# Divide-and-Coordinate by Egalitarian Utilities: Turning DCOPs into Egalitarian Worlds

Meritxell Vinyals,
J. A. Rodriguez-Aguilar
Artificial Intelligence Research Institute (IIIA)
Spanish Scientific Research Council (CSIC)
Campus UAB, Bellaterra, Spain
{meritxell,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

A Distributed Constraint Optimization Problem (DCOP) [7, 6] is a formal framework that can model many cooperative multi-agents domains. The *Divide-and-Coordinate* (DaC) framework [11] is one of the few general frameworks for solving DCOPs that provides bounds on solution quality for incomplete algorithms. In this paper, we formulate a novel DaC algorithm, the so-called Egalitarian Utilities Divide-and-Coordinate (EU-DaC) algorithm. The intuition behind EU-DaC is that agents would get closer to the agreement, that is to the optimal solution in DaC, when they communicate their local utilities for their decisions instead of their preferred decisions. We empirically show how this new algorithm outperforms DaCSA [11], the other DaC algorithm proposed so far, in all instances. We also show that it is very competitive when compared with bounded MGM k-optimal algorithms [5, 4], eventually outperforming them on some problem topologies. Our results also show how bounds provided by the DaC framework are much tighter than 2-optimal and 3-optimal bounds.

## 1. INTRODUCTION

In many cooperative multi-agents domains, such as sensor networks [13], distributed scheduling [10], and the configuration of power networks [10] a set of agents choose a set of individual actions whose rewards are dependent on the actions of other agents. A Distributed Constraint Optimization Problem (DCOP) [7, 6] is a formal framework proposed to model these cooperative networks where agents need to coordinate in a decentralized manner to find the joint action that maximize their joint reward.

Since solving a DCOP is NP-Hard [7], complete algorithms that focus on obtaining optimal solutions (e.g. ADOPT [7], OptAPO [6]) are usually unsuitable for dynamic and/or large-scale problems due to their computational and communication costs. Because of the unaffordable price of optimality, researchers have also formulated incomplete algorithms (e.g. DSA [13], DBA [13], max-sum [3]), which provide locally optimal solutions and only require a small amount of computation and local communication per agent. However, although these algorithms scale very well to large networks, they can converge to very poor solutions or even fail to converge. Another limitation is that they do not provide any quality guarantee on their solution, leaving agents with high uncertainty about the goodness of their decisions. As argued in [11, 9], quality guarantees can make a significant difference on incomplete algorithms because they allow agents to reason if it is worth investing more resources on improving their current decisions, to trade-off quality versus cost, by providing a bound on their maximum error. Some works [9, 11] have started to make headway on this direction by defining general frameworks that can provide quality guarantees over DCOP solutions even for incomplete algorithms.

On the one hand, there is the k-optimality framework [9] which defines quality guarantees for k-optimal solutions: solutions that can not be improved by changing any group of k or fewer agents decisions. The Maximum Gain Message algorithms [5, 4], namely MGM-2 and MGM-3, are DCOP approximate algorithms that converge to 2-optimal and 3-optimal solutions respectively.[1] On the other hand, Vinyals et al. [11] recently proposed the *Divide-and-Coordinate* (DaC) framework. In DaC agents iteratively divide an intractable DCOP into simpler local problems that can be individually solved by each agent and thereafter coordinate to reach an agreement over their assigments. As shown in [11], the DaC framework provides an upper bound on the quality of the optimal solution that agents can use to return per-intance quality guarantees. The Divide-and-Coordinate Sugradient Algorithm (DaCSA) [11] is a computational realization of the DaC framework, a bounded approximate DCOP algorithm in which agents coordinate by exchanging their preferred decision. However, several works [3, 12] have shown that communicating the utility of variables assignments instead of only the preferred assignments can lead to benefits in terms of solution quality.

It is this issue that we address in this paper, and to this end, we present a novel DaC algorithm, the Egalitarian Utilities Divide and Coordinate algorithm (EU-DaC). Agents running EU-DaC coordinate by exchanging the local utilities of their variables assigments with their neighbours. Concretely, this paper makes the following contributions:

*This work has been funded by projects IEA (TIN2006-15662-C02-01), Agreement Technologies (CONSOLIDER CSD2007-0022, INGENIO 2010) and EVE (TIN2009-14702-C02-01, TIN2009-14702-C02-02). Meritxell Vinyals is supported by the Spanish Ministry of Education (FPU grant AP2006-04636). JAR thanks JC2008-00337.

[1]MGM-1 is a k-optimal algorithm but no guarantees are given in the k-optimal framework for k=1.

- We formulate a novel computational realization of the DaC approach for which agents: (1) coordinate with their direct neighbours by exchanging their local utilities for shared variables assignments; and (2) update their local problems with the aim of getting closer to the utilities of their neighbours.

- We empirically evaluate the quality solutions of EU-DaC on different network topologies against state-of-the-art algorithms that provide quality guarantees (DaCSA, MGM-2 and MGM-3). Our empirical results show how EU-DaC outperforms DaCSA in all tested scenarios confirming the advantatges of communicating utilities instead of simply decisions. Moreover, it also shows that EU-DaC is competitive when compared with k-optimal algorithms (MGM-2 and MGM-3): EU-DaC solutions quality is similar to k-optimal algorithms and better on structured topologies.

- We experimentally compare the quality guarantees given by the different benchmarked algorithms: DaC quality bounds (EU-DaC and DaCSA) and k-optimal quality bounds (3-optimal for MGM-3 and 2-optimal for MGM-2). Results show that the bounds provided by EU-DaC are much tighter than k-optimal bounds, which for $k = 2$ and $k = 3$ are very loose.

This paper is structured as follows. In section 2 we give an overview of DCOPs and of the DaC framework. Next, in section 3 we describe our decentralised coordination algorithm, the EU-DaC algorithm. In section 4 we present our empirical evaluation of EU-DaC with respect to other state-of-the-art approximate algorithms with quality guarantees. Finally, we draw some conclusions and set paths to future work in section 5.

# 2. DCOP AND DIVIDE-AND-COORDINATE

## 2.1 DCOP Definition

A Constraint Optimization Problem (COP) consists of a set of variables, each one taking on a value out of a finite discrete domain. Each constraint (or 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:

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

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 [7, 6] 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 $r_{ij}$.

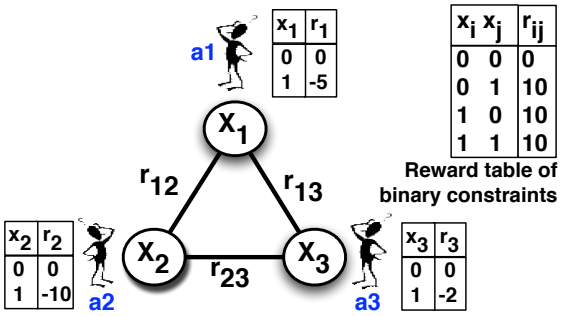



**Figure 1: Example of a DCOP constraint graph .**

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 binary DCOP in which agents choose values from $\{0, 1\}$ 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 relation $r_{13}$ with $a_3$. Each relation shows its rewards in a table. Thus, agent 3 has a reward of -2 to set its variable to 1 and each pair of agents have a reward of 10 when they set at least one of their variables to 1.

## 2.2 Divide-and-Coordinate framework

This section defines the Divide-and-Coordinate (DaC) framework, first introduced elsewhere [11]. The DaC framework is an approach that allows agents to distributedly solve a DCOP by exploiting the concept of agreement. The key idea behind the DaC approach is the following: 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. Therefore, in the DaC framework, agents start with the so-called *divide* stage in which they distributedly break the original problem into subproblems and individually solve them. Figure 2 shows an initial division in which each agent creates its local subproblem from its local relations. If a relation is shared among multiple agents, they split the relation by dividing the rewards in equal parts. Thus, the local problem of agent $a_1$ is composed of its local relation over its variable $x_1$ and all binary relations that include its variable, namely $r_{12}$ and $r_{23}$ with splitted rewards (table on the left shows rewards for binary relations). Naturally, when solving individual subproblems agents may assign different values to their sharing variables getting in conflict about their values. For instance, in the example of figure 2 variable $x_1$ is assigned by the three agents independently getting in conflict agent $a_1$ and $a_2$ over the value of its assignment.

Thus, agents proceed to coordinate, in the so-called *coordinate* stage, by exchanging some coordination information, namely $\{\Psi\}$ about their disagreements with their neighbours. Agents subsequently employ such information to update their underlying local subproblems to create a new division, in the next *divide* step, that brings them closer to an agreement. Thus, in example of figure 2, when coordinating agents $a_1$ and $a_2$ will exchange information

about their conflict over $x_1$ that will use to update their local subproblems.

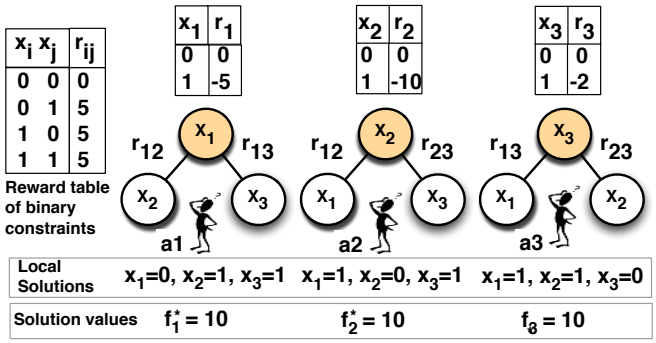



**Figure 2: Subproblems for the DCOP in figure 1, divide step ($t = 0$) .**

The DaC framework allows agents to distributedly provide bounded solutions for DCOPs by making use of the following two properties (described in [11]):

(**Proposition 1**) *the sum of the solutions of individual agents' subproblems is always an upper bound on the quality of the global (optimal) DCOP solution.*

(**Proposition 2**) *if all agents reach an agreement on a joint solution when optimizing their local subproblems, such a solution is the optimal one.*

Thus, to solve a DCOP by DaC, agents update their local subproblems by exchanging information with their neighbours, exploring the space of valid divisions, to find a division such that the solution of individual subproblems agree (since they know by proposition 2 that this will be the solution of the DCOP). However, even when agents do not agree on their assignments, they can provide with bounded anytime solutions by generating assignments closer to the agreement (that are expected to be better than randomly generated) bounded by the upper bound on its quality of proposition 1.

The DaC framework is an abstract approach that can result in different bounded approximate algorithms for DCOPs because the information that is exchanged among agents in the *coordinate* step and how agents use such information to update their problems in the *divide* step is not specified. The DaC framework only requires that after each agent updates each underlying problem the set of problems still are a original division of the DCOP. Thus, in [11], Vinyals et al. formulated the first DaC algorithm, the Divide-and-Coordinate Subgradient algorithm (DACSA) where agents coordinate by exchanging their preferred decisions on a formalism based on Lagrangian dual decomposition and subgradient methods.

Next, we will formulate a novel particular computational realization of the DaC approach.

# 3. EGALITARIAN UTILITIES DIVIDE-AND-COORDINATE

In this section we formulate the so-called Egalitarian Utilities Divide and Coordinate algorithm (EU-DaC), a novel computational realisation of the DaC approach where agents coordinate by exchanging their max-marginal utilities to set their shared decision variables to particular values. Several work in optimization [3, 12] have shown that agents lead to better solutions when they explicitly communicate their utilities for taking particular decisions than when they simply exchange their preferred decisions.

In the DaC algorithm proposed so far, DaCSA [11], agents coordinate by communicating their preferred decisions. Here we propose a new DaC algorithm, EU-DaC, that has each agent: (1) exchanges its local utilities for its shared variables with its neighbours (*coordinate* stage); and (2) updates its local problem with the aim of approaching its local utilities for its shared variables to its neighbours' utilities (*divide* stage) .

The intuition behind the EU-DaC is the following: when agents have the same utilities for setting their shared variables to particular values, they agree on their local assignments[2]. As explained in section 2.2, in the DaC framework this agreement situation also implies that they have found a DCOP solution.

In EU-DaC agents will start by exchanging their max-marginal utilities over their shared variables with their neighbours. The max-marginal utilities of an agent $a_s$ to set some decision variables to particular values is the best utility given by its local subproblem $\Phi_s$ when restricted to assignments that satisfy this condition. More formally, the max-marginal utility of an agent $a_s$ for setting a subset of decision variables $\mathcal{X}\rho \subseteq \mathcal{X}_s$ to some values $d_\rho \in \mathcal{D}_\rho$, namely $\mathcal{U}^s_\rho(d_\rho)$, is defined as:

$$\mathcal{U}^s_\rho(d_\rho) = \max_{d \in \mathcal{D}_{\mathcal{X}_s \setminus \mathcal{X}\rho}} f_s(d_\rho; d) \qquad (2)$$

where $f_s$ is the local objective function of $a_s$. Take as example figure 3(a) that shows agents' utilities exchanged during these coordinate step given subproblems of figure 2. Observe that agent $a_1$ exchanges with its neighbour $a_2$ a message that contains its local utilities for their shared variables, namely $x_1$ and $x_2$. In the example, agent $a_1$ assesses its local max-marginal utilities for its variable $x_1$ as:

$$\mathcal{U}^1_1(0) = \max_{d \in \mathcal{D}_{23}} r_1(0) + r_{12}(0,d) + r_{13}(0,d) = 10$$
$$\mathcal{U}^1_1(1) = \max_{d \in \mathcal{D}_{23}} r_1(1) + r_{12}(1,d) + r_{13}(1,d) = 5$$

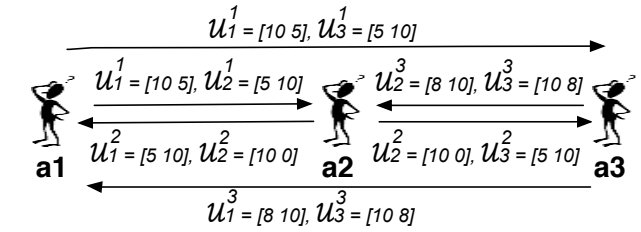
Hence, agent $a_1$ reports a local max-marginal utility of 10 when setting its variable $x_1$ to 0 and a local max-marginal utility of 5 when setting it to 1.

Once received these max-marginal utilities, agents proceed to use these information to update its local subproblems in order to get utilities closer to those reported by their neighbours. Thus, agents aim to finding a division of subproblems $\{\Phi_1, \ldots, \Phi_m\}$ such that the set of local max-marginal utilities $\{\mathcal{U}^{s=1,\ldots,m}\}$ for shared variables among the different agents are equal. Formally:

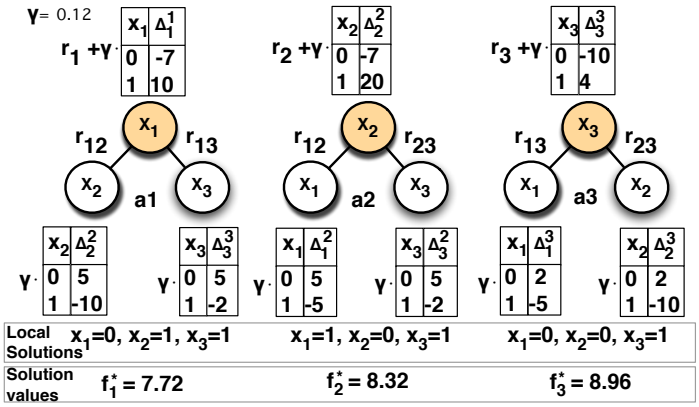
$$\mathcal{U}^i_i(k) = \mathcal{U}^v_i(k) \quad \forall x_i \in \mathcal{X}\ \forall k \in \mathcal{D}_i\ \forall x_v \in N(x_i) \qquad (3)$$

Thus, with the aim of satisfying equation 3, each agent $a_s$ proceeds to update its max-marginal utilities for shared variables by adding the difference between its own local utilities $\{\mathcal{U}^s\}$ and the max-marginal utilities reported by each of its neighbour $a_v \in N(a_s)$, namely $\{\mathcal{U}^v\}$. Thus, each agent $a_s$ updates its max-marginal utili-

[2]This statement is subject to having no ties in agents' local utilities: agent's local utilities to set its shared variables to particular values in their domain are all different.




(a) Coordination step ($t = 0$)




(b) Divide step ($t = 1$)

**Figure 3: EU-DaC execution**

ties $\{\mathcal{U}^s\}$ using the following equation:

$$\{\mathcal{U}^s\} = \{\mathcal{U}^s\} + \sum_{a_v \in N(a_s)} [\{\mathcal{U}^s\} - \{\mathcal{U}^v\}] \quad (4)$$

Each agent problem $\Phi_s$ can be reparameterized in terms of a set of max-utilities over its variables, namely $\{\mathcal{U}^s\}$.[3] Concretely, in the particular case of a binary subproblem $\Phi_s = \langle \mathcal{X}^s, \mathcal{D}^s, \mathcal{R}^s \rangle$ with a tree topology (as the ones shown in figure 2) they can be represented in function of its max-marginal utilities over single and pairwise variables as follows:

$$f_s(d) = \sum_{x_i \in \mathcal{X}^s} \mathcal{U}_i^s(d) + \sum_{r_{ij}^s \in \mathcal{R}^s} \left[\mathcal{U}_{ij}^s(d) - \mathcal{U}_i^s(d) - \mathcal{U}_j^s(d)\right]$$

Thus, in the example of figure 2 agent $a_1$ can represent its objective function as $f_1(d_1, d_2, d_3) = \mathcal{U}_1^1(d_1) + \mathcal{U}_2^1(d_2) + \mathcal{U}_3^1(d_3) + \mathcal{U}_{12}^1(d_1, d_2) - \mathcal{U}_1^1(d_1) - \mathcal{U}_2^1(d_2) + \mathcal{U}_{13}^1(d_1, d_3) - \mathcal{U}_1^1(d_1) - \mathcal{U}_3^1(d_3)$.

Notice that, using this representation, max-marginal utilities over single variables appear at least once in each problem where the variable is included. Therefore, at each iteration $t$, each agent $a_s$ updates each subproblem by adding, for each variable in its subproblem $x_i \in \mathcal{X}^s$, the (weighted) difference between its max-marginal utilities and those reported by agents with which it shares such variable. Hence, the agent objective function is updated as:

$$f_s^t(d) = f_s^{t-1}(d) + \gamma \cdot \left[\Delta_s^{s,t} + \sum_{x_i \in N_s} \Delta_i^{s,t}\right] \quad (5)$$

[3]This can be proved by the junction tree theorem[2], that states that any distribution $F$ compiled into a junction tree can be reparameterized in function of the gains (max-marginals) of its cliques $\{\mathcal{C}\}$ and separators $\{\mathcal{S}\}$

where $\Delta_s^{s,t}$ is the coordinator parameter related to its variable $x_s$:

$$\Delta_s^{s,t} = \textstyle\sum_{x_i \in N_s} \left[\mathcal{U}_s^{i,t}(d) - \mathcal{U}_s^{s,t}(d)\right], \quad (6)$$

$\Delta_i^{s,t}$ is the coordinator parameter related to variable $x_i \in N(x_s)$ :

$$\Delta_i^{s,t} = \mathcal{U}_i^{i,t}(d) - \mathcal{U}_i^{s,t}(d), \quad (7)$$

and $\gamma \in (0, 1]$ is a damping parameter that weighs the change over the subproblem.

Next, we describe in detail the phases that agents execute during the EU-DaC algorithm.

## 3.1 Algorithm description

In this section we fully describe the EU-DaC algorithm, a bounded anytime DCOP algorithm that computationally realises and interleaves the *divide* and *coordinate* stages. On the one hand, during each *divide* stage, each agent updates its local problem by adding the difference between its local max-marginals utilities with those of its neighbours according to equation 5. Then, each agent solves its updated local problem to update its preferred assignments and its assignments value. On the other hand, during the *coordinate* stage, each agent exchanges its local max-martinal utilities over single variables shared with its neighbours. In order to provide anytime solutions even in the case of disagreement, each agent generates at each iteration what is called a *candidate solution* for its variable in the exactly the same way as in DaCSA.

Algorithm 1 presents the pseudocode for EU-DaC. In what follows we describe the main stages of EU-DaC using the trace in figure 3 of a run over the DCOP in figure 1.

***Initialization* stage** (lines 1-2). At the beginning of the algorithm each agent $a_i$ creates its local problem $\bar{\Phi}_i^0$ using its local relations. Relations shared with its neighbours (binary relations) are split in equal parts. Notice that for binary DCOPs, these are always tree-structured problems (acyclic). An example of these initial division for the DCOP of figure 1 is given in figure 2.

***Divide* stage** (lines 4-6). During a *divide* stage, each agent updates its current local problem $\bar{\Phi}_i^t$ with coordination information to subsequently solve it. Firstly, each agent $a_i$ updates its local subproblem by using the coordination information gathered during the last *coordinate* stage, namely $\Psi_i^t$, using equation 5 (line 5,implemented in method $modifySubproblem$). These coordination messages contain the local max-marginal utilities of their neighbours over their shared variables. Secondly, each agent $a_i$ solves the acyclic COP that composes its local subproblem to obtain its optimal assignments, $d_i^*$, its value $f_i^*$ and its max-marginal utilities to have their individual variables in each particular state, $\{\mathcal{U}^i\}$ using the max-sum solver [3] (line 6, implemented in method $solveSubproblem$) [4]. Notice that in the very first iteration, agent do not have coordination information and therefore they solve the very initial subproblem. Figure 2 shows agents' local solutions and their values for the initial subproblems. Observe that each agent $a_i$ prefers to set its variable $x_i$ to 0 and the rest of variables to 1 reporting an individual utility $f_i$ of 10. Figure 3(b) shows the same example but in the next *divide* step, where agents have coordination information to update their local subproblems. Thus, for instance, agent $a_1$ creates a new subproblem that is composed of

[4]Although one can use other solvers such that can solve acyclic problems performing a linear number of operations, max-sum is useful because it returns at the same time the max-marginal utilities over single variables

**Algorithm 1 EU-DaC**$(\Phi, \gamma)$

Each agent $a_i$ runs:

1: $bound \leftarrow \infty$; $\{\Psi_i^0\}, \{\Delta^{i,0}\}, solution, C_i \leftarrow \emptyset$; $bestValue \leftarrow -\infty$;
2: $\overline{\Phi}_i^0 \leftarrow$ createSubproblem$(\langle \mathcal{X}^i, \mathcal{D}^i, \mathcal{R}^i \rangle)$;
3: **repeat**
4: **/* Divide stage */**
5: $\overline{\Phi}_i^t \leftarrow$ modifySubproblem$(\overline{\Phi}_i^{t-1}, \{\Delta^{i,t}\}, \gamma)$;
6: $(d_i^{*,t}, f_i^{*,t}, \{\mathcal{U}^i\}) \leftarrow$ solveSubproblem$(\overline{\Phi}_i^t)$;
7: **/* Coordinate stage */**
8: **for** $x_v \in N(x_i)$ **do**
9: $\Psi_i^v \leftarrow$ makeCoordInfo$(d_i^{*,t}, f_i^{*,t}, \langle \mathcal{U}_v^i, \mathcal{U}_i^i \rangle, C_i^{t-1}, \{\Psi\})$;
10: $\Psi_v^i \leftarrow$ exchangeCoordInfo$(\Psi_i^v)$;
11: **end for**
12: $\{\Delta_i^{t+1}\} \leftarrow$updateCoordParams$(\{\Psi_i^t\})$;
13: $C_i^t \leftarrow$ selectCandidateSolutions$(x_i, C_i^{t-1})$;
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: **until** any termination condition satisfied
21: **return** $\langle solution, bestValue, bound \rangle$

its initial relations $r_1$. $r_{12}$ and $r_{13}$ along with a weighted coordination parameter $\Delta_i^1$ (weighted by a damping factor $\gamma$) for each one of its variables $x_i \in \mathcal{X}^1$. Using a damping factor $\gamma = 0.12$, agent $a_3$ changes its optimal solution respect to the first iteration. Moreover all agents get a lower value for their local solution than respect to the first iteration. Notice that getting lower utilities for subproblems' solutions is a good indicator because their addition is an upper bound on the optimal solution. Thus, in the DaC framework when agents report lower solution values, their values and their solutions are closer to the optimal ones.

***Coordinate* stage**. During a *coordinate* stage, each agent exchanges coordination information with its neighbours and updates its coordination parameters trying to balance the disagreement among them. Before updating the coordination parameters, each agent $a_i$ exchanges a message $\Psi_i^v$ with each one of its neighbours $a_v$ that contains its max-marginal utilities for their common variables, namely $x_i$ and $x_v$ (lines 8-11). Figure 3(a) shows the max-marginal utilities that are exchanged among agents during the first coordination stage in the example of figure 2. Thus, for example, agent $a_1$ sends to $a_2$ the max-marginal utility over its variable $x_1$, namely $[\mathcal{U}_1^1(0) = 10, \mathcal{U}_1^1(1) = 5]$, and the max-marginal utility over $x_2$, namely $[\mathcal{U}_2^1(0) = 5, \mathcal{U}_2^1(1) = 10]$. Next, each agent $a_1$ uses the max-marginal utilities received from its neighbours to update the coordination parameters $\{\Delta^i\}$ following the updates in equations 6 and 7 (line 12). Thus, in the example of figure 3(a), agent $a_1$ assesses the coordination parameters $\Delta_1^1(0)$ as the difference between the local utility of $a_2$, $\mathcal{U}_1^2(0) = 5$, and its local gain, $\mathcal{U}_1^1(0) = 10$, plus the difference between the local utility of $a_3$, $\mathcal{U}_1^3(0) = 8$, and its local utility: $\Delta_1^1(0) = (5 - 10) + (8 - 10) = -7$.

Also, in each *coordinate* stage, each agent $a_i$ selects what is called a *candidate* solution for its decision variable $x_i$ (line 13). This value, namely $c_i$, does not have to be the same as the preferred assignment of $a_i$ for $x_i$ because to generate candidate solutions agents use the coordinate information received from its neighbours in addition to their local assignment. Although agents can use different strategies to generate their candidate solutions, the intuition is to generate assignments, in presence of disagreement, as much close to the agreement as they can. For example, a typical strategy is that each agent $a_i$ selects the solution $c_i$ for its variable $x_i$ that most agents agree on. Following this strategy, in the example of figure 2, each agent $a_i$ assigns $c_i$ to 1 as a candidate solution for its variable $x_i$ because all neighbours assigned $x_i$ to 1 except from $a_i$ that set it to 0. Thus, the global candidate solution selected is $c_1, c_2, c_3 = 1$. In contrast, in the next coordination step agents $a_1$ and $a_2$ will select as candidate solutions $c_1 = c_2 = 0$ (see figure 3(b) ), two of the three agents assigned $x_1$ and $x_2$ to 0. One can use different strategies simultaneously to generate the selected values. That is why we use $\mathcal{C}_i$ to note the set of candidate solutions (one for each strategy) for variable $x_i$. Finally, each agent communicates the candidate solution for its variable to its neighbours as a coordination information using the messages exchanged during the next coordinate stage (line 9-10).

**Calculate bound and anytime solutions**. In order to allow agents to return bounded anytime solutions from the optimal we need to provide agents with a protocol that allows them to distributedly evaluate their candidate solutions and assess the bound. By anytime we mean that agents in EU-DaC hold the best assignment that was generated throughout the search. However it does not mean that solutions will always increment their quality or that the optimal solution will be found if is given more time because EU-DaC can converge to a local optimum. In a distributed environment, each agent $a_i$ only knows for each iteration its local solution $f_i^*$ and the local value for the candidate solution $f_i(\{C\}_i)$. Thus, in the example of figure 2, agent $a_1$ only knows for the first iteration the value of its local solution, namely $f_1^{*,1} = 10$, and its local value for the candidate solution, namely $f_1(c_1 = 1, c_2 = 1, c_3 = 1) = 5$. Thus, agents need a protocol that allows them to distributedly assess the value of the candidate solution, defined as the sum of all local candidate solutions values, and the bound, defined as the sum of the optimal value of all subproblems. With that purpose, as in DaCSA, EU-DaC implements the protocol detailed in [14] that allows agents to calculate these aggregations of data and synchronize their bound and anytime solutions updates. This protocol requires no additional network load (it uses the coordination messages $\Psi$ already exchanged during the coordination stage to propagate their data) and small (linear) additional space. When all agents have received the coordination information related to the aggregated data for an iteration, they use it to update the bound (lines 14-16) and the anytime solution (lines 17-19), if applies. Thus, in the example of figure 2, agents will have, after a number of propagation cycles, the value of the candidate solution for the first iteration $f(c_1 = 1, c_2 = 1, c_3 = 1) = 13$ and the value of the bound $bound = 30$. Thus, agents will update to solution $[c_1 = 1, c_2 = 1, c_3 = 1]$ with a quality guarantee that the current solution has at least $13/30 \cdot 100 = 43.33\%$ of the quality of the optimal solution. In the next iteration (see figure 3(b)), agents receives the value of the candidate solution for the second iteration $f(c_1 = 0, c_2 = 0, c_3 = 1) = 18$ and a $bound = 25$. Thus, agents update their best solution to $[c_1 = 0, c_2 = 0, c_3 = 1]$ with a quality guarantee of $18/25 \cdot 100 = 75\%$.

**Termination conditions**. At each iteration, each agent checks if some termination condition is satisfied. Typical termination conditions for EU-DaC are: (1) the gap between the bound and the value

of the anytime solution is lower than a threshold; (2) max-marginal utilities are equal across agents (equation 3 is satisfied); or (3) the number of current iterations exceeds a maximum. Notice that unlike DaCSA, in EU-DaC agents can detect convergence even in the case when they have not found the solution. Thus, if condition (2) is satisfied it means that the max-marginal utilities are equal across agents but contains ties, so EU-DaC will not be able to improve after that point.

# 4. EMPIRICAL EVALUATION

In this section we provide an empirical evaluation of EU-DaC on different network topologies where agents have highly-coupled dependencies. Moreover we benchmark EU-DaC against other DCOP algorithms that can also provide quality guarantees: DaCSA [11] and bounded k-optimal Maximum-Gain-Message (MGM) algorithms [5, 4], namely MGM-2 and MGM-3 [5], by comparing their solution quality over time and the accurancy of their quality guarantees.

Firstly, we describe the different network topologies and how we generate the relations' weights in section 4.1. Next, we analyze our empirical results over these datasets in section 4.2.

## 4.1 Problem generation

Following [11], we perform our comparison on different network topologies where agents have highly-coupled dependencies. Thus, in our experiments we analyze three network topology alternatives:

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

**Small-world** We generate constraint graphs that show the small-world effect using the model proposed in [8]. The graphs are created by starting from a ring, where each node its connected to its two closer neighbours and adding a small number of random edges. In particular, for each node we use a probability $p = 0.3$ of adding a new edge that connects it to another random node.

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

As in [11], we are interested in evaluating our algorithm on the presence of strong dependencies among agents. Therefore we also generate constraint values by using mixed Ising model weights [1]. Following an Ising model, the weight of each binary relation $r_{ij}$, is determined by first sampling a value $\kappa_{ij}$ 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 $\kappa_{ij}$ is positive and forces them to be different when $\kappa_{ij}$ is negative. The $\beta$ parameter controls the average strength of interactions. In our experiments we set $\beta$ to 1.6. The weight for each unary constraint $r_i$ is determined by sampling $\kappa_i$ from a uniform distribution $U[-0.05, 0.05]$ and then assigning $r_i(0) = \kappa_i$ and $r_i(1) = -\kappa_i$.

[5]MGM-1 is excluded of the comparative because it is a 1-optimal algorithm and no bound can be provided with $k = 1$ (see [9] )

## 4.2 Results

Next, we provide details on the particular parameters selected for EU-DaC, MGM-{2,3}[6] and DaCSA in these experiments.

For MGM-2 and MGM-3 we set the probability $q$ of being an offerer to .9, a value that is shown to reach the highest average solution quality on the experiments reported in [5]. Regarding DaC algorithms (EU-DaC and DaCSA), we must specify the strategy used by agents to generate configurations at each pair of *divide* and *coordinate* stages. We use the same two strategies proposed in [11]: (1) each agent assigns to its variable the value in which more agents agree on; and (2) each agent assigns to its variable the value in which more agents agree on when the remaining variables in its subproblem are given by the values selected by the candidate solution in the previous iteration. For EU-DaC we set the value of the damping parameter $\gamma$ to .5. Finally for DaCSA we used the same step-size for the subgradient step as the one reported in [11], also using a constant step-size of .001 during the first steps when agents do not know any subgradient value.

### *4.2.1 EU-DaC solution quality*

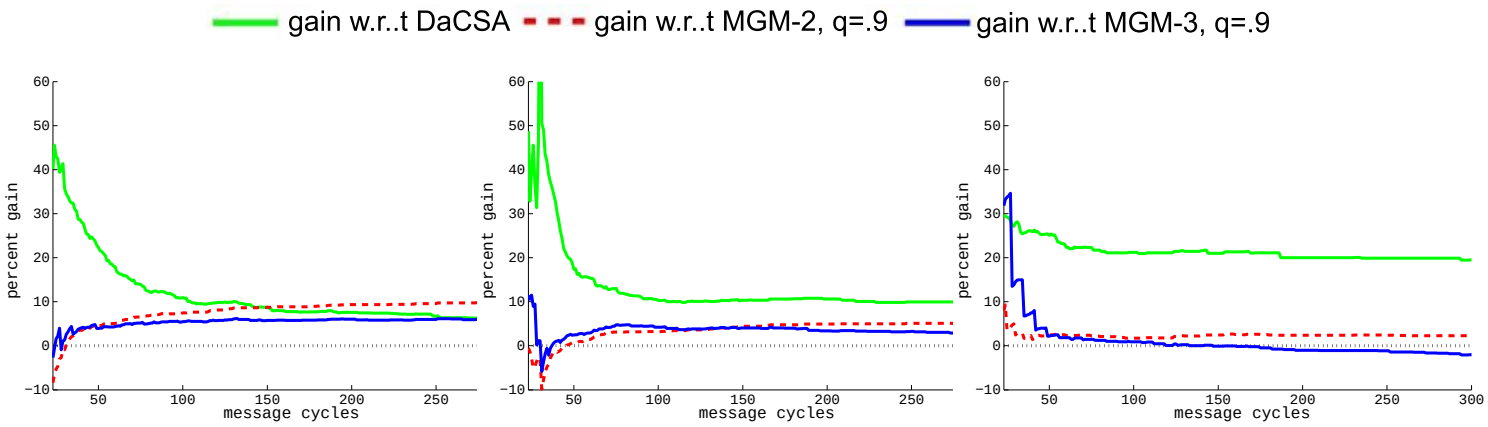
Firstly, we compare these algorithms based on the solution obtained in a number of message cycles. The number of message cycles is a commonly used measure for algorithm efficiency in the DCOP literature [9, 7, 6]. It is specially adequate to our case because all the algorithms benchmarked are low-overhead algorithms. To normalize plots, instead of using the mean of the quality of the solutions we plot the percent gain of EU-DaC respect each benchmarked algorithm. The percent gain of EU-DaC with respect to an algorithm $A$ at iteration t is assessed as $100 \cdot (\frac{q_D - q_A}{q_A})$ where $q_D$ is the value of the solution of EU-DaC algorithm and $q_A$ is the value of the solution of $A$ algorithm. Thus positive values in graphs represent positive gains of EU-DaC respect to other algorithm (higher is better).

Figures 4 (a) (b) and (c) show the results for networks of 100 agent networks on a small-world, regular grid and random topology respectively. Each graph shows the mean among 25 instances of the percent gain of EU-DaC respect to DaCSA, MGM-2 and MGM-3 when varying the number of message cycles.

First, observe that in all experimented topologies EU-DaC outperforms DaCSA, the other DaC algorithm. These results show that indeed when agents explicitly communicate their max-marginal utilities to be in a particular state instead of their decisions and try to get balanced on them they get better results closer to an agreement. Observe that while EU-DaC obtain higher gains, around $40 - 60\%$ respect to DaCSA on structured topologies (small-world and random networks, figures 4 (a)(b)) when increasing messages cycles these gains quickly reduce to around $10\%$. In random networks (4 (c)), however, the gains of EU-DaC respect to DaCSA are initially lower (around $30\%$) but remain more constant when increasing the number message cycles (around $20 - 30\%$).

Secondly, when comparing with MGM algorithms (MGM-2 and MGM-3) we observe that EU-DaC outperforms MGM-2 in all the scenarios and the same applies to MGM-3 on structured topologies. The gains of EU-DaC respect to MGM-2 and MGM-3 are lower than respect to DaCSA (around $5 - 10\%$). In random topologies EU- DaC get even negative gains respect to MGM-3 on the

[6]For MGM-2 and MGM-3 we use the code provided in *http://teamcore.usc.edu/dcop/*




(a) Small world 100 variables (b) Regular grids 100 variables (c) Random 100 variables

**Figure 4: Graphs showing the percent gain of EU-DaC with respect to DaCSA, MGM-2 and MGM-3 when varying the number of message cycles over different topologies and with mixed Ising weights $\beta = 1.6$**

long run. Thus, we can conclude that EU-DaC is very competitive when compared with MGM-2 and MGM-3 getting similar results and even outperforming them on some problem topologies.

### 4.2.2 *EU-DaC bound quality*

In this section we compare the quality guarantees of EU-DaC with those of DaCSA, MGM-2 and MGM-3 by plotting the percent bound quality of their solution when varying the number of message cycles. The percent bound quality of an algorithm $A$ is assessed as $100 \cdot \frac{q_A}{ub_A}$ where $q_A$ is the value of the solution of $A$ algorithm and $ub_A$ is an upper bound on the value of the optimal solution. Intuitively, a percent bound quality of $y$ says that the current algorithm solution has at least a $y$ percent of the quality of the optimal.

Before analysing the results we should make some comments over the quality guarantees provided by each algorithm. One the one hand, the quality guarantees of EU-DaC and DaCSA are those given by the DaC framework. As explained in section 2.2, DaC algorithms can explicitly calculate an upper bound on the quality of the optimal solution defined as the sum of all individual subproblems solutions in a division. Then, agents assess the percent bound quality using the value of the best evaluated candidate solution so far and the lower upper bound among all tested divisions. Hence, DaC quality guarantees are what are called *instance-per-basis* quality guarantees which depend on the specific problem instance, are known on runtime and vary along the execution of the algorithm. Furthermore, since agents are able to calculate the upper bound in an explicit manner, they can give quality guarantees for every solution generated by the algorithm.

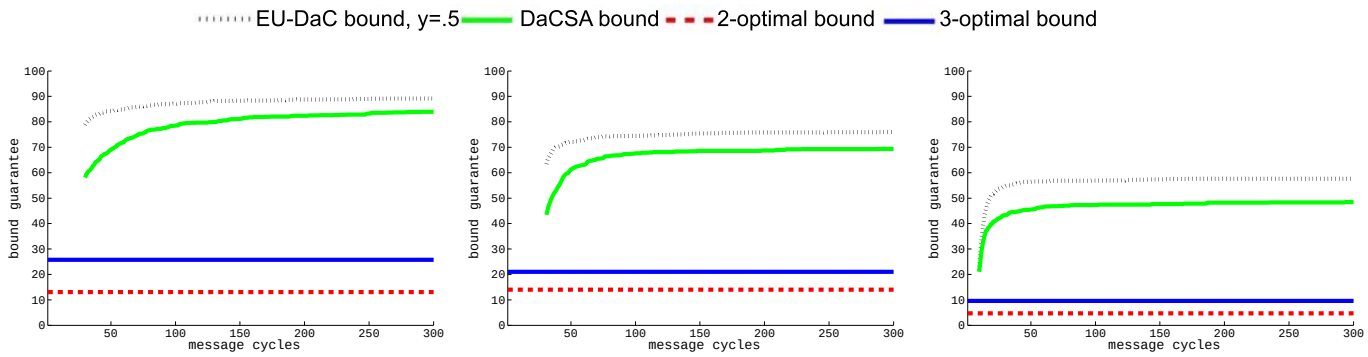
On the other hand the quality guarantees given by MGM algorithms are those of the k-optimal framework: a *worst-case* bound over k-optimal solutions. Two different quality guarantees have been formulated for k-optimal solutions [9]: (1) *general quality guarantees* that only depend on the number of variables and number of relations of the problem; and (2) *graph-based quality guarantees* that use the knowledge of the topology to obtain tighter guarantees. In our experiments, we always plot the *graph-based quality guarantees* which are assured to be better than the general ones. To assess graph-based quality guarantees agents need to solve a linear-fractional problem. That MGM algorithms quality guarantees are worst-case bounds on k-optimal solutions implies that : (a) they have a fixed quality guarantee for any of their converged solutions which can be calculated offline; and (b) they can only provide quality guarantees for solutions generated on convergence (MGM algorithms only achieve a k-optimal solution on convergence).

Figure 5 shows the mean of percent bound qualities provided by DaC algorithms (EU-DaC and DaCSA) and the graph-based guarantees of MGM-2 and MGM-3 over their converged solutions when varying the number of message cycles on the different topologies. First observe that topology influences the quality guarantee of all tested algorithms. In all cases, quality guarantees are higher on small-world topologies than on regular grids than on random topologies. Secondly, results show that the DaC bounds are significantly higher than k-optimal bounds. Moreover, the bounds provided by EU-DaC are always higher (5-10%) than those provided by DaCSA. It is not surprising since DaC quality guarantees use the quality of the best solution, which is higher for EU-DaC, to calculate the bound. For small-world topologies, EU-DaC gives a mean of percent bound of around $85\%$ whereas those of MGM-2 and MGM-3 are around $15\%$ and $30\%$ respectively. For regular grids, EU-DaC gives a mean of percent bound of around $70\%$, a meaningful bound when compared with those of MGM-2 and MGM-3 of around $15\%$ and $20\%$ respectively. Finally, for random instances EU-DaC gives a mean of percent bound of around $55\%$ whereas those of MGM-2 and MGM-3 are around $5\%$ and $10\%$ respectively.

Therefore, from these empirical results we can conclude that: (1) EU-DaC bounds (and in general DaC bounds) are meaningful enough to provide agents with an awareness of the quality of their solution and to trade-off quality vs resources; and (2) k-optimal guarantees for k=2 and k=3 are very loose and considerably underestimates the solution provided by MGM-2 and MGM-3 on the experiments (results on solution quality show how the value of MGM-2 and MGM-3 solutions are close to those of EU-DaC but the latter gives much higher quality guarantees).

## 5. CONCLUSIONS AND FUTURE WORK

Our contribution in this paper is twofold. Our first contribution is a novel DaC algorithm, the so-called Egalitarian Utilities Divide and Coordinate (EU-DaC) algorithm. The *DaC* framework [11] is one of the few general DCOP frameworks that can provide bounds on




(b) Small world 100 variables (c) Regular grids 100 variables (d) Random 100 variables

**Figure 5: Graphs showing the percent bound qualities when varying the number of message cycles over different topologies and with mixed Ising weights $\beta = 1.6$**

DCOP solutions on incomplete algorithms. Unlike DaCSA [11], the other DaC algorithm proposed so far, in EU-DaC agents coordinate by communicating their local max-marginal utilities for the different values of their decisions, instead of only their preferred decisions. Our empirical results show how our novel algorithm outperforms DaCSA in all experimented scenarios.

Our second contribution is to provide the first empirical comparison between the bounds provided by the DaC framework and those provided by the k-optimal framework [9]. Experiments show that DaC bounds improve the accuracy of k-optimal bounds: whereas DaC algorithms get bounds between around $55\%$ and $85\%$(varying on the scenario), 2-optimal and 3-optimal bounds never go above $15\%$ and $30\%$ respectively in any scenario. Despite of these results, as argued in [9, 4], one advantage of k-optimal bounds, not shared by DaC bounds, is that allows to provide an offline trade-off quality versus time.[7] However, as shown in our experiments, 2-optimal and 3-optimal bounds may very loose and you may be wasting a lot of resources when using this criteria. Therefore, it is reasonable to argue that it may be better for agents to know at runtime the bounds on the maximum-error of their current solution instead of offline bounds that overestimates the error of their converged solution.

As future work we plan to study some unexplored aspects of the DaC framework to allow a broad applicability of this class of algorithms. Firstly, we would like to study the privacy aspects of the DaC framework. Although privacy aspects do not limit DaC algorithms to be applied to domains in which distribution has reasons of parallelism, communication costs and/or robustness (e.g sensor networks, traffic control or the configuration of power networks[10]) we still do not know its applicability to some domains, such as distributed scheduling [10], where privacy is the main issue. Secondly, we aim at designing versions of DaC algorithms that adapts to changes so that it can be applied to dynamic environments.

**Acknowledgements**. The authors would like to thank Zhengyu Yin, Manish Jain and Milind Tambe to answer our questions about k-optimality and to provide the code for MGM algorithms and graph-based k-bounds to run our experiments.

## 6. REFERENCES

[7]Assuming that problem structure is known beforehand (k-optimal bounds are reward independent)

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

[2] R. G. Cowell, S. L. Lauritzen, A. P. David, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[3] 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.

[4] H. Katagishi and J. P. Pearce. Kopt: Distributed dcop algorithm for arbitrary k-optima with monotonically increasing utility. In *Ninth DCR Workshop (CP-07)*, 2007.

[5] R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for dcop: A graphical-game-based approach. In *ISCA PDCS*, pages 432–439, 2004.

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

[7] 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.

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

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

[10] A. Petcu and B. Faltings. Distributed constraint optimization applications in power networks. *International Journal of Innovations in Energy Systems and Power*, 3(1), 2008.

[11] M. Vinyals, M. Pujol, J. A. Rodriguez-Aguilar, and J. Cerquides. Divide and Coordinate: solving DCOPs by agreement. In *AAMAS*, 2010. To appear. http://www.iiia.csic.es/publications/list/author/278.

[12] M. J. Wainwright, T. Jaakkola, and A. S. Willsky. Map estimation via agreement on (hyper)trees: Message-passing and linear programming. *CoRR*, abs/cs/0508070, 2005.

[13] 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.

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