junction tree [20]. Discuss the relationship between these two structures, and their impact with respect to computation and communication on DCOP solution techniques. Can you find a junction tree that allows a GDL algorithm to solve a DCOP with significantly less computation (and/or communication) than when using a pseudo-tree? What about DFS trees? We suggest the reader start from [35] where these questions were investigated for unrestricted pseudo-trees.

9. Level 2 Provide a DCOP problem with at least five variables and a solution which is 3-optimal but not 4-optimal.

10. Level 3 Consider the bound expressed by Equation 12.9 for the bounded max-sum approach. Assuming that you know the constraint network topology, and the maximum and minimum value for all the functions (but not the actual values of the functions), in a particular DCOP example, modify the bounded max-sum technique to provide a bound in this setting. Can you provide some bounds even before running the max-sum algorithm? Can you extend the analysis assuming you do not know the constraint network topology?

11. Level 3 Consider the bounded max-sum technique presented in Section 6.2. Elaborate on how this technique can be applied to problems that include hard constraints. In particular, how is the approximation ratio affected by hard constraints? Under which assumptions can this technique still provide useful bounds?

12. Level 4 Consider the problem of minimizing the running time of a DCOP solution algorithm when agents have heterogeneous computation and communication. In this settings it might be beneficial to delegate computation to agents that have additional computation capabilities, or to minimize message exchange between agents that are connected by poor communication

links. Formalize this problem and provide an approach that can take such heterogeneity into account.

References

[1] S. M. Aji, S. B. Horn, and Robert J. Mceliece. On the convergence of iterative decoding on graphs with a single cycle. In *Proceedings of the International Symposium on Information Theory (ISIT)*, pages 276–282, 1998.

[2] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[3] S. M. Ali, S. Koenig, and M. Tambe. 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, 2005.

[4] Fahiem Bacchus, Xinguang Chen, Peter van Beek, and Toby Walsh. Binary vs. non-binary constraints. *Artif. Intell.*, 140(1/2):1–37, 2002.

[5] E. Bowring, J. Pearce, C. Portway, M. Jain, and M. Tambe. On k-optimal distributed constraint optimization algorithms: New bounds and algorithms. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent systems*, pages 607–614, 2008.

[6] A. Chapman, A. Rogers, N. R. Jennings, and D. Leslie. A unifying framework for iterative approximate best response algorithms for distributed constraint optimisation problems. *The Knowledge Engineering Review*, 26(4):411–444, 2011.

[7] A. Chechetka and K. Sycara. No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1427–1429, 2006.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms, second edition*. The MIT press, 2001.

[9] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[10] R. Dechter and R. Mateescu. And/or search spaces for graphical models. *Artificial Intelligence*, 171:73–106, 2007.

[11] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems*, pages 639–646, 2008.

[12] S. Fitzpatrick and L. Meetrens. *Distributed Sensor Networks: A multiagent perspective*, chapter Distributed coordination through anarchic optimization, pages 257–293. Kluwer Academic, 2003.

[13] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[14] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, 1983.

[15] A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward bounding for distributed COPs. *Journal Artificial Intelligence Research*, 34:61–88, 2009.

[16] P. Gutierrez and P. Meseguer. Saving redundant messages in BnB-ADOPT. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 1259–1260, 2010.

[17] Katsutoshi Hirayama and Makoto Yokoo. Distributed partial constraint satisfaction problem. In *Principles and Practice of Constraint Programming*, pages 222–236, 1997.

[18] C. Kiekintveld, Z. Yin, A. Kumar, and M. Tambe. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 133–140, 2010.

[19] R. J. Kok and N. Vlassis. Using the max-plus algorithm for multiagent decision making in coordination graphs. In *RoboCup-2005: Robot Soccer World Cup IX*, 2005.

[20] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 42(2):498–519, 2001.

[21] D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[22] R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical game-based approach. In *Proceedings of the Seventeenth International Conference on Parallel and Distributed Computing Systems*, pages 432–439, 2004.

[23] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems*, pages 438–445, 2004.

[24] P. J. Modi. *Distributed constraint optimization for multiagent systems*. PhD thesis, Department of Computer Science, University of Southern California, 2003.

[25] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal*, (161):149–180, 2005.

[26] C. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[27] J. P. Pearce and M. Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1446–1451, 2007.

[28] A. Petcu. *A Class of Algorithms for Distributed Constraint Optimization*. PhD thesis, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland), 2007.

[29] A. Petcu and B. Faltings. A-DPOP: Approximations in distributed optimization. In *Principles and Practice of Constraint Programming*, pages 802–806, 2005.

[30] A. Petcu and B. Faltings. DPOP: A scalable method for multiagent constraint optimization. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 266–271, 2005.

[31] A. Petcu and B. Faltings. MB-DPOP: A new memory-bounded algorithm for distributed optimization. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 1452–1457, 2007.

[32] A. Rogers, A. Farinelli, R. Stranders, and N. R Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence Journal*, 175(2):730–759, 2011.

[33] M. Vinyals, J. Cerquides, A. Farinelli, and J. A. Rodriguez-Aguilar. Worst-case bounds on the quality of max-product fixed-points. In *Neural Information Processing Systems*, pages 2325–2333, 2010.

[34] M. Vinyals, M. Pujol, J. A. Rodriguez-Aguilar, and J. Cerquides. Divide-and-coordinate: DCOPs by agreement. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, pages 149–156, 2010.

[35] M. Vinyals, J. A. Rodriguez-Aguilar, and J. Cerquides. Constructing a unifying theory of dynamic programming DCOP algorithms via the Generalized Distributive Law. *Journal of Autonomous Agents and Multi Agent Systems (JAAMAS)*, pages 1–26, 2010.

[36] M. Vinyals, E. Shieh, J. Cerquides, J. A. Rodriguez-Aguilar, Z. Yin, M. Tambe, and M. Bowring. Quality guarantees for region optimal DCOP algorithms. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems*, pages 133–140, 2011.

[37] Y. Weiss and W. T. Freeman. On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):723–735, 2001.

[38] Y. Weiss and W.T. Freeman. Correctness of belief propagation in Gaussian graphical models of arbitrary topology. *Neural Computation*, 13(10):2173–2200, 2001.

[39] W. Yeoh, A. Felner, and S. Koenig. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 591–598, 2008.

[40] W. Yeoh, X. Sun, and S. Koenig. Trading off solution quality for faster computation in DCOP search algorithms. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence*, pages 354–360, 2009.

[41] Makoto Yokoo. *Distributed constraint satisfaction: Foundations of cooperation in multi-agent systems*. Springer-Verlag, 2001.