QR: A New Reparameterization Algorithm for Quick Response DCOPs

Emma Rollon and Javier Larrosa

Universitat Politecnica de Catalunya

Abstract. In this paper we consider algorithms for DCOPs that require a quick response. In this context it is important to have any-time algorithms that give fast sub-optimal solutions and error bounds. Then the agents may consider the urgency of their reaction and the quality of their current solution and decide to wait a little longer for a likely better solution. Under this framework we propose QR, a new algorithm that combines a tree-like relaxation with reparameterization. It improves over previous tree-relaxation algorithms in that it is any-time. It improves over lagrangian-relaxation algorithms in that it gets more accurate bounds at the very first iterations.

Introduction

Distributed Constraint Optimization Problems (DCOP) capture combinatorial problems distributed among a set of agents. In typical applications [7, 16, 11, 12] solving the problem directly corresponds to deciding how each agent (or a central authority) must behave. Thus, agents must, in the first place, solve the problem in a coordinated way, and then carry out the corresponding actions. In this paper we consider problems that require a *quick response*. This type of problems occur in dynamic situations where the agents must react before the problem evolves and renders the current solution useless (e.g. rescue robotics [14]) and/or in geographically distributed problems where agents have very low computing and communication capabilities (e.g. wide-area surveillance [8] or monitoring environmental phenomena [9]). Since DCOPs are NP-hard in general, the main goal is then to design distributed algorithms that provide a quick near-optimal solution. Additionally, it is desirable that those algorithms are any-time and they provide a bound of their error. The conjunction of these two properties is very relevant because depending on the urgency of their reaction and the error rate, the agents may prefer to wait for a more accurate solution.

State-of-the-art algorithms for this type of problems usually use a *Tree Relaxation* (TR) [12, 13]. Roughly, the idea is to transform the original problem into an acyclic one which can be solved efficiently with well-known assyncronous message-passing algorithms such as MaxSum [4] or other variants of the GDL framework [1] such us Belief Propagation [10]. The optimal solution of the relaxation is taken as an approximation of the optimal solution of the original problem. These algorithms provide a bound of their error and have shown experimentally that they can provide reasonably accurate solutions (typical error rates range 10-30%). Their complexity is very low, since each agent must compute only one message and serialization is only on the diameter of the

tree. The main disadvantage, and the motivation of our work, is that TR algorithms are not anytime. Thus, they are unable to take advantage of additional time, if available.

As an alternative to TR, several authors have explored different iterative approaches. Zhang et al [20] propose the Distributed Stochastic Algorithm which is a distributed version of stochastic search. Although being distributed, the algorithm needs to iterate a large number of times (tens or hundreds of iterations in medium size problems) before achieving good solutions, which is not acceptable to the type of applications that we consider. Farinelli et al [4] explore the use of MaxSum in cyclic problems. Their experiments show that MaxSum also needs to iterate a too large number of times before convergence (sometimes it even diverges). Neither of these two algorithms can provide quality guarantees.

Algorithms based on *Lagrangian Relaxations* (LR) [15] are becoming very popular in the centralized context where they have shown impressive performance in *image processing* and *natural language* applications. Like TR algorithms, they also solve a relaxation of the problem. In this case, the relaxation decouples the original problem replicating variables. The relaxed problem can be solved efficiently using local operations (each factor is processed in isolation). Similarly to TR algorithms, the optimal of the relaxation is an upper bound of the optimal. However, algorithms based on LR have an important difference with respect to TR algorithms: relaxation is enhanced with an iterative phase in which they transform the original problem into an equivalent, but more explicit, one. The transformation is achieved as a sequence of so-called *reparameterizations*. Because reparameterization preserves the problem, solving the reparameterized relaxation also provides an upper bound of the problem's optimum and the goal of reparameterization is to tight the bound us much as possible. Interestingly, the reparameterization can also be used to compute a potentially good assignment, which is evaluated in the original problem producing a lower bound. Consequently, LR algorithms can also bound the error and provide quality guarantees.

In the centralized setting, the importance of early iterations of LR are disregarded. The reason is that in that context the urgency of response is not as dramatic. A central CPU can run hundreds or thousands of iterations fairly quickly and convergence is usually achieved before that. Apart from [17, 5], LR algorithms have received little attention in the multi-agents field. This may come to as a surprise, since they also have a natural assyncronous message passing implementation.

In this paper we show that LR can be useful in DCOP, even in problems that require a quick response. Our first contribution is to show that one does not need to wait until LR algorithms converge. On the contrary, we show that just after the very firsts iterations LR algorithms may already provide fairly accurate solutions. The second and main contribution is a new algorithm that combines reparameterization with a tree relaxation. The algorithm inherits the best of each world: it is any-time, provides error bounds and, most importantly, provides very accurate solutions in a very small number of iterations. Our experiments show that the combination outperforms both LR and TR.

Our work differs from [20, 4, 17] in that we specifically focus on quick accuracy. It is clear that our algorithm shares elements to [17], but has several advantages: it gets better bounds at the earliest iterations (due to a better reparameterization rule and a

stronger tree-based relaxation) and it has no parameters to be tuned before-hand (due to a more robust updating rule).

Preliminaries

A *Constraint Optimization Problem* (COP) is a tuple (X, D, F), where $X = \{x_1, \ldots, x_n\}$ and $D = \{d_1, \ldots, d_n\}$ are variables and their (finite) domains. The objective function to be optimized is

$$F(X) = \sum_{j=1}^{r} f_j(X^j)$$

Each f_j is called a *factor* [1] over a subset $X^s \subseteq X$ of the variables. It associates values to the possible assignments of its scope X^j. A *solution* is a complete assignment X. The goal in a COP is to find an *optimal solution*, which is a solution X^* such that

$$\forall X,\ F(X^*) \geq F(X)$$

Finding the optimal solution of a COP is in general an NP-hard problem [3]. A usual assumption is that factors in isolation can be processed efficiently including at least the computation of their local optima $\max_{X^j} f_j(X^j)$, and their marginals

$$z(x_i) = \max_{X^j - \{x_i\}} f_j(X^j)$$

The *factor graph* of a COP instance (X, D, F) is a bipartite graph with two types of nodes: variable and function nodes. There is one variable node for each variable $x_i \in X$ and one factor node for each local function f_j. There is an edge connecting variable node x_i to factor node f_j if x_i is an argument of f_j (i.e, $x_i \in X^j$).

In a *Distributed* COP (DCOP) the problem is distributed among a set of agents. Each agent *knows* some of the cost functions along with their variables and domains. In the DCOP environment, the agents must search for the optimum via decentralized coordination. Each agent can communicate with a (small) number of neighbors. In this paper we make the usual assumption of making a one-to-one mapping between cost functions and agents. Two agents are neighbors if their functions have a common variable in their scope.

The factor graph of a DCOP instance is a very convenient tool for identifying its complexity. It is well-known that when the graph is acyclic, the problem can be solved very efficiently with a number of algorithms. If the graph is cyclic, the cost of solving the instance is exponential on its *degree of cyclicity*. There are several measures of cyclicity, such as the *tree* or *induced width* which are essentially equivalent [3].

Tree-relaxation Algorithms

Tree-relaxation algorithms (TR) like BMS [12] or IBMS [13] are approximation algorithms with very low overhead. They relax the original problem transforming it into a tree, which can be solved efficiently.

[1] Factors are also called utility or cost functions.

Algorithm 1 shows a detailed algorithmic description of a generic tree relaxation algorithm. First (line 1), the algorithm computes a tree T from the original factor graph by removing some dependencies among functions and variables. Generally, it heuristically measures the impact of removing each dependency in the optimal solution and then it removes the ones with less impact. Next (line 2), the original problem P is transformed into an acyclic one (A, X, D, F^T) having the spanning tree as factor graph. In IBMS the transformation is done as follows: For each edge (i, j) in the original graph that does not belong to the tree, the cost function f_j is transformed into another function f'_j with $X'^j := X^j - \{x_i\}$ as,

$$f'_j(X'^j) := \max_{x_i}\{f_j(X^j)\}$$

Then (line 3), the algorithm solves $\max_X\{F^T(X)\}$ efficiently with one of the many available algorithms. Let X^T be the solution. Finally, the algorithm returns X^T as a sub-optimal solution for the original problem P, $F(X^T)$ is a lower bound of the true optimum, and $F^T(X^T)$ is an upper bound.

Lagrangian-relaxation Algorithms

Lagrangian-relaxation algorithms (LR) like MaxSum Diffusion (MSD) [19] or Factor Graph LP (FGLP) [6] are iterative methods. In the COP context, they work by combining two ideas: *reparameterization* and *relaxation*.

The relaxation used by these methods is extremely simple. The idea is to replicate each variable x_i once for each factor $f_j(X^j)$ where it occurs. Let X' denote the augmented set of variables and $F^L(X')$ its augmented relaxation objective function. When solving $F^L(X')$, unlike in the original problem, one can introduce the maximization inside the summation which gives,

$$\max_{X'}\{F^L(X')\} = \sum_{j=1}^{r} \max_{X^j}\{f_j(X^j)\}$$

This is assumed to be efficiently solvable by local computations. Clearly, the solution of the relaxation is an upper bound of the optimum of the original problem.

As a stand alone technique, it is obvious that this is a very weak relaxation, and LR algorithms thight the bound with a reparameterization pre-process. Reparameterization can be nicely explained in terms of the so-called *dual variables* δ. For every scope X^j, every variable $x_i \in X^j$ and every value in its domain, a variable $\delta_{ij}(x_i)$ is defined. Then, the original objective function $F(X)$ is re-phrased as

$$F(\delta, X) = \sum_{x_i}(\sum_{j \in M_i} \delta_{ij}(x_i)) + \sum_{j=1}^{r}(f_j(X^j) - \sum_{x_i \in X^j} \delta_{ij}(x_i))$$

where $M_i = \{j \mid x_i \in X^j\}$ is a vector of function indexes, indicating which functions contains x_i in their scope.

Algorithm 1: Generic Tree Relaxation

Input : A DCOP problem $P = (A, X, D, F)$.
Output: An sub-optimal solution, a lower bound, and an upper bound of P.

$T \leftarrow$ `ComputeTree(`P`)`;
$F^T \leftarrow$ `TreeRelaxation(`F, T`)`;
$(X^T, ub^T) \leftarrow$ `Solve(`$F^T(X)$`)`;
$lb^T \leftarrow F(X^T)$;
return (X^T, lb^T, ub^T);

Algorithm 2: Generic Lagrangian Relaxation

Input : A DCOP problem $P = (A, X, D, F)$, and an integer value k.
Output: A sub-optimal solution, a lower bound, and an upper bound of P.

$F(\delta, X) \leftarrow$ Rephrase F using dual variables δ;
Initialize all δ to 0;
repeat
    `Update(`δ`)`;
    $F^L(\delta, X') \leftarrow$ `LagrangianRelaxation(`$F(\delta, X)$`)`;
    $(X'^L, ub^L) \leftarrow$ `Solve(`$F^L(\delta, X')$`)`;
    for $x_i \in X$ **do** $X_i^L \leftarrow \arg\max_{x_i} \sum_{j \in M_i} \delta_{ij}$;
    ;
    $lb^L \leftarrow F(X^L)$;
    $k \leftarrow k - 1$;
until *fixed point or* $k = 0$;
return (X^L, lb^L, ub^L);

Algorithm 3: Quick Response

Input : A DCOP problem $P = (A, X, D, F)$, and an integer value k.
Output: A sub-optimal solution, a lower bound, and an upper bound of P.

$T \leftarrow$ `ComputeTree(`P`)`;
$F(\delta, X) \leftarrow$ Rephrase F using dual variables δ;
Initialize all δ to 0;
repeat
    `Update(`δ`)`;
    $F^Q \leftarrow$ `TreeRelaxation(`$F \cup \delta, T$`)`;
    $(X^Q, ub^Q) \leftarrow$ `Solve(`$F^Q(X)$`)`;
    $lb^Q \leftarrow F(X^Q)$;
    $k \leftarrow k - 1$;
until *fixed point or* $k = 0$;
return (X^Q, lb^Q, ub^Q);

The key property of $F(\delta, X)$ is that, no matter which values take the reparameterization variables, it is equivalent to the original problem (namely $F(X) = F(\delta, X)$ for all δ). Consequently,

$$\forall \delta,\ \max_X\{F(X)\} = \max_X\{F(\delta, X)\}$$

The Lagrangian function is

$$L(\delta) = \max_{X'} F^L(\delta, X')$$

and the goal is to find the reparameterization that produces the best relaxation, that is, to find δ^* such that,

$$L(\delta^*) = \min_\delta L(\delta)$$

or, equivalently,

$$L(\delta^*) = \min_\delta\{\sum_{x_i \in X} \max_{x_i}\{\sum_{j \in M_i} \delta_{ij}(x_i)\} + \sum_{j=1}^{r} \max_{X^j}\{f_j(X^j) - \sum_{x_i \in X^j} \delta_{ij}(x_i)\}\}$$

which is the dual optimization problem. The weak duality property, $L(\delta^*) \geq F(X^*)$, is direct.

LR algorithms optimize $L(\delta)$ using iterative methods usually inspired in the subgradient and coordinate descent methods [2]. Algorithm 2 shows the pseudo-code of a generic Lagrangian relaxation algorithm. Initially, the algorithm computes the Lagrangian of the original problem P (line 1), and sets dual variables to zero (line 2). Then, the algorithm iterates until convergence (or time out). At each iteration dual variables are updated (line 4) using some reparameterization rule. Under a message passing point of view, $\delta_{i,j}(x_i)$ may be interpreted as a message that factor f_j sends to variable x_i about its *belief* on the different values of x_i being in the optimal solution. After reparameterization, the Lagrangian-relaxation $F^L(\delta, X')$ is computed (line 5) and efficiently solved (line 6). The optimum of the relaxation is the current iteration upper bound ub^L. Dual variables are also used to compute the current sub-optimal assignment X^L (lines 7) as,

$$X_i^L := \arg\max_{x_i} \sum_{j \in M_i} \delta_{ij}(x_i)$$

Finally, $F(X^L)$ (line 8) is the current lower bound.

Combining Reparameterization and Tree relaxation

Now we evaluate the performance of the first iterations of the generic Lagrangian relaxation algorithm (Algorithm 2) with two different reparameterization schemas that have shown good performance in the centralized context. We do not include a pure subgradient method because it has been extensively observed that its convergence is much slower than the alternative coordinate descend updates considered here [15].

Let $\alpha_{ij}(x_i) = \max_{X^j - \{x_i\}}\{f_j(X^j) - \sum_{i' \in X^j} \delta_{i'j}(x_{i'})\}$. Then, the reparameterizations considered in this paper are:

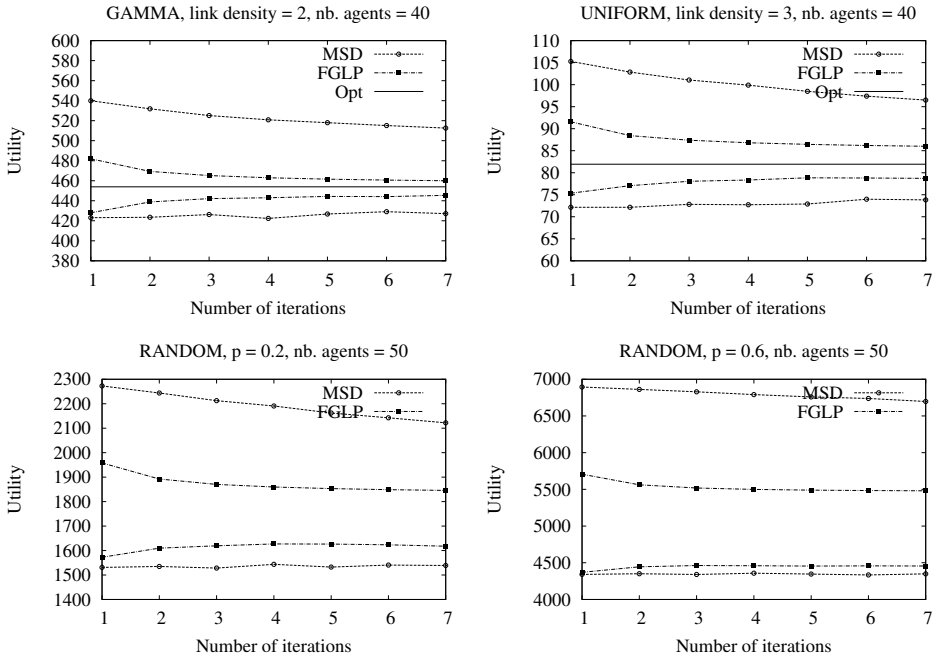

Fig. 1. Evolution of the upper and lower bound as a function of the number of iterations for the generic Lagrangian-relaxation algorithm using FGLP and MSD updates.

– Factor Graph LP (FGLP) [6], the algorithm computes for all (i, j),

$$\delta_{ij}(x_i) := \alpha_{ij}(x_i) - \frac{1}{|M_i|} \sum_{j' \in M_i} \alpha_{ij'}(x_i)$$

– Max-Sum Diffusion (MSD) [19], the algorithm computes for all j, j' such that $X^j \cap X^{j'} \neq \emptyset$,

$$\delta_{ij}(x_i) := \frac{1}{2}(\alpha_{ij'} - \alpha_{ij})$$

$$\delta_{ij'}(x_i) := \frac{1}{2}(\alpha_{ij} - \alpha_{ij'})$$

We consider the same set of problems from the ADOPT repository[2] used in [12] as well as random problems varying the number of agents in the range $\{10, .., 100\}$, each variable having 3 domain values, and two different constraint densities $p = 0.2$ and $p = 0.6$. The costs of the constraints are uniformly generated in the range $\{1, .., 10\}$. For each configuration, we generated 25 instances. In the following, for each algorithm, we report its upper and lower bound evolution. For comparison purposes, we

[2] http://teamcore.usc.edu/dcop

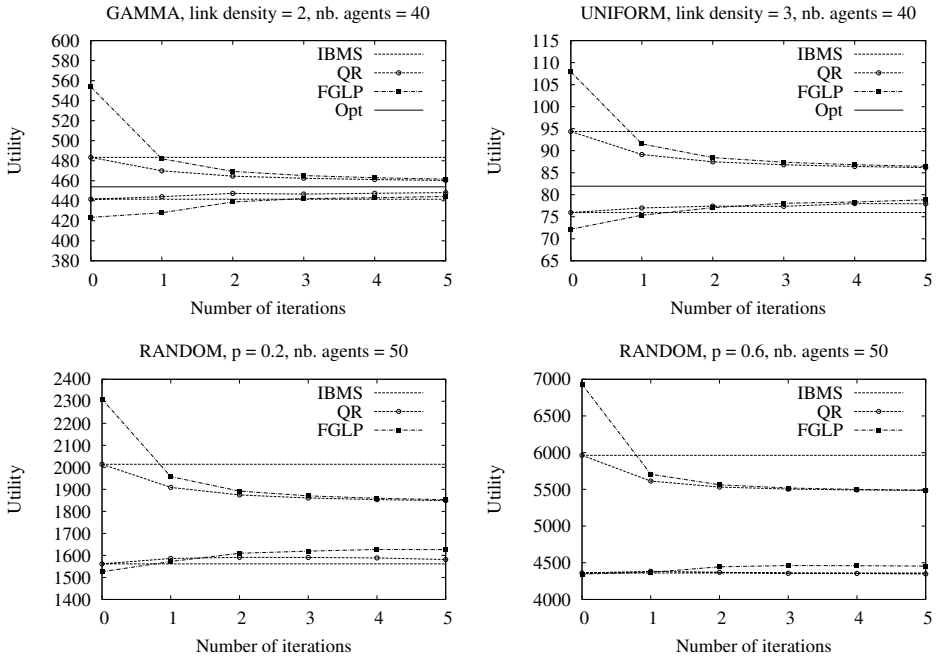

Fig. 2. Evolution of the upper and lower bound as a function of the number of iterations for the QR and generic Lagrangian relaxation algorithm using the FGLP update.

also report the true optimum (Opt) when the problem can be solved exactly with some centralized/distributed complete algorithm. We only report on the following instances: GAMMA link density 2 and UNIFORM link density 3 with 40 agents, and random with p=0.2 and p=0.6 with 50 agents. We omit results on other classes of problems, since the behaviour pattern is very similar.

Figure 1 reports results of the 7 initial iterations. Our first observation is that FGLP's update clearly outperforms MSD's from the earliest iterations and this seems to be the case systematically over all classes of problems where we performed experiments. Our second observation is that the two algorithms show a typical behaviour of iterative methods. The quality of the bounds at the first steps is poor but it improves very quickly once the reparameterization starts tighting the relaxation.

As we discussed before we are not so interested in the quality of bounds at convergence, but at the earliest iterations, where iterative methods may not be so good because the reparameterization has not yet been able to tight the relaxation. At these first iterations it seems reasonable to rely on the stronger tree-like relaxation. This is the idea of our *Quick Response* (QR) algorithm (see Algorithm 3). The algorithm iterates until time out. At each iteration one reparameterization step is computed using the updating rule of choice (line 5). Then the tree relaxation is computed with the reparameterized problem (line 6) and the tree-like problem is solved (line 7) resulting into a sub-optimal

solution X^Q and an upper bound ub^Q of the original problem. The sub-optimal solution is used to compute a lower bound $F(X^Q)$ (line 8).

Figure 2 reports the any-time behaviour of QR using FGLP's reparameterization rule. We compare it with the FGLP Lagrangian-relaxation's version discussed in the previous section. As can be seen QR and FGLP improves over time. Moreover, QR outperforms both IBMS and FGLP along the first iterations and, in particular, at the first and second ones. This means that QR on the first iteration provides quite accurate solutions within a small approximation ratio which indicates that this algorithm seems suitable for situations where a quick-response in needed.

Conclusion and Future Work

In this paper we have focused on DCOPs that require a quick response. We have discussed the importance of any-time algorithms that provide an error bound. We have introduced an algorithm that combines two elements: tree relaxation and reparameterization. Our experiments indicate that this is a promising approach.

In our current implementation we only have reparameterization rules that were designed to work well with Lagrangian relaxations and we found quite surprising that they worked well with a tree relaxation. In the future, we want to develop updating rules especialized for the tree-relaxation. Probably, analyzing tree-reparameterization algorithms [18] developed in the centralized context are a good starting point for the continuation of our work.

Acknowledgment

This work was supported by the Spanish MINECO under the grant TIN2013-45732-C4-3-P.

References

1. S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Trx. on Information Theory*, 46(2):325–343, 2000.
2. D. Bertsekas. *Dynamic Programming*. Prentice Hall, Englewook Cliffs, 1987.
3. R. Dechter. *Constraint Processing*. Morgan Kaufmann, San Francisco, 2003.
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*, pages 639–646, 2008.
5. D. Hatano and K. Hirayama. Deqed: An efficient divide-and-coordinate algorithm for dcop. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 1325–1326, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.
6. A. T. Ihler, N. Flerova, R. Dechter, and L. Otten. Join-graph based cost-shifting schemes. In Nando de Freitas and Kevin P. Murphy, editors, *UAI*, pages 397–406. AUAI Press, 2012.
7. R. Junges and A. L. C. Bazzan. Evaluating the performance of dcop algorithms in a real world, dynamic problem. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, 2008, Volume 2*, pages 599–606, 2008.

8. A. Makarenko and H. Durrant-whyte. Decentralized data fusion and control in active sensor networks. In *In Proceedings of the Seventh International Conference on Information Fusion*, 2004.
9. P. Padhy, R. K. Dash, K. Martinez, and N. R. Jennings. A utility-based sensing and communication model for a glacial sensor network. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *AAMAS*, pages 1353–1360. ACM, 2006.
10. J. Pearl. *Probabilistic Reasoning in Intelligent Systems. Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.
11. A. Rogers, A. Farinelli, and N. R. Jennings. Self-organising sensors for wide area surveillance using the max-sum algorithm. In *SOAR*, pages 84–100, 2009.
12. A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artif. Intell.*, 175(2):730–759, 2011.
13. E. Rollon and J. Larrosa. Improved bounded max-sum for distributed constraint optimization. In *CP*, pages 624–632, 2012.
14. P. E. Rybski, S. Stoeter, M. L. Gini, D. F. Hougen, and N. Papanikolopoulos. Effects of limited bandwidth communications channels on the control of multiple robots. In *IROS*, pages 369–374. IEEE, 2001.
15. D. Sontag, A. Globerson, and T. Jaakkola. Introduction to dual decomposition for inference. In Suvrit Sra, Sebastian Nowozin, and Stephen J. Wright, editors, *Optimization for Machine Learning*. MIT Press, 2011.
16. R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. Decentralised coordination of mobile sensors using the max-sum algorithm. In *IJCAI*, pages 299–304, 2009.
17. M. Vinyals, M. Pujol, J. A. Rodríguez-Aguilar, and J. Cerquides. Divide-and-coordinate: Dcops by agreement. In Wiebe van der Hoek, Gal A. Kaminka, Yves Lespérance, Michael Luck, and Sandip Sen, editors, *AAMAS*, pages 149–156. IFAAMAS, 2010.
18. M. J. Wainwright, T. Jaakkola, and A. S. Willsky. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, 51(7):2313–2335, 2005.
19. T. Werner. A linear programming approach to max-sum problem: A review. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(7):1165–1179, 2007.
20. 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.