Distributed Algorithms for DCOP: A Graphical-Game-Based Approach

Rajiv T. Maheswaran, Jonathan P. Pearce and Milind Tambe
Department of Computer Science
University of Southern California
Los Angeles, CA, 90089

Abstract

This paper addresses the application of distributed constraint optimization problems (DCOPs) to large-scale dynamic environments. We introduce a decomposition of DCOP into a graphical game and investigate the evolution of various stochastic and deterministic algorithms. We also develop techniques that allow for coordinated negotiation while maintaining distributed control of variables. We prove monotonicity properties of certain approaches and detail arguments about equilibrium sets that offer insight into the tradeoffs involved in leveraging efficiency and solution quality. The algorithms and ideas were tested and illustrated on several graph coloring domains.

1 Introduction

A distributed constraint optimization problem (DCOP) [7, 11] is a useful formalism in settings where distributed agents, each with control of some variables, attempt to optimize a global objective function characterized as the aggregation of distributed constraint utility functions. DCOP can be applied to many multiagent domains, including sensor nets, distributed spacecraft, disaster rescue simulations, and software personal assistant agents. For example, sensor agents may need to choose appropriate scanning regions to optimize targets tracked over the entire network or personal assistant agents may need to schedule multiple meetings in order to maximize the value of their users' time. As the scale of these domains become large, current complete algorithms incur immense computation costs. A large network of personal assistant agents for instance, would require DCOP global optimization over hundreds of agents and thousands of variables, which is currently very expensive. On the other hand, if we let each agent or variable react on the basis of its local knowledge of neighbors and constraints, we create a system that removes the necessity for tree-based communication structures, scales up very easily and is far more robust to dynamic environments.

Recognizing the importance of local search algorithms, researchers initially introduced DBA[12] and DSA[1] for Distributed CSPs, which were later extended to DCOPs [13]. We refer to these as algorithms without coordination or *1-coordinated* algorithms. While detailed experimental analyses of such algorithms on DCOPs is available[13], we still lack theoretical tools that allow us to understand the evolution and performance of such algorithms on arbitrary DCOP problems. Our fundamental contribution in this paper is the decomposition of a DCOP into an equivalent graphical game. Current literature on graphical games considers general reward functions [3, 10] not necessarily tied to an underlying DCOP setting. This decomposition provides a framework for analysis of 1-coordinated algorithms and furthermore suggests an evolution to *k-coordinated* algorithms, where a collection of k agents coordinate their actions in a single negotiation round.

The paper is organized as follows. In Section 2, we present a formal model of the DCOP framework. In Section 3, we introduce a decomposition of the DCOP into a game, where the players are the variables whose utilities are aggregates of their outgoing constraint utilities. We prove that the optimal solution of the DCOP is a Nash equilibrium in an appropriate game. In Section 4, two algorithms that consider only unilateral modifications of values are presented. We prove monotonicity properties of one approach and discuss its significance. In Section 5, we devise two extensions to the unilateral algorithms that support coordinated actions and prove the monotonicity of one of the extensions, which indicates justification for improved solution quality. In Section 6, we discuss experiments and results and we conclude in Section 7.

2 DCOP: Distributed Constraint Optimization

We begin with a formal representation of a distributed constraint optimization problem and an exposition to our

notational structure. Let $V \equiv \{v_i\}_{i=1}^{N}$ denote a set of variables, each of which can take a value $v_i = x_i \in X_i$, $i \in \mathcal{N} \equiv \{1, \ldots N\}$. Here, X_i will be a domain of finite cardinality $\forall i \in \mathcal{N}$. Interpreting each variable as a node in a graph, let the symmetric matrix E characterize a set of edges between variables/nodes such that $E_{ij} = E_{ji} = 1$ if an edge exists between v_i and v_j and $E_{ij} = E_{ji} = 0$, otherwise ($E_{ii} = 0 \ \forall i$). For each pair (i, j) such that $E_{ij} = 1$, let $U_{ij}(x_i, x_j) = U_{ji}(x_j, x_i)$ represent a reward obtained when $v_i = x_i$ and $v_j = x_j$. We can interpret this as a utility generated on the edge between v_i and v_j, contingent simultaneously on the values of both variables and hence referred to as a *constraint*. The global or team utility $\overline{U}(x)$ is the sum of the rewards on all the edges when the variables choose values according to the assignment $x \in X \equiv X_1 \times \cdots \times X_N$. Thus, the goal is to choose an assignment, $x^* \in X$, of values to variables such that $x^* \in \arg\max_{x \in X} \overline{U}(x) = \arg\max_{x \in X} \sum_{i,j:E_{ij}=1} U_{ij}(x_i, x_j)$ where x_i is the i-th variable's value under an assignment vector $x \in X$. This constraint *optimization* problem completely characterized by (X, E, U), where U is the collection of constraint utility functions, becomes *distributed* in nature when control of the variables is partitioned among a set of autonomous agents. For the rest of this paper, we make the simplifying assumption that there are N agents, each in control of a single variable.

3 DCOP Games

Various complete algorithms [7] have been developed to solve a given DCOP. Though heuristics that significantly speed up convergence have been developed [6], the complexity is still prohibitive in large-scale domains. The tree-based communication structures are not robust to dynamics in problem structure. Finding a solution to a slightly modified problem requires a complete rerun which is expensive and may never terminate if the time-scale of the dynamics are faster than the time-scale of the complete algorithm.

Thus, we focus on non-hierarchical variable update strategies based on local information consisting of neighbors' values and constraint utility functions on outgoing edges. We remove the need to establish a parent-child relationship between nodes. Essentially, we are creating a game where the players are the variables, the actions are the choices of values and the information state is the context consisting of neighbor's values. The key design factor is how the local utility functions are constructed from the constraint utility functions. We present a particular decomposition of the DCOP (or equivalently a construction of local utility functions) below.

Let v_j be called a *neighbor* of v_i if $E_{ij} = 1$ and let $\mathcal{N}_i \equiv \{j : j \in \mathcal{N}, E_{ij} = 1\}$ be the indexes of all neighbors of the i-th variable. Let us define $x_{-i} \equiv [x_{j_1} \cdots x_{j_{K_i}}]$, hereby referred to as a *context*, be a tuple which captures the values assigned to the $K_i \equiv |\mathcal{N}_i|$ neighboring variables of the i-th variable, i.e. $v_{j_k} = x_{j_k}$ where $\cup_{k=1}^{K_i} j_k = \mathcal{N}_i$. We now define a local utility for the i-th agent (or equivalently the i-th variable) as: $u_i(x_i; x_{-i}) \equiv \sum_{j \in \mathcal{N}_i} U_{ij}(x_i, x_j)$. We now have a DCOP game defined by (X,E,u) where u is a collection of local utility functions.

A *Nash equilibrium* assignment is a tuple of values $\hat{x} \in X$ where no agent can improve its local utility by unilaterally changing its value given its current context: $\hat{x}_i \in \arg\max_{x_i \in X_i} u_i(x_i; \hat{x}_{-i}), \ \forall i \in \mathcal{N}$. Given a DCOP game (X,E,u), let $X_{NE} \subseteq X$ be the subset of the assignment space which captures all Nash equilibrium assignments: $X_{NE} \equiv \{\hat{x} \in X : \hat{x}_i \in \arg\max_{x_i \in X_i} u_i(x_i; \hat{x}_{-i}), \ \forall i \in \mathcal{N}\}$.

Proposition 1 *The assignment x^* which optimizes the DCOP characterized by (X, E, U) is also a Nash equilibrium with respect to the graphical game (X, E, u).*

Proof. Let us assume that x^* optimizes the DCOP (X, E, U) yet is not a Nash equilibrium assignment. Then, some agent i can improve its local utility by ϵ by altering the value of its variable. However, because i's utility is made up of the sum of $U_{ij}(x_i, x_j)$ for all its neighbors j, if i's utility improves by ϵ, then the utility of i's neighbors, as a group, must also increase by ϵ, for a net increase of 2ϵ for the whole system, leading to a higher overall utility than the optimal solution x^*, which is a contradiction. ■

Because we are optimizing over a finite set, we are guaranteed to have an assignment that yields a maximum. By the previous proposition, an assignment that yields a maximum is also a Nash equilibrium, thus, we are guaranteed the existence of a pure-strategy Nash equilibrium. This claim cannot be made for any arbitrary graphical game [3, 10]. Though it has been shown to exist in congestion games without unconditional independencies [9, 8], we have shown that the games derived from DCOPs have this property in a setting with unconditional independencies. The mapping to and from the underlying distributed constraint optimization problem yields additional structure. If there were only two variables, the agents controlling each variable would be coupled by the fact that they would receive identical payoffs from their constraint. In a general graph, DCOP-derived local utility functions reflect the amalgamation of multiple such couplings which reflects an inherent benefit to cooperation.

4 Algorithms without Coordination

Given this game-theoretic framework, how will agents' choices for values of their variables evolve over time? In

a purely selfish environment, agents might be tempted to always react to the current context with the action that optimizes their local utility, but this behavior can lead to an unstable system [5]. Imposing structure on the dynamics of updating values can lead to stability and to improved rates of convergence [4]. We begin with 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) [12] focused solely on gain message passing. MGM is not a novel algorithm, but simply the name we use 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 to a truly distributed system, as it requires global knowledge of solution quality. The second is DSA (Distributed Stochastic Algorithm) [1], which is a homogeneous stationary randomized algorithm. Our analysis will focus on synchronous applications of these algorithms.

Let us define a *round* as the duration to execute one run of a particular algorithm. This run could involve 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 algorithm 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, 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). 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. We note that MGM has a cost of two cycles per round while DSA only has a cost of one cycle per round. Through our game-theoretic framework, we are able to prove the following monotonicity property of MGM.

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

Proof. Later in the paper, we provide a mathematically rigorous proof of the monotonicity of a new algorithm, MGM-2, which encompasses MGM, so here we provide a brief proof description.

In MGM, if the i-th variable is allowed to modify its value in a particular round, then its gain is higher than all its neighbors' gains. Consequently, all its neighbors would have received a gain message higher than their own and thus, would not modify their values in that round.

From the proof of Proposition 1, because the i-th variable's utility is made up of the sum of $U_{ij}(x_i, x_j)$ for all its neighbors j, if i's utility improves by ϵ, then the utility of i's neighbors, as a group, must also increase by ϵ. Since no two neighbors can move simultaneously, the global utility $\overline{U}(x^{(n)})$ is strictly increasing. And, since global utility cannot be higher than the optimal solution (meaning it cannot increase forever), MGM must reach a point at which no variable can realize a gain > 0. Thus, MGM yields monotonically increasing global utility until equilibrium. ■

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.

Example 1 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: $U(red, red) = 0, U(red, green) = U(green, red) = 1, U(green, green) = -1000$. Turning this DCOP into a game would require the agent for each variable to take the utility of the single constraint as its local utility. If (red, 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 $(green, green)$. When using DSA, there is always a positive probability for any time horizon that $(green, green)$ will be the resulting assignment.*

In domains such as independent path planning of trajectories for UAVs or rovers, in environments where communication channels are unstable, bad assignments could lead to crashes whose costs preclude the use of methods without guarantees. This is illustrated in Figure 1 which displays sample trajectories for MGM and DSA with identical starting conditions for a high-stakes scenario described in Section 6. The performance of both MGM and DSA with respect to a various graph coloring problems are investigated and discussed in Section 6.

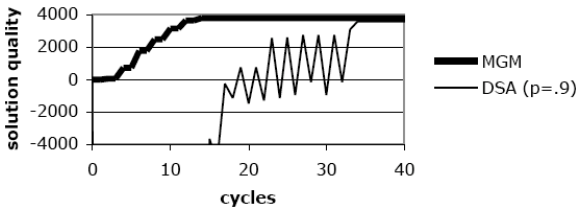

Figure 1: Sample Trajectories of MGM and DSA for a High-Stakes Scenario

5 Algorithms with Coordination

When applying algorithms without coordination, the evolution of the assignments will terminate at a Nash equilibrium point within the set X_{NE} 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-coordinated* algorithms, where agents can coordinate actions with one other agent. Let us refer to the set of terminal states of the class of *2-coordinated* 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 $x \in X_{2E}$. Clearly the terminal states of a coordinated algorithm will depend on what metric the coordinating agents will use to determine if a particular joint action is acceptable or not. In a team setting (and in our analysis), a joint action that increases the sum of the utilities of the acting agents is considered acceptable, even if a single agent may see a loss in utility. This would be true in a purely selfish environment as well, if agents could compensate each other for possible losses in utility. An alternative choice would be to make a joint action acceptable only if both agents see utility gains. We consider the former notion of an acceptable joint action and define the terminal states as follows:

$$X_{2E} = \Big\{\hat{x} : (\hat{x}_i, \hat{x}_j) = \arg\max_{(x_i,x_j)} \{u_i(x_i; \mu_{-i}(x_j, \hat{x}_{-ij})) + u_j(x_j; \mu_{-j}(x_i, \hat{x}_{-ji}))\},\ \forall i, j \in \mathcal{N}, i \neq j\Big\}$$

where x_{-ij} is a tuple consisting of all values of variables except the i-th and j-th variable, and $\mu_{-i}(x_j, x_{-ji})$ is a function that converts its arguments into an appropriate vector of the form of x_{-i} described earlier, i.e. μ_{-i} takes values from the variables indexed by $\{j\} \cup \{\mathcal{N} \setminus \{i \cup j\}\}$ to a vector composed of the variables indexed by $_{\mathcal{N}-i}$.

Proposition 3 *For a given DCOP* (X, E, U) *and its equivalent game* (X, E, u)*, we have* $X_{2E} \subseteq X_{NE}$.

Proof. We show this by proving the contrapositive. Suppose $x \notin X_{NE}$. Then, there exists a variable i such that $u_i(\hat{x}_i; x_{-i}) > u_i(x_i; x_{-i})$ for some $\hat{x}_i \neq x_i$. This further implies that there exists some variable $j \in \mathcal{N}_i$, for which $U_{ij}(\hat{x}_i, x_j) > U_{ij}(x_i, x_j)$. We then have $u_i(\hat{x}_i; \mu_{-i}(x_j, x_{-ij})) > u_i(x_i; \mu_{-i}(x_j, x_{-ij}))$ and $u_j(x_j; \mu_{-j}(\hat{x}_i, x_{-ji})) > u_j(x_j; \mu_{-j}(x_i, x_{-ij}))$ which implies that $x \notin X_{2E}$. ■

Essentially, we are saying that a unilateral move which improves the utility of a single agent must improve the constraint utility of at least one link which further implies that the local utility of another agent must also increase given that the rest of its context remains the same. The interesting phenomenon is that our definition of X_{2E} above is sufficient to capture unilateral and bilateral deviations within the context of bilateral deviations. This is due to the underlying DCOP structure and not true for a general game.

It has been proposed that coordinated actions be achieved by forming coalitions among variables. In [2], each coalition was represented by a *manager* who made the assignment decisions for all variables within the coalition. These methods inherently undermine the distributed nature of the decision-making by essentially replacing multiple variables with a single variable in the graph. It is not possible in all situations for this to occur because utility function information and the ability to communicate with the necessary neighbors may not be transferable (due to infeasibility or preference). We introduce two algorithms that allow for coordination while maintaining the underlying distributed decision making process and the same constraint graph: 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 becomes 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 *receivers*. Each offerer will choose a neighbor at random (uniformly) and send it an offer message consisting 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. 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 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, causing both offerer and receiver to be committed. Otherwise, it sends a *reject* message to the offerer, and neither one 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 *go* message if all the gain messages they received were less than the calculated global gain for the coordinated move and send a *no-go* message, otherwise. A committed agent will only modify its value if it receives a go message from its partner. We note that MGM-2 requires five cycles (value, offer, accept/reject, gain, go/no-go) per round. 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* $\overline{U}(x^{(n)})$ *is strictly increasing with respect to the round (n) until* $x^{(n)} \in X_{2E}$.

Proof. We begin by introducing some notation. At the end of the n-th round, let $C^{(n)} \subset \mathcal{N}$ denote the set of agents who are committed, $M^{(n)} \subset \mathcal{N}$ denote the set of uncommitted agents who are active, and $S^{(n)} \equiv \{C^{(n)} \cup M^{(n)}\}^C \subset \mathcal{N}$ denote the uncommitted agents who are inactive. Let $p(i) \in C^{(n)}$ denote the partner of a committed agent $i \in C^{(n)}$. The global utility can then be expressed as:

$$
\begin{aligned}
&\overline{U}\left(x^{(n+1)}\right) \\
&= \sum_{i,j:E_{ij}=1} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) \\
&= \sum_{\substack{i,j:i\in C^{(n)},\\ j\in C^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) + \sum_{\substack{i,j:i\in C^{(n)},\\ j\in S^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) \\
&\quad + \sum_{\substack{i,j:i\in S^{(n)},\\ j\in C^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) + \sum_{\substack{i,j:i\in S^{(n)},\\ j\in S^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) \\
&\quad + \sum_{\substack{i,j:i\in M^{(n)},\\ j\in S^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) + \sum_{\substack{i,j:i\in S^{(n)},\\ j\in M^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) \\
&= \sum_{i\in C^{(n)}} U_{ip(i)}\left(x_i^{(n+1)}, x_{p(i)}^{(n+1)}\right) + \sum_{i\in C^{(n)}} \sum_{j\in\mathcal{N}_i\setminus\{p(i)\}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) \\
&\quad + \sum_{j\in C^{(n)}} \sum_{i\in\mathcal{N}_j\setminus\{p(j)\}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) + \sum_{j\in C^{(n)}} U_{jp(j)}\left(x_{p(j)}^{(n+1)}, x_j^{(n+1)}\right) \\
&\quad - \sum_{j\in C^{(n)}} U_{jp(j)}\left(x_{p(j)}^{(n+1)}, x_j^{(n+1)}\right) + \sum_{\substack{i,j:i\in S^{(n)},\\ j\in S^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) \\
&\quad + \sum_{i\in M^{(n)}} u_i\left(x_i^{(n+1)}, x_j^{(n+1)}\right) + \sum_{j\in M^{(n)}} u_j\left(x_i^{(n+1)}, x_j^{(n+1)}\right) \\
&= \sum_{i\in C^{(n)}} U_{ip(i)}\left(x_i^{(n+1)}, x_{p(i)}^{(n+1)}\right) + \sum_{i\in C^{(n)}} \sum_{j\in\mathcal{N}_i\setminus\{p(i)\}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n)}\right) \\
&\quad + \sum_{j\in C^{(n)}} \sum_{i\in\mathcal{N}_j\setminus\{p(j)\}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n)}\right) + \sum_{j\in C^{(n)}} U_{jp(j)}\left(x_{p(j)}^{(n+1)}, x_j^{(n+1)}\right) \\
&\quad - \sum_{j\in C^{(n)}} U_{jp(j)}\left(x_{p(j)}^{(n+1)}, x_j^{(n+1)}\right) + \sum_{\substack{i,j:i\in S^{(n)},\\ j\in S^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n)}, x_j^{(n)}\right) + \\
&\quad + \sum_{i\in M^{(n)}} u_i\left(x_i^{(n+1)}, x_j^{(n)}\right) + \sum_{j\in M^{(n)}} u_j\left(x_i^{(n)}, x_j^{(n+1)}\right) \\
&= \sum_{i\in C^{(n)}} u_i(x_i^{(n+1)}; \mu_{-i}(x_{p(i)}^{(n+1)}, x_{-ip(i)}^{(n)})) \\
&\quad + \sum_{j\in C^{(n)}} u_j(x_j^{(n+1)}; \mu_{-j}(x_{p(j)}^{(n+1)}, x_{-jp(j)}^{(n)})) \\
&\quad - \sum_{j\in C^{(n)}} U_{jp(j)}\left(x_{p(j)}^{(n+1)}, x_j^{(n+1)}\right) + \sum_{\substack{i,j:i\in S^{(n)},\\ j\in S^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n)}, x_j^{(n)}\right) \\
&\quad + \sum_{i\in M^{(n)}} u_i\left(x_i^{(n+1)}, x_j^{(n)}\right) + \sum_{j\in M^{(n)}} u_j\left(x_i^{(n)}, x_j^{(n+1)}\right) \\
&> \sum_{i\in C^{(n)}} u_i(x_i^{(n)}; \mu_{-i}(x_{p(i)}^{(n)}, x_{-ip(i)}^{(n)})) + \sum_{j\in C^{(n)}} u_j(x_j^{(n)}; \mu_{-j}(x_{p(j)}^{(n)}, x_{-jp(j)}^{(n)})) \\
&\quad - \sum_{j\in C^{(n)}} U_{jp(j)}\left(x_{p(j)}^{(n)}, x_j^{(n)}\right) + \sum_{\substack{i,j:i\in S^{(n)},\\ j\in S^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n)}, x_j^{(n)}\right) \\
&\quad + \sum_{i\in M^{(n)}} u_i\left(x_i^{(n)}, x_j^{(n)}\right) + \sum_{j\in M^{(n)}} u_j\left(x_i^{(n)}, x_j^{(n)}\right) = \overline{U}\left(x^{(n)}\right).
\end{aligned}
$$

The first equality is by definition. The second equality partitions the indexes into update class, eliminating cross indexes of $M^{(n)}$ with anything other than $S^{(n)}$. In the third equality, we simplify the summations involving committed agents using expressions for partners and neighbors, we insert a zero value term in parenthesis, and transform the summations involving active agents into local utilities. In the fourth equality, we modify the round index for the inactive agents. In the fifth, we transform the summations involving committed agents into local utilities. The inequality is due to the fact that the global utility on the links of the committed partners and the local utility of the active agents must increase due to the positive gain messages. The key is

that by setting $j = p(i)$ in the second and third summations, we recover the gain message of the committed teams. Note the subtraction of the utility gain on the link between partners to avoid double counting. The final equality comes by reversing the transformation to yield the previous round's global utility. Thus, MGM-2 yields monotonically increasing global utility until equilibrium is reached. ■

Example 2 Meeting Scheduling. *Consider two agents trying to schedule a meeting at either 7am or 1pm 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* $(7,7)$, *no 1-coordinated algorithm would be able to reach a global optimum, while 2-coordinated algorithms would.*

A 2-coordinated algorithm solves the problem because both agents together can see all possible solutions. However, it is not obvious that a 2-coordinated algorithm always yields a better solution than a 1-coordinated algorithm. In fact, there are DCOPs and initial conditions for which a 1-coordinated algorithm yields a better solution than a 2-coordinated algorithm, since we cannot predict each algorithm's exact trajectory. However, the above proposition gives us some confidence that 2-coordinated algorithms will perform better on average due to the following:

Corollary 1 *For every initial condition* $x_0 \in X_{NE} \setminus X_{2E}$, *MGM-2 yields a better solution than either MGM or DSA.*

Proof. Since $x_0 \in X_{NE}$, neither MGM nor DSA will move and the solution quality will be that obtained at the assignment x_0. However, since $x_0 \notin X_{2E}$, MGM-2 will continue to evolve from x_0 until it reaches an assignment in X_{2E}. Because MGM-2 is monotonic in global utility, any solution it reaches in X_{2E} will have a higher global utility than x_0. ■

Thus, MGM-2 dominates DSA and MGM for initial conditions in $X_{NE} \setminus X_{2E}$ and is identical to DSA and MGM on X_{2E} (as neither algorithm will evolve from there). The unknown is the behavior on $X \setminus X_{NE}$. It is difficult to analyze this space because one cannot pinpoint the trajectories due to the probabilistic nature of their evolution. If we assume that iterations beginning in $X \setminus X_{NE}$ are taken to points in X_{NE} in a relatively uniform manner on average with all algorithms, then we might surmise that the dominance of MGM-2 should yield a better solution quality. The performance of MGM-2 and SCA-2 in several domains are investigated and discussed in Section 6.

6 Experiments

We considered three different domains for our experiments. The first was a graph-coloring scenario in which a cost of one is incurred if two neighbors 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 problem. The second was a DCOP in which every combination of values on a constraint between two neighbors was assigned a random reward chosen uniformly from $\{1, \ldots, 10\}$. In both domains, we considered 10 randomly generated graphs with 40 variables, three values per variable, and 120 constraints. For each graph, we ran 100 runs of each algorithm, with a random start state. The third domain simulates a scenario in which miscoordination is very costly. In this high-stakes environment, agents negotiate over the use of resources, and if two agents decide to use the same resource, the result could be catastrophic. Such an example might be a set of unmanned aerial vehicles (UAVs) negotiating over sections of airspace, or rovers negotiating over sections of terrain. Here, if two neighbors take the same value, a large penalty (-1000) is incurred; otherwise they obtain a reward chosen uniformly from $\{10, \ldots, 100\}$. Because miscoordination is costly, we introduced a *safe* (zero) value for all agents. An agent with this value is not using any resource. If two neighbors choose zero, neither a reward nor a penalty is obtained. In this scenario, a randomized start state would be a poor choice, especially for an anytime algorithm, as it would likely contain many large penalties. So, rather than using randomized start states, all agents started with zero. To induce agents to move, so that 1-coordinated algorithms would not be useless, a reward of one was introduced for the case where one agent has the zero value, and its neighbor has a nonzero value. In this domain, we also performed 100 runs on each of 10 randomly generated graphs with forty variables and 120 constraints, but due to the addition of the safe value, the agents here had four possible values.

For each of the three domains, we ran: MGM, DSA with $p \in \{.1, .3, .5, .7, .9\}$, MGM-2 with $q \in \{.1, .3, .5, .7, .9\}$ and SCA-2 with all combinations of the above values of p and q (where q is the probability of being an offerer and p is the probability of an uncommited agent acting). Each table shows an average of 100 runs on 10 randomly generated examples with some selected values of p and q. 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, as we move from 1- to 2-coordinated algorithms, the computational cost each agent i must incur can increase by a factor of as much as $\sum_j |X_j|$ as the agent can now consider the combination of its and all its neighbors' moves. However, in the 2-coordinated algorithms we present, each agent randomly picks a single neighbor j to coordinate with, and so its computation is increased by a factor of only $|X_j|$. Al-

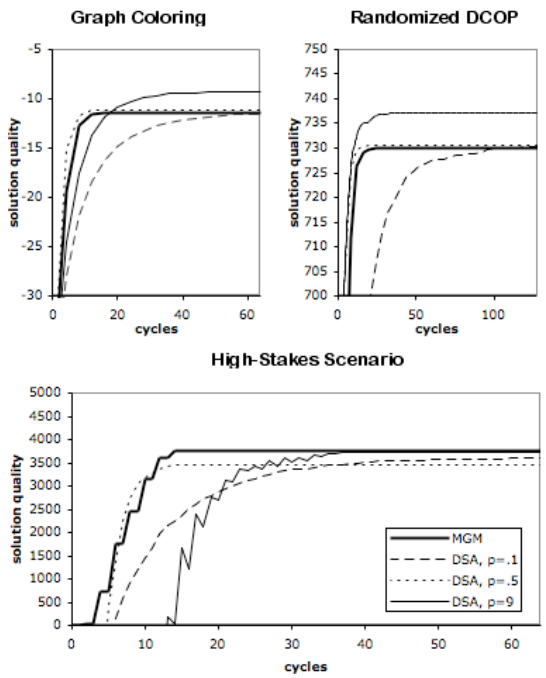

Figure 2: Performance comparison of MGM and DSA

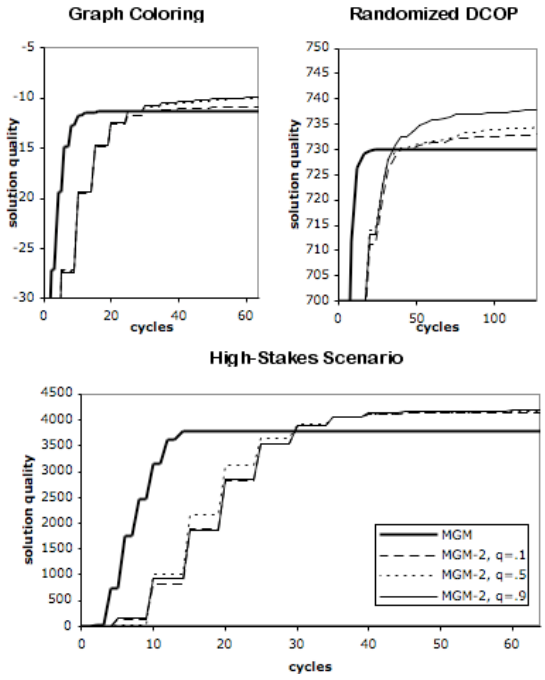

Figure 3: Performance comparison of MGM and MGM-2

though each run was for 256 cycles, most of the graphs display a cropped view, to show the important phenomena.

Figure 2 compares MGM and DSA for several p values. For graph coloring, MGM is dominated, first by DSA with $p = .5$, and then by DSA with $p = .9$. For the randomized DCOP, MGM is completely dominated by DSA with $p = .9$. MGM does better in the high-stakes scenario as all DSA algorithms have negative solution quality (not shown in the graph) for the first few cycles, because at the beginning of a run, almost every agent wants to move. As p increases, more agents act simultaneously, and so many pairs of neighbors choose 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 [13] and show that DSA is not necessarily the algorithm of choice vs. DBA in all domains.

Figure 3 shows a comparison between MGM and MGM-2, for several values of q. In all domains, MGM-2 eventually reaches a higher solution quality after about 30 cycles, despite the algorithms' initial slowness. The stair-like shape of the MGM-2 curves is due to the fact that agents are changing values only once out of every five cycles, due to the cycles used in communication. Of the three values of q shown in the graphs, MGM-2 rises fastest when $q = .5$, but eventually reaches its highest average solution quality when $q = .9$, for all three domains. We note that, in the high-stakes domain, the solution quality is positive at every cycle, due to the monotonic property of both MGM and MGM-2. Thus, these experiments clearly verify the monotonicity of MGM and MGM-2, and also show that MGM-2 reaches a higher solution quality as expected.

Figure 4 shows a comparison between DSA and SCA-2, for $p = .9$ and several values of q. DSA starts out faster, but SCA-2 eventually overtakes it. The result of the effect of q on SCA-2 appears inconclusive. Although SCA-2 with $q = .9$ does not achieve a solution quality above zero for the first 65 cycles, it eventually achieves a solution quality comparable to SCA with lower values of q.

We also note that the phase transition mentioned in [13] (where DSA's performance degrades for $p > .8$) is not replicated in our results. In fact, our solution quality gets better as $p > .8$, though with slower convergence.

7 Summary

Key contributions of this paper include: (i) decomposition of a DCOP into an equivalent graphical game, (ii) proof of monotonicity for MGM, a 1-coordinated algorithm, (ii) development of 2-coordinated algorithms that maintain distributed control of variables, (iii) proof of monotonicity of MGM-2, and (iv) experimental verification and discovery when applying these algorithms to a variety of problems. The key theoretical idea is that breaking a DCOP down to a game can lead to algorithms where we

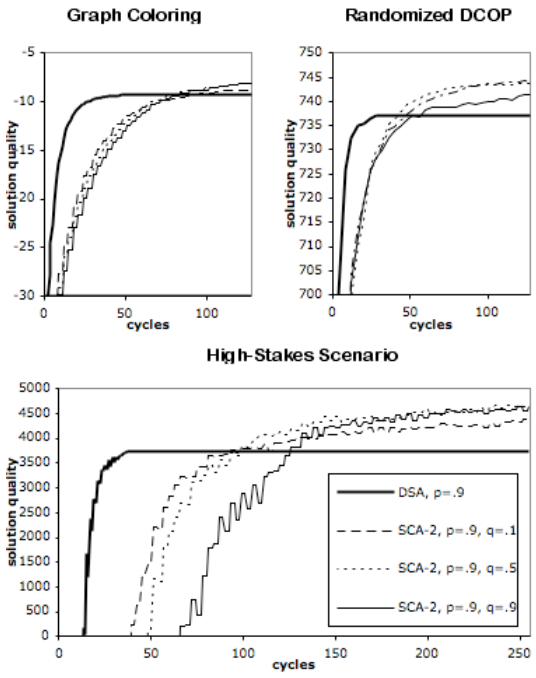

Figure 4: Performance comparison of DSA and SCA-2

can guarantee strict improvement in solution quality over time, which is critical in anytime application in high-stakes environments. Also important is the idea of *k*-coordinated algorithms leading to progressively nested sets of equilibria, which yield both a higher average solution quality and a higher likelihood of obtaining a globally optimal solution. Through experiments, we show that randomized algorithms though very efficient are not always ideal. Initial results imply that the nature of the constraint utility function makes a fundamental difference in the solution structure rather than the graph structure. Future work will entail development of *k*-coordinated algorithms and deeper analysis of stochastic schemes to obtain analytic reasoning for choosing particular update rates. Also, it would be interesting to see if heterogeneous dynamic randomized algorithms can reduce convergence rates. Finally, we plan to apply our work to larger examples to more closely approximate real-world conditions.

References

[1] S. Fitzpatrick and L. Meertens. Distributed coordination through anarchic optimization. In V. Lesser, C. L. Ortiz Jr., and M. Tambe, editors, *Distributed Sensor Networks: A Multiagent Perspective*, pages 257–295. Kluwer Academic Publishers, 2003.

[2] K. Hirayama and J. Toyoda. Forming coalitions for breaking deadlocks. In *Proc. ICMAS*, pages 155–162, 1995.

[3] M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. In *Proc. UAI*, pages 253–260, 2001.

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

[5] R. T. Maheswaran and T. Başar. Decentralized network resource allocation as a repeated noncooperative market game. In *Proc. CDC*, Orlando, FL, December 2001.

[6] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking DCOP to the real world: efficient complete solutions for distributed multi-event scheduling. In *AAMAS 2004*, New York, NY, July 2004.

[7] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proceedings of the Second International Conference on Autonomous Agents and Multi-Agent Systems*, Sydney, Australia 2003.

[8] D. Monderer and L. S. Shapley. Potential games. *Games and Economic Behavior*, 14:124–143, 1996.

[9] R. W. Rosenthal. A class of games possessing pure-stratgy nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

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

[11] 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(5):673–685, 1998.

[12] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction and optimization problems. In *Proc. ICMAS*, pages 401–408, 1996.

[13] 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*, pages 185–192, Melbourne, Australia, July 2003.