

# Solving Multiagent Networks using Distributed Constraint Optimization

**Jonathan P. Pearce, Milind Tambe**

Computer Science Dept.  
University of Southern California  
Los Angeles, CA 90089  
{jppearce,tambe}@usc.edu

**Rajiv Maheswaran**

Information Sciences Institute  
University of Southern California  
Marina del Rey, CA 90292  
maheswar@isi.edu

## Abstract

In many cooperative multiagent domains, the effect of local interactions between agents can be compactly represented as a network structure. Given that agents are spread across such a network, agents directly interact only with a small group of neighbors. A distributed constraint optimization problem (DCOP) is a useful framework to reason about such networks of agents. Given agents' inability to communicate and collaborate in large groups in such networks, we focus on an approach called *k-optimality* for solving DCOPs. In this approach, agents form groups of one or more agents until no group of  $k$  or fewer agents can possibly improve the DCOP solution; we define this type of local optimum, and any algorithm guaranteed to reach such a local optimum, as *k-optimal*. The article provides an overview of three key results related to *k-optimality*. The first set of results are 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. The second set of results are upper bounds on the number of *k*-optima that can exist in a DCOP. These results are useful in domains where a DCOP must generate a set of solutions rather than single solution. Finally, we sketch algorithms for *k-optimality* and provide some experimental results for 1-, 2- and 3-optimal algorithms for several types of DCOPs.

## Introduction

In many multi-agent domains, including sensor networks, teams of unmanned air vehicles, or teams of personal assistant agents, a set of agents chooses a joint action as a combination of individual actions. 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, such as a distributed constraint optimization problem (DCOP) (Modi *et al.* 2005; Mailler & Lesser 2004) for cooperative domains or by a graphical game (Kearns, Littman, & Singh 2001; Vickrey & Koller 2002) for noncooperative domains. Each of these models can take the form of a graph in which each

node is an agent and each (hyper)edge denotes a subset of locally interacting agents. In particular, associated with each such hyperedge is a reward matrix that indicates the costs or rewards incurred due to the joint action of the subset of agents involved, either to the agent team (in DCOPs) or to individual agents (in graphical games).<sup>1</sup> Local interaction is a key property captured in such graphs; not all agents interact with all other agents. This article focuses on the team setting, using DCOP, whose applications include multi-agent plan coordination (Cox, Durfee, & Bartold 2005), sensor networks (Zhang *et al.* 2003), meeting scheduling (Petcu & Faltings 2005) and RoboCup soccer (Vlassis, Elhorst, & Kok 2004).

Traditionally, researchers have focused on obtaining a single, globally optimal solution to DCOPs, introducing complete algorithms such as Adopt (Modi *et al.* 2005), OptAPO (Mailler & Lesser 2004), and DPOP (Petcu & Faltings 2005). However, because DCOP is NP-hard (Modi *et al.* 2005), 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 existing incomplete algorithms, such as DSA (Fitzpatrick & Meertens 2003) and DBA (Yokoo & Hirayama 1996; Zhang *et al.* 2003), agents are bounded in their ability to aggregate information about one another's constraints; in these algorithms, each individual agent optimizes based on its individual constraints, given the actions of all its neighbors, until a local optimum is reached where no single agent can improve the overall solution. Unfortunately, no guarantees on solution quality currently exist for these types of local optima.

This article presents an overview of a generalization of such incomplete algorithms via an approach called *k-optimality*. The key idea is that agents within small DCOP subgraphs optimize such that no group of  $k$  or fewer agents

<sup>1</sup>Here, costs refer to negative real numbers and rewards refer to positive real numbers to reflect intuition; we could use either costs or rewards exclusively if we assume they can span all real numbers.

can possibly improve the solution; we define this type of local optimum as a *k*-optimum. (Note that we do not require completely connected DCOP subgraphs for *k*-optimality.) According to this definition of *k*-optimality, for a DCOP with *n* agents, DSA and DBA are 1-optimal, while all complete algorithms are *n*-optimal. Here, we focus on *k*-optima and *k*-optimal algorithms for  $1 < k < n$ . The *k*-optimality concept provides an algorithm-independent classification for local optima in a DCOP that allows for quality guarantees.

In addition to the introduction of *k*-optimality itself, the article presents an overview of three sets of results about *k*-optimality. The first set of results are 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.

As an example of the use of these worst-case bounds, consider a team of mobile sensors that must quickly choose a joint action in order to observe some transitory phenomenon. This problem can be represented as a DCOP graph, where constraints exist between nearby sensors, and there are no constraints between far-away sensors. In particular, the combination of individual actions by nearby sensors may generate costs or rewards to the team, and the overall utility of the joint action is determined by the sum of these costs and rewards. Given such a large-sized DCOP graph, and time-limits to solve it, an incomplete, *k*-optimal algorithm, rather than a complete algorithm, must be used to find a solution. However, what is the right level of “*k*” in this *k*-optimal algorithm? Answering this question is further complicated because the actual DCOP rewards may not be known until the sensors are deployed. In this case, worst-case quality guarantees for *k*-optimal solutions for a given *k*, that are independent of the actual costs and rewards in the DCOP, are useful to help decide which *k*-optimal algorithm to use. Alternatively, the guarantees can help to choose between different sensor formations, i.e., different constraint graphs.

The second set of results are upper bounds on the number of *k*-optima that can exist in a DCOP. These results are valuable in domains where we need the DCOP to generate a set of *k*-optimal assignments, i.e., multiple assignments to the same DCOP. Generating sets of assignments is useful in domains such as disaster rescue (to provide multiple rescue options to a human commander) (Schurr *et al.* 2005) or patrolling (to execute multiple patrols in the same area) (Ruan *et al.* 2005) and others. Given that we need a set of assignments, the upper bounds are useful given two key features of the domains of interest. First, each *k*-optimum in the set consumes some resources that must be allocated in advance. Such resource consumption arises because: (i) a team actually executes each *k*-optimum in the set, or (ii) the *k*-optimal set is presented to a human user (or another agent) as a list of options to choose from, requiring time. In

each case, resources are consumed based on the *k*-optimal set size. Second, while the existence of the constraints between agents is known *a priori*, the actual rewards and costs on the constraints depend on conditions that are not known until runtime, and so resources must be allocated before the rewards and costs are known and before the agents generate the *k*-optimal set.

To understand the utility of these upper bounds, consider another domain involving a team of disaster rescue agents that must generate a set of *k*-optimal joint actions. This set is to be presented as a set of diverse options to a human commander, so the commander can choose one joint action for the team to actually execute. Constraints exist between agents whose actions must be coordinated (i.e., members of subteams) but their costs and rewards depend on conditions on the ground that are unknown until the time when the agents must be deployed. Here, the resource is the time available to the commander to make the decision, and examination of each option consumes this resource. Thus, presenting too many options will cause the commander to run out of time before considering them all, but presenting too few may cause high-quality options to be omitted. Knowing the maximal number of *k*-optimal joint actions that could exist for a given DCOP allows us to choose the right level of *k* given the amount of time available or allocate sufficient resources (time) for a given level of *k*.

The third result is a set of 2- and 3-optimal algorithms and an experimental analysis of the performance of 1-, 2- and 3-optimal algorithms on several types of DCOPs. Although we now have theoretical lower bounds on solution quality of *k*-optima, experimental results are useful to understand average-case performance on common DCOP problems.

## DCOP

A Distributed Constraint Optimization Problem (DCOP) consists of a set of variables, each assigned to an agent which must assign a value to the variable; these values correspond to individual actions that can be taken by the agents. Constraints exist between subsets of these variables that determine costs and rewards to the agent team based on the combinations of values chosen by their respective agents. Because in this article we assume each agent controls a single variable, we will use the terms “agent” and “variable” interchangeably.

Formally, a DCOP is a set of variables (one per agent)  $\mathcal{N} := \{1, \dots, n\}$  and a set of domains  $\mathcal{A} := \{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ , where the  $i^{th}$  variable takes value  $a_i \in \mathcal{A}_i$ . We denote the joint action (or assignment) of a subgroup of agents  $S \subset N$  by  $a_S$  and the joint action of the multi-agent team by  $a = [a_1 \dots a_n]$ .

Valued constraints exist on various minimal subsets  $S \subset N$  of these variables. 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  $a_S$ . By minimality of  $S$ , we mean that the reward obtained by that subset of agents,  $R_S$ , cannot be decomposed further through addition. We will refer to these subsets  $S$  as “constraints” and the functions  $R_S(\cdot)$  as “constraint reward functions.” The cardinality of  $S$  is also

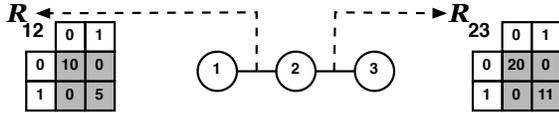


Figure 1: DCOP example

referred to as the arity of the constraint. Thus, for example, if the maximum cardinality is of  $S$  is two, the DCOP is a binary DCOP. The solution quality for a particular complete assignment  $a$ ,  $R(a)$ , is the sum of the rewards for that assignment from all constraints (captured in the set denoted by  $\theta$ ) in the DCOP.

**Example 1** Figure 1 shows a binary DCOP in which agents choose actions from the domain  $\{0, 1\}$ , with rewards shown for the two constraints  $S_{1,2} = \{1, 2\}$  and  $S_{2,3} = \{2, 3\}$ . Thus for example, if agent 2 chooses action 0, and agent 3 chooses action 0, the team gets a reward of 20.

As discussed earlier, several algorithms exist for solving DCOPs: complete algorithms, which are guaranteed to reach a globally optimal solution, and incomplete algorithms, which reach a local optimum, and do not provide guarantees on solution quality. These algorithms differ in the numbers and sizes of messages that are communicated among agents. In this article, we provide  $k$ -optimality as an algorithm-independent classification of local optima, and show how their solution quality can be guaranteed.

### $k$ -Optimality

Before formally introducing the concept of  $k$ -optimality, we must define the following terms. For two assignments,  $a$  and  $\tilde{a}$ , the *deviating group*,  $D(a, \tilde{a})$  is 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  $D([1\ 1\ 1], [0\ 1\ 0]) = \{1, 3\}$ . The *distance* between two assignments,  $d(a, \tilde{a})$  is the cardinality of the deviating group, i.e., the number of agents with different actions. The *relative reward* of an assignment  $a$  with respect to another assignment  $\tilde{a}$  is  $\Delta(a, \tilde{a}) := R(a) - R(\tilde{a})$ . We classify an assignment  $a$  as a  *$k$ -optimal assignment* or  *$k$ -optimum* if

$$\Delta(a, \tilde{a}) \geq 0 \quad \forall \tilde{a} \quad \text{such that} \quad d(a, \tilde{a}) \leq k.$$

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  $k$  or less can improve the overall reward by choosing different actions; every such subgroup is acting optimally with respect to its context. Let  $A_q(n, k)$  be the set of all  $k$ -optima for a team of  $n$  agents with domains of cardinality  $q$ . It is straightforward to show  $A_q(n, k+1) \subseteq A_q(n, k)$ : all  $k+1$  optimal solutions are also  $k$ -optimal.

If no ties occur between the rewards of DCOP assignments that are a distance of  $k$  or less apart, then a collection of  $k$ -optima must be mutually separated by a distance of at

least  $k+1$  as they each have the highest reward within a radius of  $k$ . Thus, if no ties occur, higher  $k$  implies that the  $k$ -optima are farther apart, and each  $k$ -optima has a higher reward than a larger proportion of assignments. (This assumption of no ties will be exploited only in our results related to upper bounds in this article.)

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 action from 1 to 0, the reward on  $S_{1,2}$  decreases from 5 to 0 (and hence the team reward from 16 to 11). If agent 2 changes its action 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 action 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  $k \in \{1, 2, 3\}$ .

### Properties of $k$ -Optimal DCOP Solutions

We now show, in an experiment, the advantages of  $k$ -optimal assignment sets as capturing both diversity and high reward compared with assignment sets chosen by other metrics. Diversity is important in domains where many  $k$ -optimal assignments are presented as choices to a human or the agents must execute multiple assignments from their set of assignments. In either case, it is important that all these assignments are not essentially the same with very minor discrepancies either to provide a useful choice or to ensure that the agents actions cover different parts of the solution space.

This illustrative experiment is based on a patrolling domain. In particular, domains requiring repeated patrols in an area by a team of UAVs (unmanned air vehicles), UGVs (unmanned ground vehicles), or robots, for peacekeeping or law enforcement after a disaster, provide one key illustration of the utility of  $k$ -optimality. Thus, given a team of patrol robots in charge of executing multiple joint patrols in an area as in (Ruan *et al.* 2005), each robot may be assigned a region within the area. Each robot is controlled by a single agent, and hence, for one joint patrol, each agent must choose one of several possible routes to patrol within its region. A joint patrol is an assignment, where each agent's action is the route it has chosen to patrol, and rewards and costs arise from the combination of routes patrolled by agents in adjacent or overlapping regions. For example, if two nearby agents choose routes that largely overlap on a low-activity street, the constraint between those agents would incur a cost, while routes that overlap on a high-activity street would generate a reward. Agents in distant regions would not share a constraint.

Given such a patrolling domain, the lower half of Figure 2(a) shows a DCOP graph representing a team of 10 patrol robots, each of whom must choose one of two routes to patrol in its region. The nodes are agents and the edges represent binary constraints between agents assigned to overlapping regions. The actions (i.e., the chosen routes) of these agents combine to produce a cost or reward to the team. For

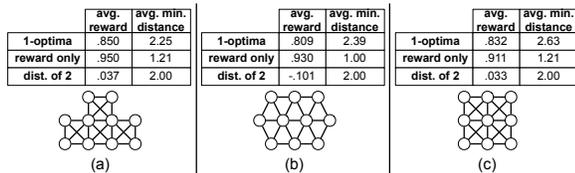


Figure 2: 1-optima vs. assignment sets chosen using other metrics

each of 20 runs, the edges were initialized with rewards from a uniform random distribution. The set of all 1-optima was enumerated. Then, for the same DCOP, we used two other metrics to produce equal-sized sets of assignments. For one metric, the assignments with highest reward were included in the set, and for the next metric, assignments were included in the set by the following method, which selects assignments purely based on diversity (expressed as distance). We repeatedly cycled through all possible assignments in lexicographic order, and included an assignment in the set if the distance between it and every assignment already in the set was not less than a specified distance; in this case 2. The average reward and the diversity (expressed as the minimum distance between any pair of assignments in the set) for the sets chosen using each of the three metrics over all 20 runs is shown in the upper half of Figure 2(a). While the sets of 1-optima come close to the reward level of the sets chosen purely according to reward, they are clearly more diverse (t-tests significance within .0001%). If a minimum distance of 2 is required in order to guarantee diversity, then using reward alone as a metric is insufficient; in fact the assignment sets generated using that metric had an average minimum distance of 1.21, compared with 2.25 for 1-optimal assignment sets (which guarantee a minimum distance of  $k + 1 = 2$ ). The 1-optimal assignment set also provides significantly higher average reward than the sets chosen to maintain a given minimum distance, which had an average reward of 0.037 (t-test significance within .0001%). Similar results with equal significance were observed for the 10-agent graph in Figure 2(b) and the nine-agent graph in Figure 2(c). Note also that this experiment used  $k = 1$ , the lowest possible  $k$ . Increasing  $k$  would, by definition, increase the diversity of the  $k$ -optimal assignment set as well as the neighborhood size for which each assignment is optimal.

In addition to categorizing local optima in a DCOP,  $k$ -optimality provides a natural classification for DCOP algorithms. Many known algorithms are guaranteed to converge to  $k$ -optima for  $k = 1$ , including DBA (Zhang *et al.* 2003), DSA (Fitzpatrick & Meertens 2003), and coordinate ascent (Vlassis, Elhorst, & Kok 2004). Complete algorithms such as (Modi *et al.* 2005), OptAPO (Mailler & Lesser 2004) and DPOP (Petcu & Faltings 2005) are  $k$ -optimal for  $k = n$ .

### Lower Bounds on Solution Quality

In this section we introduce the first known guaranteed lower bounds on the solution quality of  $k$ -optimal DCOP assignments. 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. For example, consider a team of mobile sensors mentioned earlier that must quickly choose a joint action in order to observe some transitory phenomenon. While we wish to use a DCOP formalism to solve this problem, and utilize a  $k$ -optimal algorithm, exactly what level of  $k$  to use remains unclear. In attempting to address this problem, we also assume the actual costs and rewards on the DCOP are not known *a priori* (otherwise the DCOP could be solved centrally ahead of time; or all  $k$ -optima could be found by brute force, with the lowest-quality  $k$ -optimum providing an exact guarantee for a particular problem instance).

In cases such as these, worst-case quality guarantees for  $k$ -optimal solutions for a given  $k$ , that are independent of the actual costs and rewards in the DCOP, are useful to decide which algorithm (level of  $k$ ) to use. Alternatively, these guarantees can help to choose between different constraint graphs, e.g. different sensor network formations. To this end, this section provides reward-independent guarantees on solution quality for any  $k$ -optimal DCOP assignment. We provide a guarantee for a  $k$ -optimal solution as a fraction of the reward of the optimal solution, assuming that all rewards in the DCOP are non-negative (the reward structure of any DCOP can be transformed to meet this condition if no rewards of negative infinite exist).

**Proposition 1** *For any DCOP of  $n$  agents, with maximum constraint arity of  $m$ , where all constraint rewards are non-negative, and where  $a^*$  is the globally optimal solution, then, for any  $k$ -optimal assignment,  $a$ , where  $R(a) < R(a^*)$  and  $m \leq k < n$ , we have  $R(a) \geq \frac{\binom{n-m}{k-m}}{\binom{n}{k} - \binom{n-m}{k}} R(a^*)$ .*

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  $a^*$ . This set allows us to express the relationship between  $R(a)$  and  $R(a^*)$ . While a detailed proof of this proposition appears in (Pearce & Tambe 2007), we provide an example providing an intuitive sense for this result.

Consider a DCOP with five variables numbered 1 to 5, with domains of  $\{0,1\}$ . Suppose that this DCOP is a fully connected binary DCOP with constraints between every pair of variables (i.e.,  $|S| = 2$  for all  $R_S$ ). Suppose that  $a = [0\ 0\ 0\ 0\ 0]$  is a 3-optimum, and that  $a^* = [1\ 1\ 1\ 1\ 1]$  is the global optimum. Then  $d(a, a^*) = 5$ . We now consider the set  $\hat{A}^{a,k}$  discussed above ( $k = 3$ , in this example). We can show that  $\hat{A}^{a,k}$  contains  $\binom{d(a,a^*)}{k} = 10$  assignments, listed below:

$$[1\ 1\ 1\ 0\ 0], [1\ 1\ 0\ 1\ 0], [1\ 1\ 0\ 0\ 1], [1\ 0\ 1\ 1\ 0], [1\ 0\ 1\ 0\ 1], \\ [1\ 0\ 0\ 1\ 1], [0\ 1\ 1\ 1\ 0], [0\ 1\ 1\ 0\ 1], [0\ 1\ 0\ 1\ 1], [0\ 0\ 1\ 1\ 1].$$

These are all the assignments that deviate from  $a$  by 3 actions and take the value from the optimal solution in

those deviations. We can now use this set  $\hat{A}^{a,k}$  to express  $R(a)$  in terms of  $R(a^*)$ . We begin by noting that  $R(a) \geq R(\hat{a}), \forall \hat{a} \in \hat{A}^{a,k}$ , because  $a$  is a 3-optimum. Then, by adding this over all the assignments in  $\hat{A}^{a,k}$  we get:  $10 \cdot R(a) \geq \sum_{\hat{a} \in \hat{A}^{a,k}} R(\hat{a})$ .

We can then engage in a counting exercise by types of constraints. For example, for  $S = \{1, 2\}, a_1^* = \hat{a}_1 = 1$  and  $a_2^* = \hat{a}_2 = 1$  for  $\hat{a} = [1 \ 1 \ 1 \ 0 \ 0], [1 \ 1 \ 0 \ 1 \ 0],$  and  $[1 \ 1 \ 0 \ 0 \ 1]$ . Here we are counting the number of assignments where for a given constraint, all values are the same as the optimal assignment. We can also count how often for a given constraint, all values are the same as the  $k$ -optimal assignment. For example, for  $S = \{1, 2\}, a_1 = \hat{a}_1 = 0$  and  $a_2 = \hat{a}_2 = 0$  only for  $\hat{a} = [0 \ 0 \ 1 \ 1 \ 1]$ . By doing this counting over all the constraints, we can bound the sum of rewards of the assignments in  $\hat{A}^{a,k}$  in terms of the rewards  $R(a^*)$  and  $R(a)$ . For our example, we can obtain a bound that states  $10 \cdot R(a) \geq \sum_{\hat{a} \in \hat{A}^{a,k}} R(\hat{a}) \geq 3 \cdot R(a^*) + 1 \cdot R(a)$ , and hence  $R(a) \geq \frac{3}{10-1} R(a^*) = \frac{1}{3} R(a^*)$ . By solving for  $R(a)$ , we get the bound in the proposition.  $\square$

### Graph-Based Quality Guarantees

The guarantee for  $k$ -optima in the previous section applies to all possible DCOP graph structures. However, knowledge of the structure of constraint graphs can be used to obtain tighter guarantees. If the graph structure of the DCOP is known, we can exploit it by refining our definition of  $\hat{A}^{a,k}$ . In the previous section, we defined  $\hat{A}^{a,k}$  as all assignments  $\hat{a}$  that have a distance of  $k$  from a  $k$ -optimal assignment  $a$  where the values of the deviating variables take on those from the optimal solution  $a^*$ . If the graph is known, let  $\hat{A}^{a,k}$  contain all possible deviations of  $k$  *connected agents* rather than all possible deviations of  $k$  agents. With this refinement, we can produce tighter guarantees for  $k$ -optima in sparse graphs.

To illustrate graph-based bounds, let us consider binary DCOPs where all constraint rewards are non-negative and  $a^*$  is the globally optimal solution. For ring graphs (where each variable has two constraints), we have  $R(a) \geq \frac{k-1}{k+1} R(a^*)$ . For star graphs (each variable has one constraint except the central variable, which has  $n-1$ ), we have  $R(a) \geq \frac{k-1}{n-1} R(a^*)$ . These are provable tight guarantees.

Finally, bounds for DCOPs with arbitrary graphs and non-negative constraint rewards can be found using a linear-fractional program (LFP) (Pearce & Tambe 2007). A linear-fractional program consists of a set of variables, an objective function of these variables to be maximized or (in this case) minimized, and a set of equations or inequalities representing constraints on these variables. It is important to distinguish between the variables and constraints in the LFP and the variables and constraints in the DCOP. In our LFP, we have two variables for each constraint  $S$  in the DCOP graph. The first variable, denoted as  $R_S(a^*)$ , represents the reward that occurs on  $S$  in the globally optimal solution  $a^*$ . The other,  $R_S(a)$ , represents the reward that occurs on  $S$  in a  $k$ -optimal solution  $a$ . The objective is to find a complete set of rewards that minimizes our objective function:

$$\frac{R(a)}{R(a^*)} = \frac{\sum_{S \in \theta} R_S(a)}{\sum_{S \in \theta} R_S(a^*)} \quad (1)$$

In other words, we want to find the set of rewards that will bring about the worst possible  $k$ -optimal solution to get our lower bound.

We minimize this objective function subject to a set of constraints that ensure that  $a$  is in fact  $k$ -optimal. We have one constraint in the LFP for every assignment  $\tilde{a}$  in the DCOP such that: (i) all variables in  $\tilde{a}$  take only values that appear in either  $a$  or  $a^*$  (ii) the distance between  $a$  and  $\tilde{a}$  is less than or equal to  $k$ . These constraints are of the form  $R(a) - R(\tilde{a}) \geq 0$  for every possible  $\tilde{a}$ . The first condition ensures  $R(a)$  and  $R(\tilde{a})$  can be expressed in terms of our variables  $R_S(a)$  and  $R_S(a^*)$ . The second condition ensures that  $a$  will be  $k$ -optimal.

LFPs have been shown to be reducible to standard Linear Programs (LPs) (Boyd & Vandenberghe 2004). This method gives a tight bound for any graph, but requires a globally optimal solution to the resulting LP, in contrast to the constant-time guarantees of the bounds for the fully-connected, ring and star graphs.

### Experimental Results for Lower Bounds

While we have so far focused on theoretical guarantees for  $k$ -optima, this section provides an illustration of these guarantees in action, and how they are affected by constraint graph structure. Figures 3a, 3b, and 3c show quality guarantees for binary DCOPs with fully connected graphs, ring graphs, and star graphs, calculated directly from the bounds discussed earlier.

Figure 3d shows quality guarantees for DCOPs whose graphs are binary trees, obtained using the LFP mentioned in the previous section. Constructing these LFPs and solving them optimally with LINGO 8.0 global solver took about two minutes on a 3 GHz Pentium IV with 1GB RAM.

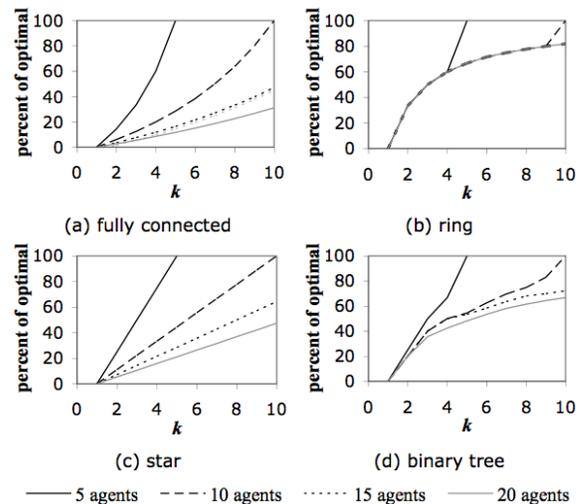


Figure 3: Quality guarantees for  $k$ -optima with respect to the global optimum for DCOPs of various graph structures.

For each of the graphs, the  $x$ -axis plots the value chosen for  $k$ , and the  $y$ -axis plots the lower bound for  $k$ -optima as a percentage of the optimal solution quality for systems of 5, 10, 15, and 20 agents. These results show how the worst-case benefit of increasing  $k$  varies depending on graph structure. For example, in a five-agent DCOP, a 3-optimum is guaranteed to be 50% of optimal whether the graph is a star or a ring. However, moving to  $k = 4$  means that worst-case solution quality will improve to 75% for a star, but only to 60% for a ring. For fully connected graphs, the benefit of increasing  $k$  goes up as  $k$  increases; whereas for stars it stays constant, and for chains it decreases, except for when  $k = n$ . Results for binary trees are mixed.

### Upper Bounds on the Number of $k$ -Optima

Traditionally, researchers have focused on obtaining a single DCOP solution, expressed as a single assignment of actions to agents. However, in this section, we consider a multi-agent system that generates a *set* of  $k$ -optimal assignments, i.e., multiple assignments to the same DCOP. Generating sets of assignments is useful in domains such as disaster rescue (to provide multiple rescue options to a human commander) (Schurr *et al.* 2005), patrolling (to execute multiple patrols in the same area) (Ruan *et al.* 2005), training simulations (to provide several options to a student) and others (Tate, Dalton, & Levine 1998). As discussed earlier in the context of the patrolling domain, when generating such a set of assignments, use of rewards alone leads to somewhat repetitive and predictable solutions (patrols). Picking diverse joint patrols at random on the other hand leads to low-quality solutions. Using  $k$ -optimality directly addresses such circumstances; if no ties exist between the rewards of patrols a distance  $k$  or fewer apart,  $k$ -optimality ensures that all joint patrols differ by at least  $k + 1$  agents' actions, as well as ensuring that this diversity would not come at the expense of obviously bad joint patrols, as each is optimal within a radius of at least  $k$  agents' actions.

Our key contribution in this section is addressing efficient resource allocation for the multiple assignments in a  $k$ -optimal set, by defining tight upper bounds on the number of  $k$ -optimal assignments that can exist for a given DCOP. These bounds are necessitated by two key features of the typical domains where a  $k$ -optimal set is applicable. First, each assignment in the set consumes some resources that must be allocated in advance. Such resource consumption arises because: (i) a team actually executes each assignment in the set, as in our patrolling example above, or (ii) the assignment set is presented to a human user (or another agent) as a list of options to choose from, requiring time. In each case, resources are consumed based on the assignment set size. Second, while the existence of the constraints between agents is known *a priori*, the actual rewards and costs on the constraints depend on conditions that are not known until runtime, and so resources must be allocated before the rewards and costs are known and before the agents generate the  $k$ -optimal assignment set. In the patrolling domain, constraints are known to exist between patrol robots assigned to adjacent or overlapping regions. However, their costs and rewards depend on recent field reports of adversarial activ-

ity that are not known until the robots are deployed. At this point the robots must already be fueled in order for them to immediately generate and execute a set of  $k$ -optimal patrols. The resource to be allocated to the robots is the amount of fuel required to execute each patrol; thus it is critical to ensure that enough fuel is given to each robot so that each assignment found can be executed, without burdening the robots with wasted fuel that will go unused. Recall the other domain mentioned earlier of a team of disaster rescue agents that must generate a set of  $k$ -optimal assignments in order to present a set of diverse options to a human commander. Upper bounds were useful in this domain to choose the right level of  $k$ . Choosing the wrong  $k$  would result in possibly presenting too many options, causing the commander to run out of time before considering them all, or presenting too few (causing high-quality options to be omitted).

Thus, because each assignment consumes resources, it is useful to know the maximal number of  $k$ -optimal assignments that could exist for a given DCOP. Unfortunately, we cannot predict this number because the costs and rewards for the DCOP are not known in advance. Despite this uncertainty, reward-independent bounds can be obtained on the size of a  $k$ -optimal assignment set, i.e., to safely allocate enough resources for a given value of  $k$  for any DCOP with a particular graph structure. In addition to their uses in resource allocation, these bounds also provide insight into the problem landscapes.

### From Coding Theory to Upper Bounds

To find the first upper bounds on the number of  $k$ -optima (i.e. on  $|A_q(n, k)|$ ) for a given DCOP graph, we discovered a correspondence to coding theory (Ling & Xing 2004), yielding bounds independent of both reward and graph structure. In the next section, we provide a method to use the structure of the DCOP graph (or hypergraph of arbitrary arity) to obtain significantly tighter bounds.

In error-correcting codes, a set of codewords must be chosen from the space of all possible words, where each word is a string of characters from an alphabet. All codewords are sufficiently different from one another so that transmission errors will not cause one to be confused for another. Finding the maximum possible number of  $k$ -optima can be mapped to finding the maximum number of codewords in a space of  $q^n$  words where the minimum distance between any two codewords is  $d = k + 1$ . We can map DCOP assignments to words and  $k$ -optima to codewords as follows: an assignment  $a$  taken by  $n$  agents each with a domain of cardinality  $q$  is analogous to a word of length  $n$  from an alphabet of cardinality  $q$ . The distance  $d(a, \tilde{a})$  can then be interpreted as a Hamming distance between two words. Then, if  $a$  is  $k$ -optimal, and  $d(a, \tilde{a}) \leq k$ , then  $\tilde{a}$  cannot also be  $k$ -optimal by definition. Thus, any two  $k$ -optima must be separated by distance  $\geq k + 1$ . (In this section, we assume no ties in  $k$ -optima).

Three well-known bounds on codewords are the Hamming, Singleton and Plotkin bounds (Ling & Xing 2004). We will refer to  $\beta_{HSP}$  is the best of these (graph-independent) bounds.

## Graph-Based Upper Bounds

The  $\beta_{HSP}$  bound depend only on the number of agents  $n$ , the degree of optimality  $k$  and the number of actions available to each agent  $q$ . It ignores the graph structure and thus how the team reward is decomposed onto constraints, i.e. the bounds are the same for all possible sets of constraints  $\theta$ . For instance, the bound on 1-optima for Example 1 according to  $\beta_{HSP}$  is 4, and it ignores the fact that agents 1 and 3 do not share a constraint, and yields the same result independent of the DCOP graph structure. However, exploiting this structure (as captured by  $\theta$ ) can significantly tighten the bounds on the number of  $k$ -optimal solution that could exist. In particular, when obtaining the bounds in the previous section, pairs of assignments were mutually exclusive as  $k$ -optima (only one of the two could be  $k$ -optimal) if they were separated by a distance  $\leq k$ . We now show how some assignments separated by a distance  $\geq k + 1$  must also be mutually exclusive as  $k$ -optima.

If we let  $G$  be some subgroup of agents, then let  $D_G(a, \tilde{a})$  be the set of agents within the subgroup  $G$  who have chosen different actions between  $a$  and  $\tilde{a}$ . Let  $V(G)$  be the set of agents (including those in  $G$ ) who are a member of some constraint  $S \in \theta$  incident on a member of  $G$  (e.g.,  $G$  and the agents who share a constraint with some member of  $G$ ). Then,  $V(G)^C$  is the set of all agents whose contribution to the team reward is independent of the values taken by  $G$ .

**Proposition 2** *Let there be an assignment  $a^* \in A_q(n, k)$  and let  $\tilde{a} \in A$  be another assignment for which  $d(a^*, \tilde{a}) > k$ . If  $\exists G \subset \mathcal{N}$ ,  $G \neq \emptyset$  for which  $|G| \leq k$  and  $D_{V(G)}(a^*, \tilde{a}) = G$ , then  $\tilde{a} \notin A_q(n, k)$ .*

In other words, if an assignment  $\tilde{a}$  contains some group  $G$  that is facing the same context as it does in the  $k$ -optimal assignment  $a^*$ , but chooses different values than those in  $a^*$ , then  $\tilde{a}$  cannot be  $k$ -optimal even though its distance from  $a^*$  is greater than  $k$ .

Proposition 2 provides conditions where if  $a^*$  is  $k$ -optimal then  $\tilde{a}$ , which may be separated from  $a^*$  by a distance greater than  $k$  may not be  $k$ -optimal, thus tightening bounds on  $k$ -optimal assignment sets. With Proposition 2, since agents are typically not fully connected to all other agents, the *relevant context* a subgroup faces is not the entire set of other agents. Thus, the subgroup and its relevant context form a view (captured by  $V(G)$ ) that is not the entire team. We note that this proposition does not imply any relationship between the reward of  $a^*$  and that of  $\tilde{a}$ .

Figure 4(a) shows  $G$ ,  $V(G)$ , and  $V(G)^C$  for a sample DCOP of six agents with a domain of two actions, white and gray. Without Proposition 2,  $\tilde{a}_1$ ,  $\tilde{a}_2$ , and  $\tilde{a}_3$  could all potentially be 2-optimal. However, Proposition 2 guarantees that they are not, leading to a tighter bound on the number of 2-optima that could exist. To see the effect, note that if  $a^*$  is 2-optimal, then  $G = \{1, 2\}$ , a subgroup of size 2, must have taken an optimal subgroup joint action (all white) given its relevant context (all white). Even though  $\tilde{a}_1$ ,  $\tilde{a}_2$ , and  $\tilde{a}_3$  are a distance greater than 2 from  $a^*$ , they cannot be 2-optimal, since in each of them,  $G$  faces the same relevant context (all white) but is now taking a suboptimal subgroup joint action (all gray).

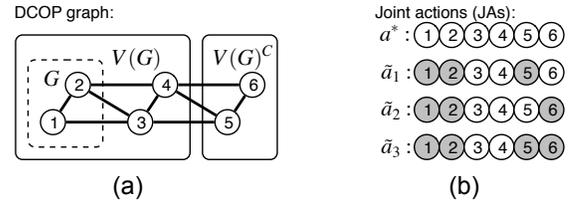


Figure 4: A visual representation of the effect of Proposition 2.

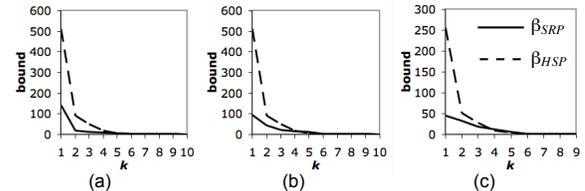


Figure 5:  $\beta_{SRP}$  vs.  $\beta_{HSP}$  for DCOP graphs from Figure 2

Based on Proposition 2 we investigated heuristic techniques to obtain an upper bound on  $|A_q(n, k)|$  that exploits DCOP graph structure. One key heuristic we developed is the *Symmetric Region Packing bound*,  $\beta_{SRP}$ . More details about  $\beta_{SRP}$  are presented in (Pearce, Tambe, & Maheswaran 2006).

## Experimental Results for Upper Bounds

We present two evaluations to explore the effectiveness of the different bounds on the number of  $k$ -optima. First, for the three DCOP graphs shown in Figure 2, Figure 5 provides a concrete demonstration of the gains in resource allocation due to the tighter bounds made possible with graph-based analysis. The  $x$  axis in Figure 5 shows  $k$ , and the  $y$  axis shows the  $\beta_{HSP}$  and  $\beta_{SRP}$  bounds on the number of  $k$ -optima that can exist. To understand the implications of these results on resource allocation, consider a patrolling problem where the constraints between agents are shown in the 10-agent DCOP graph from Figure 2(a), and all agents consume one unit of fuel for each assignment taken. Suppose that  $k = 2$  has been chosen, and so at runtime, the agents will use MGM-2 (to be described in the next section), repeatedly, to find and execute a set of 2-optimal assignments. We must allocate enough fuel to the agents *a priori* so they can execute up to all possible 2-optimal assignments. Figure 5(a) shows that if  $\beta_{HSP}$  is used, the agents would be loaded with 93 units of fuel to ensure enough for all 2-optimal assignments. However,  $\beta_{SRP}$  reveals that only 18 units of fuel are sufficient, a five-fold savings. (For clarity we note that on all three graphs, both bounds are 1 when  $k = n$  and 2 when  $n - 3 \leq k < n$ .)

Second, to systematically investigate the impact of graph structure on bounds, we generated a large number of DCOP graphs of varying size and density. We started with complete binary graphs (all pairs of agents are connected) where each node (agent) had a unique ID. To gradually make each graph sparser, edges were repeatedly removed according to the fol-

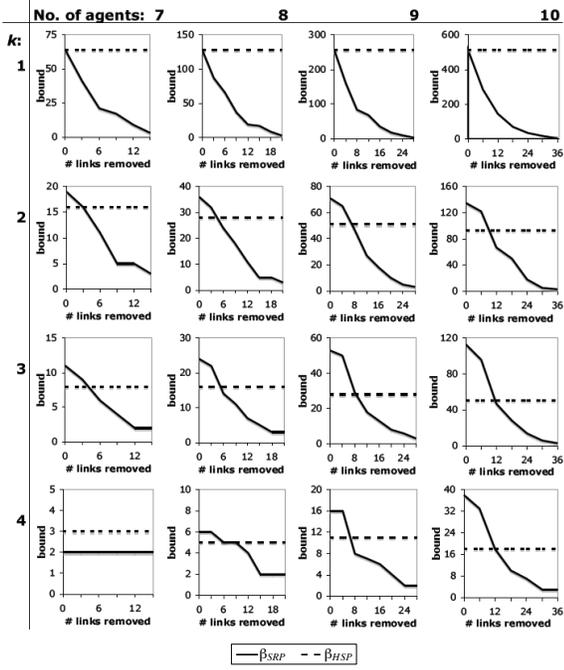


Figure 6: Comparisons of  $\beta_{SRP}$  vs.  $\beta_{HSP}$

lowing two-step process: (1) Find the lowest-ID node that has more than one incident edge. (2) If such a node exists, find the lowest-ID node that shares an edge with it, and remove this edge. Figure 6 shows the  $\beta_{HSP}$  and  $\beta_{SRP}$  bounds for  $k$ -optima for  $k \in \{1, 2, 3, 4\}$  and  $n \in \{7, 8, 9, 10\}$ . For each of the 16 plots shown, the  $y$  axis shows the bounds and the  $x$ -axis shows the number of links removed from the graph according to the above method.

While  $\beta_{HSP} < \beta_{SRP}$  for very dense graphs,  $\beta_{SRP}$  provides significant gains for the vast majority of cases. For example, for the graph with 10 agents, and 24 links removed, and a fixed  $k = 1$ ,  $\beta_{HSP}$  implies that we must equip the agents with 512 resources to ensure that all resources are not exhausted before all 1-optimal actions are executed. However,  $\beta_{SRP}$  indicates that a 15-fold reduction to 34 resources will suffice, yielding a savings of 478 due to the use of graph structure when computing bounds.

## Algorithms

This section contains a description of existing 1-optimal algorithms, new 2- and 3-optimal algorithms, as well as a theoretical analysis of key properties of these algorithms and experimental comparisons.

### 1-Optimal Algorithms

We begin with two algorithms that only consider unilateral actions by agents in a given context. The first is the MGM (Maximum Gain Message) Algorithm which is a modification of DBA (Distributed Breakout Algorithm) (Yokoo & Hirayama 1996) focused solely on gain message passing. MGM is not a novel algorithm, but simply a name chosen

---

### Algorithm 1 DSA (myNeighbors, myValue)

---

- 1: SendValueMessage(myNeighbors, myValue)
  - 2: currentContext = GetValueMessages(myNeighbors)
  - 3: [gain, new Value] = BestUnilateralGain(currentContext)
  - 4: **if** Random(0,1) < Threshold **then**
  - 5:     myValue = new Value
- 

---

### Algorithm 2 MGM (myNeighbors, myValue)

---

- 1: SendValueMessage(myNeighbors, myValue)
  - 2: currentContext = GetValueMessages(myNeighbors)
  - 3: [gain, new Value] = BestUnilateralGain(currentContext)
  - 4: SendGainMessage(myNeighbors, gain)
  - 5: neighborGains = ReceiveGainMessages(myNeighbors)
  - 6: **if** gain > max(neighborGains) **then**
  - 7:     myValue = new Value
- 

to describe DBA without the changes on constraint costs that DBA uses to break out of local minima. We note that DBA itself cannot be applied in an optimization context, as it would require global knowledge of solution quality (it can be applied in a satisfaction context because any agent encountering a violated constraint would know that the current solution is not a satisfying solution). The second is DSA (Distributed Stochastic Algorithm) (Fitzpatrick & Meertens 2003), which is a randomized algorithm. Our analysis will focus on synchronous applications of these algorithms.

Let us define a *round* as involving multiple broadcasts of *messages*. Every time a messaging phase occurs in a round, we will count that as one *cycle* and cycles will be our performance metric for speed, as is common in DCOP literature. Let  $x^{(n)} \in X$  denote the assignments at the beginning of the  $n$ -th round. We assume that every agent will broadcast its current value to all its neighbors at the beginning of the round taking up one cycle. Once agents are aware of their current contexts (i.e. values of neighboring agents), they will go through a process as determined by the specific algorithm to decide which of them will be able to modify their value. Let  $M^{(n)} \subseteq \mathcal{N}$  denote the set of agents allowed to modify the values in the  $n$ -th round. For MGM, 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 all its neighbors (ties can be broken through variable ordering or another method) (Yokoo & Hirayama 1996). For DSA, each agent generates a random number from a uniform distribution on  $[0, 1]$  and acts if that number is less than some threshold  $p$  (Fitzpatrick & Meertens 2003). We note that MGM has a cost of two cycles per round while DSA only has a cost of one cycle per round. Pseudocode for DSA and MGM is given in Algorithms 2 and 3 respectively.

We are able to prove the following monotonicity property of MGM. Let us refer to the set of terminal states of the class of 1-optimal algorithms as  $X_E$ , i.e. no unilateral modification of values will increase sum of all constraint utilities connected to the acting agent(s) if  $x \in X_E$ .

**Proposition 3** When applying MGM, the global utility  $\bar{U}(x^{(n)})$  is strictly increasing with respect to the round ( $n$ ) until  $x^{(n)} \in X_E$ .

Why is monotonicity important? In anytime domains where communication may be halted arbitrarily and existing strategies must be executed, randomized algorithms risk being terminated at highly undesirable assignments. Given a starting condition with a minimum acceptable global utility, monotonic algorithms guarantee lower bounds on performance in anytime environments. Consider the following example.

**The Traffic Light Game.** Consider two variables, both of which can take on the values *red* or *green*, with a constraint that takes on utilities as follows:

- $U(\text{red}, \text{red}) = 0$ .
- $U(\text{red}, \text{green}) = U(\text{green}, \text{red}) = 1$ .
- $U(\text{green}, \text{green}) = -1000$ .

If  $(\text{red}, \text{red})$  is the initial condition, each agent would choose to alter its value to *green* if given the opportunity to move. If both agents are allowed to alter their value in the same round, we would end up in the adverse state  $(\text{green}, \text{green})$ . When using DSA, there is always a positive probability for any time horizon that  $(\text{green}, \text{green})$  will be the resulting assignment.

## 2-Optimal Algorithms

When applying 1-optimal algorithms, the evolution of the assignments will terminate at a 1-optimum within the set  $X_E$  described earlier. One method to improve the solution quality is for agents to coordinate actions with their neighbors. This allows the evolution to follow a richer space of trajectories and alters the set of terminal assignments. In this section we introduce two 2-optimal algorithms, where agents can coordinate actions with one other agent. Let us refer to the set of terminal states of the class of 2-optimal algorithms as  $X_{2E}$ , i.e. neither a unilateral nor a bilateral modification of values will increase sum of all constraint utilities connected to the acting agent(s) if  $x \in X_{2E}$ .

We now introduce two algorithms that allow for coordination while maintaining the underlying distributed decision making process: MGM-2 (Maximum Gain Message-2) and SCA-2 (Stochastic Coordination Algorithm-2).

Both MGM-2 and SCA-2 begin a round with agents broadcasting their current values. The first step in both algorithms is to decide which subset of agents are allowed to make *offers*. We resolve this by randomization, as each agent generates a random number uniformly from  $[0, 1]$  and considers themselves to be an *offerer* if the random number is below a threshold  $q$ . If an agent is an offerer, it cannot accept offers from other agents. All agents who are not offerers are considered to be *receivers*. Each offerer will choose a neighbor at random (uniformly) 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 agent and the offerer's local utility gain for each value pair. Each receiver

will then calculate the global utility gain for each value pair in the offer message by adding the offerer's local utility gain to its own utility change under the new context and (very importantly) subtracting the difference in the link between the two so it is not counted twice. If the maximum global 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.

At this point, the algorithms diverge. For SCA-2, any agent who is not committed and can make a local utility gain with a unilateral move generates a random number uniformly from  $[0, 1]$  and considers themselves to be *active* if the number is under a threshold  $p$ . At the end of the round, all committed agents change their values to the committed offer and all active agents change their values according to their unilateral best response. Thus, SCA-2 requires three cycles (value, offer, accept/reject) per round. In MGM-2 (after the offers and replies are settled), each agent sends a gain message to all its neighbors. Uncommitted agents send their best local utility gain for a unilateral move. Committed agents send the global gain for their coordinated move. Uncommitted agents follow the same procedure as in MGM, where they modify their value if their gain message was larger than all the gain messages they received. 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 *confirm* message from its partner. We note that MGM-2 requires five cycles (value, offer, accept/reject, gain, confirm/deconfirm) per round, and has less concurrency than SCA-2 (since no two neighboring groups in MGM-2 will ever move together). Given the excess cost of MGM-2, why would one choose to apply it? We can show that MGM-2 is monotonic in global utility.

**Proposition 4** When applying MGM-2, the global utility  $\bar{U}(x^{(n)})$  is strictly increasing with respect to the round ( $n$ ) until  $x^{(n)} \in X_{2E}$ .

Furthermore, 2-optimal algorithms will sometimes yield a solution of higher quality than 1-optimal algorithms as shown in the example below; however, this is not true of all situations.

**Meeting Scheduling.** Consider two agents trying to schedule a meeting at either 7:00 AM or 1:00 PM with the constraint utility as follows:  $U(7, 7) = 1$ ,  $U(7, 1) = U(1, 7) = -100$ ,  $U(1, 1) = 10$ . If the agents started at  $(7, 7)$ , any 1-coordinated algorithm would not be able to reach the global optimum, while 2-coordinated algorithms would.

## 3-Optimal Algorithms

The main complication with moving to 3-optimality is the following: With 2-optimal algorithms, the offerer could simply send all information the receiver needed to compute the optimal joint move in the offer message itself. With groups of three agents, this is no longer possible, and thus two more

message cycles are needed. MGM-3, the monotonic version of the 3-optimal algorithm thus requires seven cycles. However, SCA-3, the stochastic version only requires five.

## Experiments

We performed two groups of experiments - one for “medium-sized” DCOPs of forty variables and one for DCOPs of 1000 variables, larger than any problems considered in papers on complete DCOP algorithms.

We considered three different domains for our first group of experiments. The first was a standard graph-coloring scenario, in which a cost of one is incurred if two neighboring agents choose the same color, and no cost is incurred otherwise. Real-world problems involving sensor networks, in which it may be undesirable for neighboring sensors to be observing the same location, are commonly mapped to this type of graph-coloring scenario. The second was a fully randomized DCOP, in which every combination of values on a constraint between two neighboring agents was assigned a random reward chosen uniformly from the set  $\{1, \dots, 10\}$ . The third domain was chosen to simulate a high-stakes scenario, in which miscoordination is very costly. In this environment, agents are negotiating over the use of resources. If two agents decide to use the same resource, the result could be catastrophic. An example of such a scenario might be a set of unmanned aerial vehicles (UAVs) negotiating over sections of airspace, or rovers negotiating over sections of terrain. In this domain, if two neighboring agents take the same value, there is a large penalty incurred (-1000). If two neighboring agents take different values, they obtain a reward chosen uniformly from  $\{10, \dots, 100\}$ . In all of these domains, we considered ten randomly generated graphs with forty variables, three values per variable, and 120 constraints. For each graph, we ran 100 runs of each algorithm.

We used communication cycles as the metric for our experiments, as is common in the DCOP literature, since it is assumed that communication is the speed bottleneck. (However, we note that, as we move from 1-optimal to 2-optimal to 3-optimal algorithms, the computational cost at each agent  $i$  increases by a polynomial factor. For brevity, computational load is not discussed further in this article.) Although each run was for 256 cycles, most of the graphs display a cropped view, to show the important phenomena.

Figure 7 shows a comparison between MGM and DSA for several values of  $p$ . For graph coloring, MGM is dominated, first by DSA with  $p = 0.5$ , and then by DSA with  $p = 0.9$ . For the randomized DCOP, MGM is completely dominated by DSA with  $p = 0.9$ . MGM does better in the high-stakes scenario as all DSA algorithms have a negative solution quality (not shown in the graph) for the first few cycles. This happens because at the beginning of a run, almost every agent will want to move. As the value of  $p$  increases, more agents act simultaneously, and thus, many pairs of neighbors are choosing the same value, causing large penalties. Thus, these results show that the nature of the constraint utility function makes a fundamental difference in which algorithm dominates. Results from the high-stakes scenario contrast with (Zhang *et al.* 2003) and show that DSA is

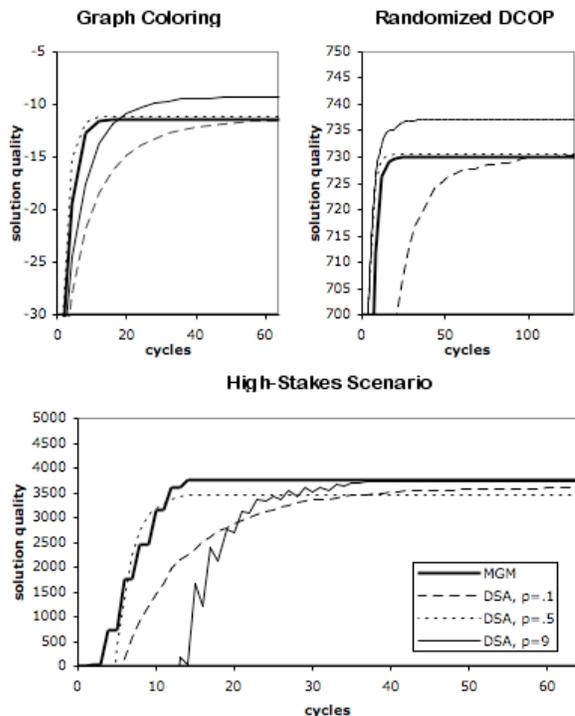


Figure 7: Comparison of the performance of MGM and DSA

not necessarily the algorithm of choice when compared with DBA across all domains.

Figure 8 compares MGM, MGM-2, and MGM-3 for  $q = 0.5$ . In all three cases, MGM-3 increases at the slowest rate, but eventually overtakes MGM-2. Similar results were observed in our comparison of DSA, SCA-2 and SCA-3.

For our second group of experiments, we considered DCOPs of 1000 variables using the graph-coloring and random DCOP domains. The main purpose of these experiments was to demonstrate that the  $k$ -optimal algorithms quickly converge to a solution even for very large problems such as these. A random DCOP graph was generated for each domain, for link densities ranging from 1 to 5, and results for MGM and MGM-3 are shown in the following tables. The tables shown represent an average of 100 runs (from a random initial set of values) for each DCOP. For comparison, complete algorithms (Modi *et al.* 2005) require 1000s of cycles just for graphs of less than 50 variables and constraint density of 3.

Density	Cycles (MGM)	Cycles (MGM-3)
1	7.12	270.62
2	11.74	3277.89
3	15.58	4708.06
4	19.92	5220.46
5	23.30	5448.10

Table 1: Results for MGM and MGM-3 for large DCOPs: Graph Coloring

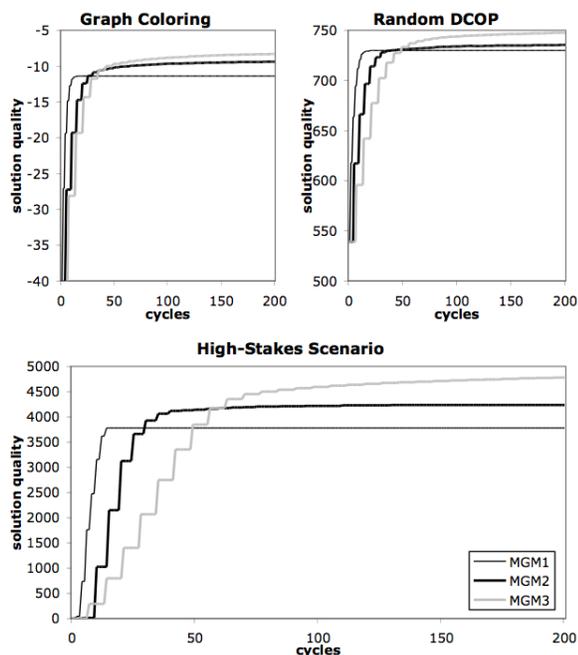


Figure 8: Comparison of the performance of MGM, MGM-2, and MGM-3

## Conclusion

In multiagent domains involving teams of sensors, or teams of unmanned air vehicles or of personal assistant agents, the effect of local interactions between agents can be compactly represented as a network structure. In such agent networks, not all agents interact with all others. DCOP is a useful to framework to reason about agents' local interactions in such networks. This article considers the case of  $k$ -optimality for DCOPs: agents optimize a DCOP by forming groups of one or more agents until no group of  $k$  or fewer agents can possibly improve the solution. The article provides an overview of three key results related to  $k$ -optimality. The first set of results are 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. The second set of results are upper bounds on the number of  $k$ -optima that can exist in a DCOP. Because each joint action consumes resources, knowing the maximal number of  $k$ -optimal joint actions that could exist for a given DCOP allows us to allocate sufficient resources for a given level of  $k$ . Finally, we sketched algorithms for  $k$ -optimality and provided some experimental results on the performance of 1-, 2- and 3-optimal algorithms for several types of DCOPs.

## References

- Boyd, S., and Vandenberghe, L. 2004. *Convex Optimization*. Cambridge U. Press.
- Cox, J.; Durfee, E.; and Bartold, T. 2005. A distributed

framework for solving the multiagent plan coordination problem. In *AAMAS*.

Fitzpatrick, S., and Meertens, L. 2003. Distributed coordination through anarchic optimization. In Lesser, V.; Ortiz, C. L.; and Tambe, M., eds., *Distributed Sensor Networks: A Multiagent Perspective*. Kluwer. 257–295.

Kearns, M.; Littman, M.; and Singh, S. 2001. Graphical models for game theory. In *Proc. UAI*.

Ling, S., and Xing, C. 2004. *Coding Theory: A First Course*. Cambridge U. Press.

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

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

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

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

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

Ruan, S.; Meirina, C.; Yu, F.; Patipati, K. R.; and Popp, R. L. 2005. Patrolling in a stochastic environment. In *10th Intl. Command and Control Research Symp.*

Schurr, N.; Marecki, J.; Scerri, P.; Lewis, J.; and Tambe, M. 2005. The DEFACITO system: Training tool for incident commanders. In *IAAI*.

Tate, A.; Dalton, J.; and Levine, J. 1998. Generation of multiple qualitatively different plan options. In *Proc. AIPS*.

Vickrey, D., and Koller, D. 2002. Multi-agent algorithms for solving graphical games. In *Proc. AAAI*, 345–351.

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

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

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