Divide-and-Coordinate: DCOPs by agreement

Meritxell Vinyals,
Marc Pujol,
J. A. Rodriguez-Aguilar
Artificial Intelligence Research Institute (IIIA)
Spanish Scientific Research Council (CSIC)
Campus UAB, Bellaterra, Spain
{meritxell,mpujol,jar}@iiia.csic.es

Jesus Cerquides
WAI, Dep. Matemàtica Aplicada i Anàlisi
Universitat de Barcelona
Gran Via 585, Barcelona, Spain
cerquide@maia.ub.es

ABSTRACT

In this paper we investigate an approach to provide approximate, anytime algorithms for DCOPs that can provide quality guarantees. At this aim, we propose the *divide-and-coordinate* (DaC) approach. Such approach amounts to solving a DCOP by iterating (1) a *divide* stage in which agents divide the DCOP into a set of simpler local subproblems and solve them; and (2) a *coordinate* stage in which agents exchange local information that brings them closer to an agreement. Next, we formulate a novel algorithm, the Divide and Coordinate Subgradient Algorithm (DaCSA), a computational realization of DaC based on Lagrangian decompositions and the dual subgradient method. By relying on the DaC approach, DaCSA provides bounded approximate solutions. We empirically evaluate DaCSA showing that it is competitive with other state-of-the-art DCOP approximate algorithms and can eventually outperform them while providing useful quality guarantees.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence—*Distributed Artificial Intelligence*

General Terms

Algorithms theory

Keywords

DCOP, Multi-agent Optimization, Divide and Coordinate, DaCSA, approximate algorithms, bounds

1. INTRODUCTION

A distributed constraint optimization problem (DCOP) [12, 10, 15] is a formalism that captures the rewards and costs of local interactions in a Multiagent System (MAS) where each agent chooses a set of individual actions. Unlike constraint optimization problems (COPs), agents in DCOPs need to coordinate in a decentralized manner. DCOP is a framework that can model a large number of coordination, scheduling, and task allocation problems in MAS. DCOPs have already been applied to domains such as sensor networks [18], meeting scheduling [15, 9], and traffic control [6].

Traditionally, researchers have focused on developing complete algorithms for DCOPs (e.g Adopt [12], OptAPO [10], DPOP [15]) that return a globally optimal solution. However, since DCOPs are shown to be NP-hard [12], complete algorithms are often unsuitable for some actual-world domains.

Therefore, approximate algorithms have also been proposed to solve DCOPs. On the one hand, stochastic algorithms such as the Distributed Stochastic Algorithm (DSA) or the Distributed Breakout Algorithm (DBA) [18] make agents locally update their decisions based on their neighbours' states. On the other hand, Max-Sum (MS) [4] is a variant of the sum-product algorithm that has been recently applied to DCOPs with success. These algorithms are low-overhead algorithms (they require very little local computation and local communication) and thus are said to be well-suited for large-scale applications. However, they may either converge to poor solutions or even diverge.

Furthermore, as pointed out in [14], an important limitation of these approximate algorithms is that they fail to provide any quality guarantees for their solutions, leaving agents with high uncertainty about the goodness of their decisions. Thus, although some works have started to make headway in this direction [14, 3], the design of new bounded, approximate DCOP algorithms that make efficient use of limited resources and can provide useful bounds is still an open issue.

We address this issue in this paper by: (1) proposing to solve DCOPs by a novel approach, the so-called *divide-and-coordinate* (DaC) approach, that provides a bound on the DCOP solution; and (2) formulating and benchmarking the first DaC algorithm for DCOPs. Concretely, this paper makes the following contributions:

- We propose to solve DCOPs by a novel approach, the *DaC* approach, where agents divide an intractable DCOP into simpler subproblems to individually solve them. Thereafter, agents solve the DCOP by searching for an agreement on the optimal assignments of their subproblems.
- We formulate a particular computational realization of the *DaC* approach using two well-known techniques in optimization: Lagrangian dual decompositions and subgradient dual methods [2]. As a result, we provide a novel approximate algorithm, the so-called Divide and Coordinate Subgradient Algorithm (DaCSA), which allows agents to distributedly solve DCOPs by providing bounded, anytime solutions.
- We empirically evaluate DaCSA on different agent network topologies by comparing it with two state-of-the-art approximate algorithms: Max-sum [4] and DSA [18]. The empirical results provide evidence of the competiveness of DaCSA, specially on structured problems, where it does outperform both Max-sum and DSA while providing significant bounds.

The use of subgradient algorithms and Lagrangian dual decompositions on distributed discrete optimization problems is not new.

Cite as: Divide-and-Coordinate: DCOPs by agreement, Meritxell Vinyals, Marc Pujol, J. A. Rodriguez-Aguilar and Jesus Cerquides, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 149-156

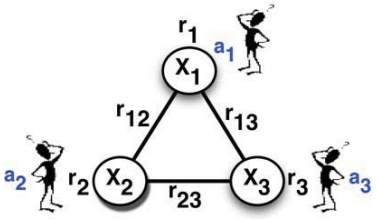


Figure 1: Example of a DCOP constraint graph .

In [8] Komodakis et al. apply a message-passing algorithm that uses dual decomposition to solve Markov Random Fields with application to computer vision problems. Furthermore, Hirayama et al. proposed in [7] a distributed version of a subgradient algorithm to solve the Generalized Mutual Assignment Problem. In contrast, we use the Lagrangian dual decomposition and subgradient algorithms as a tool for designing an algorithm that follows the more general DaC approach. Moreover, to the best of our knowledge, in this paper we present the first distributed algorithm that uses Lagrangian dual decompositions and subgradient methods to solve DCOPs.

This paper is structured as follows. In section 2 we give an overview of DCOPs. Next, in section 3 we formulate the *divide-and-coordinate* approach. In section 4 we propose DaCSA, a novel DCOP approximate algorithm. In section 5 we detail an empirical evaluation of the performance of DaCSA against other state-of-the-art approximate algorithms. Finally, we draw some conclusions and set paths to future work in section 6.

2. OVERVIEW OF DCOPS

A Constraint Optimization Problem (COP) consists of a set of variables, each one taking on a value out of a finite discrete domain. A relation in this context determines the utility of every combination of values taken by the variables in its domain. The goal of a COP algorithm is to assign values to these variables so that the total utility is maximized.

Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of variables over domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$. A *utility relation* is a function $r : \mathcal{D}_r \rightarrow \mathbb{R}^+$ with domain variables $\{x_{i_1}, \ldots, x_{i_q}\}$ in $\mathcal{D}_r = \mathcal{D}_{i_1} \times \ldots \times \mathcal{D}_{i_q}$, that assigns a utility value to each combination of values of its domain variables. Formally, a COP is a tuple $\Phi = \langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ where: $\mathcal{X}$ is a set of variables; $\mathcal{D}$ is the joint domain space for all variables; and $\mathcal{R}$ is a set of utility relations. The objective function f is described as an aggregation over the set of relations. Formally:

$$f(d) = \sum_{r \in \mathcal{R}} r(d_r) \tag{1}$$

where d is an element of the joint domain space $\mathcal{D}$ and d_r is an element of $\mathcal{D}_r$. The goal is to assess a configuration d^* with utility f^* that maximizes the objective function in equation 1. A DCOP [15, 12] is a distributed version of a COP where: (1) variables are distributed among a set of agents $\mathcal{A}$; and (2) each agent receives knowledge about all relations that involve its variable(s). Although an agent can be in charge of one or more variables, hereafter, we assume that each agent a_i is assigned a single variable x_i. Moreover, we focus on binary DCOPs (those whose utility relations involve at most two variables). Therefore, we will refer to unary constraints involving variable $x_i \in \mathcal{X}$ as r_i, and to binary constraints involving variables $x_i, x_j \in \mathcal{X}$ as r_{ij}.

DCOPs are usually represented by their constraint graphs, where nodes stand for variables and edges link variables that have some direct dependency (appear together in the domain of some relation). Figure 1 shows an example of a DCOP represented by its constraint graph. For instance, note that relation r_{12} is known by agent a_1, that controls variable x_1, and agent a_2, that controls variable x_2. In this context, the neighbours of some agent a are those that share some constraint with a. Thus, in figure 1, a_2 and a_3 are neighbours of a_1 because a_1 shares relation r_{12} with a_2 and r_{13} with a_3.

3. DIVIDE-AND-COORDINATE DCOPS

At the aim of providing a bounded, approximate algorithm for DCOPs, in this section we define a novel approach, the so-called *divide-and-coordinate* (DaC) approach. The intuition behind the DaC approach is simple. Since solving a DCOP is NP-Hard, we can think of *dividing* this intractable problem into simpler subproblems that can be individually solved by each agent. However, agents' solutions to their suproblems (local assignments) may conflict. Thus, agents *coordinate* by exchanging information about their disagreements. They subsequently employ such information to create a new division that brings them closer to an agreement. Thus, agents iteratively *divide* and *coordinate* until finding an agreement. Therefore, in order to solve a DCOP via a DaC approach, agents explore the space of divisions to find a division such that they agree on their individual solutions. This solution will be the solution to the original DCOP. Moreover, as discussed below, the DaC approach allows agents to naturally provide a bound on the DCOP solution during their search for an agreement.

According to a DaC approach, agents iteratively try to solve a DCOP Φ by interleaving two stages: a *divide stage* and a *coordinate* stage. The *divide* and *coordinate* stages ran by each agent are outlined in algorithm 1, a general DaC algorithm.

As a first step of the general DaC algorithm, agents create an initial division distributedly. At this aim, each agent only uses its local relations[1] in Φ (line 2) splitting the relations shared with other agents in equal parts. Thereafter, agents interleave *divide* and *coordinate* stages till reaching an agreement. For instance, in figure 2, agents divide the original DCOP Φ into three local subproblems (Φ_1, Φ_2, Φ_3), one per agent/variable. The right hand side of figure 2 (connected with arrows to its respective agent) details a particular division of the DCOP in figure 1 into three subproblems. Thus, the subproblem created by agent a_1 is Φ_1, where $\mathcal{X}^1 = \{x_1, x_2, x_3\}$, $\mathcal{D}^1 = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$, and $\mathcal{R}^1 = \{r_1, \frac{1}{2}r_{12}, \frac{1}{2}r_{13}\}$).

During the *divide* stage, agents solve their local subproblems (line 6). Observe now that agents' subproblems *share* variables (e.g. variable x_2 in figure 2 is shared by all agents). Hence, agents may disagree on the values of their shared variables after solving their subproblems. In figure 2, agents' local solutions (d_1^*, d_2^*, d_3^*) may disagree on the assignment to x_2. Therefore, to resolve conflicts, agents proceed with a *coordination* stage by exchanging local information with their neighbours. The information about neighbouring subproblems gathered during the *coordinate* stage (line 8) will be used by each agent, during a subsequent *divide* stage, to modify its subproblem to approach its decision to its neighbours' solution (line 5).

An important feature of the DaC approach is the requirement that the local subproblems in each *divide* stage compose the original DCOP Φ. In such case, we say that the set of subproblems are a *valid* division of the DCOP Φ. Next we formally describe the notion of a valid division of a DCOP.

DEFINITION 1 (VALID DIVISION). *Given a DCOP Φ, a set of m subproblems $\{\Phi_s | s = 1, \ldots, m\}$ is a valid division of Φ if its*

[1] As described in section 2, an agent's local relations in a DCOP are those that include the variables in its domain.

Algorithm 1 $DaC(\Phi)$

Each agent a_i runs:

1: $a_i.pCoordinate = \emptyset$; $a_i.neighbours = N(a_i)$; $a_i.x = x_i$
2: $\Phi_i \leftarrow$createSubproblem$(\langle \mathcal{X}^i, \mathcal{D}^i, \mathcal{R}^i \rangle)$
3: **repeat**
4: **/*Divide stage*/**
5: $\bar{\Phi}_i \leftarrow$ modifySubproblem$(\Phi_i, a_i.pCoordinate)$;
6: $\langle d_i^*, f_i^* \rangle \leftarrow$solveSubproblem$(\bar{\Phi}_i)$;
7: **/*Coordinate stage*/**
8: $a_i.pCoordinate \leftarrow$coordinate$(a_i.neighbours)$;
9: **until** agreement

objective function, f, can be rewritten as the sum of the objective functions of the individual subproblems, namely:

$$f(d) = f_1(d_1) + \ldots + f_m(d_m) \quad (2)$$

where f_s is the objective function for subproblem Φ_s, and d_s is the projection of d over the variables of Φ_s.

In figure 2 the set of subproblems created by agents in a *divide* stage are a valid division of the original DCOP with objective f iff: $f(x_1, x_2, x_3) = f_1(x_1, x_2, x_3) + f_2(x_1, x_2, x_3) + f_3(x_1, x_2, x_3)$. For instance, in figure 2 it is easy to check that combining the relations splitted in different subproblems recovers the original relations in the DCOP in figure 1.

The value of a division $\{\Phi_s | s = 1, \ldots, m\}$ is the sum of solutions of individual subproblems, namely $\sum_{l=1}^{m} f_s^*$, where f_s^* is the value of subproblem Φ_s evaluated at its optimal solution d_s^*.

Next, we show that the DaC approach has two important properties: (1) the sum of solutions to subproblems is always an upper bound on the quality of the global (optimal) solution; and (2) if all agents reach an agreement on a joint solution when optimizing their local subproblems, such solution is the optimal one.

Next we formulate the two propositions, which found the DaC approach, that relate the global solution and the value of a DCOP Φ with the local solutions and value of individual subproblems that compose any division of Φ.

PROPOSITION 1. *Given a DCOP Φ with objective function f, the value of a division $\{\Phi_s | s = 1 \ldots m\}$ of Φ is an upper bound on its optimal solution, namely $f^* \leq f_1^* + \ldots + f_m^*$.*

PROOF. We prove this by contradiction. Assume that there is an assignment $d \in \mathcal{D}$ whose value is greater for Φ than the value for some division $\Phi_1, \ldots, \Phi_m$ of Φ, that is $(f(d) = f_1(d_1) + \ldots + f_m(d_m)) > f_1^* + \ldots + f_m^*$. This implies a contradiction since at least some function $f_s \in \{f_1, \ldots, f_m\}$ evaluated at d_s (the projection of d over the variables in Φ_s) should be greater than its optimal solution f_s^*. □

Thus, the utility of the optimal solution of a DCOP is always lower than the value of any of its divisions. This upper bound allows to provide approximate solutions with quality guarantees.

PROPOSITION 2. *Given a DCOP Φ and a division $\{\Phi_s | s = 1 \ldots m\}$, if the solutions of all individual subproblems assign the very same value to each variable in $\mathcal{X}$, then this assignment is the optimal solution of Φ. In this case, the upper bound of proposition 1 is met with equality, $f^* = f_1^* + \ldots + f_m^*$.*

PROOF. Assume that the solutions $d_1^* \ldots d_m^*$ of the individual subproblems of a division $\{\Phi_s | s = 1 \ldots m\}$ of Φ assign the same value to each variable in $\mathcal{X}$. Let $d = d_1^* \cap \ldots \cap d_m^*$ be the values that individual subproblems assign to variables in $\mathcal{X}$. By proposition 1, we know that the value of any DCOP configuration cannot be greater than the value of any of its divisions. Thus, the value of any other configuration $d' \in \mathcal{D}$ of Φ is lower than the value of d, and hence d is the optimal solution of Φ. □

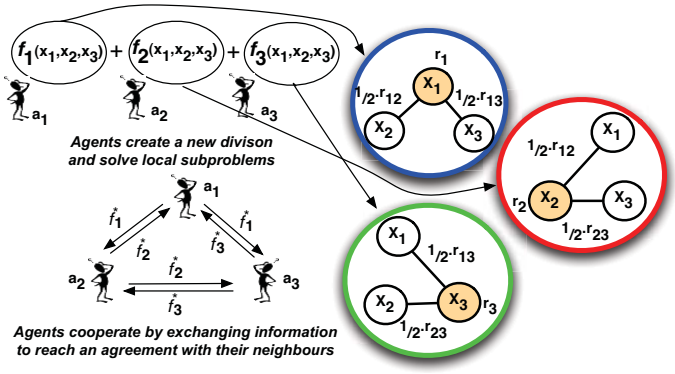


Figure 2: Divide-and-coordinate approach.

To summarise, to solve a DCOP Φ by DaC, agents explore the space of valid divisions to find a division such that the solution of individual subproblems agree (that will be the solution of Φ). However, even when the subproblems in a valid division do not agree on their assignments, agents can still provide quality guarantees on such solution using the bound of proposition 1.

Next, we will formulate a particular computational realization of the DaC approach, namely of algorithm 1, using dual decomposition techniques well-known in optimisation [2].

4. DACSA: A DAC ALGORITHM

In this section we formulate the so-called Divide And Coordinate Subgradient Algorithm (DaCSA), an anytime algorithm that allows agents to solve DCOPs while returning a quality guarantee on the solution. DaCSA is a particular computational realization of the DaC approach based on Lagrangian dual decompositions and subgradient methods. Next, in section 4.1 we provide the formal foundations of DaCSA while in section 4.2, we provide its algorithmic details.

4.1 Formal foundations

To build a computational realization of the DaC approach to solve a DCOP Φ we must define: (1) what information agents exchange about local subproblems during the *coordinate* step; and (2) how to use that information to create, at each *divide* step, a valid tractable division of Φ whose subproblems' solutions are closer to an agreement. At this aim, we propose to use Lagrange dual decomposition along with subgradient methods, both well-known techniques in optimization with strong theoretical properties (refer to [2], section 6.4).

In the remaining of the section let $\Phi = \langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ be a binary DCOP. Furthermore, let $\{\Phi_1, \ldots, \Phi_m\}$, $m = |\mathcal{X}|$ be a valid division of Φ, where $\Phi_s = \{\mathcal{X}^s, \mathcal{D}^s, \mathcal{R}^s\}$ and:

$$\mathcal{X}^s = \{x_i \ \ \forall x_i \in N(x_s) \cup \{x_s\}\}, \quad (3)$$

$$\mathcal{D}^s = \mathcal{D}_{\mathcal{X}^s}, \text{ and} \quad (4)$$

$$\mathcal{R}^s = \{r_s^s = r_s\} \cup \{r_{sj}^s = \frac{1}{2} \cdot r_{sj} \ \ \forall r_{sj} \in \mathcal{R}\} \cup \cup \{r_{is}^s = \frac{1}{2} \cdot r_{is} \ \ \forall r_{is} \in \mathcal{R}\} \quad (5)$$

In the valid division above each subproblem Φ_s is defined over variable x_s and its neighbours $N(x_s)$. Moreover, Φ_s is assigned the full unary relationship for variable x_s, and a half of every binary relation involving x_s. The right hand side of figure 2 details

the division of the DCOP in figure 1 that agents will create when following equations 3- 5. Notice that, in a binary DCOP, all these subproblems are acyclic and therefore identified as computationally tractable [16].

To apply duality we need to formalize Φ as a binary linear program (LP). With this purpose, for each subproblem Φ_s we define the following binary variables:

- $x^s_{i;k}$, that takes on value 1 when variable x_i in subproblem Φ_s takes on value k.
- $x^s_{ij;kl}$, that takes on value 1 when variables x_i,x_j in subproblem Φ_s take on values k and l respectively.

Formally, the set of binary variables of subproblem Φ_s is given by:

$$X^s = \{x^s_{i;k} : \forall i \in \mathcal{X}^s \, \forall k \in \mathcal{D}_i\} \cup \{x^s_{ij;kl} : \forall r^s_{ij} \in \mathcal{R}^s \, \forall k \in \mathcal{D}_i \, \forall l \in \mathcal{D}_j\}$$

With this set of variables we can express the objective function for subproblem Φ_s as:

$$f^s(X^s) = \sum_{k \in \mathcal{D}_s} x^s_{s;k} \cdot r^s_s(k) + \sum_{r_{ij} \in \mathcal{R}^s} \sum_{k \in \mathcal{D}_i} \sum_{l \in \mathcal{D}_j} x^s_{ij;kl} \cdot r^s_{ij}(k,l)$$

Then, solving Φ amounts to solving the following LP:

$$\max_x f(x) = \max_{\{X^s\}} \sum_{s=1}^{m} f_s(X^s) \tag{6}$$

subject to the following constraints ($\forall s \in \{1, \ldots, m\}$):

(C1) A unique value is assigned to each variable:

$$\begin{aligned} &\sum_{k \in \mathcal{D}_i} x^s_{i;k} = 1 && \forall x_i \in \mathcal{X}^s \\ &\sum_{k \in \mathcal{D}_i} \sum_{l \in \mathcal{D}_j} x^s_{ij;kl} = 1 && \forall r^s_{ij} \in \mathcal{R}^s \end{aligned} \tag{7}$$

(C2) A variable is assigned the very same value in all relations:

$$\begin{aligned} x^s_{i;k} &= \sum_{l \in \mathcal{D}_j} x^s_{ij;kl} && \forall r^s_{ij} \in \mathcal{R}^s \;\; \forall k \in \mathcal{D}_i \\ x^s_{j;l} &= \sum_{k \in \mathcal{D}_i} x^s_{ij;kl} && \forall r^s_{ij} \in \mathcal{R}^s \; \forall l \in \mathcal{D}_j \end{aligned} \tag{8}$$

(C3) Subproblems agree on variables' values:

$$\begin{aligned} x^i_{i;k} &= x^j_{i;k} && \forall x_i \in \mathcal{X} \; \forall x_j \in N(x_i) \; \forall k \in \mathcal{D}_i \\ x^i_{ij;kl} &= x^j_{ij;kl} && \forall r_{ij} \in \mathcal{R} \; \forall k \in \mathcal{D}_i \; \forall l \in \mathcal{D}_j \end{aligned} \tag{9}$$

Notice that the sets of constraints (C1) and (C2) ensure consistency in assignments inside each subproblem, whereas the set of constraints (C3) ensures consistency between subproblems.

In order to solve subproblems independently, we relax the constraints coupling subproblems, namely those in (9), by introducing them into the objective function using the technique of Lagrange multipliers [2]. Hence, the set of Lagrange multipliers $\{\lambda\}$ are:

$$\begin{aligned} &\lambda^j_{i;k} \cdot (x^i_{i;k} - x^j_{i;k}) && \forall x_i \in \mathcal{X} \; \forall x_j \in N(x_i) \; \forall k \in \mathcal{D}_i \\ &\lambda_{ij;kl} \cdot (x^i_{ij;kl} - x^j_{ij;kl}) && \forall r_{ij} \in \mathcal{R} \; \forall k \in \mathcal{D}_i \; \forall l \in \mathcal{D}_j \end{aligned} \tag{10}$$

Given a set of Lagrange multipliers $\{\lambda\}$, each original subproblem $\Phi_s = \{\mathcal{X}^s, \mathcal{D}^s, \mathcal{R}^s\}$ is transformed into a new subproblem $\overline{\Phi}_s = \{\mathcal{X}^s, \mathcal{D}^s, \overline{\mathcal{R}}^s\}$, whose objective function is defined as:

$$\begin{aligned} \overline{f^s}(X^s) = & \sum_{k \in \mathcal{D}_s} x^s_{s;k} \cdot (r^s_s(k) + \sum_{x_j \in N(x_s)} \lambda^j_{s;k}) \\ & - \sum_{x_j \in N(x_s)} \sum_{l \in \mathcal{D}_j} x^s_{j;l} \cdot \lambda^s_{j;l} \\ & + \sum_{r^s_{is} \in \mathcal{R}^s} x^s_{is;kl} \cdot (r^s_{is}(k,l) + \lambda_{is;kl}) \\ & + \sum_{r^s_{sj} \in \mathcal{R}^s} x^s_{sj;kl} \cdot (r^s_{sj}(k,l) - \lambda_{sj;kl}) \end{aligned} \tag{11}$$

Notice that equation 11 helps us implement the *divide* stage of the DaC approach. Thus, it realises the `modifySubproblem` function in the general algorithm 1 using the Lagrange multipliers as coordination parameters.

To assess a new set of Lagrange multipliers that minimize the violation of constraints, that is that bring solutions of subproblems $\{\overline{\Phi}_1, \ldots, \overline{\Phi}_m\}$ closer to an agreement, we propose to use the subgradient method [2]. The subgradient method is an iterative method that allows to update Lagrange multipliers in parallel from the solution of the neighbouring subproblems at each iteration. Following the subgradient method, Lagrange multipliers in subproblem $\overline{\Phi}_s$ at iteration t are updated as follows [2]:

$$\begin{aligned} \lambda^{j,t+1}_{s;k} &= \lambda^{j,t}_{s;k} - \gamma_t \cdot (x^{*,s}_{s;k} - x^{*,j}_{s;k}) && \forall k \in \mathcal{D}_s \\ \lambda^{s,t+1}_{j;l} &= \lambda^{s,t}_{j;l} - \gamma_t \cdot (x^{*,j}_{j;l} - x^{*,s}_{j;l}) && \forall l \in \mathcal{D}_j \\ \lambda^{t+1}_{sj;kl} &= \lambda^{t}_{sj;kl} - \gamma_t \cdot (x^{*,s}_{sj;kl} - x^{*,j}_{sj;kl}) && \forall k \in \mathcal{D}_s \; \forall l \in \mathcal{D}_j \\ \lambda^{t+1}_{is;kl} &= \lambda^{t}_{is;kl} - \gamma_t \cdot (x^{*,i}_{is;kl} - x^{*,s}_{is;kl}) && \forall k \in \mathcal{D}_i \; \forall l \in \mathcal{D}_s \end{aligned} \tag{12}$$

where $x^{*,s}_{s;k}$ is the optimal assignment for variable $x^s_{s;k}$ in subproblem $\overline{\Phi}_s$.

Notice that the equations in 12 help us implement the *coordinate* stage of the DaC approach. Thus, they realise the `coordinate` function in the general algorithm 1 by assessing the value of the Lagrange multipliers as coordination parameters. Following these equations, the value of a Lagrange multiplier is modified whenever the two subproblems involved with the multiplier disagree on the value assigned to a variable, and it remains unchanged if they agree. This can be interpreted as an attempt to reduce disagreement between subproblems. Moreover, the update of Lagrange multipliers considers a step-size γ_t (a positive real value) that weighs the impact of disagreements on updates. The larger the value of γ_t, the higher the impact of disagreements on Lagrange multipliers.

Observe that the subproblems generated by equation 11 conform a valid division of the original DCOP. Hence, propositions 1 and 2 apply to subproblems $\{\overline{\Phi}_1, \ldots, \overline{\Phi}_s\}$ so that: (1) if all subproblems agree on the value of shared variables then these values are the DCOP solution; and (2) the sum of the value of the solutions of individual subproblems provides a bound on the quality of the solution of the original problem.

4.2 DaCSA

In this section we formulate DaCSA, a bounded anytime DCOP algorithm that implements a DaC approach based on the formal foundations established in section 4.1. Agents running DaCSA start by creating simpler problems from their known relations of a DCOP, thus composing a valid division. Thereafter, they interleave *divide* and *coordinate* stages. During each *divide* stage, each agent modifies its subproblem by a set of coordination parameters (the Lagrange multipliers) using equation 11. Next, each agent solves its new subproblem to obtain the set of variable assignments that maximise its objective function. During each *coordinate* stage, each agent exchanges its local assignments with its neighbours and updates its coordination parameters using equation 12. Moreover, to return bounded anytime solutions, agents need to evaluate the candidate solution and to assess the bound.

The DaCSA algorithm is outlined in algorithm 2 (figure 3(a)). In what follows we detail each phase of DaCSA using the trace in figure 3(b) of a run over the DCOP in figure 1.

At the beginning of the algorithm, each agent a_i creates its local problem Φ_i using its local information in the DCOP (line 2) using equations 3-5. Thus, in figure 3, agent a_2 creates its local problem $\Phi_2 = \langle \mathcal{X}^2, \mathcal{D}^2, \mathcal{R}^2 \rangle$ for its variable x_2 where: (1)

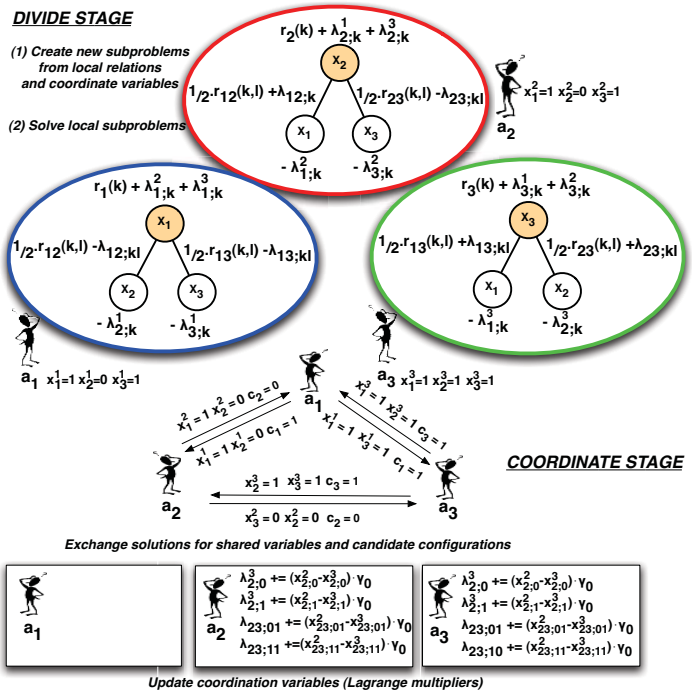
[2] At $t = 0$, each Lagrange multiplier is set to 0.

Algorithm 2. DaCSA(Φ)

Each agent a_i runs:

1: $bound \leftarrow \infty$; $\{\lambda^0\} \leftarrow 0$; $solution \leftarrow \emptyset$; $bestValue \leftarrow -\infty$; $C_i \leftarrow \emptyset$;
2: $\Phi_i \leftarrow$ createSubproblem$(\langle \mathcal{X}^i, \mathcal{D}^i, \mathcal{R}^i \rangle)$;
3: **repeat**
4: **/* Divide stage */**
5: $\overline{\Phi}_i \leftarrow$ modifySubproblem$(\Phi_i, \{\lambda_i\})$;
6: $(X^{*,i}, f_i^*) \leftarrow$ solveSubproblem(Φ_i);
7: **/* Coordinate stage */**
8: **for** $x_v \in N(x_i)$ **do**
9: $\Psi_i^v \leftarrow$makeCoordInfo$(x_i^{*,i}, x_v^{*,i}, f_i^*, C_i)$;
10: $\Psi_v^i \leftarrow$ exchangeCoordInfo(Ψ_i^v);
11: **end for**
12: $\gamma_t \leftarrow$ updateStepSize();
13: $\{\lambda\} \leftarrow$updateCoordParams$(\{\lambda\}, \gamma_t, X^{*,i})$;
14: **if** betterBoundAvailable$(\{\Psi\}, bound)$ **then**
15: Update $bound$.
16: **end if**
17: **if** betterSolAvailable$(\{\Psi\}, bestValue)$ **then**
18: Update $solution$ and $bestValue$.
19: **end if**
20: $C_i \leftarrow$ selectCandidateSolutions(x_i, C_i);
21: **until** any termination condition satisfied
22: **return** $\langle solution, bestValue, bound \rangle$

(a) DaCSA algorithm



(b) Trace of DaCSA.

Figure 3: DaCSA algorithm and its trace

$\mathcal{X}^2$ is composed of x_2 and its neighbours in the constraint graph, $\mathcal{X}^2 = \{x_1, x_2, x_3\}$; (2) $\mathcal{D}^2$ is the joint domain space for the variables in $\mathcal{X}^2$; and (3) $\mathcal{R}^2$ contains r_2, the unary relation for x_2, and a half of each binary relation involving x_2, namely $\frac{1}{2} \cdot r_{13}$ and $\frac{1}{2} \cdot r_{23}$.

***Divide* stage.** During a *divide* stage, each agent modifies its local problem with coordination information to subsequently solve it. Firstly each agent a_i obtains a new subproblem $\overline{\Phi}_i = \langle \mathcal{X}^i, \mathcal{D}^i, \overline{\mathcal{R}}^i \rangle$ (line 5) by modifying its local relations by applying equation (11) with its coordination parameters. Thus, for instance in figure 3(b), the new set of relations $\overline{\mathcal{R}}^2$ created by a_2 is composed of: (1) the unary relation r_2^2 along with Lagrange multipliers to coordinate the x_2 assignments with a_1, namely $\lambda_{2;k}^1$, and a_3, namely $\lambda_{2;k}^3$; (2) the binary relation r_{12}^2 along with the Lagrange multipliers $\lambda_{12;k}$ to coordinate the $\{x_1, x_2\}$ assignments with a_1;(3) the binary relation r_{23}^2 along with the Lagrange multipliers $\lambda_{23;kl}$ coordinate the $\{x_2, x_3\}$ assignments with a_3; and (4) Lagrange multipliers per neighbouring variables of x_2, namely $\lambda_{2;k}^2$ and $\lambda_{3;k}^2$, to coordinate the x_1 assignments with a_1 and the x_3 assignments with a_3.

Secondly, each agent a_i solves its new subproblem to obtain its optimal local solution, $X^{*,i}$, along with its value f_i^* (line 6). Since each individual subproblem $\overline{\Phi}_i$ stands for an acyclic DCOP, an agent can use any of the efficient solvers in the literature (e.g [15, 4]) to solve them. In figure 3(b), agents a_1, a_2, and a_3 solve their local subproblems to obtain the following local solutions: $X^{*,1} = \{x_1^{*,1} = 1, x_2^{*,1} = 0, x_3^{*,1} = 1\}$, $X^{*,2} = \{x_1^{*,2} = 1, x_2^{*,2} = 0, x_3^{*,2} = 1\}$, and $X^{*,3} = \{x_1^{*,3} = 1, x_2^{*,3} = 1, x_3^{*,3} = 1\}$ respectively. According to the DaC approach, agents' solutions may disagree after a *divide* stage, as it is the case in this example. Thus, agent a_2 disagrees with a_3 on the optimal configuration of variable x_2 and, consequently, on the optimal joint configuration for their variables x_2 and x_3.

***Coordinate* stage.** During a *coordinate* stage, each agent exchanges its local assignments with its neighbours and updates its coordination parameters trying to reduce the disagreement among them using subgradient method updates.

Before updating the coordination parameters, each agent a_i exchanges a message Ψ_i^v with each one of its neighbours a_v that contains the assignments for their common variables, namely x_i and x_v (lines 8-11). Thus, in figure 3, agent a_2 sends a message to a_1 with assignments $\{x_1^{*,2} = 1, x_2^{*,2} = 0\}$ and a message to a_3 with assignments $\{x_2^{*,2} = 0, x_3^{*,2} = 1\}$. Then each agent calculates the step-size γ_t for that iteration according to the chosen step-size rule (line 12). Each agent uses the assignments received from its neighbours to update the coordination parameters (Lagrange multipliers) following subgradient updates in equation 12. Thus, in figure 3, a_2 uses the assignments received from a_3 for their common variables, namely $\{x_2^{*,3} = 1, x_3^{*,3} = 1\}$ to update the coordination parameters trying to decrease the disagreement with a_3. Since both agents agree on the optimal value of x_3 the Lagrange multipliers that measure the disagreement over x_3, namely $\lambda_{3;0}^2$ and $\lambda_{3;1}^2$, remain unchanged. Moreover, since both agents agree on that joint configurations $\{x_2 = 0, x_3 = 0\}$ and $\{x_2 = 1, x_3 = 0\}$ are not optimal, the Lagrange multipliers $\lambda_{23;00}$, $\lambda_{23;01}$ also remain unchanged. Therefore, following equation 12, the only Lagrange multipliers updated by a_2 given a_3 assignments are $\lambda_{2;0}^3$, $\lambda_{2;1}^3$, $\lambda_{23;01}$ and $\lambda_{23;11}$. Thus, since a_2 sets x_2 to 0 and a_3 sets x_2 to 1, a_2 increases $\lambda_{2;0}^3$ and decreases $\lambda_{2;1}^3$ by γ_t. This decreases the utility of relation r_2 for its current optimal configuration, namely $r_2(0)$, and increments utility for a_1's optimal configuration, namely $r_2(1)$, thus making it easier for them to get to an agreement. The same happens for $\lambda_{23;01}$ and $\lambda_{23;11}$.

Additionally, when an agent a_i exchanges messages with its

neighbours for their common assignments, it also includes the candidate solution for variable x_i in the former iteration. This value, namely c_i, does not have to be the same as $x_i^{*,i}$, the value that maximises its individual subproblem. A typical strategy is that each agent a_i selects the configuration c_i for its variable x_i that most agents agree on. Following this strategy, in figure 3, agent a_2 would select $c_2 = 0$ because both a_1 and a_2 assigned 0 to x_2 whereas only a_3 assigned 1 to it. One can use different strategies simultaneously to generate the selected values. That is why we use C_i to note the set of candidate solutions (one for each strategy) for variable x_i.

Calculate bound and anytime solutions. To be able to return anytime solutions with quality guarantees, each agent must calculate the bound, assessed as the sum of solutions of all subproblems. Likewise each agent has to calculate the value of a candidate solution, assessed as the sum of all local values for that candidate solution, for each pair of *divide* and *coordinate* steps. However, in a distributed environment such as DaCSA, there is no single agent that knows these values: each agent a_i, at each iteration, only knows its local solution f_i^* and the local value for the candidate solution $f_i(\{C\}_i)$ where $\{C\}_i$ contains the candidate solution for x_i, namely c_i, and the candidate solutions for each of its neighbouring variables x_j, namely c_j, received from each neighbour during the $exchangeCoordInfo$. Thus, agents need a distributed protocol that allows them to calculate these aggregations of data and synchronize their bound and anytime solution updates. There are multiple [19, 7] protocols available in the literature to perform this task. We chose the one detailed in [19] because it allows agents to calculate the aggregations of local data without any additional message, only using the coordination messages exchanged during the execution of DaCSA. In this protocol, agents are initially arranged on a tree. For each value that has to be aggregated, each agent: (1) receives some data from its children, (2) aggregates these data and sends the results to its parent, (3) receives the aggregated value from its parent, and finally (4) relays this value to its children. These operations, that are interleaved with the DaCSA coordination messages, introduce little computation overhead. Moreover, the time and space requirements for each agent are linear the height of the chosen tree. When all agents have received the information related to the aggregated data for an iteration (e.g the value of the bound and of the candidate solution), they use it to update the bound (lines 14-16) and the anytime solution (lines 17-19), if applies. Because the aggregation process needs some message cycles to complete, agents will not have the actual anytime solution values during the first DaCSA iterations. Thus, in this initial phase, agents simply return the latest generated candidate solution without giving any guarantee on its quality.

Check termination conditions. At each iteration of the algorithm, each agent checks if some termination condition is satisfied. Typical termination conditions for DaCSA are: (1) the gap between the $bound$ and the value of the anytime solution is lower than a threshold; (2) the $bound$ has not been updated for a number of iterations; or (3) the number of current iterations exceeds a maximum.

4.3 Complexity analysis

For each iteration of the algorithm, agents run a communication round in which each agent exchanges a message with each one of its neighbours in the constraint graph. Therefore, the number of messages exchanged per iteration is $2 \cdot |E|$, where E is the set of edges of the constraint graph. A message from a_i to a_v contains the assignments for their two common variables along with the candidate solution for x_i. Therefore, the size of all exchanged messages is linear to the domain of variables. Messages are enhanced with the aggregation of data to calculate bounds and evaluate candidate configurations, which is linear to the height of the communication tree. With respect to computation, each agent at each iteration: (1) creates its own subproblem in parallel with the rest of agents; (2) solves a local problem composed of a two-level tree structured DCOP; and (3) updates the coordination parameters. The three steps (1) (2) (3) require a number of operations linear to the size of the local relations. As a result, DaCSA is a low-overhead algorithm because agents exchange a linear number of messages of linear size and performs a linear number of operations.

5. EMPIRICAL EVALUATION

In this section we compare DaCSA with two state-of-the-art approximate algorithms: Max-sum (MS) [4] and DSA [18]. Firstly, we explain the details of our experimental setup in section 5.1. Secondly, we analyze our empirical results in section 5.2.

5.1 Empirical settings

5.1.1 Problem generation

We perform our comparison over randomly generated DCOP problems with binary variables. The process of generating a DCOP is divided in two steps. Firstly, we generate a constraint graph, and afterwards we generate values for each of the relationships in the constraint graph. Several results in agent research have found that the network topology has a significant effect when solving a distributed problem (e.g. emergence of social conventions [17] or organizational adaptation [5]). In our experiments we analyze three network topology alternatives:

Small-world Many real-world networks, such a as food chains, electric power grids or social influence networks show the *small-world effect* [11], that is, the distance between any two nodes in the network is very small. We generate constraint graphs that show the small-world effect using the model proposed in [13]. The graphs are created by starting from a ring and adding a small number of random edges. In particular, for each node we use a probability $p = 0.3$ of adding a new random edge.

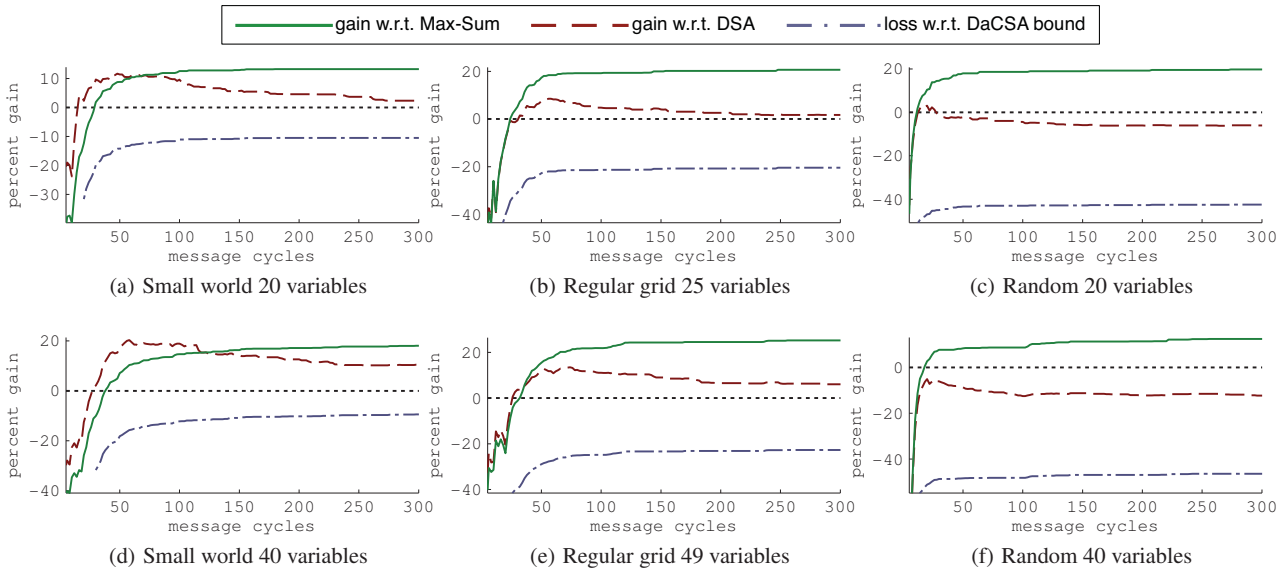
Regular grids The constraint graphs are rectangular grid where each agent is connected to its four closer neighbors.

Random networks The constraint graphs are created by randomly adding three links for each variable.

Once a constraint graph is generated, we must assess its constraints' values. We are interested in evaluating our algorithms in the presence of strong dependencies among the values of variables. At this aim, we generate constraint values by following an Ising model [1]. Ising models have been widely used in statistical physics. Following an Ising model, the weight of each binary relation r_{ij}, is determined by first sampling a value κ_{ij} from a uniform distribution $U[-\beta, \beta]$ and then assigning

$$r_{ij}(x_i, x_j) = \begin{cases} \kappa_{ij} & x_i = x_j \\ -\kappa_{ij} & x_i \neq x_j \end{cases}$$

Note that the constraint pushes both variables to be similar when κ_{ij} is positive and forces them to be different when κ_{ij} is negative. The β parameter controls the average strength of interactions. In our experiments we set β to 1.6. The weight for each unary constraint r_i is determined by sampling κ_i from a uniform distribution $U[-0.05, 0.05]$ and then assigning $r_i(0) = \kappa_i$ and $r_i(1) = -\kappa_i$.



(a) Small world 20 variables (b) Regular grid 25 variables (c) Random 20 variables

(d) Small world 40 variables (e) Regular grid 49 variables (f) Random 40 variables

Figure 4: Graphs showing the percent gain of DaCSA with respect to MS and DSA and the percent loss with respect to the DaCSA bound vs the number of message cycles on agent networks with different topologies and scales.

5.1.2 Algorithms' parameters

In this section we provide details on the particular parameters selected for DSA and DaCSA in these experiments.

For DSA we use an activation probability $p = 0.7$, a value that is reported to work well in [18]. Since DSA usually converges in a small number of iterations to get a fair comparison we restart it every time it converges keeping the best configuration among all converged solutions.

Regarding DaCSA, we must specify: (1) the strategy used by agents to generate configurations at each pair of *divide* and *coordinate* stages; and (2) how to assess the step-sizes γ_t in equation 12.

At each iteration two different candidate solutions are proposed by DaCSA. For the first one, each agent assigns to its variable the value on which more agents agree (explained in section 4.2) For the second one, each agent assigns to its variable the value on which more agents agree when the remaining variables in its subproblem are given the values selected by the candidate solution in the previous iteration. We assess the step-size at each iteration t as follows:

$$\gamma_t = \frac{1+m}{t+m} \cdot \frac{bestValue - bound}{(\sum_{\lambda \in \{\lambda\}} (\lambda^t - \lambda^{t-1})^2)^2} \qquad (13)$$

where $m = 5$. The intuition behind this formula is that the information transferred between agents for each constraint (γ_t) gets larger when the distance between the value of the best solution found so far and the bound grows, namely, when the algorithm is far from the optimal solution. Furthermore, it also gets larger when the level of disagreement among agents is smaller (there are fewer constraints among which we have to share the load). Each agent instantiates the bound, the value of the anytime configuration and the sum of Lagrange multipliers in equation 13 with the values of the last known *divide* and *coordinate* stages. At an early stage in the execution, when agents do not know yet the value of these parameters, they use a constant step-size $\gamma_t = 0.001/\sqrt{t}$.

5.2 Results

We compare these algorithms based on the solution obtained in a number of message cycles. The number of message cycles is a commonly used measure for algorithm efficiency in the DCOP literature [14, 12, 10]. It is specially adequate to our case because all the algorithms benchmarked are low-overhead algorithms.

At a concrete point in time, let q_D be the quality of the DaCSA anytime solution and q_x the quality of the solution of its competitor x. The percent gain of DaCSA with respect to x is assessed as $100 \cdot (\frac{q_D - q_x}{q_x})$. We also plotted the percent loss of DaCSA against the bound to have an idea of the accuracy of the bound.

5.2.1 Comparing DaCSA with DSA and Max-sum

Figures 4 (a) (b) and (c) show the results for 20 agents (25 in the regular grid) on a small-world, regular grid, and random structure respectively. Each graph shows the mean among 25 problem instances of the percent gain of DaCSA with respect to Max-sum (MS) and DSA when varying the number of message cycles (mcs) up to 300.

Over more realistic topologies (small world and regular grids) we observe that DaCSA outperforms Max-Sum and DSA. Concretely, at 50 message cycles, in small-world topologies, DaCSA gets a mean percent gain of around 10% with respect DSA and Max-sum. For regular grids, at 50 message cycles, the mean percent gain of DaCSA with respect to DSA and Max-sum is around 10% and 20% respectively. In both topologies, the gain with respect Max-sum remains nearly unchanged but the gain with respect to DSA is reduced although it never gets negative along the 300 messages cycles.

However, in random instances DaCSA performs slightly worse and, although it gets better results than Max-sum (it gets a mean gain of 20% at 50 message cycles) it is unable to outperform DSA.

The same conclusions (although more significant) hold for figures 4 (d), (e), and (f), as the number of variables increases to 40 (49 in the regular grid case). Hence, the network topology seems to be a key factor for DaCSA's performance.

5.2.2 DaCSA bound quality

Table 1 shows the quality guarantees provided by the DaCSA

by showing the mean of the approximation ratios given by DaCSA solution and its bound and its variance on different topologies.

Topology	Vars	Message cycles		
		50 mcs	100 mcs	300 mcs
Small-world	v20	1.17 ± 0.004	1.13 ± 0.002	1.12 ± 0.002
	v40	1.22 ± 0.004	1.14 ± 0.002	1.10 ± 0.002
Regular grids	v25	1.32 ± 0.006	1.28 ± 0.006	1.26 ± 0.004
	v49	1.41 ± 0.004	1.33 ± 0.003	1.29 ± 0.003
Random	v20	1.77 ± 0.007	1.76 ± 0.007	1.75 ± 0.007
	v40	1.96 ± 0.007	1.95 ± 0.012	1.88 ± 0.008

Table 1: Mean of approximation ratios on different topologies calculated from DaCSA solutions and bounds

The bound provided by DaCSA on realistic topologies is very accurate. The mean loss of DaCSA with respect to the bound is around 12% (approximation ratio of 1.17) in small-world and around 23% (approximation ratio of 1.32) in regular grids. The bound provided by DaCSA over random networks is less accurate.mean percent loss

These first empirical results show the potential of DaCSA because it provides good solutions on realistic topologies with highly-coupled matrices. Moreover, the results also show that the approximation ratio that the algorithm provides in this kind of problems is significant. DaCSA provides worse solutions on unstructured problems.

6. CONCLUSIONS

In this paper we addressed the problem of designing a low-overhead DCOP approximate algorithm that can assess a quality guarantee on its solutions. Thus, our first contribution is the *divide-and-coordinate* (DaC) approach, a novel approach to solve DCOPs. We have shown that any algorithm formulated in the DaC approach can naturally provide a bound on the DCOP solution while searching for an agreement.

Our second contribution is a novel bounded anytime DCOP algorithm, the so-called Divide-and-Coordinate Subgradient algorithm (DaCSA), a computational realization of the DaC approach based on Lagrangian decompositions and subgradient methods. When running DaCSA, agents can provide anytime solutions with quality guarantees while using little local computation and local communication. Finally, we provide empirical results for DaCSA showing its potential, particularly on problems over realistic topologies.

As future work, we plan to explore multiple lines. Firstly, we intend to benchmark DaCSA on a further number of datasets, to perform a better analysis of its performance and its dependence on the structure of the problem. Secondly, we aim at designing a version of DaCSA that adapts to changes so that it can be applied to dynamic environments. Finally, we plan to extend DaCSA to better trade-off communication with computation. This amounts to allowing agents to handle more complex subproblems and exchange information about their disagreement on larger combinations of variables.

Acknowledgments. This work has been funded by projects EVE (TIN2009-14702-C02-01), Agreement Technologies (CONSOLIDER CSD2007-0022, INGENIO 2010) and Generalitat de Catalunya under the grant 2009-SGR-1434. Meritxell Vinyals is supported by the Ministry of Education of Spain (FPU grant AP2006-04636).

7. REFERENCES

[1] R. Baxter. *Exactly Solved Models in Statistical Mechanics*. Academic Press, London, 1982.

[2] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2007.

[3] A. Farinelli, A. Rogers, and N. Jennings. Bounded approximate decentralised coordination using the max-sum algorithm. *IJCAI-09 Workshop on Distributed Constraint Reasoning (DCR)*, July 2009.

[4] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS*, May 2008.

[5] M. E. Gaston and M. DesJardins. Agent-organized networks for multi-agent production and exchange. In *AAAI*, pages 77–82, 2005.

[6] R. Junges and A. L. C. Bazzan. Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In *AAMAS*, pages 599–606, 2008.

[7] M. Y. Katsutoshi Hirayama, Toshihiro Matsui. Adaptive price update in distributed lagrangian relaxation protocol. In *AAMAS*, pages 1033–1040, 2009.

[8] N. Komodakis, N. Paragios, and G. Tziritas. Mrf optimization via dual decomposition: Message-passing revisited. In *ICCV*, pages 1–8. IEEE, 2007.

[9] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling. In *AAMAS*, pages 310–317, 2004.

[10] R. Mailler and V. R. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, pages 438–445, 2004.

[11] N. Mark. The structure and function of networks. *SIAM Review*, 45:167–256, 2003.

[12] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artif. Intell.*, 161(1-2):149–180, 2005.

[13] M. Newman and D. Watts. Renormalization group analysis of the small-world network model. *Phys. Lett. A.*, 263:341–346, 1999.

[14] J. P. Pearce and M. Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *IJCAI*, pages 1446–1451, 2007.

[15] A. Petcu. *A Class of Algorithms for Distributed Constraint Optimization*. PhD thesis, EPFL, Lausanne, 2007.

[16] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.

[17] J. M. Pujol, J. Delgado, R. Sangüesa, and A. Flache. The role of clustering on the emergence of efficient social conventions. In *IJCAI*, pages 965–970, 2005.

[18] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artif. Intell.*, 161(1-2):55–87, 2005.

[19] R. Zivan. Anytime local search for distributed constraint optimization. In *AAMAS*, pages 1449–1452, 2008.