# When Should There be a “Me” in “Team”? Distributed Multi-Agent Optimization Under Uncertainty

Matthew E. Taylor, Manish Jain, Yanquin Jin[⋆], Makoto Yooko[†], and Milind Tambe
University of Southern California, Los Angeles, CA {taylorm,manishja,tambe}@usc.edu
[⋆]Tsinghua University, Beijing, China jinyq06@gmail.com
[†]Kyushu University, Fukuoka, Japan yokoo@is.kyushu-u.ac.jp

## ABSTRACT

Increasing teamwork between agents typically increases the performance of a multi-agent system, at the cost of increased communication and higher computational complexity. This work examines joint actions in the context of a multi-agent optimization problem where agents must cooperate to balance exploration and exploitation. Surprisingly, results show that increased teamwork can hurt agent performance, *even when communication and computation costs are ignored*, which we term the team uncertainty penalty. This paper introduces the above phenomena, analyzes it, and presents algorithms to reduce the effect of the penalty in our problem setting.

## Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents*

## General Terms

Algorithms, Experimentation

## Keywords

Coordination, DCEE, DCOP, Multi-agent Exploration and Exploitation, Multi-agent Optimization

## 1. INTRODUCTION

A significant body of work in multiagent systems over more than two decades has focused on teamwork [4, 8, 17]. If cooperative agents can efficiently make joint decisions, so the common wisdom goes, the team as a whole will only benefit [18] — provided the increased coordination and communication overheads do not overwhelm the agents.[1] This paper introduces the *team uncertainty penalty*: joint decisions by a team of agents acting under uncertain conditions can lead to a significant degradation in team performance, relative to agents acting alone. Contrary to popular wisdom, the problem is not the cost of increased communication or computation in service of the joint decision; the problem is the joint decision itself.

Before discussing this paper's investigation of the penalty, first consider the Distributed Constraint Optimization Problem (*DCOP*) [10, 12, 14] formulation. In DCOPs, agents attempt to maximize a global reward function by changing local variables to optimize constraints. Variable settings are only observable by the agent that controls the variable and constraints are only observed by the agents sharing the constraint. Complete DCOP algorithms find optimal solutions but are NP-hard and do not scale to large domains due to computation and communication costs [12], while incomplete algorithms provide only locally optimal solutions but are able to scale significantly better.

One type of incomplete DCOP algorithm, termed *k-optimal* [9, 13], allows $k$ agents to coordinate variable changes at any given time. K-optimal algorithms have proven solution quality guarantees and they discover solutions where $k$ or fewer agents cannot change values together to improve performance. For example, k=2 corresponds to pairs of agents taking joint actions in a DCOP, reaching a 2-optimal solution. Algorithms that use higher values of $k$, increasing the amount of partial centralization, have been shown to effectively better optimize the global reward. However, this requires additional computation and communication costs [9, 13].

Rather than using the standard DCOP formulation, this study allows uncertainty in the world model. We define *Distributed Coordination of Exploration and Exploitation* (DCEE) problems in Section 2.1, in which agents must balance coordinated exploration of an unknown environment with exploitation of their rewards to best maximize the total reward. Examples of such real world problems include network routing optimization, repeated task allocation, and mobile network optimization.

DCEE algorithms must differ from DCOP algorithms along multiple dimensions. DCEE agents do not know their reward functions, are unable to exhaustively explore the environment (which is assumed to be very large or even infinite), and agents seek to maximize their online reward. It is precisely for these reasons that one might assume teamwork would always be advantageous. Unfortunately, in some settings the teamwork uncertainty penalty reduces performance. After discussing this phenomena, two types of algorithms are introduced to help ameliorate this penalty: the first improve performance by disallowing teamwork in certain settings, and the second by discounting actions that have uncertainty.

This paper has several contributions, including: (1) extending existing k=1 algorithms for DCOPs with unknown reward matrices to k=2, (2) empirically analyzing the tradeoffs of these algorithms,

[1]This paper focuses on cooperative multi-agent problems where all agents may be considered part of a single team as they share a common reward function. However, we use the term “level of teamwork” to refer to the amount of partial centralization among agents, reflected in how much information they share, how they coordinate actions, and how many agents may simultaneously perform a joint action. More precisely, higher level of teamwork will refer to higher values of $k$ in the k-optimal algorithms that follow in Sections 2.1 and 3.

**Cite as:** When Should There be a “Me” in “Team”? Distributed Multi-Agent Optimization Under Uncertainty, M. E. Taylor, M. Jain, Y. Jin, Makoto Yooko, and M. Tambe, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. XXX-XXX.

(3) explaining the surprising behavior where increased teamwork decreases performance, and (4) presenting algorithms designed to avoid this phenomena. Our hope is that in addition to algorithms for DCEE-like problems, other algorithms will also benefit from this investigation. The primary conclusion of this paper is the somewhat counter-intuitive result that when there is uncertainty, increasing the amount of teamwork may be harmful.

## 2. BACKGROUND

This section defines the DCEE problem class, describes the *mobile wireless network* problem later used as an experimental testbed, and discusses existing DCEE algorithms.

### 2.1 Problem Definition

This section formally defines DCEE domains, first introduced elsewhere [5]. A DCEE consists of a set $V$ of $n$ variables, $\{x_1, x_2, \ldots, x_n\}$, assigned to a set of agents, where each agent controls one (or more) variable's assignment. Agents have at most $T$ *rounds* to modify their variables $x_i$, which can take on any value from the finite domain $D_i$. The goal of such a problem is for agents to choose values for the variables such that the cumulative sum over a set of binary constraints and associated payoff or reward functions, $f_{ij} : D_i \times D_j \rightarrow \Re$, is maximized over time horizon $T \in \mathbb{N}$. More specifically, the agents attempt to pick a set of assignments (one per time step: $A_0, \ldots, A_T$) such that the total reward (the *return*) is maximized:

$$R = \sum_{t=0}^{T} \sum_{x_i, x_j \in V} f_{ij}(d_i, d_j),$$

where $d_i \in D_i, d_j \in D_j$ and $x_i \leftarrow d_i, x_j \leftarrow d_j \in A_k$. Take the constraints in Figure 1 as an example. $x_1$, $x_2$, and $x_3$ are variables, each with a domain and global reward function as shown. If all agents choose the value 0, the total solution quality of this complete assignment on the time step it is chosen is $7 + 15 = 22$.

As in a DCOP, DCEE agents work to maximize the team's reward, the sum of their rewards. Agents in a DCOP are traditionally assumed to have *a priori* knowledge of the reward function. In order to more flexibly model a class of real world domains, DCEE problems do not make this assumption. Thus, DCEE problems appear similar to DCOPs, but with the following features absent from DCOPs: (1) agents initially know the constraint graph but only discover rewards through exploration (i.e., a pair of agents set their values to explicitly discover a reward), (2) problems last a set amount of time, (3) there are more combinations of domain values than can be explored within this time (disallowing exhaustive exploration), and (4) agents seek to maximize the online global reward over this time horizon $T$.

### 2.2 Motivating Domain

Modeling a problem as a DCEE is warranted when a set of agents need to coordinate to maximize their shared reward, but are unable to fully explore the entire space of their possible assignments. Such a setting will most likely occur when there are many agents, a large number of possible assignments, and a limited amount of time. We explicitly do not consider centralized approaches to reduce communication and improve robustness.

The DCEE *mobile wireless network* problem, which we have simulated based on the implementation of Jain *et al.* [5], is used as an example domain throughout this paper. During natural disasters and other ad-hoc situations, personnel may quickly form such a network by placing mobile robots in an area to relay information (e.g., about endangered victims or fires). The robots must optimize the network to ensure reliable and effective communication for the

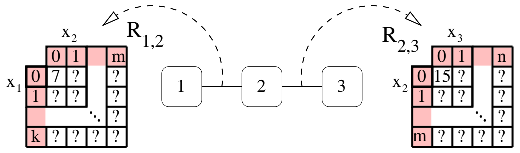


**Figure 1: This figure depicts three DCEE agents. Each agent can select a value, and the instantaneous global reward depends on pairs of agents' selected values. Constraint rewards are only known once they are explored.**

duration of the task. Value settings correspond to agent (robot) locations, constraints are determined by the network topology, and rewards are based on link quality between robots.

Our formulation of the mobile wireless network problem is based on the capabilities of current low-cost physical robots. An agent can measure the link quality between itself and each neighbor, corresponding to measuring the reward for the pair of agent assignments. The time for movement in agents is assumed to dominate communication and calculation time — we measure experiment length by the number of rounds, defined by the period in which every agent may decide to move to a new position and then reach that position. All agents may choose to either stay in their current position or explore. We also consider agents which have the ability to backtrack to a previously explored position. To be consistent with previous work [5], we assume that link qualities are uncorrelated between locations (e.g., when the link quality is determined by signal strength in a multi-path environment) and that the network topology is *static*.

### 2.3 Existing Algorithms

This section briefly discusses five algorithms previously introduced and evaluated on both physical robots and simulated agents [5].

**MGM-Omniscient-1** is based on MGM [9], where the problem's reward matrix is artificially supplied (hence "Omniscient") and therefore represents an upper bound on solution quality for k=1 agents. On each round, each agent (1) communicates its current value to all neighbors (2) calculates and communicates its bid (the expected *gain* in the reward if allowed to change its variable assignment) (3) change its value if it has the highest bid per neighborhood.

The remaining algorithms fall into two broad classes: Static Estimation (SE) and decision theoretic (i.e., Balanced Exploration, or BE). The two static estimation algorithms differ from MGM-Omniscient-1 in that they do not know their reward for unexplored states. **SE-Optimistic-1** is a greedy approach that assumes agents will receive the maximum possible reward for unexplored value settings, while **SE-Mean-1** assumes unexplored variable settings will result in the average reward.

**BE-Rebid-1** is a decision theoretic method that computes the expected reward of executing the explore or backtrack actions, assuming that the probability distribution over rewards is known. Exploring is evaluated by using order statistics while backtracking returns the agent to a known setting (and then allows it to exploit this reward for the remainder of the experiment). **BE-Stay-1** is similar to BE-Rebid-1, but does not allow backtracking. Instead, every agent compares the expected reward it would receive if it kept the same variable assignment (stay) with the expected reward of exploring (explore).

## 3. EXTENDING TO K=2

This section describes our three novel DCEE algorithms. While the previous approaches in Section 2.3 allowed only one agent per neighborhood to select a new value, algorithms in this section follow the framework of MGM-2 [13] and allow two agents in a neighborhood to coordinate a joint variable change, as discussed in Section 1. We implement the following k=2 algorithms so that they may be considered "natural extensions" to existing k=1 DCEE algorithms, where higher values of $k$ imply increased teamwork.

Algorithm 1 describes code executed by each agent on every round when executing any k=2 algorithm. The only variation in different algorithms comes in the functions *getMaxGainAndAssignmentForPair()* and *getMaxGainAndAssignment()*, which depend on the particular estimation technique used, as discussed next.

**MGM-Omniscient-2** extends MGM-Omniscient-1 so that two agents may change value per neighborhood, and is artificially provided the full reward matrix; it may be considered a re-implementation of MGM-2. Given sufficient time, MGM-Omniscient-2 will always discover a locally optimal solution where no combination of one or two neighboring agents can improve the global reward. This algorithm represents an upper bound for k=2 algorithms in that it is very unlikely that any algorithm which requires exploration would be able to surpass the performance of an Omniscient algorithm. On each round, an agent: (1) selects a neighbor and sends an *Offer* message for a joint variable change, based on the maximal gain — Algorithm 1, lines 4–5. (2) for each offer, sends an *Accept* or *Reject* message reflecting the agent's decision to pair with the offering agent. Agents accept the maximum offer (lines 6–11). (3) calculates the *joint gain* of the pair if an offer is accepted, and otherwise calculates the gain of an individual change (lines 12–14). (4) may change its variable setting if it has the highest gain in its neighborhood (lines 18-21). If the agent is part of a pair, a *ConfirmVariableChange* message is sent to the partnering agent. (5) executes the joint / individual assignment (lines 29–30). Like MGM-Omniscient-1, MGM-Omniscient-2 monotonically increases its solution quality [9]. MGM-Omniscient-2 also requires more communication than MGM-Omniscient-1, but reaches higher or similar solution quality [13]. As with other k=2 algorithms, single agents may act alone, subsuming MGM-Omniscient-1.

**SE-Optimistic-2** makes the same assumption as SE-Optimistic-1; any unexplored reward is assumed to be optimal. This algorithm (and those that follow) differ in how *getMaxGainAndAssignmentForPair()* and *getMaxGainAndAssignment()* (lines 4 and 14) calculate utilities for possible assignments. Agents individually expect to gain[2] reward equal to $n \times R_{max} - R_c$ when changing variables, where $n$ is the number of neighbors (i.e., constraints), $R_{max}$ is the maximum possible reward per constraint, and $R_c$ is the agent's current total reward. Agents that successfully pair then bid their joint gain (line 15), equal to the sum of their individual gains with the chosen neighbor, not double-counting the shared constraint, and unpaired agents bid their individual gains.

**SE-Mean-2** is identical to SE-Optimistic-2, but modifies its utilities so that unexplored variables are assumed to return the average reward. Individual agents now expect to gain $n \times R_\mu - R_c$, and compute the gain for pair based on this utility in *getMaxGainAndAssignment()*. Note that although the DCEE problem is an optimization problem (by definition), Mean is a satisficing algorithm: agents only optimize until reaching an per-constraint average of $\mu$.

**BE-Rebid-2** is an algorithm that requires information about the distribution of rewards. While such information is often available (e.g., a robot may collect a number of samples to estimate the distribution), if it is unavailable, BE methods are inapplicable.

[2]Note that the agent's expected gain is what it will receive from a variable change, in expectation. When an agent explores, it may expect to increase reward (a positive gain), but in fact receive a decreased reward (a negative gain).

**Algorithm 1** PSEUDOCODE FOR $k$=2 ALGORITHMS

```
 1: for each neighbor $n$ do
 2:     Send variable assignment and reward matrices to $n$
 3:     Receive variable assignment and reward matrices from $n$
 4: Find maximum gain, $g$, the corresponding neighbor to pair with, $p$, and
    the variable assignment, $a$:
    $g, p, a \leftarrow$ getMaxGainAndAssignmentForPair()
 5: Send Offer to $p$
 6: for all Offer messages received do
 7:     if (agent requesting to pair is $p$) then
 8:         Send Accept to $p$
 9:     else
10:         Send Reject to $p$
11: Receive responses from neighbors, if any
12: if (reject[$p$]) then
13:     $p \leftarrow \emptyset$
14:     Find max gain and preferred assignment (individual update):
        $g, a \leftarrow$ getMaxGainAndAssignment()
15: Send Bid ($g, p$) to all neighbors
16: Receive Bids (gainNeighbors$_n$) from all neighbors
17: $G \leftarrow \max_n$ gainNeighbors$_n$
18: if ($g > G$) then
19:     $bChanging \leftarrow$ True
20:     if ($p \neq \emptyset$) then
21:         Send ConfirmVariableChange to $p$
22: else
23:     $bChanging \leftarrow$ False
24:     if ($p \neq \emptyset$) then
25:         Send ProhibitVariableChange to $p$
26: Receive any messages sent by neighbors
27: if (ProhibitVariableChange[$p$] and $p \neq \emptyset$) then
28:     $bChanging \leftarrow$ False
29: if ($bChanging$) then
30:     UpdateAssignment($a$)
```

BE-Rebid-2 extends the BE-Rebid-1 algorithm by allowing pairs of agents to select coordinated explore, stay, and backtrack actions. We consider four actions: explore-explore, explore-stay, stay-backtrack, and coordinated backtrack.[3] The coordinated backtrack action is unique to this algorithm; it allows two agents to simultaneously backtrack to a previous setting.

First, consider how BE-Rebid-1 determines the value of a single agent backtracking or exploring. The value for backtracking is $V_{back} = R_b t$, where $R_b$ is the reward for the variable setting with the agent's highest discovered reward and $t$ is the number of remaining timesteps. The value for exploring, given that neighbors do not move, is $V_{explore}(R_b, T, n) =$

$$\max_{0 \le t_e \le T} \left\{ t_e \mu(n) + t_s \int_{x > R_b} x Q(x, n, t_e) dx + t_s R_b F(R_b, n)^{t_e} \right\} \quad (1)$$

After the agent sets $t_e$ such that it maximizes this equation, it can bid assuming that it can explore for $t_e$ rounds, and then exploit for $t_s = T - t_e$ rounds. $Q(x, n, t_e)$ gives the probability of $x$ being the maximum reward among the $t_e$ rewards explored (computed via order statistics). $F(x, n)^t$ is the cumulative probability of drawing a sample (from a distribution that takes into account $n$ neighbors) less than or equal to $x$ in all of the $t$ draws. $\mu(n)$ is the average reward over $n$ neighbors. The agent will thus choose to explore or backtrack depending on which action's expected value is larger.

In the k=2 variant, BE-Rebid-2, the gain of explore and backtrack actions are calculated as above. The gain of an action is the difference between the expected reward of the action and $R_c t$, where $R_c$ is the reward of the current variable setting. As in MGM-Omniscient-2, agents offer gains over joint actions to their neigh-

[3]An agent may not change its value if a neighbor executes backtrack: explore-backtrack is an invalid joint action.

bors and then agents attempt to pair with the neighbor which offers the maximum gain. The reward of joint exploration by agents $i$ and $j$, as calculated by agent $j$, is:

$$V_{exp-exp} = V_{explore:i}(R_{b:i}, T, n_i) + V_{explore:j}(R_{b:j}, T, n_j - 1)$$

which calculates the sum of two gains (not double counting the shared constraint). The joint gain of explore-stay is determined by the exploring agent. Similarly, the joint gain of stay-backtrack is calculated by the backtracking agent. Lastly, the gain of coordinated backtrack is the difference between the best joint reward experienced by the two agents and their current reward.

**BE-Stay-2** may be used by agents that cannot backtrack. Similar to Rebid, the reward for stay ($V_{stay}$) is $R_c t$. An agent's reward for exploring can be calculated recursively: $V(\cdot, 0, \cdot) = 0$ and $V(R_c, T, n) = \max(V_{stay}(R_c, T), V_{exp}(R_c, T, n)), T > 0$.

$$V_{explore}(R_c, T, n) = \int_{-\infty}^{\infty} V(x, n)(V(x, T-1) + x)dx$$

where $V(x, n)$ is the expectation of receiving the total reward $x$ when the agent has $n$ neighbors.

A combination of $V_{stay}$ and $V_{explore}$ is used to calculate the joint gains. As in BE-Rebid-2, the gain of joint exploration is calculated as the sum of individual exploration utilities, not double counting the common constraint. The gain of explore-stay is given by the gain of the exploring agent and the gain of stay-stay is 0, by definition. Again, bidding proceeds as in MGM-Omniscient-2 and agents may act individually if no team yields a higher expected reward.

## 4. EXPERIMENTAL ANALYSIS

Experiments in this section compare the performance of our DCEE algorithms with those re-implemented from Jain *et al.* [5] in multiple instances of the mobile wireless network problem. Our custom simulator generates random network topologies and reward matrices, and then sequentially tests algorithms on the same problem instances. Consistent with past work, constraint rewards in the simulations are drawn from a normal distribution with $\mu = 100$ and $\sigma = 16$ over the (truncated) range [0,200]. Experiments run 40 agents for 100 rounds over 30 independent trials.

| Topology | k=1 | k=2 |
|---|---|---|
| Chain | 136,409 | 160,098 |
| Random 1/3 | 363,876 | 386,189 |
| Random 2/3 | 509,008 | 534,400 |
| Full | 600,465 | 655,545 |

**Table 1: Gains of Omniscient Algorithms**

The results in Table 1 lists the gains (i.e., improvement in total reward over no optimization) for Omniscient k=1 and k=2 algorithms on graphs with four different topologies, including random topologies where roughly $\frac{1}{3}$ or $\frac{2}{3}$ of the number of links in a fully connected graph are randomly added to the network. All were chosen to be consistent with those used previously [5]. Recall that the Omniscient algorithm artificially provides reward matrices to the agents; this result shows that increased teamwork is beneficial, as expected from previous DCOP work. Additionally, these results help confirm that our k=2 implementation is correct.

Next, consider Figure 2(a), which shows the performance of DCEE algorithms on the same graph topologies. Lower and upper bounds were determined by disallowing all agent variable changes and using BE-Rebid-1 (the highest performing algorithm previously reported). The results are reported as a scaled gain, where 0.0 corresponds to no optimization, and 1.0 is the gain achieved by BE-Rebid-1. Any gain greater than zero represents an improvement in the total reward. k=2 algorithms typically converge to a higher final reward due to their ability to make joint decisions.

The k=2 algorithms outperform the k=1 algorithms in the majority of situations, except for SE-Optimistic-1 and BE-Rebid-1 on sparse graphs. For instance, SE-Optimistic-1 and BE-Rebid-1 outperform their k=2 counterparts on chain graphs (paired t-tests, $p < 5.3 \times 10^{-7}$), and BE-Rebid-1 outperforms BE-Rebid-2 on Random graphs with $\frac{1}{3}$ of their links (although the difference is not statistically significant).

However, the Rebid and Optimistic are the best performing algorithms, making this behavior particularly troubling. That k=2 does not dominate other approaches is a particularly surprising result precisely because previous DCOP work showed that k=2 algorithms reached higher final rewards [9, 13]. We term this phenomena the team uncertainty penalty. Note that this penalty strictly affects total reward: it does not consider any penalty from increased communication or computational complexity. Supplemental experiments[4] that vary the number of agents on different topologies and vary the experiment lengths all show that k=2 is superior to k=1 variants — the factor most critical to relative performance is the graph topology. To investigate this phenomena, we next considered a set of sparse graphs with random topologies.

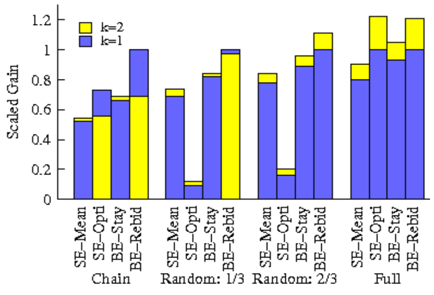
Figure 2(b) compares the relative performance of the k=1 and k=2 variants of Optimistic and Rebid on random topologies. The first trend to notice in Figure 2(b) is that BE-Rebid-1 outperforms BE-Rebid-2 on sparse graphs. The lower the average numbers of neighbors agents have in a graph, the better the k=1 variant will perform. This is in contrast to Omniscient algorithms, where k=2 always outperforms k=1. The second trend to notice is that both Optimistic algorithms perform quite poorly on random graphs, as shown before in Figure 2(a). In Optimistic algorithms, an agent's bid is going to be proportional to the number of neighbors it has; agents with high numbers of neighbors will consistently win bids, blocking others in the neighborhood, as discussed elsewhere [5].

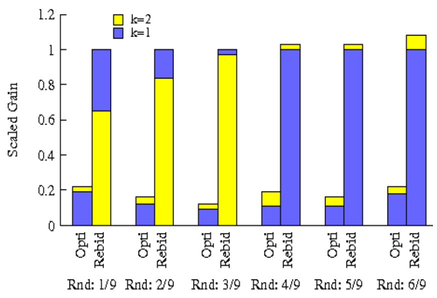
Optimistic algorithms perform poorly on random graphs; to better understand how joint moves affect performance, we generated a series of graphs with regular topology. All agents in such graphs have the same number of neighbors, allowing for clearer analysis of algorithmic performance. Figure 2(c) shows the scaled gain of the two Optimistic and the two Rebid algorithms for different regular graph structures: the x-axis varies the number of neighbors in a regular graph structure; y-axis is the agents' scaled gain. In Figure 2(b), the performance of the k=2 version of Rebid improved relative to the k=1 version as density increased. Figure 2(c) shows that this trend holds for both Rebid and Optimistic in regular graphs. In particular, k=1 outperforms k=2 for both Optimistic and Rebid in graphs with three and five neighbors per agent ($p < 1.4 \times 10^{-4}$). SE-Optimistic-2 outperforms SE-Optimistic-1 on regular graphs with twenty neighbors ($p < 8.6 \times 10^{-4}$) while the two Rebid algorithms do not have statistically significant gain differences.

To further confirm this phenomenon, we implemented SE-Optimistic-3, a k=3 version of our Optimistic algorithm. SE-Optimistic-3 differs primarily from SE-Optimistic-2 in that reward matrices need to be shared between agents at a hop distance of two, whereas SE-Optimistic-2 needs to share information only with neighbors. Pseudocode for SE-Optimistic-3 may be found online.[4]

Figure 3 shows the results from preliminary experiments where 10 agents are run for 50 rounds. In a chain graph, the k=1 version of Optimistic performs better than k=2, which performs better than k=3. In the full graph, SE-Optimistic-3 is better than k=2, which is better than k=1. All differences in the chain graphs are statistically significant ($p < 0.022$), and k=2 algorithms outperform their k=1 counterparts in the full graphs with statistical significance ($p < 0.015$). The k=3 Omniscient algorithm follows the

[4]`http://teamcore.usc.edu/dcop/`


(a) Different Graph Topologies


(b) Different Random Topologies

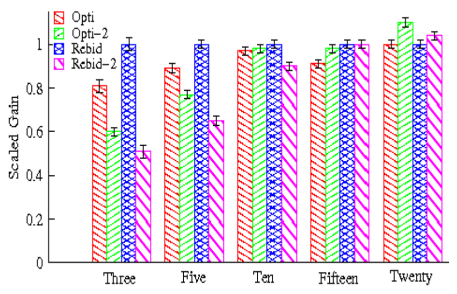

(c) Regular Graph Topologies

**Figure 2: (a) shows how the scaled cumulative gain for algorithms differ per network topologies, in (b) BE-Rebid-2 outperforms BE-Rebid-1 as graph density increases, and (c) shows relative performance of k=1 and k=2 change with the number of neighbors where the labels on the y-axis show the number of neighbors each agent has in the regular graph topology.**

trends seen earlier in Table 1 by dominating MGM-Omniscient-2, which in turn dominates MGM-Omniscient-1 ($p < 4.0 \times 10^{-4}$ for Omniscient graphs). These results confirm that higher amounts of teamwork always improve agent performance in Omniscient algorithms, but may decrease performance in non-Omniscient algorithms.

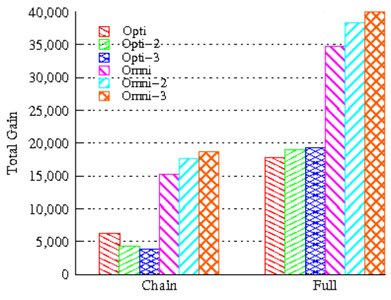


**Figure 3: SE-Optimistic-3 is worse than k=1 and k=2 on chain graphs, but is better on full graphs.**

Most previous work in teamwork and joint action, including previous results in $k$-optimal algorithms, caused us to expect that increasing the level of teamwork in decision making would lead to improved final solution quality in our results. In direct contradiction with these previous expectations, we have shown that in DCEE problems, blindly increasing the number of agents that can execute a joint action may actually decrease the final solution quality. We have been able to isolate situations where this phenomenon occurs — in graphs with low density, k=2 algorithms can perform worse than k=1 algorithms. The next section explains the origin of this team uncertainty penalty with the ultimate goal of improving our algorithms so that teamwork does not decrease performance.

## 5. UNDERSTANDING TEAM UNCERTAINTY

This section presents empirical justification for the team uncertainty penalty, as well as motivating how algorithms may be enhanced to decrease such a penalty.

### 5.1 Relative Performance of k=1 and k=2

As discussed above, the primary deciding factor in the performance of SE-Optimistic-2 and BE-Rebid-2, relative to their k=1 counterparts, is the average number of neighbors. We expected that k=2 algorithms would allow more constraints to change (i.e. change associated variable assignments), because pairs of agents can coordinate joint moves. To better understand the team uncertainty penalty, we first hypothesized that our k=2 algorithms may fail to optimize more constraints than the k=1 algorithms. Figure 4(a) shows the performance of four algorithms on different regular graphs. The x-axis displays the graph density (i.e., number of neighbors) and the y-axis shows the average number of constraints changed per round. This graph shows that k=2 does perform as expected, changing more constraints than k=1.

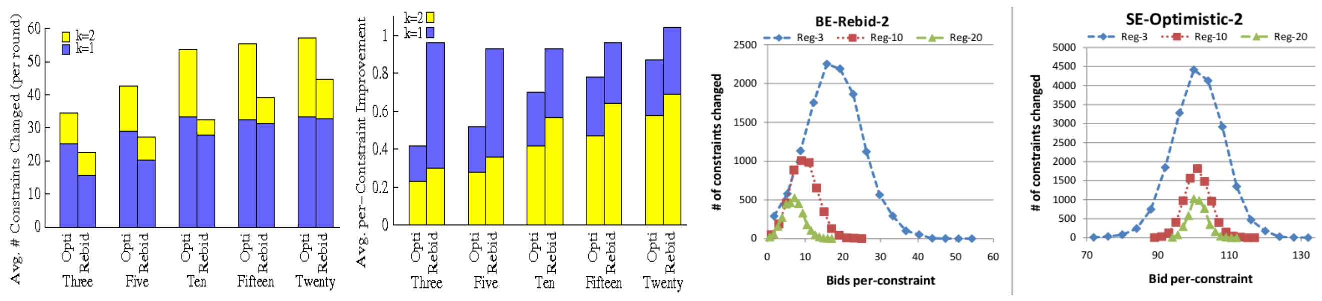
Given that k=2 changes more constraints, we next considered that k=2 changes could be less "valuable." Figure 4(b) displays the same data, but now the y-axis shows the average immediate reward improvement for all agents that change variables during the experiment, normalized by the number of constraints. For instance, in ten trials of 100 rounds, SE-Optimistic-2 agents on a density three graph changed a total of 59,410 constraints (Figure 4(a)). On average, each variable change resulted in an immediate improvement to a constraint's reward by 0.41 (Figure 4(b)).

The two important trends to note are that (1) both k=2 algorithms increased the per-constraint improvement as the density of the graphs increase, and (2) k=1 algorithms have a higher per-constraint improvement than their k=2 counterparts in every case. The k=2 performance trend will be examined in detail in Section 5.2. The trends in Figures 4(b) and 4(a) combine to explain the relative algorithmic performance of k=1 and k=2 algorithms. k=1 algorithms consistently receive a higher improvement per constraint, relative to k=2. However, k=1 algorithms affect fewer constraints than k=2 on average. As the density increases, the difference in the per-constraint improvement in k=1 and k=2 *decreases*. In contrast, the difference between the number of constraints changing in k=1 and k=2 *increases* as density increases. Thus, as the graph density increases, k=2 is able to overcome k=1.

### 5.2 Few Neighbors Curtails k=2 Performance

This section analyzes why SE-Optimistic-2 and BE-Rebid-2 provide lower average reward improvement in graphs with few neighbors per agent (Figure 4(b)). To begin, consider Figures 5(a) and 5(b). Both graphs show the average performance on regular graphs with different densities. The x-axis shows the agents' bids, normalized per-constraint, of all agents that win the bid to change variables. The y-axis plots the gain, again normalized per-constraint. Only pairs of agents are considered (the few single agent actions executed by these k=2 algorithms are ignored). Each of the trials contain thousands of points; points are binned together along the x-axis and vertical error bars show the standard error. Both figures show that when the algorithms have low bids, they *decrease* the improvement per-constraint.

It is clear in Figure 5(a) that changing variables because of low bids in low density graphs cause more harm than in high density graphs. This helps to explain the behavior in Figure 4(b) — as the graph density increases, BE-Rebid-2 improves relative to BE-Rebid-1. Figure 4(c) shows the distribution of bids in BE-Rebid-2, confirming that all different density graphs considered have a substantial number of low bids. Figure 5(b) shows a similar phenom-



(a) Avg. Constraints Changed (b) Avg. Reward Improvement (c) Bid Histograms

**Figure 4: The average number of constraints changed per round in (a) show that in regular graphs, k=2 algorithms always change more constraints (i.e. associated variable assignments). (b) shows average per-constraint improvement for different regular graphs: k=1 algorithms have higher improvement. Histograms in (c) show how often algorithms allow agents to change variables on regular graphs with different densities.**

ena for SE-Optimistic-2, although it is harder to see. In low density graphs, Optimistic agents have a wider range of possible bids than in a high density graph.[5] This allows agents in a low density graph to make larger "mistakes" than agents in a high density graph. Figure 4(c) shows that SE-Optimistic-2 bids follow a roughly normal frequency distribution for each graph density and that lower density graphs produce more bids that are less than 100 (resulting in an average negative gain).

The experiments in these sections show that when bids are low, agents are more likely to receive a negative gain. Furthermore, as the graph density decreases, agents make lower bids, risking larger negative gains (relative to high density graphs). This behavior suggests that both BE-Rebid-2 and SE-Optimistic-2 can be improved by changing the bidding behavior — it appears that the algorithms are overly aggressive, and prohibiting them from changing constraints that have relatively low bids may increase their achieved gains. These insights will be leveraged in Section 6 to improve algorithmic performance, ameliorating the team uncertainty penalty.

## 5.3 In Contrast: Mean and Stay

Recall that in Figure 2(b) SE-Mean-2 and BE-Stay-2 outperform their k=1 counterparts — these two algorithms do not exhibit the team uncertainty penalty. The primary reason is that they are all very *conservative* algorithms in their bidding. Mean agents assume that a changed variable will only return the mean link value, causing it to drastically lower its bids relative to Optimistic. Likewise, Stay does not allow agents to backtrack, forcing the agents must be more cautious than Rebid. If an agent reaches a high valued location in Rebid, it may continue exploring because it can return to the position in the future, whereas if a Stay agent explores, it can never return to a previous location.

Regular Graph: Density 5

| Algorithm | Constraints Changed | Per-constraint Improvement |
|---|---|---|
| Opti-1 | 28.9 | 0.5 |
| Opti-2 | 42.6 | 0.3 |
| Mean-1 | 0.8 | 6.6 |
| Mean-2 | 0.9 | 6.4 |
| Rebid-1 | 20.1 | 0.9 |
| Rebid-2 | 27.2 | 0.4 |
| Stay-1 | 1.3 | 5.1 |
| Stay-2 | 1.4 | 3.6 |

**Table 2: Both Mean and Stay algorithms change fewer constraints and receive higher gain per constraint change than Optimistic and Rebid.**

Both k=1 and k=2 variants of Mean and Stay change variables much less frequently than Optimistic and Rebid. Additionally, this results in a higher per-constraint improvement than SE-Optimistic-2 and BE-Rebid-2. Table 2 gives an example comparison of the algorithms' performance on a regular graph of density five. Optimistic and Rebid algorithms change one to two orders of magnitude more constraints than Mean and Stay (per round). The cautious nature of Mean and Stay avoids the team uncertainty penalty, but at the expense of overall decreased total on-line reward (relative to Optimistic and Rebid, shown in Figure 2(b)). Each variable change in Mean and Stay is relatively good, but so few changes are made, performance suffers (relative to Optimistic and Rebid).

[5]The (per agent) average link value has a low variance in high density graphs due to many values being averaged, while the reverse is true for low density graphs.

## 6. DCEE ALGORITHM EXTENSIONS

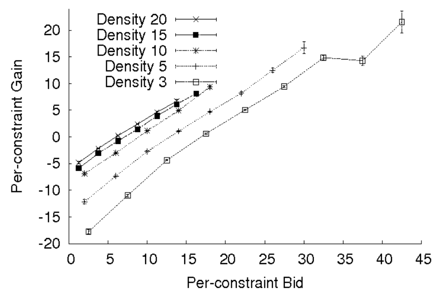
This section revisits the SE-Optimistic-2 and BE-Rebid-2 algorithms to ameliorate the team uncertainty penalty. In principle, one could evaluate a graph and *a priori* decide whether a k=1 or k=2 is likely to be superior, based on the graph density. A more robust solution is to design a k=2 algorithm which can also perform well at low densities, potentially even outperforming the existing k=2 algorithms. The algorithms introduced here empirically demonstrate the soundness of the arguments in the previous section and show that the utility of teamwork can be *improved* by accounting for the team uncertainty penalty. The first method uses a threshold to determine when joint actions are allowed. The second method discounts actions in unknown parts of the reward matrix; teamwork is always useful when rewards are known, *but may be harmful to agents when exploring*.

### 6.1 Discouraging Joint Actions

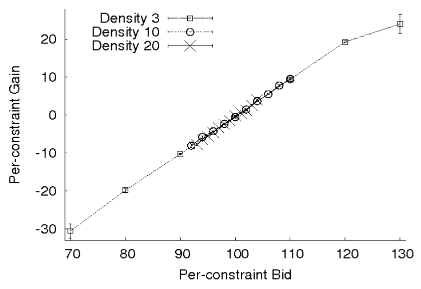
The first solution to decrease the team uncertainty penalty is to discourage joint actions with low bids. Recall that in Figures 5(a) and 5(b), pairs of agents that made low bids (and won the right to change variables) were much more likely to receive negative gains than those that made high bids. In this section, we consider using a threshold parameter: if a pair of agents do not bid to improve by at least $t$ units of reward per constraint in the next round, the algorithm disallows the joint action and reverts to k=1. Put differently, this method uses a parameter to decide when to use teamwork and when agents should act alone.

**SE-Threshold-2** is identical to SE-Optimistic-2, except that agents are allowed to form a pair only if their bid will be at least $t\times$*(number of agents that neighbor the pair)*. **BE-Threshold-2** is similar: pairs may form only if the pair has a bid above $t\times$*(number of agents that neighbor the pair)*. Agents only execute joint actions when their bids are quite high (i.e., there should be a significant advantage to the joint move) and otherwise "play it safe" with k=1.

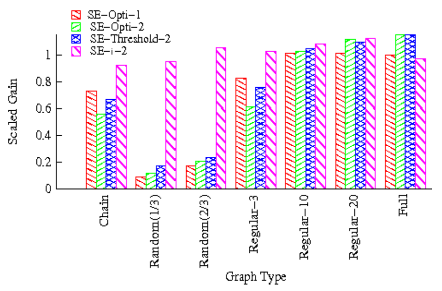
To test these two algorithms, we first ran preliminary experiments testing out roughly ten different values of $t$ for both. Fig-



(a) Rebid Gains



(b) Optimistic Gains



(c) Improved SE Algorithms

**Figure 5: Graphs (a) and (b) shows the per-constraint gain per per-constraint bid for pairs that change variables in BE-Rebid-2 and SE-Optimistic-2 on different regular graph densities. (c) shows SE-Threshold-2 with $t = 100$ and SE-i-2 with $i = 110$.**

ures 5(c) and 6(a) shows the performance of SE-Threshold-2 and BE-Threshold-2, respectively. In these two graphs, both algorithms use a single value of $t$, confirming that performance can be improved by discouraging teamwork. SE-Threshold-2 outperforms SE-Optimistic-2 on very low density graphs (for instance, SE-Threshold-2 outperforms SE-Optimistic-2, $p < 3.4 \times 10^{-3}$, but underperforms SE-Optimistic-1, $p < 8.0 \times 10^{-2}$), and dominates SE-Optimistic-1 on high density graphs. Likewise, Rebid-Threshold-2 outperforms BE-Rebid-2 on low density graphs and outperforms BE-Rebid-1 on high density graphs. Using a threshold reduces the team uncertainty penalty in both cases.

A second set of results tested how the algorithms performed on a single graph type with different parameter settings. Figure 6(b) shows the results of BE-Threshold-2 on regular graphs of varying densities. The y-axis shows the net gain on different graph types. The points at the far left represent the performance of BE-Rebid-1, the points at the far right BE-Rebid-2, and the points connected by lines show the performance of BE-Threshold-2 for different values of $t$ (shown on the x-axis). As expected, different thresholds maximize performance for different graphs. An important open question is whether these parameters can be automatically tuned. A fixed threshold setting shows substantial improvements, but even higher gains could be achieved if the algorithmic parameters can be set automatically per graph, or even per agent.

## 6.2 Discounting All Bids Under Uncertainty

The second solution is to discount *all* bids. Both SE-Optimistic-2 and BE-Rebid-2 receive negative average reward with low bids. Reducing all exploration bids via a per-constraint discount discourages agents from changing variables when they have low bids, resulting in algorithms that are less aggressive. Put differently, our results show that joint actions are more likely to make mistakes than single agent actions in the presence of uncertainty. This parameter reduces the value of exploration, encouraging exploitation.

**SE-i-2** is similar to both SE-Mean-2 and SE-Optimistic-2 in that all unexplored variable assignments are assumed to receive a reward per link of $i$ (where $i$=$\mu$ for Mean and $i$=*MAX* for Optimistic), effectively controlling the calculated utility of exploration. While Optimistic agents change their values throughout all trials, SE-i-2 agents will stop optimizing once all agents have a reward of at least $i \times$*numNeighbors*. **BE-i-2** generalizes the BE-Rebid-2 algorithm so that all utilities for the explore action are discounted, proportional to the number of constraints that would be changed. This shift in utility discourages agents from being overly Optimistic with their bids and encourages agents to exploit (i.e., stay and/or backtrack), rather than explore.

Figure 5(c) demonstrates that SE-i-2 generally outperforms all other algorithms, regardless of density. Particularly impressive is the performance on chain and random graphs, substantially outperforming the Optimistic algorithms as SE-Threshold-2. The discount factor allows agents with high degree to more easily decline to move such that they do not dominate their neighbors (as discussed earlier in Section 4). SE-i-2 is the highest performing SE algorithm, with the exception of full graphs. The performance of BE-i-2 in Figure 6(a) demonstrates BE-i-2 is similar to BE-Threshold-2, in that it outperforms BE-Rebid-2 on sparse graphs and outperforms BE-Rebid-1 on dense graphs. These results suggest that BE-Threshold-2 is the best k=2 BE algorithm, in that it more often outperforms BE-i-2 than not, and avoids the team uncertainty penalty (unlike BE-Rebid-2). Note that BE-Rebid-2 outperforms both BE-Threshold-2 and BE-i-2 on the full graph; it is not surprising that the most aggressive algorithm (BE-Rebid-2) does well on this high density graph where 40 agents are fully connected.

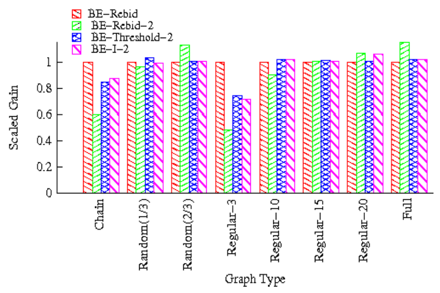
Figure 6(c) is analogous to Figure 6(b): it shows how the performance of SE-i-2 changes on a single graph type as the parameter $i$ is varied. As before with BE-Threshold-2, the parameter value that produces maximal in SE-i-2 gain depends on the graph type. If a single parameter is used, SE-i-2 is a significant improvement over SE-Optimistic-2. If multiple parameters may be tuned, or set automatically, performance can be increased still further.

This section has presented novel algorithms that explicitly account for the team uncertainty penalty. In particular, the SE-i-2 algorithm worked surprisingly well, outperforming all other SE algorithms, with the exception of full graphs. These algorithms show that both reducing the calculated utility of joint moves and reducing the utility of acting under uncertainty improve performance. This represents an important confirmation that the team uncertainty penalty can be reduced, if not avoided, by explicit consideration during an algorithm's design.

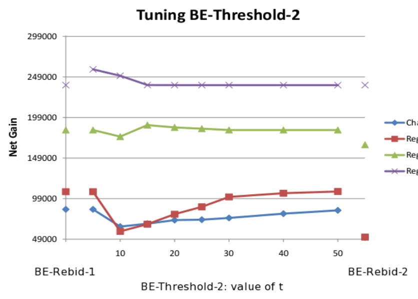
## 7. RELATED WORK

In multi-agent work, teamwork is typically considered beneficial, although if too much is shared, computational requirements will explode [3]. To our knowledge, this is the first work to show that increasing the amount of teamwork may actually be harmful, without counting communication or computation cost. For example, previous work [6] has focused on "level of cooperativeness" in DisCSPs (distributed constraint satisfaction), as well as on the size of a mediation group to select asynchronous partial overlay DisCSP algorithms [1]. While these results show that increased cooperativeness may not always improve performance, the focus of that research was run-time performance rather than solution quality, which is the exclusive focus of our work.

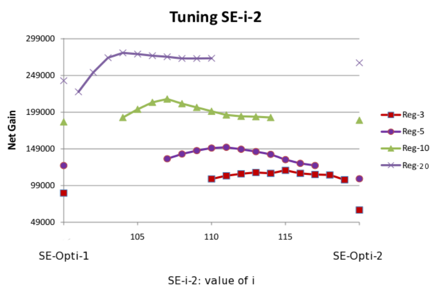
Zhang *et al.* [19] analyze the distributed stochastic search algorithm (DSA) and show that it often performs better than the distributed breakout algorithm (DBA) along multiple dimensions, en-



(a) New BE Algorithms



(b) Tuning BE-Threshold-2



(c) Tuning SE-i-2

**Figure 6: (a) shows the performance of BE-Threshold-2 using $t = 15$ and SE-i-2 using $i = 4$. (b) shows how the performance of BE-Threshold-2 change as its parameters are tuned (BE-Rebid-1 is at the far left and BE-Rebid-2 is at the far right, for comparison). (c) shows the performance of SE-i-2 as $i$ is tuned, in comparison to SE-Optimistic-1 (left) and SE-Optimistic-2 (right).**

couraging many researchers to extend the DSA framework (c.f., Zivan [20]). Because MGM is similar to DBA, we also investigated using DSA as a base for algorithms in the DCEE problems, although DSA requires tuning of a real-valued parameter that affects its probabilistic update rule. However, our results (omitted for brevity) showed that our MGM-based algorithms often outperformed DSA, and substantially outperformed DSA algorithms if its free parameter was not carefully tuned.

Other researchers have examined lower-level concerns in mobile wireless networks (c.f., the LANdroids[6] project) rather than explicit multi-agent coordination issues. Marder *et al.* [11] formulate dynamic sensor coverage as a "potential game," similar to a DCOP. Stranders *et al.* [16] use the max-sum algorithm to coordinate mobile sensor movement over a small grid. Kok and Vlassis [7] use a reinforcement learning approach that applies to multi-agent tasks with coordination graphs.

While single-agent approaches to exploration vs. exploitation may provide insight into DCEE problems (c.f., multi-armed bandits [15]), coordinated exploration is significantly more difficult because rewards depend on pairs of agents: agent coordination is critical. Distributed POMDPs [2] are also not directly relevant: DEC-POMDPs plan over uncertainty in agent action outcomes, whereas DCEE actions have known outcomes but uncertain rewards.

## 8. CONCLUSION

A major contribution of this paper is establishing that increased teamwork under uncertainty may sometimes degrade performance. Additionally, it has presented and empirically tested a set of k=2 algorithms, investigated the team uncertainty penalty in the context of DCEE problems, and presented a second set of improved algorithms that reduce the penalty's impact. The two types of algorithmic solutions, disallowing low valued joint actions and discounting the utility of actions under uncertainty, are very general. In the future, we plan to investigate how well these teamwork ideas generalize to, and how the team uncertainty penalty manifests in, other settings, such as multi-agent reinforcement learning. Additionally, we intend to explore how to best incorporate prior knowledge about the reward function, and to attempt to more formally quantify the amounts of teamwork used by different algorithms in DCEE domains.

## Acknowledgements

We would like to thank W. Bradley Knox, Chris Kiekintveld, James Pita, Atul Kumar, Scott Alfeld, and the anonymous reviewers for helpful comments and suggestions. This work was supported in part by Perceptronics Solutions.

[6]`www.darpa.mil/ipto/programs/ld/ld.asp`

## 9. REFERENCES

[1] M. Benisch and N. Sadeh. Examining DCSP coordination tradeoffs. In *AAMAS*, 2006.

[2] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov decision processes. In *UAI*, 2000.

[3] E. H. Durfee. Blissful ignorance: Knowing just enough to coordinate well. In *AAAI*, 1995.

[4] B. J. Grosz and C. L. Sidner. Plans for discourse. In P. R. Cogent, J. Morgan, and M. Pollack, editors, *Intentions in Communication*. MIT Press, 1990.

[5] M. Jain, M. E. Taylor, M. Yokoo, and M. Tambe. DCOPs meet the real world: Exploring unknown reward matrices with applications to mobile sensor networks. In *IJCAI*, 2009.

[6] H. Jung and M. Tambe. Argumentation as distributed constraint satisfaction: Applications and results. In *AAMAS*, 2001.

[7] J. R. Kok and N. Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *JMLR*, 7:1789–1828, 2006.

[8] H. J. Levesque, P. R. Cohen, and J. Nunes. On acting together. In *AAAI*, 1990.

[9] R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical-game-based approach. In *PDCS*, 2004.

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

[11] J. Marden, G. Arslan, and J. Shamma. Connections between cooperative control and potential games illustrated on the consensus problem. In *European Control Conference*, 2007.

[12] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *AIJ*, 161:149–180, 2005.

[13] J. P. Pearce, M. Tambe, and R. Maheswaran. Solving multiagent networks using distributed constraint optimization. *AI Magazine*, 29(3):47–66, 2008.

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

[15] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58:527–535, 1952.

[16] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. Decentralized coordination of mobile sensors using the max-sum algorithm. In *IJCAI*, 2009.

[17] M. Tambe. Towards flexible teamwork. *JAIR*, 7:83–124, 1997.

[18] Y. Xu, P. Scerri, B. Xu, S. Okamato, M. Lewis, and K. Sycara. An integrated token-based algorithm for scalable coordination. In *AAMAS*, 2005.

[19] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problem in sensor networks. *AIJ*, 161:55–87, 2005.

[20] R. Zivan. Anytime local search for distributed constraint optimization. In *AAAI*, 2008.