K-optimal Algorithms for Distributed Constraint Optimization: Extending to Domains with Hard Constraints

Jonathan Pearce, Emma Bowring, Christopher Portway, Milind Tambe
Computer Science Dept.
University of Southern California
Los Angeles, CA 90089
{jppearce,bowring,tambe,portway}@usc.edu

Abstract

Distributed constraint optimization (DCOP) has proven to be a promising approach to address coordination, scheduling and task allocation in large-scale multiagent networks, in domains involving sensor networks, teams of unmanned air vehicles, or teams of software personal assistants and others. Locally optimal approaches to DCOP suggest themselves as appropriate for such large-scale multiagent networks, particularly when such networks are accompanied by lack of high-bandwidth communications among agents. K-optimal algorithms provide an important class of these locally optimal algorithms, given analytical results proving quality guarantees. Previous work on k-optimality, including its theoretical guarantees, focused exclusively on soft constraints. This paper extends the results to DCOPs with hard constraints. It focuses in particular on DCOPs where such hard constraints are resource constraints which individual agents must not violate. We provide two key results in the context of such DCOPs. First we provide reward-independent lower bounds on the quality of k-optima in the presence of hard (resource) constraints. Second, we present algorithms for k-optimality given hard resource constraints, and present detailed experimental results over DCOP graphs of 1000 agents with varying constraint density.

Introduction

In a large class of multi-agent scenarios, teams of agents must take joint actions, generated as a combination of individual actions, to achieve joint goals. Often, the locality of agents' interactions means that the utility generated by each agent's action depends only on the actions of a subset of the other agents. In this case, the outcomes of possible joint actions can be compactly represented by graphical models, in particular, as a distributed constraint optimization problem (DCOP)(Modi *et al.* 2005a; Mailler and Lesser 2004; Zhang *et al.* 2003). A DCOP can be represented in the form of a graph in which each node is an agent and each edge denotes a subset of agents whose actions, when taken together, incur costs or rewards, either to the agent team. Applications of DCOP include multi-agent plan coordination (Cox *et al.* 2005), sensor networks (Modi *et al.* 2005a), meeting scheduling (Petcu and Faltings 2005a) and RoboCup soccer (Vlassis *et al.* 2004).

Traditionally, researchers have focused on obtaining a single, globally optimal solution to DCOPs, introducing complete algorithms such as Adopt (Modi *et al.* 2005a), OptAPO (Mailler and Lesser 2004), and DPOP (Petcu and Faltings 2005b). However, because DCOP has been shown to be NP-hard(Modi *et al.* 2005a), as the scale of these domains become large, current complete algorithms can incur large computation or communication costs. For example, a large-scale network of personal assistant agents might require global optimization over hundreds of agents and thousands of variables. However, incomplete algorithms in which agents form small groups and optimize within these groups can lead to a system that scales up easily and is more robust to dynamic environments.

In previous work, *k-optimal* algorithms have emerged as a promising approach in building such incomplete algorithms(Pearce and Tambe 2007; Pearce *et al.* 2006; Pearce 2007). In k-optimal algorithms, agents optimize by forming groups of one or more agents until no group of k or fewer agents can possibly improve the solution; this type of local optimum is defined as a *k-optimum.* A major advantage of k-optimal algorithms are the theoretical guarantees available. For example, we are able to provide worst-case guarantees on the solution quality of k-optima in a DCOP. These guarantees can help determine an appropriate k-optimal algorithm, or possibly an appropriate constraint graph structure, for agents to use in situations where the cost of coordination between agents must be weighed against the quality of the solution reached. If increasing the value of k will provide a large increase in guaranteed solution quality, it may be worth the extra computation or communication required to reach a higher k-optimal solution. Another example of theoretical guarantees on k-optimality are the upper bounds on the number of k-optima that can exist in a DCOP. Finally, k-optimal algorithms have been defined, and promising experimental results have been provided.

Previous work on k-optimality, including its theoretical guarantees, focused exclusively on soft constraints. This paper extends the results to DCOPs with hard constraints. Hard constraints are constraints with infinite negative cost, and they are important and unavoidable in many domains. For example, a DCOP representing the location of unmanned air-vehicles (UAVs) must be able to represent the hard constraint that two UAVs must not occupy the same space (or they will crash). We focus in particular on an important subcase, where such hard constraints are resource constraints which individual agents must not violate. These resource constraints include for example the amount of fuel that a UAV may carry. Unfortunately, previous research k-optimality is unable to provide guarantees on solution quality in the presence of such hard constraints. Furthermore, algorithms for k-optimality have not been focused on such hard constraints, in particular the resource constraints mentioned earlier. We provide two key results in the context of such DCOPs with hard constraints. First we provide reward-independent lower bounds on the quality of k-optima in the presence of hard (resource) constraints. Second, we present algorithms for k-optimality given hard resource constraints, and present detailed experimental results over DCOP graphs of 1000 agents with varying constraint density.

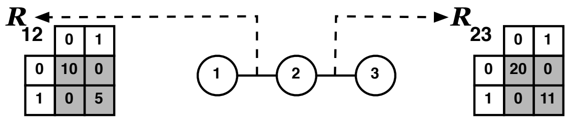
Background

This section formally introduces our notation for DCOPs and discusses the concept of k-optimality.

DCOP and k-optimality

Formally, a DCOP is a set of variables (one per agent) $N := \{1, \ldots, n\}$ and a set of domains $\mathcal{A} := \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, where the i^{th} variable takes value $a_i \in \mathcal{A}_i$. We denote the assignment of a subgroup of agents $S \subset \mathcal{I}$ by $a_S := \times_{i \in S} a_i \in \mathcal{A}_S$ where $\mathcal{A}_S := \times_{i \in S} \mathcal{A}_i$ and the assignment of the multi-agent team by $a = [a_1 \cdots a_n]$. Valued constraints exist on various minimal subsets $S \subset N$ of these variables. By minimality, we mean that the reward component R_S cannot be decomposed further through addition. If we denote by θ the set of all such subsets S on which a constraint exists, then we can express the minimality condition mathematically as follows: Mathematically: $\forall S \in \theta, R_S(a_S) \neq R_{S_1}(a_{S_1}) + R_{S_2}(a_{S_2})$ for any $R_{S_1}(\cdot) : \mathcal{A}_{S_1} \to R, R_{S_2}(\cdot) : \mathcal{A}_{S_2} \to R, S_1, S_2 \subset \mathcal{N}$ such that $S_1 \cup S_2 = S, S_1, S_2 \neq \emptyset$.

A constraint on S is expressed as a reward function $R_S(a_S)$. This function represents the reward to the team generated by the constraint on S when the agents take assignment a; costs are expressed as negative rewards. For convenience, we will refer to these subsets S as "constraints" and the functions $R_S(\cdot)$ as "constraint reward functions." The solution quality for a particular complete assignment a is the sum of the rewards for that assignment from all constraints in the DCOP: $R(a) = \sum_{S \in \theta} R_S(a) = \sum_{S \in \theta} R_S(a_S)$.



12	0	1
0	10	0
1	0	5

23	0	1
0	20	0
1	0	11

Figure 1: DCOP example

For example, Figure 1 shows a binary DCOP in which agents choose actions from the domain $\{0, 1\}$, with rewards shown for the two constraints (minimal subgroups) $S_{1,2} = \{1, 2\}$ and $S_{2,3} = \{2, 3\}$.

Before formally introducing the concept of k-optimality, we must define the following terms. For two assignments, a and $\tilde{a}$, the *deviating group* is $D(a, \tilde{a}) := \{i \in \mathcal{I} : a_i \neq \tilde{a}_i\}$, i.e., the set of agents whose actions in assignment $\tilde{a}$ differ from their actions in a. For example, in Figure 1, given an assignment [1 1 1] (agents 1, 2 and 3 all choose action 1) and an assignment [0 1 0], the deviating group D([1 1 1], [0 1 0]) = $\{1, 3\}$. We can now define *distance* as $d(a, \tilde{a}) := |D(a, \tilde{a})|$ where $|\cdot|$ denotes the cardinality of the set. The *relative reward* of an assignment a with respect to another assignment $\tilde{a}$ is

$$\Delta(a, \tilde{a}) := R(a) - R(\tilde{a}) = \sum_{S \in \theta : S \cap D(a,\tilde{a}) \neq \emptyset} [R_S(a_S) - R_S(\tilde{a}_S)] .$$

In this summation, only the rewards on constraints incident on deviating agents are considered, since the other rewards remain the same.

We classify an assignment a as a *k-optimal assignment* or *k-optimum* if $\Delta(a, \tilde{a}) \geq 0 \; \forall \tilde{a}$ such that $d(a, \tilde{a}) \leq k$. That is, a has a higher or equal reward to any assignment a distance of k or less from a. Equivalently, if the set of agents have reached a k-optimum, then no subgroup of cardinality $\leq k$ can improve the overall reward by choosing different actions; every such subgroup is acting optimally with respect to its context.

For illustration, let us go back to Figure 1. The assignment $a = [1\ 1\ 1]$ (with a total reward of 16) is 1-optimal because any single agent that deviates reduces the team reward. For example, if agent 1 changes its value from 1 to 0, the reward on $S_{1,2}$ decreases from 5 to 0. If agent 2 changes its value from 1 to 0, the rewards on $S_{1,2}$ and $S_{2,3}$ decrease from 5 to 0 and from 11 to 0, respectively. If agent 3 changes its value from 1 to 0, the reward on $S_{2,3}$ decreases from 11 to 0. However, [1 1 1] is not 2-optimal because if the group $\{2, 3\}$ deviated, making the assignment $\tilde{a} = [1\ 0\ 0]$, team reward would increase from 16 to 20. The globally optimal solution, $a^* = [0\ 0\ 0]$, with a total reward of 30, is k-optimal for all $k \in$ {1,2,3}.

Algorithms for k-optimality

We discuss two algorithms, MGM-1 and MGM-2, which attain k-optimal results for $k = 1, 2$ respectively, i.e.

they are 1-optimal and 2-optimal algorithms. Other variations of 1-optimal and 2-optimal algorithms are also discussed in previous work (Maheswaran and Başar 1998), and the key techniques we discuss would apply to these other algorithms as well.

MGM-1: A 1-optimal algorithm only considers unilateral actions by agents in a given context. The 1-optimal algorithm being built upon in this paper is the MGM-1 (Maximum Gain Message-1) Algorithm (Maheswaran *et al.* 2004a; Pearce 2007) which is a modification of DBA (Distributed Breakout Algorithm) (Yokoo and Hirayama 1996).

Variables begin in MGM-1 by taking on an initial randomly selected assignment. Then execution continues in rounds. A *round* is defined as the duration to move from one value assignment to the next. A round could involve multiple messaging phases. Every time a messaging phase occurs, it is counted as one *cycle*. During a round of MGM-1, each agent broadcasts a gain message to all its neighbors that represents the maximum change in its local utility if it is allowed to act under the current context. An agent is then allowed to act if its gain message is larger than all the gain messages it receives from its neighbors (ties can be broken through variable ordering or another method). For example, if the variables in the example in Figure 1 had initially selected the assignment {0,1,0} then x_1 would send a gain message to x_2 indicating it could switch values from 0 to 1 and achieve a gain of 5. x_3 would also send a gain message to x_2 indicating it could change its value and achieve a gain of 11. x_2 would send a proposal message to both x_1 and x_3 indicating it could switch values and achieve a gain of 30. Since x_2 has the highest gain, it will switch its value to 0 and the other two variables will remain at their current assignment. Execution continues until no further proposals are made. MGM-1 requires two cycles per round (Maheswaran *et al.* 2004a; Pearce 2007).

MGM-2: When applying a 1-optimal algorithm, the evolution of the assignments will terminate at a 1-optimum. One method to improve the solution quality is for agents to coordinate actions with their neighbors. This allows the algorithm to break out of some of the local optima. This section introduces the 2-optimal algorithm MGM-2 (Maximum Gain Message-2) (Pearce 2007).

As with MGM-1, agents initially take on a random assignment and then begin executing rounds of the MGM-2 algorithm. The first step is to decide which subset of agents are allowed to make *offers*. This is resolved by randomization, as each agent is randomly assigned to be an *offerer* or a *receiver*. Each offerer will choose a neighbor at random and send it an offer message which consists of all coordinated moves between the offerer and receiver that will yield a gain in local utility to the offerer under the current context. The offer message will contain both the suggested values for each player and the offerer's local utility gain for each value pair. For example, suppose the agents in the example in Figure 1 had currently taken on the assignment $\{x_1 \leftarrow 1, x_2 \leftarrow 1, x_3 \leftarrow 0\}$ and x_1 had been assigned to be an offerer, while x_2 and x_3 had been made receivers. Agent x_1 could send a proposal to x_2 that they change values from $\{x_1 \leftarrow 1, x_2 \leftarrow 1\}$ to $\{x_1 \leftarrow 0, x_2 \leftarrow 0\}$ for a gain of 5 from x_1's perspective. Given that x_3 is not a neighbor of any receivers, it will not receive an offer in this round.

Each receiver will then calculate the overall utility gain for each value pair in the offer message by adding the offerer's local utility gain to it's own utility change under the new context and subtracting the difference in the link between the two so it is not counted twice: $\sum_{y \in Neighbors(x_i)} Gain_{iy} + \sum_{z \in Neighbors(x_j)} Gain_{jz} - Gain_{ij}$. In the example in Figure 1, Agent x_2 would receive the offer and calculate that their combined gain from this move would be 25. If the maximum overall gain over all offered value pairs is positive, the receiver will send an *accept* message to the offerer with the appropriate value pair and both the offerer and receiver are considered to be committed. Otherwise, it sends a *reject* message to the offerer, and neither agent is committed.

Uncommitted agents choose their best local utility gain for a unilateral move and send a proposal message. Uncommitted agents follow the same procedure as in MGM-1, where they modify their value if their gain message was larger than all the gain messages they received. Committed agents send the global gain for their coordinated move. Committed agents send their partners a *confirm* message if all the gain messages they received were less than the calculated global gain for the coordinated move and send a *deconfirm* message, otherwise. A committed agent will only modify its value if it receives a go message from its partner. MGM-2 requires five cycles (value, offer, accept/reject, gain, confirm/deconfirm) per round in contrast to MGM-1's 2 cycles per round.

DCOP with Hard constraints: Multiply Constrained DCOPs

In many domains, hard constraints exist, and solutions that violate a hard constraint are not useful. For example, a schedule where two people disagree on the time of their meeting with each other may be considered useless, no matter how good the schedule is for other people. We define a hard constraint as a constraint in which at least one combination of values produces a sufficiently large cost (a negative reward many times larger than the sum of all positive rewards in the DCOP).

One major source of such hard constraints, presented in (Bowring *et al.* 2006), is the presence of resource constraints at individual agents, e.g. travel budgets, battery power. Multiply-constrained DCOP (MC-DCOP) helps address such resource constraints. In MC-DCOP, we have a reward function as before (referred to below as "f") and a resource cost (referred to below as "g").

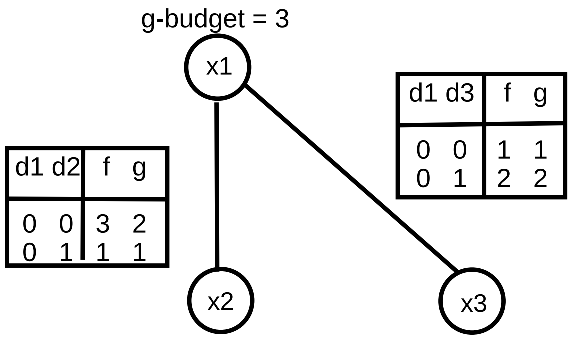


Figure 2: Multiply Constrained DCOP example

In other words, MC-DCOP adds a new cost function g_{ij} on a subset of x_i's links and a g-budget G_i which the accumulated g-cost must not exceed. Together this g-function and g-budget constitute a g-constraint. Figure 2 shows an example g-constraint on x_1. Each link has both a f-reward and a g-cost. In the example, if $x_1 = 0, x_2 = 0, x_3 = 1$, this leads to an optimal "f" reward 4, but the g-cost is 4, which violates x_1's g-budget of 3. Violation of g-budgets leads to a hard-constraint violation.

Quality Guarantees for DCOPs with Hard Constraints

Previous work(Pearce and Tambe 2007) presented quality guarantees for DCOPs in which all rewards were restricted to being non-negative. A DCOP with both costs and rewards could be normalized to one that met this restriction as long as it contained no hard constraints (constraints with an infinitely large cost). Given the importance of hard constraints it is important to address such hard constraints however.

This section shows how, in some cases, we can guarantee the solution quality of k-optimal DCOP solutions even when the DCOP contains hard constraints. To obtain these guarantees we will assume that, in the DCOPs with hard constraints that we are considering, there always exists a solution that does not violate any hard constraints (i.e. that the globally optimal solution is a feasible solution). Given this assumption, we will show how the methods for obtaining guarantees given in the previous sections can be modified to allow for the existence of hard constraints. Finally, we will assume that we know *a priori* which constraints in the DCOP graph are hard constraints (but not which combinations of values on these constraints cause them to be violated).

One complicating issue is that in some DCOPs with hard constraints, a k-optimal solution may be infeasible; that is, the k-optimal solution violates at least one hard constraint, and any deviation of k or fewer agents also results in an infeasible solution. If a k-optimum is infeasible, then it is impossible to guarantee its solution quality with respect to the global optimum. Therefore, to avoid these cases we will restrict our analysis to the following kind of DCOP: Consider a subgraph H of the DCOP constraint graph that consists all the hard constraints in the DCOP only. We will only consider DCOPs where the largest connected subgraph of H contains k or fewer agents (nodes). That is to say, we will not consider any DCOP for which a connected subgraph of H exists that contains more than k agents. In the DCOPs we are considering, no k-optimum could be infeasible. To see this, suppose an infeasible k-optimum did exist in such a DCOP. This means that at least one hard constraint in one of these such subgraphs is being violated. Since the number of agents in the subgraph is k or less, all these agents could change their values to the values that they take in the optimal solution. This change would ensure that no constraints in the subgraph were being violated, improving the overall reward. However, if k or fewer agents can improve the reward of a given assignment, it cannot be a k-optimum in the first place.

Given our k-optimal assignment a and the global optimal a^*, the key to proving this proposition is to define a set $\hat{A}^{a,k}$ that contains all assignments $\hat{a}$ where exactly k variables have deviated from their values in a, and these variables are taking the same values that they take in a^*. Note that $R(a) \geq \hat{a}, \forall \hat{a} \in \hat{A}^{a,k}$ This set $\hat{A}^{a,k}$ allows us to express the relationship between $R(a)$ and $R(a^*)$.

For any assignment $\hat{a} \in \hat{A}$, the constraints θ in the DCOP can be divided into three discrete sets, given a and $\hat{a}$:

- $\theta_1(a, \hat{a}) \subset \theta$ such that $\forall S \in \theta_1(a, \hat{a}), S \subset D(a, \hat{a})$.
- $\theta_2(a, \hat{a}) \subset \theta$ such that $\forall S \in \theta_2(a, \hat{a}), S \cap D(a, \hat{a}) = \emptyset$.
- $\theta_3(a, \hat{a}) \subset \theta$ such that $\forall S \in \theta_3(a, \hat{a}), S \notin \theta_1(a, \hat{a}) \cup \theta_2(a, \hat{a})$.

$\theta_1(a, \hat{a})$ contains the constraints that include only the variables in $\hat{a}$ which have deviated from their values in a; $\theta_2(a, \hat{a})$ contains the constraints that include only the variables in $\hat{a}$ which have not deviated from their values in a; and $\theta_3(a, \hat{a})$ contains the constraints that include at least one of each.

Thus, we can express the sum of the global rewards of all assignments $\hat{a} \in \hat{A}^{a,k}$ as:

$$\sum_{\hat{a}\in\hat{A}^{a,k}} \sum_{S\in\theta_1(a,\hat{a})} R_S(\hat{a}) + \sum_{\hat{a}\in\hat{A}^{a,k}} \sum_{S\in\theta_2(a,\hat{a})} R_S(\hat{a}) + \sum_{\hat{a}\in\hat{A}^{a,k}} \sum_{S\in\theta_3(a,\hat{a})} R_S(\hat{a}).$$

We know from our assumptions that

$$\sum_{\hat{a}\in\hat{A}^{a,k}} \sum_{S\in\theta_1(a,\hat{a})} R_S(\hat{a}) > 0$$

and

$$\sum_{\hat{a}\in\hat{A}^{a,k}} \sum_{S\in\theta_2(a,\hat{a})} R_S(\hat{a}) > 0$$

because otherwise the globally optimal solution or the k-optimal solution would contain a violated hard constraint, and that is not possible in the DCOPs we are considering. Finally we also exclude all assignments $\hat{a} \in \hat{A}^{a,k}$ where $S_{hard} \in \theta_3(a,\hat{a})$ for any hard constraint S_{hard}. Excluding all assignments $\hat{a} \in \hat{A}^{a,k}$ where $S_{hard} \in \theta_3(a,\hat{a})$ means including only the assignments $\hat{a} \in \hat{A}^{a,k}$ where $S_{hard} \in \theta_1(a,\hat{a}) \cup \theta_2(a,\hat{a})$, meaning only the assignments where all variables involved in all hard constraints are taking the same values as in the optimal solution, or where all variables involved in all hard constraints are taking the same values as in the k-optimal solution.

To see this, first consider a DCOP of six variables with domains of {0,1} on a star-shaped graph with agent 1 at the center (i.e. $\theta = \{S : 1 \in S, |S| = 2\}$). Suppose that $a = [0\ 0\ 0\ 0\ 0\ 0]$ is a 4-optimum, and that $a^* = [1\ 1\ 1\ 1\ 1\ 1]$ is the global optimum. In the case of no hard constraints, $d(a, a^*) = 6$, and $\hat{A}^{a,k}$ contains $\binom{d(a,a^*)-1}{k-1} = 10$ assignments, listed below:

[1 1 1 1 0 0], [1 1 1 0 1 0], [1 1 1 0 0 1], [1 1 0 1 1 0], [1 1 0 1 0 1],[1 1 0 0 1 1], [1 0 1 1 1 0], [1 0 1 1 0 1], [1 0 1 0 1 1], [1 0 0 1 1 1].

Now suppose that $S_{hard} = \{1, 2\}$ is a hard constraint. As mentioned above, we modify $\hat{A}^{a,k}$ to exclude all assignments $\hat{a} \in \hat{A}^{a,k}$ where $S_{hard} \in \theta_3(a,\hat{a})$. This means removing all assignments where $a_1 = 0$ and $a_2 = 1$ and all assignments where $a_1 = 1$ and $a_2 = 0$. Now, we are left with 6 assignments in $\hat{A}^{a,k}$ =

[1 1 1 1 0 0], [1 1 1 0 1 0], [1 1 1 0 0 1], [1 1 0 1 1 0], [1 1 0 1 0 1],[1 1 0 0 1 1].

Since $R(a) \geq \hat{a}, \forall \hat{a} \in \hat{A}^{a,k}$, then $6 \cdot R(a) \geq \sum_{\hat{A}^{a,k}} R(\hat{a})$. Now, $\forall S \neq S_{hard} \in \theta, \forall i \in S, a^*_i = \hat{a}_i$ for exactly $\binom{n-2}{k-2} = 3$ assignments in $\hat{A}^{a,k}$. For example, for $S = \{1, 3\}, a^*_1 = \hat{a}_1 = 1$ and $a^*_2 = \hat{a}_2 = 1$ for $\hat{a} = [1\ 1\ 1\ 1\ 0\ 0]$, $[1\ 1\ 1\ 0\ 1\ 0]$, and $[1\ 1\ 1\ 0\ 0\ 1]$. Therefore, $\forall S \in \theta, R_S(a^*) = R_S(\hat{a})$ for these $\binom{n-2}{k-2} = 3$ assignments. Since this is a star graph, $\forall S \in \theta, \forall i \in S, a_i = \hat{a}_i$ for zero assignments in $\hat{A}^{a,k}$. Thus, $6 \cdot R(a) \geq \sum_{\hat{A}^{a,k}} R(\hat{a}) \geq 3 \cdot R(a^*) + 0 \cdot R(a)$, and so $R(a) \geq \frac{3}{6-0} R(a^*) = \frac{1}{2} R(a^*)$.□

The following proposition gives a more general proof for the case of more than one hard constraint in a star-shaped graph.

Proposition 1 *For any binary DCOP of n agents with a star graph structure, where all constraint rewards are non-negative except for h hard constraints, and a^* is the globally optimal solution, then, for any k-optimal assignment, a, where $k < n$ and $0 < h < n - 1$.*

$$R(a) \geq \frac{k-h-1}{n-h-1} R(a^*). \tag{1}$$

Proof:

In a star graph, there are $\binom{n-1}{k-1}$ connected subgraphs of k variables, and so for the case of no hard constraints, $|\hat{A}^{a,k}| = \binom{n-1}{k-1}$ because each assignment $\hat{a} \in \hat{A}^{a,k}$ corresponds to one of those connected subgraphs. However, with hard constraints, we only include assignments in $\hat{A}^{a,k}$ where $S_{hard} \in \theta_1(a,\hat{a}) \cup \theta_2(a,\hat{a})$ for all hard constraints S_{hard}. In a star graph, there are $\binom{n-h-1}{k-h-1}$ connected subgraphs of k variables that include all variables involved in a hard constraint (corresponding to the assignments $\hat{a}$ where $S_{hard} \in \theta_1(a,\hat{a})$ for all hard constraints S_{hard}). Additionally, because all hard constraints must involve the central variable, there are (in the worst case, where $d(a, a^*) = n$) zero connected subgraphs of k variables that include all variables in the k-optimal solution($S_{hard} \in \theta_2(a,\hat{a})$ for all hard constraints S_{hard}). Therefore, $|\hat{A}^{a,k}| = \binom{n-h-1}{k-h-1}$. For every non-hard constraint S, there are $\binom{n-h-2}{k-h-2}$ connected subgraphs of k variables that contain S, and therefore,

$$\sum_{\hat{a}\in\hat{A}^{a,k}} \sum_{S\in\theta_1(a,\hat{a})} R_S(\hat{a}) = \binom{n-h-2}{k-h-2} R(a^*).$$

Finally, just as in the case without hard constraints, there are no ways to choose a constraint S so that it does not include any variable in a given connected subgraph of k variables. Therefore,

$$\sum_{\hat{a}\in\hat{A}^{a,k}} \sum_{S\in\theta_2(a,\hat{a})} R_S(\hat{a}) = 0 \cdot R(a).$$

and therefore Equation 1 holds. ■

When $h = n-1$ (all constraints are hard constraints), it is easy to see that only $k = n$ will guarantee optimality; otherwise, no guarantee is possible.

For other graph types, we can apply an modification to the linear fractional program (LFP) method in (Pearce and Tambe 2007). That LFP was a minimization of $\frac{R(a)}{R(a^*)}$ such that $\forall \tilde{a} \in \tilde{A}, R(a) - R(\tilde{a}) \geq 0$, given $\tilde{A}$, where $\tilde{A}$ contains any assignment $\tilde{a}$ such that $d(a, \tilde{a}) \leq k$ (note that reward $R(\tilde{a}) \leq R(a)$). With the existence of hard constraints in the DCOP that can have an infinitely large negative reward, this constraint in the LFP no longer holds for all $\tilde{a}$. Instead this constraint holds only for those $\tilde{a} \in \tilde{A}$ where $S_{hard} \in \theta_1(a,\tilde{a}) \cup \theta_2(a,\tilde{a})$ for all hard constraints S_{hard}. This occurs because any assignment $\tilde{a}$ that does not meet this condition could violate a hard constraint, resulting in a large negative value for $R(\tilde{a})$. Not including these assignments produces a smaller $\tilde{a}$ and thus a smaller set of constraints in the LFP, resulting in a lower guarantee in the presence of hard constraints. This can be implemented by, when constructing the LFP, omitting any constraint in the LFP which corresponds to

Finally, we note that, these guarantees for k-optima in standard DCOPs with hard constraints also apply to multiply-constrained DCOPs (MC-DCOP) introduced earlier. In MC-DCOP g-constraint on an agent with m

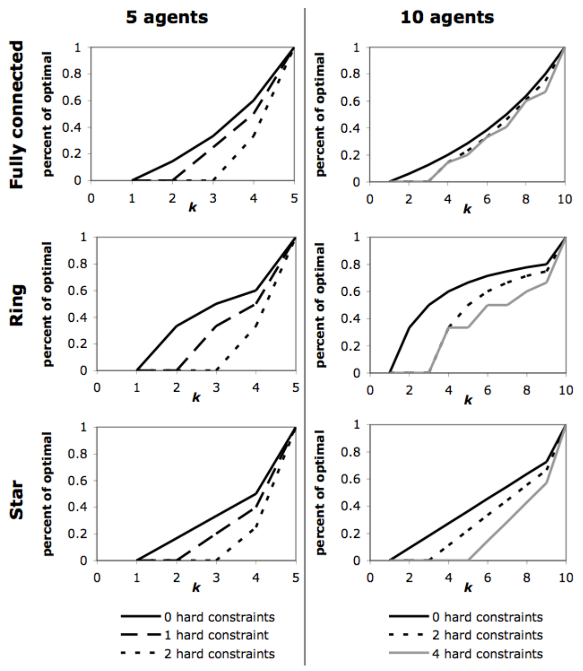


Figure 3: Quality guarantees for k-optima in DCOPs containing hard constraints.

neighbors is a shorthand expression for the more general notion of a hard $m+1$-ary constraint between the agent and all its neighbors (where any assignment where the g-budget is exceeded is considered infeasible). Thus, the methods of this section can be applied to obtain guarantees on k-optima in multiply-constrained DCOPs as well; this is shown in the experimental results later in this paper.

Figure 3 shows quality guarantees in the presence of hard constraints for fully connected graphs, ring graphs, and star graphs. These experiments began with a DCOP containing all soft constraints and no hard constraints, and gradually more and more soft constraints were made into hard constraints. The left column shows the effect of one and two hard constraints in a DCOP of five agents, and the right column shows the effect of two and four hard constraints in a DCOP of 10 agents. The constraints that were set as hard constraints were, in order, {0,1}, {2,3}, {4,5}, and {6,7}. This methodology was chosen so that no agent would be subject to more than one hard constraint, and so that k-optimal solutions would always be feasible. For star graphs, the guarantee from Proposition 1 was used; while for the others, the LFP method was used.

Figure 4 shows quality guarantees for a multiply-constrained DCOP with 30 agents, arranged in a ring structure. These experiments begin with a DCOP containing no agents with g-constraints. The number of agents with g-constraints was gradually increased by assigning a g-constraint to every third agent, starting with agent 0, until there were 10 agents with g-constraints.

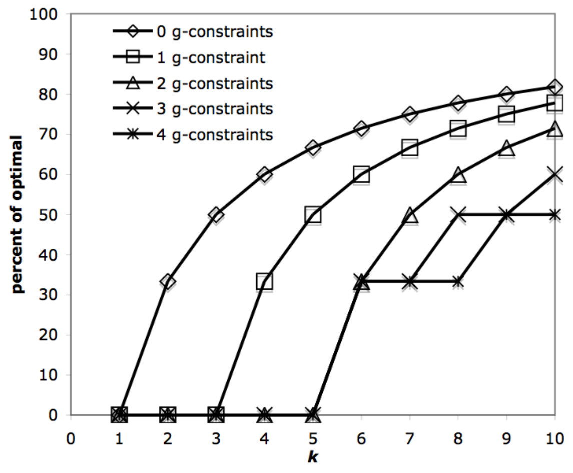


Figure 4: Quality guarantees for k-optima in a multiply-constrained ring DCOP.

Similar to the previous experiments, this methodology was chosen so that k-optimal solutions would always be feasible. The LFP method was used to calculate the guarantees. Figure 4 shows guarantees for up to 4 agents with g-constraints as k increases. For the cases of 5 to 10 agents with g-constraints the guarantee was the same as for 4 agents with g-constraints.

K-optimal Algorithms with Hard Constraints

Multiply Constrained MGM

While the MGM algorithms described earlier could be applied with standard DCOP formulations, they cannot be directly applied in the context of MC-DCOPs. In particular, there are four main challenges that must be addressed in designing locally optimal MC-DCOP algorithms. First, agents' additional resource constraints add to DCOP search complexity. Hence, agents must quickly prune unproductive search paths. Second, harnessing state-of-the-art DCOP algorithms is useful since a lot of research has already been done to maximize their efficiency (Modi *et al.* 2005b; Yokoo *et al.* 1998; Yokoo 2001; Ali *et al.* 2005; Petcu and Faltings 2005a; Maheswaran *et al.* 2004b; Pearce 2007). However, existing algorithms are not designed to handle resource constraints that are local and defined over n-ary domains. Third, algorithms must exploit constraint revelation to gain efficiency. The fourth challenge is that of detecting when the bounded optimization constraints have rendered the problem unsatisfiable. This challenge is more relevant to incomplete than complete algorithms because complete algorithms systematically consider all possible assignments and thus can easily detect unsatisfiability. However, locally optimal algorithms, particularly hill-climbing algorithms like MGM, explore only part of the search space and therefore have two options: a) require a valid initial starting point and maintain a satisfaction invariant or b) detect when the search has covered all assignments without any being found that are satisfying. This paper uses approach a), but this still leaves the challenge of detecting when it will be impossible to find a valid start point. The incomplete Multiply-Constrained DCOP algorithms presented in this paper make use of *constraint-graph transformation* and *dynamically-constraining search* to meet the first three challenges above. To address the unsatisfiability detection challenge a carefully defined dummy value is added to variables' domains so that they can easily find a valid start point and also flag a local constraint violation that remains even at termination.

MC-MGM-1

The Multiply Constrained - Maximum Gain Message - 1 (MC-MGM-1) algorithm maintains the invariant that at no point during execution does the current assignment violate any agent's g-constraint. All moves are calculated to go from one assignment where this invariant holds to another assignment where it holds. However, the tricky part is finding the first assignment where this holds. In order to address this issue as well as to provide a mechanism for determining when a problem is unsatisfiable, MC-MGM-1 performs an initialization step when it first begins where it adds a dummy value to each variable's domain. This dummy value is used as the starting value for each variable. The dummy value, d', is defined to have the following constraint function on all links:

- $f(d', d') = c$
- $g(d', d') = 0$
- $f(d', d_i) = f(d_i, d') = k$
- $g(d', d_i) = g(d_i, d') = 0$
- c and k are constants such that $c < k < 0$
- d_i is a regular domain value

This means that all variables can start at an assignment that spends 0 g, which is a satisfying solution. However, since the quality in terms of f is by definition lower than any real assignment, MC-MGM-1 will attempt to find a solution that does not involve dummy values. The reason that c must be smaller than k is that since MC-MGM-1 only allows one variable to move at a time, it must be profitable for nodes to move from a dummy value to a real value even if their neighbors are all still set to their dummy values. Otherwise the initial assignment becomes an mc-1-optimum and termination occurs immediately. If MC-MGM-1 fails to find a valid assignment containing no dummy values then the problem is 1-optimally unsatisfiable, since MC-MGM-1 will always favor real domain values over the dummy one. So the dummy values also allow for easy detection of 1-optimal unsatisfiability.

After initialization, each variable repeatedly runs rounds of the pseudo-code shown in Algorithm 1. Note that each round involves multiple cycles of communication and execution. The first thing the variables do is send value messages to their neighbors informing them of their current value as well as how much g-budget is currently available for use by that particular neighbor. The available-g sent to node x_j equals total g minus the g currently consumed by all of x_i's other neighbors (line 2 in Algorithm 1).

Algorithm 1 MC-MGM-1 (allNeighbors, currentValue)

```
1: SendValueMessages(allNeighbors, currentValue, available-g-budget)
2: currentContext = GetValueMessages(allNeighbors)
3: for newValue in EffectiveDomain(currentContext) do
4:     [gain,newValue] = BestUnilateralGain(currentContext)
5: if gain > 0 then
6:     SendGainMessage(allNeighbors,gain, newValue)
7: neighborGains = ReceiveGainMessages(allNeighbors, NeighborValues)
8: if GConstraintViolated(newContext) then
9:     n = SelectNeighborsToBlock()
10:    SendBlockMessages(n)
11: if gain > max(neighborGains) and !ReceivedBlockMessage() then
12:    currentValue = newValue
```

After receiving the value messages, each node calculates its effective domain (line 2). This means removing from consideration any values that given the current context would violate either the variable's own g-constraint or any of its neighbors' available-g's. The node then considers all of the values in the effective domain and picks the one that would allow it to gain the largest increase in local f, if selected (lines 3-4). It then sends a gain message to all of its neighbors proposing the move and listing the gain that it would achieve (lines 5-7).

Upon receipt of gain messages, two things occur. First the variable looks to see if any of its neighbors can achieve a better gain and if so rescinds its intention to move (line 11). Second, each variable checks to see whether the combined expenditure from all of its neighbors' proposed moves violates its g-constraint (line 8). An example of this situation can be seen in Figure 2, suppose x_1 has taken on the value 0 and receives two move proposals from its neighbors x_2 and x_3. The neighbors are assumed to currently have taken on the dummy value, thus each sees an available-g of 3. They make the following move proposals, neither of which individually violate the available-g: x_2 proposes taking on 0 and x_3 proposes taking on 1. However, if both moves are made, x_1's g-constraint will be violated. If a node detects this situation, then it will send a blocking message to a subset of the offending neighbors (lines 9-10). For example, x_1 may choose to send a blocking message to x_3. (The heuristics for selecting neighbors will be discussed in the next section.) If a variable receives a block message, then it will not move in the current round. The blocks are temporary and thus an agent is free to consider moving in the next round. Those agents with the highest local gain in the current round who don't receive blocking messages will move. In the example from Figure 2, x_3 will not change from the dummy value, but x_2 will go ahead and take on the value 0. The available-g will then be updated and x_3 will be informed in the next round that it can only propose moves that spend no more than 1 unit of g. These rounds repeat until all agents have ceased to propose moves.

Heuristics in MC-MGM

The number of blocking messages sent is the minimum number that will prevent x_i's g-constraint from being violated, which will be at worst one fewer than the number of x_i's neighbors proposing moves (since each individual move is legal). There are various possible heuristics for selecting which neighbor(s) to block. Heuristics can be deterministic or stochastic. Deterministic heuristics have the advantage of not ignoring local information in deciding who to block. However, local information may not indicate the globally optimal choice and so when run multiple times, stochastic heuristics have the advantage of being able to eventually find the optimal set of neighbors to block. Additionally, there are two ways to handle a blocking message: 1) x_j maintains its old value and chooses not to make its proposed move (monotonic) 2) x_j resets itself to a value that consumes less g (non-monotonic). Monotonic heuristics are guaranteed to terminate, however non-monotonic heuristics provide more options for breaking out of a local optimum. The following three different heuristics were selected as representative examples of possible heuristics.

- *Monotonic*: x_i selects one or more random neighbors and sends blocking messages. The blocking messages are interpreted by each neighbor x_j to mean that x_j should refrain from changing its value in the current round. The advantage of this heuristic is that it maintains the property of monotonicity which was a property of the original MGM algorithms. The global utility never decreases during execution which allows proof of termination to be guaranteed. However, the disadvantage is that, experimentally, it is the worst performing of the heuristics on random examples.
- *Random Reset*: x_i selects one or more random neighbors and sends blocking messages which are interpreted by each neighbor x_j to mean that it should reset its value to the dummy value. This heuristic, when run multiple times, allows MC-MGM to eventually send its blocking message(s) to the optimal neighbor(s). The disadvantage is that monotonicity can not be guaranteed and that random reset will not consider changing its own value to prevent the violation.
- *Self*: x_i sends no blocking messages but instead resets itself to the dummy value. In this case, one fewer cycle of communication is required. However, monotonicity is also not guaranteed and this is a deterministic heuristic.

MC-MGM-2

Multiply-Constrained MGM-2 operates much like MC-MGM-1 with the exception that in addition to making individual moves, it can propose joint moves between pairs of agents. The pseudo-code has been omitted due to a lack of space, but some of the basic structure is described below.

At the beginning of the round agents are stochastically designated to be either offerers or receivers, this designation helps reduce redundant computation where multiple agents propose the same move. If a node, x_i, is an offerer, it will randomly select a neighbor, x_j and search for the best joint move that does not violate x_i's g-constraint or the available-g of any of x_i's neighbors including x_j. Variable x_i will then send a proposal message to x_j. As with regular MGM-2, x_j will not accept a proposal if it is itself an offerer. However, those receivers that receive a proposal for a joint move will check to see if it violates their g-constraint of the available-g of any of their neighbors. If the proposed move is legal and has the highest local gain, then the node will send an acceptance message.

Proofs

This section will present proofs that when using the monotonic heuristic MC-MGM-1 is monotonic and terminates. Additionally, when MC-MGM-1 terminates it has found an mc-1-optimal solution. Similarly this section will show proofs that when using the monotonic heuristic MC-MGM-2 is monotonic and terminates at an mc-2-optimal solution. While many heuristics are available for use within MC-MGM, the proofs in this section will assume use of the monotonic heuristic from Section . While the other three heuristics discussed were experimentally found to always terminate, they cannot be proven to always terminate.

The following definitions are used in the proofs that follow:

- Variables are denoted as $x_i \in X$, where X denotes the set of all variables, and values of variables as $d_i \in D_i$ where D_i is the finite domain of the variable x_i.
- Let $d^{(n)}$ refer to the assignment of values to variables in the DCOP due to MC-MGM at the beginning of the n-th cycle of the algorithm. Let the global utility of this assignment be $U(d^{(n)})$.
- Let $L(d_i; d_{-i})$ refer to the local utility of agent i, given the current context denoted as d_{-i}. Then, $L(d_i; d_{-i}) = \sum_{x_j \in Neighbors(x_i)} f_{ij}(d_i, d_j)$.
- $Gain_i$ is the change in local utility of x_i due to a unilateral change in its value from d_i to d'_i, given fixed context d_{-i}, i.e. the difference between $L(d_i; d_{-i})$,

and $L(d'_i; d_{-i})$. $Gain_{ij}$ is the change in local utility of x_i and x_j due to the 2-coordinated change in values from d_i to d'_i and d_j to d'_j. This equals $L(d'_i; d_{-i}) + L(d'_j; d_{-j}) + f(d_i, d_j) - f(d'_i, d'_j)$

- In computing $Gain_i$, x_i only considers its *Effective-Domain*, i.e. values that do not violate its own g-constraint, or the available-g sent by it's g-constraint neighbors. Thus, references to Gain below, refer to gain over x_i's EffectiveDomain.
- Once an agent in MC-MGM takes on a value from its real domain, it will never go back to its dummy starting value, because the gain will be negative.
- MC-MGM maintains as an invariant that no agents' g-constraint is violated at the end of any round of execution.

Proposition 2 *When running MC-MGM-1, the global utility $U(d^{(n)})$ never decreases.*

Proof:
There are two separate cases to handle: *non-blocking* and *blocking.* In the non-blocking case, no block messages are issued. In the blocking case at least one blocking message is issued.

In the non-blocking case, if x_i is intending to modify its value in round r, then:

- $Gain_i > Gain_j, \forall x_j \in Neighbors(x_i)$
- $Gain_i > 0$

Since x_i's neighbors would have received x_i's message proposing $Gain_i$, they will not modify their values in round r. Thus, no two neighboring variables will change values simultaneously. So, when x_i changes its value, $Gain_i$ will be realized. Since x_i's gain is amassed from the sum of utilities on each link connected to x_i, x_i's gain implies that the global utility $U(d^{(r)})$ increases. If multiple variables change values simultaneously, they are guaranteed to be non-neighbors, and thus, each of their gains will add to $U(d^{(r)})$.

In the blocking case, a blocking message within a round r will cause a variable x_i which had intended to change its value to not make the change. Such a block would cause x_i to realize a gain of 0, which does not cause a decrease in $U(d^{(r)})$. The message does not affect any other variables.

■

Proposition 3 *When running MC-MGM-2, the global utility $U(d^{(n)})$ never decreases.*

Proof:
The proof is similar to Proposition 2, and has been omitted due to length.

■

Given that MC-MGM sends out blocking messages, there is a possibility of entering deadlock. A cycle of blocking messages could block all variables from changing values, even though they had not yet reached an mc-k-optimum. An example of deadlock in MC-MGM-1 is shown in Figure 5. There are four agents, with

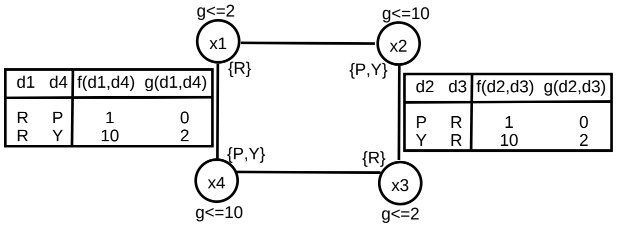


Figure 5: A Deadlock example for MC-MGM-1

domains as shown. The f reward and g cost are as shown in the table. The DCOP is initialized with all agents taking on a dummy value of 0. x_1 and x_3 switch from 0 to value R. At this point, x_2 and x_4 propose switching to Y, which gives them each a gain of 20, and Y is under the available-g of 2 (for both x_1 and x_3). Since x_1 and x_3's budgets would be violated if both x_2 and x_4 switched to the value Y, they must send blocking messages. If x_1 randomly selected x_2 to block and x_3 randomly selected x_4 neither x_2 or x_4 could change values, and this would create a deadlock situation.

Fortunately, since the agents being blocked are randomly selected (in the monotonic heuristic), remaining in deadlock indefinitely is impossible. Suppose agents can enter deadlock with probability p, where $p \in [0, 1)$. Since agents randomly select who to block each round, there is a probability of $1 - p$ of escaping deadlock in every round. After N rounds of execution, the probability of remaining in deadlock is p^N. Since execution continues until there are no longer any proposal messages being sent, N approaches ∞ and p^N approaches 0. Furthermore, once one variable is allowed to change its value, the budgets available at the remaining variables change and the old deadlock is resolved. For example in Figure 5, if x_2 is allowed to change its value to Y, then the available-g at x_1 and x_3 changes to 0, and x_4 will propose taking the value P in the next round.

Proposition 4 *Given that deadlocks are resolved using randomization, MC-MGM-1 will terminate at an mc-1-optimal solution.*

Proof:
In Proposition 2, it was shown that MC-MGM-1 will lead to a monotonically increasing global utility $U(d^{(n)})$. Since $U(d^{(n)})$ cannot be higher than the finite globally optimal solution, MC-MGM-1 cannot keep increasing $U(d^{(n)})$ forever. Thus, assuming it eventually resolves any deadlocks it enters, MC-MGM-1 will terminate.

Termination in MC-MGM-1 occurs when no variable x_i is able to propose a move from d_i to d'_i given d_{-i} where $Gain_i > 0$ and no g-constraints are violated after the move. This situation is the definition of an mc-1-optimal, so when MC-MGM-1 terminates, the agents have reached an mc-1-optimal.

■

Proposition 5 *Given that deadlocks are resolved using randomization, MC-MGM-2 will terminate at an mc-2-optimal solution.*

Proof: The proof is similar to Proposition 4, and has been omitted due to length.
■

Experimental Results

Experiments were conducted on randomly generated graph coloring graphs of 1000 nodes. It is worth noting that similar work in complete algorithms for a DCOP (Modi *et al.* 2005a; Mailler and Lesser 2004; Petcu and Faltings 2005b) are focused on problems of 20 to 100 nodes. We generated separate graphs for f-constraint link densities of 3 and 6. For each of these we formed graphs with randomly generated g-constraints along the existing f-constraints. We generated graphs with a total of 160 g-constraints.

Results for four experiments are given below as an average of 30 runs of the algorithms on graphs of the described type. On all graphs g-budgets were uniformly assigned to each agent from 0 to 50 along the x axis. There are three key aspects of solution quality which are shown along the y axis in Figures 6 - 15:

- Cycles required to reach termination of the algorithm
- Global utility of the final solution reached
- Number of agents which never left the dummy value d' due to blocking messages

Our first experiment focused on running MC-MGM-1 and MC-MGM-2 on 1000 node graphs, with graph link density of 3 and 6, where subgraphs of g-constraints had only two agents (see our section on quality results for the definition and importance of these subgraphs). Figure 7 shows run-time in message cycles on a log-scale along the y-axis. For example, we see that for graphs of density 6, when g-budget is 10, the MC-MGM-2 algorithm "peaks" at about 500 cycles. Key observations from these results are as follows. By comparing MC-MGM-1 and MC-MGM-2 results for graphs of density 3, we see that MC-MGM-1 runs between 2.3 and 9.5 times faster than MC-MGM-2. The difference in run-time for density 6 was 11.6 to 17.1 fold. Global utility varied by less than 2% between the algorithms for both densities, as shown in Figure 6. The number of agents reamining at d' was identical across the g-bugdet range, resulting in only 2 visible lines in Figure 8. At g-budgets of 0, all agents could not satify their budgets, resulting in 320 agents remaining at d' regardless of density.

The second experiment focused on subgraphs of no greater than 3 agents. The differences between MC-MGM-1 and 2 are more pronounced than with subgraphs of only 2 agents. We again found MC-MGM-1 is significantly faster than MC-MGM-2, as seen in Figure 10. For example, the difference in run-time peaks at a g-budget of 10, where MC-MGM-2 takes 132.7 times as long to terminate as MC-MGM-1. Figure 9 and Figure 11 show the global utility and number of agents

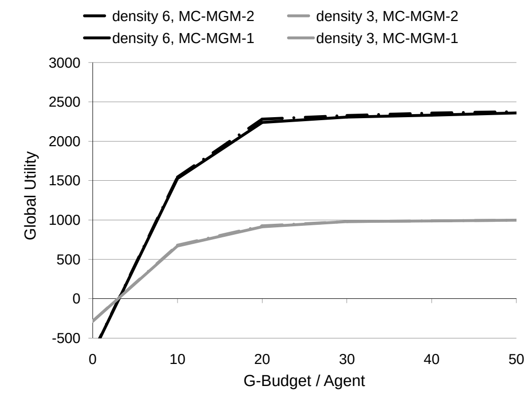


Figure 6: Global utility of MC-MGM-1 and 2 for subgraphs of g-resource constraints of 2 agents

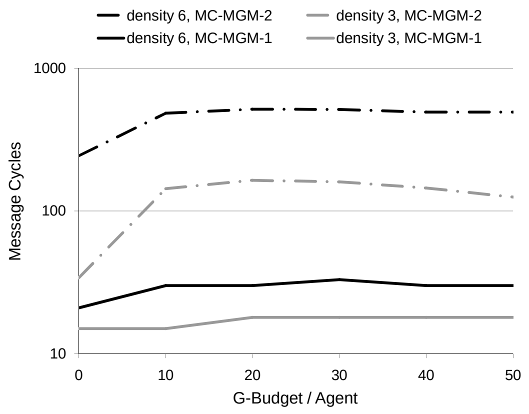


Figure 7: Cycles to termination for MC-MGM-1 and 2 for subgraphs of g-resource constraints of 2 agents

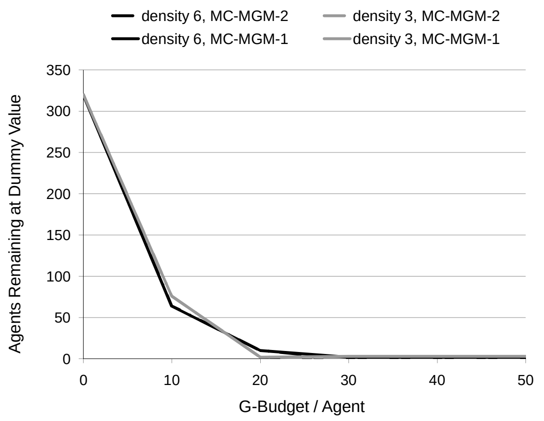


Figure 8: Number of Agents which never escaped d' using MC-MGM-1 and 2 for subgraphs of g-resource constraints of 2 agents

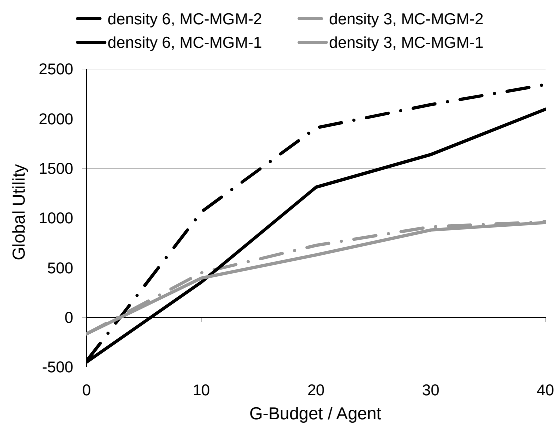


Figure 9: Global utility of MC-MGM-1 and 2 for subgraphs of g-resource constraints of 3 agents or less

Figure 10: Cycles to termination for MC-MGM-1 and 2 for subgraphs of g-resource constraints of 3 agents or less

reamining at d', respectively. MC-MGM-2 resulted in solutions on average 49% and 6% higher in global utility for densities 3 and 6, respectively. At it's peak, MC-MGM-2's solution had 3 times the global utility, when the link density was 6.

We continued to increase the subgraph size in our third experiment to cover both subgraphs of 4 and 5 agents or less. Results in Figure 12 show a continued trend of MC-MGM-2 solutions dominating MC-MGM-2 in terms of utility. Results in this set resembled those for subgraphs of 3 or less agents.

For our fourth experiment, we examined two additional blocking techniques in comparison with the monotonic technique used in all previous experiments. Changes in the blocking heuristic showed random reset neighbor performed poorly, as seen in Figure 14. For both densities, it reached the arbitrary cut-off point in our experiment of 6000 cycles. The monotonic and self reset heuristics were within similar cycle ranges to each other, peaking for density 6 at 3184 and 3021, respectively. Monotonic worked faster at the higher g-budgets, and random reset was faster when the g-budget was lower. In addition, the quality of the solution as shown in Figure 13 corresponds favorably: higher global utility for monotonic at higher, looser g-budgets and higher utility for self reset at lower, tighter g-budgets. The measure of solution quality with regard to unsatisfied g-budgets shown in Figure 13 suggests self reset dominates both other algorithms, resulting in consistantly less agents who are never able to leave the invalid start state, d'. While this is promising, this is a deterministic heuristic, and not guaranteed to terminate.

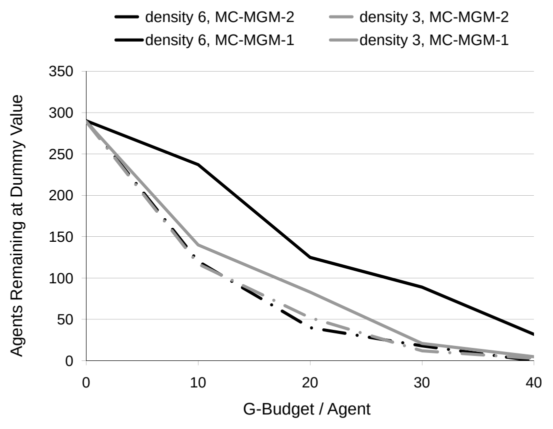


Figure 11: Number of Agents which never escaped d' using MC-MGM-1 and 2 for subgraphs of g-resource constraints of 3 agents or less

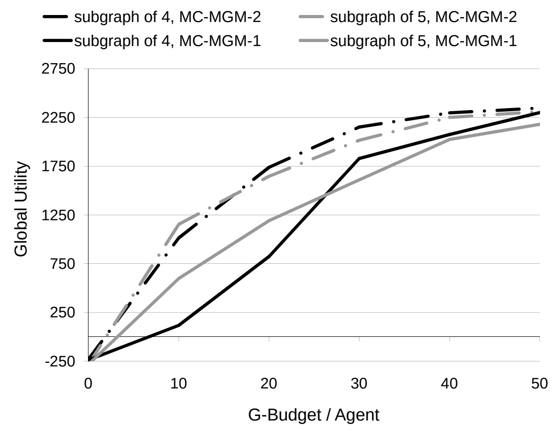


Figure 12: Global utility of MC-MGM-1 and 2 for subgraphs of g-resource constraint of no more than 4 or 5 agents, density of 6

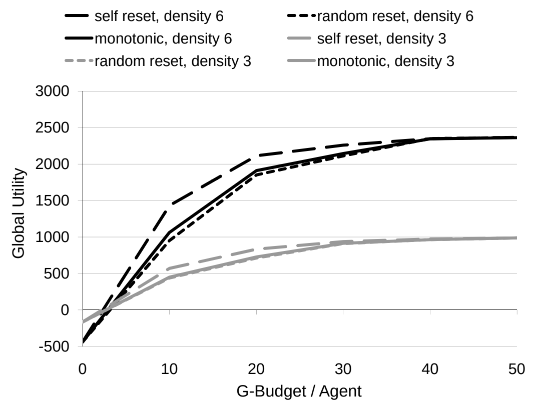


Figure 13: Global utility of the 3 heuristics, using MC-MGM-2 on subgraphs of 3 agents or less

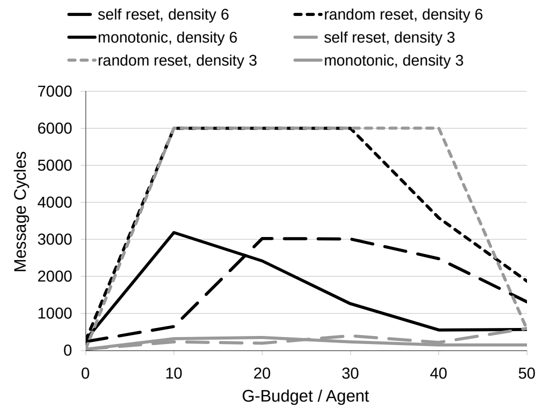


Figure 14: Cycles to termination of the 3 heuristics, using MC-MGM-2 on subgraphs of 3 agents or less

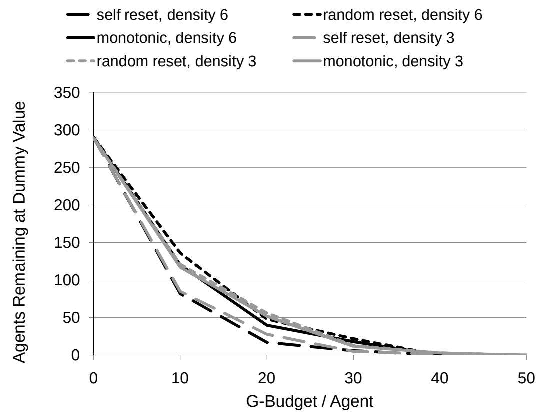


Figure 15: Number of Agents which never escaped d' for the 3 heuristics on subgraphs of 3 agents or less

Conclusion

Distributed constraint optimization (DCOP) has proven to be a promising approach to address coordination, scheduling and task allocation in large-scale multiagent networks, in domains involving sensor networks, teams of unmanned air vehicles, or teams of software personal assistants and others. K-optimal algorithms provide an important class of locally optimal algorithms for such DCOP domains, given analytical results proving quality guarantees. Previous work on k-optimality, including its theoretical guarantees, focused exclusively on soft constraints. This paper extends the results to DCOPs with hard constraints. It focuses in particular on DCOPs where such hard constraints are resource constraints which individual agents must not violate. Such hard constraints are important in many domains, and thus extending k-optimal algorithms to address such domains is a critical open research issue.

We provide two key results in the context of such DCOPs. First we provide reward-independent lower bounds on the quality of k-optima in the presence of hard (resource) constraints. Second, we present algorithms for k-optimality given hard resource constraints, and present detailed experimental results over DCOP graphs of 1000 agents with varying constraint density.

References

S. Ali, S. Koenig, and M. Tambe. Preprocessing techniques for accelerating the dcop algorithm adopt. In *AAMAS*, 2005.

E. Bowring, M. Tambe, and M. Yokoo. Multiply-constrained dcop for distributed planning and scheduling. In *AAMAS*, 2006.

J. Cox, E. Durfee, and T. Bartold. A distributed framework for solving the multiagent plan coordination problem. In *AAMAS*, 2005.

Rajiv T. Maheswaran and T. Başar. Multi-user flow control as a nash game: Performance of various algorithms. In *Proc. CDC*, Tampa, FL, December 1998.

R. Maheswaran, J. Pearce, and M. Tambe. Distributed algorithms for dcops: A graphical game based approach. In *Proceedings of the International Conference on Parallel and Distributed Computing (PDCS)*, 2004.

R. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking DCOP to the real world : Efficient complete solutions for distributed event scheduling. In *AAMAS*, 2004.

R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, 2004.

P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005.

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

J. Pearce and M. Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization. In *Proceedings of the International Joint Conference on AI (IJCAI)*, 2007.

J. Pearce, M. Tambe, and R. Maheswaran. Solution sets for dcops and graphical games. In *Proceedings of the International Joint Conference on Agents and Multiagent Systems (AAMAS)*, 2006.

J. Pearce. *Local Optimization in Cooperative Agent Networks*. PhD dissertation, University of Southern California, Computer Science Department, 2007.

A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, Edinburgh, Scotland, Aug 2005.

A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, 2005.

N. Vlassis, R. Elhorst, and J. R. Kok. Anytime algorithms for multiagent decision making using coordination graphs. In *Proc. Intl. Conf. on Systems, Man and Cybernetics*, 2004.

M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction and optimization problems. In *ICMAS*, 1996.

M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10:673–685, 1998.

Makoto Yokoo. *Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems*. Springer, 2001.

W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In *AAMAS*, 2003.