Advances in Complex Systems

Distributed On-line Multi-Agent Optimization Under Uncertainty: Balancing Exploration and Exploitation

Matthew E. Taylor[1], Manish Jain[2], Prateek Tandon[3], Makoto Yokoo[4], and Milind Tambe[2]

[1] *Computer Science Department, Lafayette College*
Easton, PA 1804, USA
taylorm@cs.lafayette.edu

[2] *Computer Science Department, The University of Southern California*
Los Angeles, CA 90089, USA
{*manishja, tambe*}*@usc.edu*

[3] *Robotics Institute, Carnegie Mellon University*
Pittsburgh, PA 15213, USA
prateekt@andrew.cmu.edu

[4] *Department of Informatics, Kyushu University*
Fukuoka, 819-0395, Japan
yokoo@inf.kyushu-u.ac.jp

Received (received date)
Revised (revised date)

A significant body of work exists on effectively allowing multiple agents to coordinate to achieve a shared goal. In particular, a growing body of work in the Distributed Constraint Optimization (DCOP) framework enables such coordination with different amounts of teamwork. Such algorithms can implicitly or explicitly trade-off improved solution quality with increased communication and computation requirements. However, the DCOP framework is limited to planning problems; DCOP agents must have complete and accurate knowledge about the reward function at plan time.

We extend the DCOP framework, defining the *Distributed Coordination of Exploration and Exploitation* (DCEE) problem class to address real-world problems, such as ad-hoc wireless network optimization, via multiple novel algorithms. DCEE algorithms differ from DCOP algorithms in that they (1) are limited to a finite number of actions in a single trial, (2) attempt to maximize the on-line, rather than final, reward, (3) are unable to exhaustively explore all possible actions, and (4) may have knowledge about the distribution of rewards in the environment, but not the rewards themselves. Thus, a DCEE problem is not a type of planning problem, as DCEE algorithms must carefully balance and coordinate multiple agents' exploration and exploitation.

Two classes of algorithms are introduced: *static estimation* algorithms perform simple calculations that allow agents to either stay or explore, and *balanced exploration* algorithms use knowledge about the distribution of the rewards and the time remaining in an experiment to decide whether to stay, explore, or (in some algorithms) backtrack to a previous location. These two classes of DCEE algorithms are compared in simulation and on physical robots in a complex mobile ad-hoc wireless network setting. Contrary

to our expectations, we found that increasing teamwork in DCEE algorithms may *lower* team performance. In contrast, agents running DCOP algorithms improve their reward as teamwork increases. We term this previously unknown phenomenon the *team uncertainty penalty*, analyze it in both simulation and on robots, and present techniques to ameliorate the penalty.

Keywords: Cooperative multi-agent systems, Exploration, Optimization, Distributed Constraint Optimization (DCOP), Distributed Coordination of Exploration and Exploitation (DCEE)

1. Introduction

As physical and virtual agent technology gains in popularity, robust *teamwork* reasoning becomes increasingly important. Consider, for instance, a team of robots that must collaborate to achieve some goal. At one extreme, all agents could pool their observations on every timestep and the leader could calculate and communicate an action for each agent. At the other extreme, every agent could act independently, taking only its own observations and action into account, ignoring other agents' knowledge. This article investigates an intermediate solution: agents reason in a distributed, rather than centralized, manner and agents make decisions based on a subset of the team's observations.

One currently popular approach to addressing such distributed problems is to frame tasks within the *Distributed Constraint Optimization Problem* (DCOP) framework [52, 26, 30, 15, 54, 5]. In a DCOP, cooperative agents coordinate to maximize a team reward. Examples include multiagent plan coordination [8], sensor networks [23], and directing autonomous vehicles to survey underwater structures [53]. Because the utility of an agent's action depends on the actions of others, agents must coordinate their individual actions to achieve joint goals and different algorithms may use different levels of teamwork.[a] One potentially significant shortcoming of the DCOP framework is that it assumes the team's reward function is fully and perfectly known – DCOP agents plan off-line what actions to take. However, in many real-world multi-agent domains, a full model of the system is not known *a priori* and must instead be learned (e.g., any time the full model is unknown, such as when using ad-hoc teams for disaster response [39, 44], in multi-agent exploration scenarios [28, 46], or for multi-agent security [38, 1] against an unknown adversary).

This article presents the *Distributed Coordination of Exploration and Exploitation* (DCEE) framework, significantly extending the DCOP framework, so that agents can balance *exploration* with *exploitation* in a cooperative multi-agent sys-

[a]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 coordination 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 and k-movement algorithm frameworks, as discussed in Sections 2.1 and 2.2.

tem. Unlike DCOPs, in a DCEE,

(1) there is a fixed number of actions agents can execute within a single trial,
(2) the agents attempt to maximize the on-line, cumulative reward within this fixed number of actions,
(3) the cross product of all of its possible actions cannot be fully explored within a given trial (in fact, a single agent may not even be able to explore all possible actions, such as when the action space is continuous), and
(4) agents may have prior knowledge about the distribution of possible rewards, but must explore the environment to sample the reward function.

These challenges disallow direct application of current DCOP algorithms as they implicitly assume knowledge of the full payoff matrix. Furthermore, time constraints disallow using a globally optimal algorithm as agents cannot fully explore their environment. This paper introduces and analyzes a family of novel DCEE algorithms to address these challenges, based on both decision theoretic exploration strategies and simpler "static estimation" strategies. These algorithms are based on two key ideas: (1) exploration and exploitation are interleaved and (2) different settings may require different exploration strategies.

A significant body of work in multiagent systems over more than two decades has focused on teamwork [16, 24, 43]. If cooperative agents can efficiently make joint decisions, so the common wisdom goes, the team as a whole will only benefit [48] — provided the increased coordination and communication overheads do not overwhelm the agents. This has been shown multiple times in the DCOP framework [25, 34]: as agents share more information and collaboratively calculate joint actions, the team reward increases at the cost of higher communication and computation. In the DCEE framework, we have discovered that this is not always the case; in some algorithms and in some situations, increasing the amount of teamwork can actually decrease the reward obtained by the team. Investigations in both simulation and on robots show the presence of the *team uncertainty penalty*, where joint decisions by a team of agents acting under uncertain conditions can lead to a significant degradation in team performance, relative to agents acting with less coordination. Contrary to popular wisdom, the problem is not the cost of increased communication or computation in service of the joint decision; the problem stems from the joint decision itself in the presence of uncertainty.

This article has several contributions,[b] including:

[b]This article summarizes and extends a pair of papers presented at the IJCAI [18] and AAMAS [45] conferences. The most significant additions and extensions are (1) a more detailed explanations of the DCEE algorithms, (2) additional robotic experiments, (3) a comparison to the DSA algorithm, (4) the release of our simulator and robotic source code, (5) a significantly expanded investigation of the team uncertainty penalty, (6) additional discussion of algorithms that reduce the team uncertainty penalty, including the novel SE-i-1 and BE-i-1 algorithms, and (7) expanded related and future work sections.

- formulating the DCEE problem,
- introducing two classes of algorithms designed to maximize the team reward under different amounts of domain knowledge and agent capabilities,
- incorporating different levels of teamwork into these algorithms,
- empirically analyzing the trade-offs of these algorithms in both simulation and on physical robots,
- presenting empirical investigations of the team uncertainty penalty, and
- introducing and empirically evaluating additional algorithms designed to ameliorate the penalty.

The remainder of the article is organized as follows. Section 2 provides background information on DCOP algorithms, presents the DCEE formulation, and introduces the motivating domain for this article in both a simulated and physical environment. Section 3 presents an artificial "omniscient" algorithm and two sets of algorithms for DCEE agents. Both the *static estimation* and *balanced exploration* algorithms are designed to be compatible with different amounts of teamwork (i.e., amounts of partial centralization). Results in the simulated and physical domains are detailed in Sections 4 and 5, including those that clearly demonstrate the team uncertainty penalty. Section 6 introduces two algorithmic modifications to ameliorate the penalty in static estimation and balanced exploration algorithms. Related work is discussed in Section 7. Section 8 discusses future work and concludes the article.

2. Problem Description

This section of the article first presents background on DCOPs and then Section 2.2 details the DCEE problem formulation. Section 2.3 defines the mobile ad-hoc wireless network optimization task in both simulation and for physical robots, which is used as a testbed for DCEE algorithms.

2.1. *DCOP*

In DCOP problems, cooperative agents attempt to maximize a global reward function by changing local variables to optimize constraints, which may be considered an extension of the older *distributed constraint satisfaction* problem [49, 50]. Variable settings are only observable by the agent that controls the variable and constraints are only observed by the agents sharing the constraint.

More formally, a DCOP consists of a set V of n variables, $\{x_1, x_2, \ldots, x_n\}$, assigned to a set of agents. We assume that each agent controls one variable's assignment for simplicity of exposition, but an agent may control more than one variable in the general case. Variable x_i can take on any value from the discrete finite domain D_i. The goal is to choose values for the variables such that the sum over a set of constraints[c] and associated payoff or reward functions, $f_{ij} : D_i \times D_j \rightarrow \Re$, is

[c]For exposition purposes, we will assume that the constraints are binary, but n-ary constraints are

maximized. Specifically the goal is for the team to find an assignment of variables, A, such that $F(A)$ is maximized.

$$F(A) = \sum_{x_i, x_j \in V} f_{ij}(d_i, d_j), \text{ where } d_i \in D_i,\ d_j \in D_j, \text{ and } (x_i \leftarrow d_i,\ x_j \leftarrow d_j) \in A.$$

For example, in Figure 1, x_1, x_2, and x_3 are variables, each with a domain of {0,1}, and the reward function is as shown. If agents 2 and 3 choose the value 1, the agent pair receives a reward of 9. If agent 1 chooses value 1 as well, the total solution quality of this assignment is 12, which is locally optimal as no single agent can change its value to improve its own reward (and that of the entire DCOP). $F((x_1 \leftarrow 0), (x_2 \leftarrow 0), (x_3 \leftarrow 0)) = 22$ and is globally optimal.

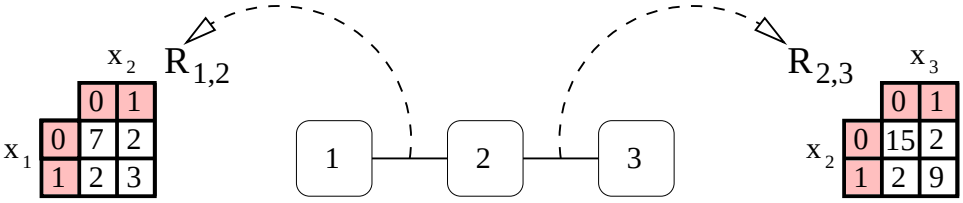


Fig. 1: This figure depicts a three-agent DCOP. Each agent can select a value for its variable and the instantaneous global reward depends on pairs of agents' selected values.

Significant progress has been made in the design and analysis of globally optimal DCOP algorithms [26, 30, 35, 27]. However, given that DCOPs are NP-Hard [30], "complete" algorithms that find optimal solutions require significant communication and computation overhead, motivating the need for "incomplete" algorithms that find locally optimal configurations. Such locally optimal solutions enable DCOPs to scale to significantly larger tasks [52, 33].

While there are multiple classes of incomplete algorithms (c.f., Vinyals et al. [47]), we focus on the class termed *k-optimal* [25, 34]. *k-optimal* algorithms allow 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, eventually reaching a 2-optimal solution. In Figure 1, $F((x_1 \leftarrow 1), (x_2 \leftarrow 1), (x_3 \leftarrow 1)) = 12$ is a 1-optimal solution because no one agent can change its value and improve the team reward. However, it is not a 2-optimal solution because two agents can change their values and improve the total team reward to $F((x_1 \leftarrow 1), (x_2 \leftarrow 0), (x_3 \leftarrow 0))$ $= 17$.

Algorithms that use higher values of k, increasing the amount of partial centralization, have been proven to improve the bound on the minimum possible solution.

also possible.

In practice, increasing k has been shown to effectively increase the team reward. In the extreme case when k is set equal to the number of agents in the DCOP, the algorithm essentially becomes centralized and will find the global optimum. However, increasing teamwork does require additional computation and communication resources [25, 34].

2.2. *DCEE*

This section describes the *Distributed Coordination of Exploration and Exploitation* (DCEE) framework, extending DCOPs along multiple dimensions. Specific DCEE algorithms will be discussed later in Section 3.

DCEE algorithms must differ from DCOP algorithms in three novel ways as DCEE agents:

- do not know their reward functions,
- are unable to exhaustively explore the environment (which is very large or even infinite), and
- seek to maximize their on-line (total) reward within a fixed amount of time.

These challenges disallow direct application of current DCOP algorithms as they implicitly assume knowledge of the full payoff matrix. Furthermore, time constraints disallow using a globally optimal algorithm as agents cannot fully explore their environment. Examples of real world problems which may have the above characteristics include multi-agent exploration scenarios [28, 46], disaster response [39, 44], multi-agent security [38, 1] against an unknown adversary, network routing optimization [51], and sensor network optimization [41].

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, in the general case, more) variable's assignment. Agents have at most T *rounds* to modify their variables x_i, which can take on any value from the 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 $T \in \mathbb{N}$. The agents attempt to pick a set of assignments (one per time step: $A_0, \ldots, A_T$) such that the total reward (i.e., the return) is maximized:

$$R = \sum_{t=0}^{T} \sum_{x_i, x_j \in V} f_{i,j}(d_{i,t}, d_{j,t}), \text{where } d_{i,t} \in D_i, d_{j,t} \in D_j, \text{and } (x_i \leftarrow d_{i,t}, x_j \leftarrow d_{j,t}) \in A_t.$$

An agent knows its own action, but only knows the actions of its neighbors through communication. Once an has explored a particular setting, it knows the reward for each of its reward functions for the current setting (assuming its neighbors have communicated their previous actions). Additionally, we assume that the team reward is easily measurable, although individual agents do not need this information.

Take the constraints in Figure 2 as an example. x_1, x_2, and x_3 are variables, each with a domain and global reward function as shown (which has been partially explored). If all agents choose the value 0, the total solution quality of this complete assignment on the time step it is chosen is $7 + 15 = 22$. Unexplored rewards are denoted with question marks.

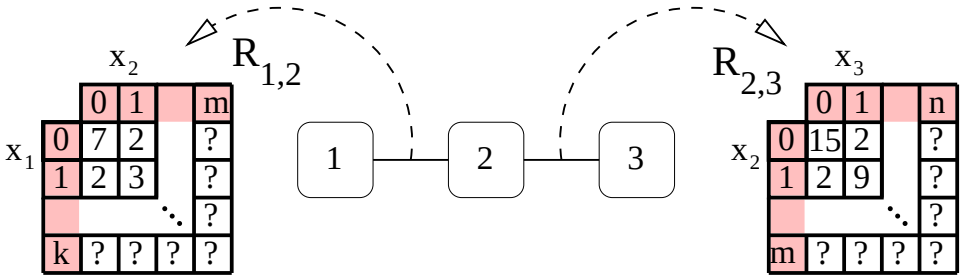


Fig. 2: This figure depicts a three-agent DCEE. As in a DCOP, each agent can select a value for its variable and the instantaneous global reward depends on pairs of agents' selected values. In contrast, constraint rewards in a DCEE are only known once they have been sampled (i.e., explored).

The previous section defined the concept of k-optimal DCOP algorithms. However, such a concept does not directly translate to DCEE algorithms because there will always be additional configurations that are unexplored (by definition). Therefore, agents would never know if they were in a locally optimal configuration because they could potentially explore and find a higher reward. Furthermore, the goal of the algorithm is not to maximize the final reward, as in a DCOP, but to maximize the on-line reward (i.e., the total cumulative team reward). Rather than discard this idea of teamwork, we classify DCEE algorithms that perform partial centralization in terms of *k-movement*, where at most k agents per neighborhood can perform joint moves. For instance, if agent x_2 in Figure 2 changes its value in a 1-movement algorithm, neither x_1 or x_3 could also change values. However, in a 2-movement algorithm, if x_2 changes its value, x_1 or x_3 (but not both), could also change their values. A 2-movement algorithm has no guarantee about final (or on-line) optimality, but if agents converge to a fixed configuration, it is because no single agent, nor pairs of agents, wish to explore. Thus, the primary expected benefit of increasing k is that larger joint moves can be performed, potentially improving the team's reward.

One may assume, as the authors first did, that k-movement algorithms would behave similarly to k-optimal algorithms. Specifically, we had expected that as teamwork increased (i.e., higher values of k were used), the total reward would increase. Surprisingly, as shown in Section 4, sometimes increasing the amount of teamwork in k-movement algorithms *decreases* the total reward accrued by the team.

2.3. *Motivating Domain*

This section introduces the *mobile wireless ad-hoc network* domain, used as a testbed throughout this article. There is one agent per network node and each agent has some movement capability. For the purposes of this article, we consider the goal of maximizing the signal strength between all pairs of communicating nodes, but many other complimentary goals are possible.[d] We assume agents have limited movement abilities such that they may attempt to optimize the wireless signal strengths but not modify the topology of the network. Even though agent movements are small, the time taken for movement is much less than the time needed for communication in our DCEE algorithms. Each agent knows its *neighbors*, i.e., the agents with whom it can directly communicate. During natural disasters, rescue personnel may quickly form such a network by placing mobile agents around a disaster site to relay information about endangered victims, fires, etc. In such situations, agents would need to optimize the network to ensure reliable and effective communication for the duration of the task.

In general, wireless radio communication has a predictable signal strength loss inversely proportional to the square of the distance between transmitter and receiver. However, such regularity disappears in urban or indoor settings because scattering, reflection, and diffraction create a *multi-path* setting. Such radio wave interference, termed *small scale fading*, results in significant signal strength differences over small distances, and modeling or predicting these signal strengths is overwhelmingly difficult [31]. This article assumes that small scale fading dominates: if a wireless agent moves at least half a wavelength, it will measure a new (uncorrelated) signal strength from each neighbor. Such signals can be modeled as an independent random number drawn from some distribution [22]. In initial experiments on physical robots, we found that the distribution of signal strengths could be approximated by a Normal distribution. Our experiments assume a Normal distribution, but our algorithms are distribution-independent — other reward distributions can be easily substituted.

Given a network of agents and the trial duration (i.e., an experiment length), the team's goal is to maximize the sum of signal strengths on all network links over this time. Each experiment is discretized into synchronized *rounds*. A round ends after all agents perform the required computation, finish communication, and execute an action. An agent's action can be to either explore or stay.

Robot experiments in this article use a set of Creates (mobile agents from iRobot, shown in Figure 3), and a custom simulator modeled after the Creates, which we

[d]For instance, the team's reward could depend on the minimum signal strength in the network, the throughput between some source and sink in the network, the average bandwidth between nodes, or the latency between a subset of important nodes. More generally, the agents could work to minimize energy usage while optimizing network communication, or they could form a sensor network that attempts to maximize coverage while maintaining connectivity.

have made publicly available.[e] Robots rely on odometry to localize and the Creates used have an odometry error of approximately 5%. All agents have the ability to execute the stay or the explore action on a given round. Depending on the specifications of the agent/robot, it may also be able to backtrack (returning to a previous location). In our physical implementation, robots are able to backtrack (ignoring odometry errors), but we also run experiments assuming they cannot.

Fig. 3: Three iRobot Creates with on-board computers

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 agent actions (i.e., agent locations in this domain), and there is a limited amount of time. We explicitly do not consider fully centralized approaches in order to reduce communication and computation requirements, as well as to improve robustness and scalability.

In the context of the mobile ad-hoc wireless network domain, each robot in the network is a DCEE agent. Direct communication links between robots represent constraints between agents. The reward is based on signal strength measurements between robots. The different physical positions of a robot constitute the domain of values possible for agents. An agent can accurately measure the signal strength between its current location and the current location of each of its neighbors only when they explore that particular physical configuration.

3. Solution Techniques

This section describes our novel DCEE algorithms. Our algorithms belong to two classes: (1) *static estimation* (SE) algorithms, which assign a constant estimate to

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

all unexplored constraints, and (2) *balanced exploration* (BE) algorithms, which use decision theoretic reasoning to compute the expected utility of each action. As discussed later with experimental results, the SE and BE classes, and specific algorithms within each class, may be preferred depending on:

- what is known about the distribution of possible rewards,
- the agents' action capabilities,
- the desired amount of teamwork, and/or
- the topology of the network.

Given the inapplicability of globally optimal algorithms to very large systems (because complete algorithms are NP-Hard), we extend locally optimal DCOP algorithms to the DCEE framework. The *Maximal Gain Messaging* (MGM) algorithm [33] and the *distributed stochastic search algorithm* (DSA) [13, 52] are natural candidates, but DSA has an additional probability parameter that must be set which has a significant impact on its performance [25], as shown in Section Appendix C. While all the algorithms presented below are in the framework of MGM, the key ideas can be embedded in any locally optimal DCOP framework. With the exception of Section Appendix C, results in this article use the MGM framework to ensure fair comparisons.

In addition to the distinction between static estimation and balanced exploration techniques, our algorithms can be classified by the amount of teamwork allowed. As discussed in the previous section, different numbers of agents can form teams to execute joint actions. In k=1 algorithms, only a single agent can change variables per neighborhood, and our k=1 algorithms are described using the MGM framework (see Algorithm 1). k=2 and k=3 algorithms (described in Algorithms 2 and 3, respectively), use the framework of MGM-2 and MGM-3 [34], allowing additional teamwork. The k=2 and k=3 algorithms may be considered "natural extensions" to the k=1 DCEE algorithms. Each algorithm will be introduced first with the 1-movement variant, and in most cases, an explanation of the 2-movement extension will follow. We further extend two of the algorithms to a 3-movement variation as a proof-of-concept.

Table 1 summarizes the DCEE algorithms by agent ability, what reward information is required, and whether the number of rounds is needed by the algorithm in question. If the rewards are assumed to follow a normal distribution, the PDF and the CDF ($f(x, n)$ and $F(x, n)$, respectively) can be directly calculated from the mean, μ, and standard deviation, σ, of the Gaussian describing the reward distribution.

3.1. *k=1 Algorithms*

After first defining an "omniscient" algorithm, we describe the SE algorithms in section 3.1.2 and the BE algorithms in section 3.1.3. The omniscient algorithm operates within the DCOP framework, and "k" represents the value of k-optimality.

Table 1: The DCEE algorithms discussed in this section have different abilities and knowledge requirements, as discussed in the text.

Method Name	Agents Can Backtrack	Reward Information	# Rounds Remaining	Variants Tested k=1	k=2	k=3
Omniscient		all rewards		✓	✓	✓
SE-Optimistic		maximum reward		✓	✓	✓
SE-Mean		average reward		✓	✓	
BE-Backtrack	✓	PDF and CDF	✓	✓		
BE-Rebid	✓	PDF and CDF	✓	✓	✓	
BE-Stay		PDF and CDF	✓	✓	✓	

The remaining DCEE algorithms overload k to mean the number of agents that can execute joint moves in k-movement algorithms.

3.1.1. *Omniscient Algorithms*

We first implement MGM and artificially provide agents with the reward for each possible value, converting the DCEE problem into a DCOP. Provided such a matrix, Omniscient algorithms will find a locally optimal assignment of values for all agents, providing an (unrealistic) upper bound on algorithmic performance for incomplete algorithms on a given DCEE problem. Additionally, we assume that the DCEE algorithms cannot explore all value settings — an Omniscient algorithm may select an assignment that *no* DCEE algorithm could discover without exhaustive exploration (which is impossible within a time-limited trial, by definition).

Consider the MGM-1 framework in Algorithm 1. **Omniscient-1** defines a round as a period in which every agent:

(1) communicates its current assignment to all its neighbors, (Algorithm 1, lines 1–3),[f]
(2) calculates and communicates its *bid* (i.e., the expected gain in its local reward if it is allowed to change values) to all its neighbors (line 4), and
(3) if it has the maximum gain of its neighbors (line 9), change its value (line 13). Ties for winning the bid to move are split randomly.

At quiescence, no agents will bid to change variables because they have reached a locally optimal team reward; no one agent can deviate from the (1-optimal) equilibrium assignment without decreasing the team's reward.

The computational complexity of Omniscient-1 is primarily in the *getMaxGainAndAssignment()* procedure. On each round, each agent computes what its total reward would be if it changed its variable setting to each of the variable's

[f]If warranted, extra bookkeeping can eliminate the communication on lines 1-3 in the algorithm for all but the first round, reducing the communication overhead.

Algorithm 1 Pseudocode for agents using k=1 algorithms

```
1: for each neighbor i do
2:    Send variable assignment and reward matrices to i
3:    Receive variable assignment and reward matrices from i
4: Find max gain and preferred assignment: g, a ← getMaxGainAndAssignment()

5: Send Bid g to all neighbors
6: Receive Bids from all n neighbors
7: G ← max_n Bids_n
8: if g > G then
9:    bChanging← True
10: else
11:    bChanging← False
12: if bChanging then
13:    UpdateAssignment(a)
```

possible settings, assuming that its neighbors did not change their variable settings. The agent would then submit a bid equal to the difference between this maximum reward and its current reward.

3.1.2. *Static Estimation (SE) Algorithms*

Unlike Omniscient algorithms, SE algorithms operate on the full DCEE problem and must explicitly reason about exploration and exploitation. In particular, on every round, an agent may stay with its current variable setting, or explore a new variable setting. Although each agent has many possible variable settings, it has no reason to prefer to explore one particular setting over another, as we assume there is no prior knowledge about individual unexplored variable settings. Without loss of generality, we can assume that an agent has only two choices: stay with its current variable setting, or explore the next variable setting (according to an arbitrary ordering of variables).

SE-Optimistic-1 agents estimate their gain for exploration, if allowed to change variables, by assuming that they will receive the maximum reward on the next unexplored constraint. For instance, in the mobile ad-hoc network domain, this equates to assuming that if an agent moves to an unexplored location, it will maximize the signal between itself and all neighbors. On every round, each agent bids its expected gain:[g]

$$NumberLinks \times MaximumReward - R_c,$$

[g]An agent's expected gain is what it expects to receive from a variable change. When an agent explores, it may expect to increase reward (a positive gain), but in fact receive a decreased reward (a "negative gain").

where R_c is the current reward. The algorithm then proceeds as in Omniscient-1. This is similar to a 1-step greedy approach where agents with the lowest rewards have the highest bid. Throughout the experiment, at least one agent will typically explore per round and the agents' variables never stabilize to a fixed setting.[h]

SE-Mean-1 differs from SE-Optimistic-1 by assuming that visiting an unexplored value will result in the average reward on all constraints (denoted μ) instead of the maximum reward. Agents have an expected gain of

$$NumberLinks \times \mu - R_c,$$

causing the agents to explore until their average constraint reward is at least μ. This allows agents to eventually converge to a fixed configuration, unlike in SE-Optimistic-1. Although the DCEE problem is an optimization problem (by definition), the SE-Mean algorithms are satisficing: agents only optimize until reaching a per-constraint average of μ. SE-Mean assumes that the mean of the distribution is known, or can be estimated. In contrast, SE-Optimistic only needs to have the value of a reward that cannot be exceeded. (SE-Optimistic can also use an unrealistically high value, rather than the actual maximum reward, without affecting behavior.)

3.1.3. *Balanced Exploration (BE) Algorithms*

Balanced Exploration algorithms allow agents to more accurately estimate the expected utility of exploration given a time horizon, as well as precisely when to stop exploring within this time horizon. Each agent compares the expected gain from exploring new variable settings or exploiting known variable setting rewards. This gain from exploration depends on: (1) the number of rounds remaining in the trial, (2) the distribution of rewards, and (3) the current reward of the agent, or the best explored reward if the agent can return to a previously explored setting. Such decision theoretic calculations require knowledge about the distribution of rewards (specifically, the probability density function and/or the cumulative density function) and require significantly more computation than the SE algorithms. As in the previous algorithms, the agent with the highest bid per neighborhood is allowed to change variable settings.

In addition to executing the stay or explore actions, agents may also be able to backtrack to a previous variable location. Such backtracking allows the agents to better exploit past knowledge and to explore more aggressively, knowing that they may return later to a previous variable assignment. In some settings, agents will be able to easily return to previous assignments, while in others they will not (for instance, a robot may or may not have sensors accurate enough to precisely return to a previous physical location). The BE-Backtrack and BE-Rebid algorithms allow backtracking, while the BE-Stay algorithm does not. As the reader may suspect,

[h]This due to our selection of Gaussian rewards — if the maximum value had a non-trivial probability of being reached, an optimistic algorithm would eventually stabilize.

empirical results in Section 4 show that the ability to backtrack can significantly improve DCEE agents' performance.

BE-Backtrack-1 differs from all other algorithms discussed in this article because it makes commitments to explore for *multiple* timesteps. No other algorithms make such commitments, but reason about the value of different actions on every round. As we discuss later, this commitment actually reduces the team's overall reward, but we present this BE algorithm first for expository purposes. BE-Backtrack-1 calculates the value for backtracking to the best known variable setting, V_{back}, and the value for exploring a new variable setting, $V_{explore}$.

V_{back} is the expected utility of backtracking to the best known variable setting (which the agent can reach in a single round) and then not changing the setting for the remainder of the trial, t rounds. Throughout the trial, each BE-Backtrack-1 agent keeps track of the value with the highest total received reward (R_b). At any point, the agent may return to this value (e.g., a location in the mobile ad-hoc wireless network domain) if its neighbors have not changed their values. The *state* of the agent can thus be defined as (R_b, T); this notion of the agent's state differs from the actual value of the agent's assignment. The utility of backtracking is:

$$V_{back}(R_b, t) = R_b t.$$

The utility of exploring is based on the reward of the best expected value found during exploration. Let the number of rounds for which the agent explores be t_e. An exploration policy would be in the form "explore for t_e rounds, backtrack to the best value found on round $t_e + 1$, and then stay with that value for the remainder of the experiment for t_s rounds," where $t = t_e + t_s$.

$V_{explore}(R_b, t)$ can be calculated by summing three separate components:

(1) the expected utility accrued while exploring for t_e steps,
(2) the utility accrued after exploration multiplied by the probability of finding a reward better than R_b, and
(3) the utility accrued after exploration multiplied by the probability of failing to find a reward better than R_b.

The first component, the expected utility accrued while exploring, is $t_e \times \mu \times n$, where μ is the average reward and n is again the number of neighbors (i.e., constraints) for the agent. The second component depends on the probability of finding values with a total reward higher than R_b, multiplied by the number of steps left in the trial. The expected best reward in this case is described by the distribution:

$$\int_{x > R_b} x \cdot Q(x, n, t_e) dx$$

where $Q(x, n, t_e)$ gives the probability of x being the maximum sample among the t_e samples drawn and is defined as:

$$Q(x, n, t_e) = t_e \times f(x, n) \times F(x, n)^{t_e - 1}.$$

This n^{th} order statistics equation calculates the probability that x will be the maximum reward found in t_e values. n is the number of neighbors, $f(x,n)$ is the probability of drawing x as a sample given that the agent has n neighbors (the PDF), and $F(x,n)$ is the cumulative probability of drawing a sample less than or equal to x (the CDF) given n neighbors, defined as

$$\int_{y \leq x} f(y,n)dy.$$

Informally, $Q(x,n,t_e)$ is calculated by drawing a sample x from any of the t_e samples with a probability $f(x,n)$, and drawing the rest of the $t_e - 1$ samples, such that their values are less than x, with a probability of $F(x,n)^{t_e-1}$.

The third component will depend on how likely it is that we fail to discover a reward greater than R_b, times the number of steps left in the trial. After the agent explores, it will backtrack and receive R_b for the remaining t_s rounds. Again, the cumulative probability of drawing a sample less than or equal to R_b in t_e samples is defined as $F(R_b,n)^{t_e}$, where $F(x,n)$ is defined as before. Thus, $V_{explore}(R_b,t) =$

$$\max_{0 \leq t_e \leq t} \left\{ t_e \mu(n) + t_s \int_{x > R_b} xQ(x,n,t_e)dx + t_s R_b F(R_b,n)^{t_e} \right\}. \quad (1)$$

The value of t_e that maximizes $V_{explore}$ gives the number of exploration steps. The expected reward of state (R_b,t) for BE-Backtrack-1 is:

$$V(R_b,t) = \max \left\{ V_{back}(R_b,t), V_{explore}(R_b,t) \right\}.$$

The bid of the agent is $V(R_b,t) - R_c t$ and the agent with the highest bid per neighborhood will immediately backtrack, or explore for t_e rounds and then backtrack to the value of the highest reward found thus far. Agents that do not win the bid to change variables execute the stay action. BE-Backtrack-1 assumes that agents will not change values after backtracking in the $(t_e+1)^{th}$ round. However, if the agent's neighbors later change their value, R_c (the agent's current reward) may change, and the agent may subsequently choose to explore rather than staying at its backtracked value setting.

Notice that when an agent's neighbors explore and then backtrack, they could not have reduced the instantaneous team reward. In particular, the reward of an agent that has backtracked after exploring cannot be lower than its reward at the time it started exploring (although it may be lower during exploration). This is because only this agent was allowed to change values in its neighborhood and the agent could have backtracked to its initial value (and, thus initial reward) if it were unable to find a better configuration. However, an agent that explores is preventing others from exploring, which may reduce the total on-line reward (relative to a different agent acting or to no agents changing values). As we will show in Section 4, the BE-Backtrack-1 algorithm is dominated by BE-Rebid-1, discussed next.

BE-Rebid-1 agents calculate their gain using the BE-Backtrack equations but all agents re-calculate and rebid on every timestep. Prior work in different decision making contexts has shown that such reevaluation at each timestep can lead to better performance in practice [32]. Equation 1 again calculates the expected gain of exploring for t_e steps, but now the agent may execute fewer than t_e exploratory steps. If this happens, it will be due to rewards received after moving: either the moving agent has found itself in a favorable position and no more exploration is needed, or the cumulative reward has decreased significantly and one or more of its neighbors has now won the bid to change values.

BE-Stay-1 may be used by agents that are unable to backtrack. Based on experiments showing that BE-Rebid-1 dominated BE-Backtrack-1, BE-Stay was designed as another approach where agents make a decision every round. In this algorithm, every agent considers its current total reward and compares the expected reward it would receive if it kept the same variable assignment (V_{stay}) with the expected reward of exploring ($V_{explore}$). The reward of exploring, given the current reward R_c, is calculated recursively as:

$$V(R_c, t) = \begin{cases} V_{stay}(R_c, 0) = V_{explore}(R_c, 0) = 0 & \text{for } t = 0 \\ \max(V_{stay}(R_c, t), V_{explore}(R_c, t)) & \text{for } t > 0. \end{cases}$$

The expected reward from stay will be the current reward multiplied by the time left in the trial:

$$V_{stay}(R_c, t) = R_c t.$$

The expected reward of exploring will depend on the probability of achieving a given reward in the next state, the reward received for that one timestep, and the expected reward of the rest of the trial:

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

where $f(x, n)$ is the probability of receiving the total reward x in an unexplored location (i.e., the PDF).

Therefore, the expected value of having a current signal strength of R_c with t timesteps remaining is:

$$V(R_c, t) = \begin{cases} 0 & \text{if t = 0,} \\ \max\left(R_c t, \int_{-\infty}^{\infty} f(x, n)(V(x, t-1) + x)dx\right) & \text{otherwise.} \end{cases}$$

In each round, agents calculate V_{stay} and $V_{explore}$. If explore has the higher expected reward, an agent will bid to change its value. If the value of stay is higher than the value of explore, the agent will execute the stay action and will not bid to act. BE-Stay-1 differs from BE-Rebid-1 even when the backtrack state is the current state: BE-Rebid-1 assumes the agent may backtrack to this state in the future, which BE-Stay-1 does not.

3.2. *k=2 Algorithms*

This section discusses increasing the amount of teamwork to k=2, with the exception of BE-Backtrack-1, which was dominated by BE-Rebid-1 (as shown later in Section 4.1).

Omniscient-2 extends Omniscient-1 so that two agents may change value per neighborhood. This algorithm is again artificially provided the full reward matrix; it may be considered a re-implementation of MGM-2. Given sufficient time, Omniscient-2 will always discover a 2-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. (It is not impossible for an algorithm that must explore to outperform an Omniscient algorithm, as the latter algorithm finds a locally, not globally, optimal solution.) On each round, an agent:

(1) selects the neighbor (Algorithm 2, line 4) that will provide the highest joint gain and sends an *Offer* message for a joint variable change (line 5),
(2) sends an *Accept* message if its selected neighbor offers to pair with it (line 9),
(3) calculates its individual gain if the pairing failed (line 14),
(4) finds out if it is part of a pair that has a higher gain than its neighbors, or is unpaired and has a higher gain than its neighbors (line 18), and
(5) changes its value (line 29), unless it does not have the highest gain in the neighborhood (line 21) or it is part of a pair and its neighbor cannot change variables (line 27).

Like Omniscient-1, Omniscient-2 monotonically increases its solution quality [25]. Omniscient-2 also requires more communication than Omniscient-1, but generally reaches higher or similar solution quality [34]. As with other k=2 algorithms, single agents may act alone.

The *getMaxGainAndAssignment()* procedure in Omniscient-2 is unchanged from Omniscient-1. The new procedure, *getMaxGainAndAssignmentForPair()* is more computationally complex. Every agent must consider every neighbor as a possible pair. Then, for each pair, it considers all possible variable assignments for the two agents, assuming all neighbors of the pair do not change variable settings. Having found the maximum reward for each pair, it finds the pair with the highest reward, and will offer to pair with this agent.

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

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()*

Algorithm 2 PSEUDOCODE FOR k=2 ALGORITHMS

1: **for** each neighbor i **do**
2: Send variable assignment and reward matrices to i
3: Receive variable assignment and reward matrices from i
4: Find maximum gain, g, the corresponding neighbor to pair with, p, and the variable assignment, a:
$g, p, a \leftarrow$ *getMaxGainAndAssignmentForPair()*
5: Send `OfferPair` to agent p
6: *doPair* $\leftarrow$ False
7: **for** all `OfferPair` messages received **do**
8: **if** agent requesting to pair is p **then**
9: Send `Accept` to agent p
10: *doPair* $\leftarrow$ True
11: Receive `Accept` message, if any
12: **if** (Did not received `Accept` from p) or (not *doPair*) **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 n `Bids` from all neighbors, ignoring message from p
17: $G \leftarrow \max_n$ `Bids`$_n$
18: **if** $g > G$ **then**
19: *bChanging*$\leftarrow$ True
20: **else**
21: *bChanging*$\leftarrow$ False
22: **if** $p \neq \emptyset$ **then**
23: Send `ProhibitVariableChange` to agent p
24: Receive any messages sent by neighbors
25: **if** (bChanging) and ($p \neq \emptyset$) **then**
26: **if** Received `ProhibitVariableChange` from agent p **then**
27: *bChanging*$\leftarrow$ False
28: **if** *bChanging* **then**
29: UpdateAssignment(a)

calculate utilities for possible assignments. Agents individually expect to gain reward equal to $NumberLinks \times MaximumReward - R_c$. Agents that successfully pair with their selected neighbor (Algorithm 2, line 10) bid their joint gain. The joint gain is equal to the sum of their individual gains, not double-counting the shared constraint.

SE-Mean-2 is similar to SE-Optimistic-2, but modifies its utilities so that unexplored variables are assumed to return the average reward. As in SE-Mean-1, individual agents expect to gain $NumberLinks \times \mu - R_c$ and compute the gain for

pairs by considering all links connected to the pair of agents.

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, backtrack-stay, and coordinated backtrack. An agent may not change its value if a neighbor executes backtrack: explore-backtrack is an invalid joint action because if one agent explores, the constraint shared with the other agent would take on the value of a new variable, and that other agent would not be able to backtrack by definition. The coordinated backtrack action is unique to this algorithm; it allows two agents to simultaneously backtrack to a previous setting.[i]

BE-Rebid-2, follows Algorithm 2 and calculates the gain of explore and backtrack actions as was done for the BE-Backtrack-1 algorithm. The gain of an action is based on the difference between the expected reward of the action and R_ct. As in Omniscient-2, agents calculate gains over joint actions with their neighbors 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:

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

which calculates the sum of two gains (not double counting the shared constraint). $V_{explore:i}$ is the value of exploration by agent i and $R_{b:i}$ is the best reward found for agent i. This estimate, in effect, is the sum of agent i and agent j exploring independently, while not double-counting the shared constraint. The joint gain of explore-stay is determined by the exploring agent. Similarly, the joint gain of backtrack-stay 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 is similar to BE-Stay-1, but now 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 Omniscient-2 and an agent may act individually if it fails to pair but still wins the bid.

Section 4.2 discusses the relative performance of k=1 and k=2 algorithms. Two k=3 algorithms are used to bolster the claim that the trends observed continue to higher values of k. An explanation of **Omniscient-3** and **SE-Optimistic-3** can be found in Appendix A.

[i]Note that a joint backtrack of k agents would require coordination between all k agents — full centralization would be required if all agents were to return to a previous configuration, which is out of scope for this article.

4. Experimental Results

Experiments in this section compare the performance of our DCEE algorithms in multiple settings. In our domain, the number of variable settings is always more than can be fully explored. Signal strengths are drawn from a normal distribution defined by $\mathcal{N}(100, 16)$.[j] To allow for better comparisons, the random seeds were pre-generated and kept consistent across the different algorithms. Throughout, error bars denote the standard error.

In our simulator experiments we set the number of possible variable settings to be one more than the number of rounds (i.e., in a 100 round experiment, there are 101 possible variable settings per agent), so that no agent could possibly explore all of its variable settings. We simulate signal strength values as non-negative integers, allowing our implementations to use summations rather than integrations.

The questions this section addresses are:

(1) Can DCEE algorithms effectively improve the on-line team reward?
(2) How does the performance of DCEE algorithms compare to the performance of the Omnisicient algorithm?
(3) What are the highest performing DCEE algorithms and why?
(4) What factors change the relative performance of different DCEE algorithms?
(5) How does the performance of 1-movement algorithms compare to 2-movement algorithms?
(6) Can results found in simulation be replicated on a multi-robot platform?

4.1. *Simulation Results for k=1*

This section presents three sets of results, each of which varies a different component of the problem domain: the number of agents, the time horizon, or the network topology. However, we first graph the performance of different algorithms on a chain of agents. Figure 4 shows *learning curves* for 5 DCEE algorithms, compared to the DCOP Omniscient-1 algorithm and no optimization (i.e., no agents move during the trial). The x-axis shows the round number and the y-axis shows the total signal strength. We say that the team "learns" if it is able to improve reward over time and an algorithm's performance is measured by the total area under the curve. 40 agents run for 100 rounds, and curves are averaged over 30 independent trials. For each algorithm, the total cumulative signal strength is the area under the curve and the gain is the area between the curve and the NoMovement line. SE-Mean-1 converges quickly to a comparatively low value while SE-Optimistic-1 explores continually, attempting to achieve the maximal signal strength. BE-Stay-1 cannot backtrack and must be cautious; it converges the fastest of all three BE methods. BE-Rebid-1 performed the best, leveraging its ability to backtrack to a previous location and

[j]Normal functions are infinite; we constrained signals to the range [0,200], covering 99.999% of the probability mass of this distribution.

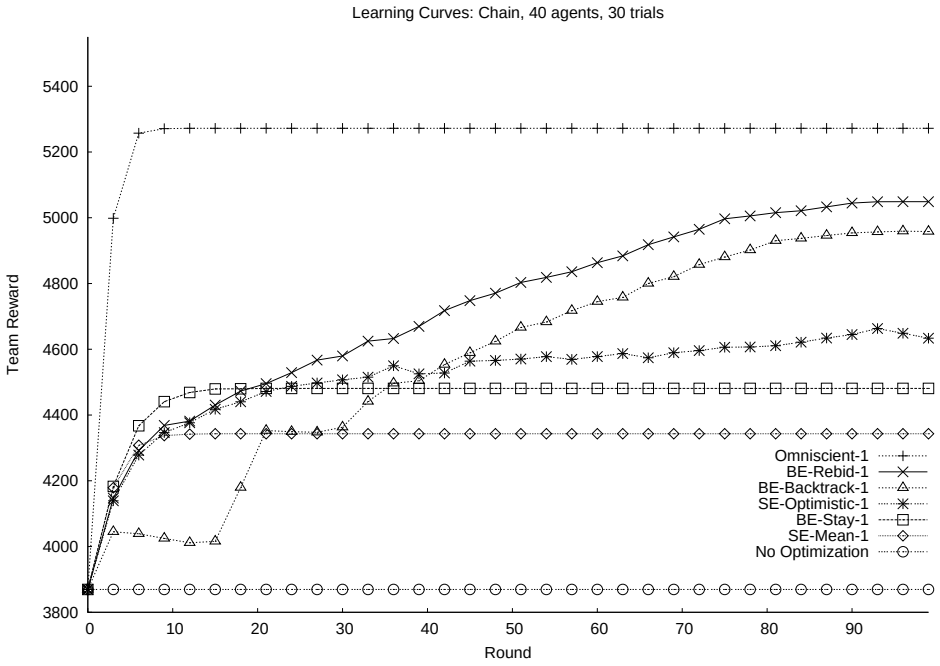


Fig. 4: A learning curve for 40 agents in a chain topology where $T = 100$, averaged over 30 trials. Averages are plotted every 3 rounds for clarity.

to use knowledge about the reward distribution. BE-Backtrack-1, which commits to explore for some number of rounds without re-evaluating, is dominated by BE-Rebid-1.

Having considered how different algorithms improve their reward over time, we now focus on the total reward gained, which is the quantity we aim to maximize. Results are reported as a scaled gain, where 0 corresponds to no optimization and 1 corresponds to the improvement of BE-Rebid-1 (as BE-Rebid-1 was the highest performing 1-movement DCEE algorithm). Any gain greater than zero represents an improvement directly due to the algorithm. Such a metric helps isolate the improvement due to agent movement and scales across tasks with different numbers of links, agents, and time lengths.

Figure 5(a) shows the algorithms' relative performance over different experiment lengths. The y-axis measures the scaled gain. The x-axis shows the five values of T, the total number of rounds in a trial. All trials use random sparse graphs with 15–20 links and 10 agents. Each result is averaged over 30 independent trials. The difference between scaled gain for each pair of algorithms is statistically significant within a single value of T (paired Student's t-tests calculate $p < 0.05$), except for $T = 5$. When the time horizon is very small, all but SE-Optimistic-1 perform roughly the same because all four algorithms explore very little. As the number

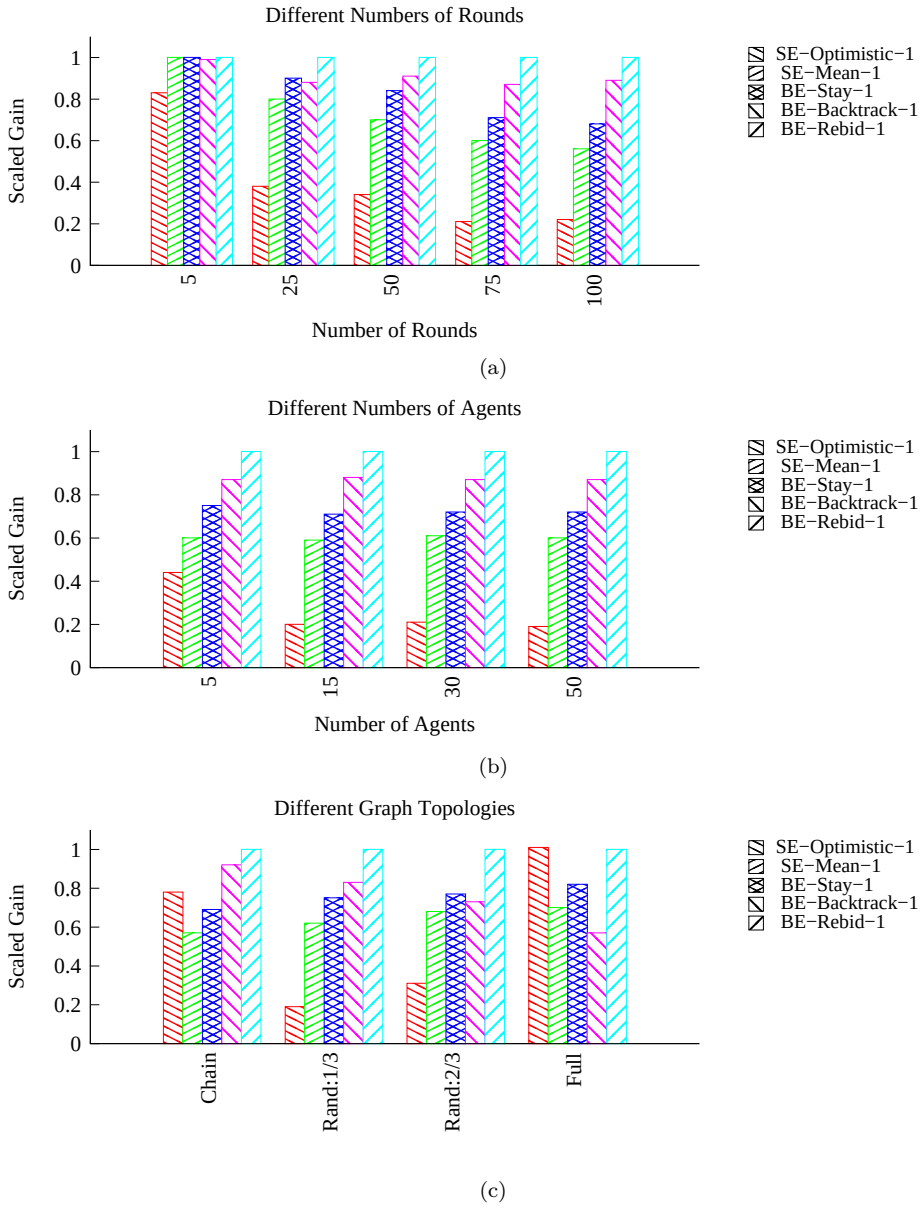


Fig. 5: The performance of different algorithms is shown where the y-axis is the scaled gain (0 represents No-Movement and 1 represents the gain of BE-Rebid-1) and results are averaged over 30 independent trials. In (a), 10 agents are run on random graphs where $\frac{1}{3}$ of the possible constraints are active (corresponding to a DCEE graph with 15–20 edges), and the x-axis reports the number of rounds in the experiment. (b) shows the performance for different numbers of agents on the same random graphs run for 100 rounds. (c) shows the performance of 20 agents run for 100 rounds on four different topologies, including random graphs with $\frac{1}{3}$ and $\frac{2}{3}$ of the possible links added.

of rounds per experiment increases, BE algorithms outperform SE algorithms, and BE-Rebid-1 consistently achieves the highest scaled gain.

The second set of results, shown in Figure 5(b), varies the number of agents and again uses random sparse graphs. The time horizon is 100 rounds. The x-axis shows the number of agents, varied from 5 to 50. Paired Student's t-tests determine all results to be statistically significantly different ($p < 0.05$), confirming that BE-Rebid-1 outperforms all other algorithms.

The third set of results shown in Figure 5(c) compares the performance on different graph topologies: a chain structure, random structures (with $\frac{1}{3}$ or $\frac{2}{3}$ of all possible links enabled), and a fully connected topology. Each test uses 20 agents and 100 rounds. All results within a single topology are again statistically different ($p < 0.05$).

Three trends in Figure 5(c) are worth noting. First, BE-Rebid-1 statistically significantly ($p < 0.05$) outperforms all other algorithms in all topologies tested, except in the fully connected graph (where it is similar to SE-Optimistic-1 as only one agent can move per round and both algorithms generally select the agent with the lowest reward to optimize its value). Fully connected graphs are thus one setting where the static estimation algorithms can perform just as well as the more complex BE algorithms.

Second, as the link density of the graph is increased, the relative performance of BE-Backtrack-1 *decreases*, with statistical significance ($p < 0.05$), due to the aggressive nature of the algorithm. A BE-Backtrack-1 agent will explore for t_e steps, preventing all neighbors from moving during this time. Thus, as the link density increases, higher numbers of agents are not allowed to move until after t_e steps.

Third, SE-Mean-1 outperforms SE-Optimistic-1 in randomly generated graphs, but not in chain and fully connected graphs. Unlike in chain and fully connected graphs, agents in random graphs can have a high variance in their degrees of network connectivity. We analyze the number of agents that were able to optimize their rewards. While 40% of the agents moved when running SE-Mean-1, only 18.5% agents could do so when running SE-Optimistic-1 in random graphs with density $\frac{1}{3}$. SE-Optimistic-1 agents with a high degree of connectivity monopolize movement opportunities because they bid unrealistically high rewards. Their bid is relatively large when compared to the bids of agents with lower degrees. There exists a large correlation (Pearson's coefficient of $\rho > 0.5$) between the degree of the agent and the number of moves made by the agent in SE-Optimistic-1. In contrast, SE-Mean-1 agents allow others to win bids once the reward of an agent reaches $\mu \times n$, where n is the number of neighbors for the agent. There exists only a weak correlation with a Pearson's coefficient of $\rho < 0.05$ between the degree of the agent and the number of moves made by the agent for SE-Mean-1, explaining this difference in performance.

The results also show that BE-Stay-1 is statistically significantly dominated by BE-Rebid-1, demonstrating that the ability to backtrack can lead to significantly better performance. In random graphs, SE-Optimistic-1 has the worst performance,

Omniscient Performance

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 2: Gains of Omniscient Algorithms

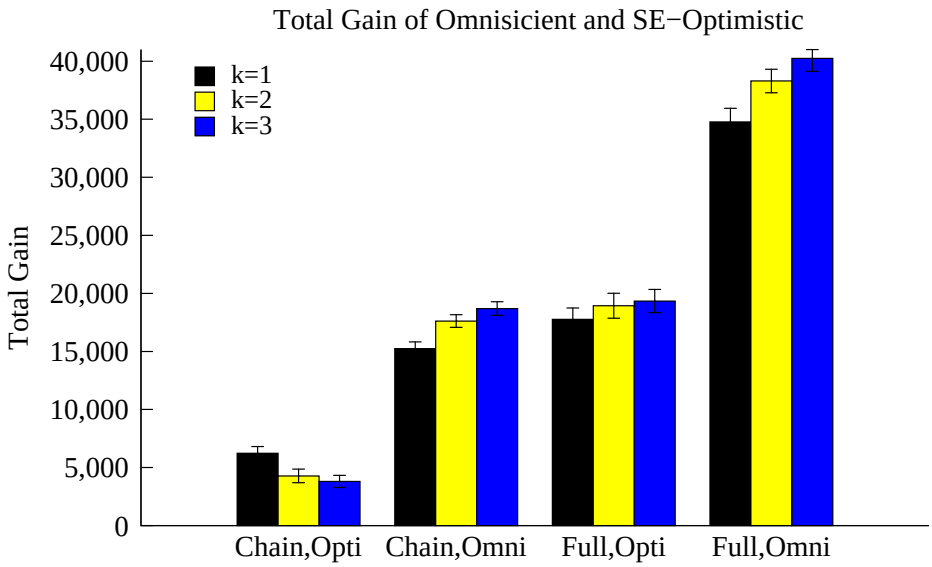


Fig. 6: In chain graphs, SE-Optimistic-1 outperforms SE-Optimistic-2, which in turn outperforms SE-Optimistic-3. Increasing values of k improves performance in SE-Optimistic on full graphs, and in SE-Omniscient in both chain and full graphs.

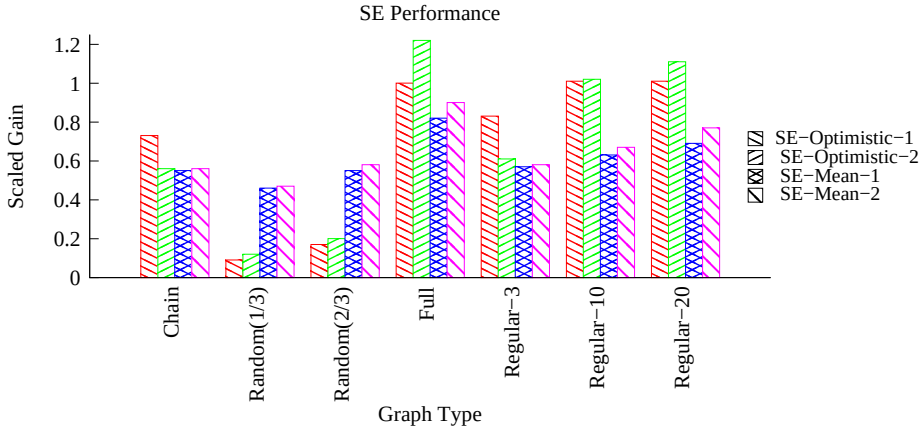
but in chain and full graphs it dominates SE-Mean-1. SE-Optimistic-1 performs similar to BE-Rebid-1 in full graphs.

4.2. *Simulation Results, k=2*

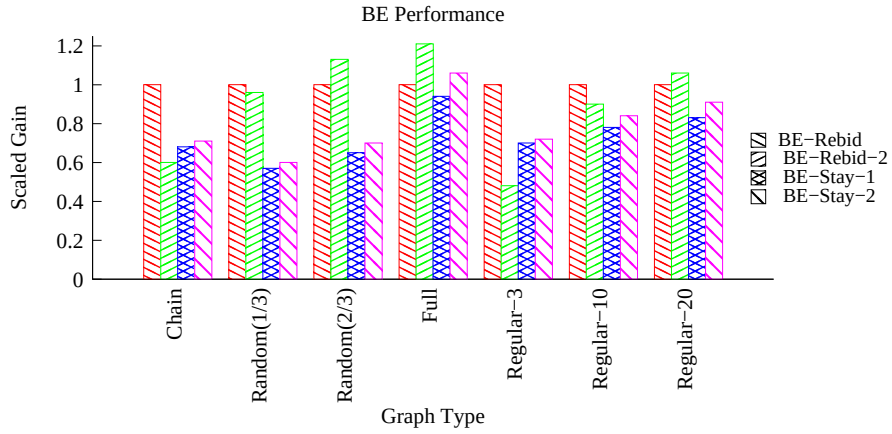
The results in Table 2 list the total gains for Omniscient-1 and Omniscient-2 algorithms on graphs with four different topologies. Experiments run 40 agents for 100 rounds over 30 independent trials. Recall that the Omniscient algorithms artificially provide reward matrices to the agents; this result shows that increased teamwork is beneficial, as expected from previous DCOP work. Additionally, these results help confirm that our k=2 implementation is correct.

Figure 6 shows the results from experiments where 10 agents are run for 50 rounds. In a chain graph, the k=1 version of SE-Optimistic performs better than k=2, which performs better than k=3. In the full graph, SE-Optimistic-3 is better than SE-Optimistic-2, which is better than SE-Optimistic-1. All differences

in the chain graphs are statistically significant ($p < 0.05$), and k=2 algorithms outperform their k=1 counterparts in the full graphs with statistical significance ($p < 0.05$). The Omniscient-3 algorithm follows the trend seen earlier in Table 2 by dominating Omniscient-2, which in turn dominates Omniscient-1 ($p < 0.05$ for the Omniscient gains). These results confirm that higher amounts of teamwork improve agent performance in Omniscient algorithms, but may decrease performance in non-Omniscient algorithms.



(a) SE-Optimistic and SE-Mean



(b) BE-Rebid and BE-Stay

Fig. 7: Figures (a) and (b) report the scaled gain of the static estimation and balanced exploration algorithms, respectively. SE-Optimistic-2 and BE-Rebid-2 are outperformed by SE-Optimistic-1 and BE-Rebid-1 on low density graphs. SE-Mean-2 and BE-Stay-2 always outperform their k=1 counterparts, but typically have lower performance than the SE-Optimistic and BE-Rebid algorithms.

Next, consider Figures 7(a) and 7(b), 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 1-movement DCEE algorithm). 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, $p < 0.05$), 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). The results of SE-Optimistic-3 in Figure 6 serve to further confirm this phenomenon: increasing values of k reduce performance on chain graphs, but improve performance on full graphs.

However, the BE-Rebid and SE-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 [25, 34]. We term this phenomenon the *team uncertainty penalty*. This penalty strictly affects total reward: it does not consider any penalty from increased communication or computational complexity. To investigate this phenomenon, we next considered a set of sparse graphs with random topologies.

Figure 8 compares the relative performance of the k=1 and k=2 variants of SE-Optimistic and BE-Rebid on random topologies. The first trend to notice in Figure 8 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. Again, this is in contrast to Omniscient algorithms, where k=2 always outperforms k=1. The second trend to notice is that both SE-Optimistic algorithms perform quite poorly on random graphs, as shown before in Figures 5(c) & 7(a). In SE-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 earlier.

Comment 1. Fig 8 poor quality

To better understand why SE-Optimistic algorithms perform poorly on random graphs, we considered 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 9 shows the scaled gain of the two SE-Optimistic and the two BE-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 8, the performance of BE-Rebid-2 improved relative to the k=1 version as the density was increased. Figure 9 shows that this trend holds for both BE-Rebid and SE-Optimistic in regular graphs. In particular, k=1 outperforms k=2 for both SE-Optimistic and BE-Rebid in graphs with three and five neighbors per agent ($p < 0.05$). SE-Optimistic-2 outperforms SE-Optimistic-1 on regular graphs with twenty neighbors ($p < 0.05$) while the two BE-Rebid algorithms do not have

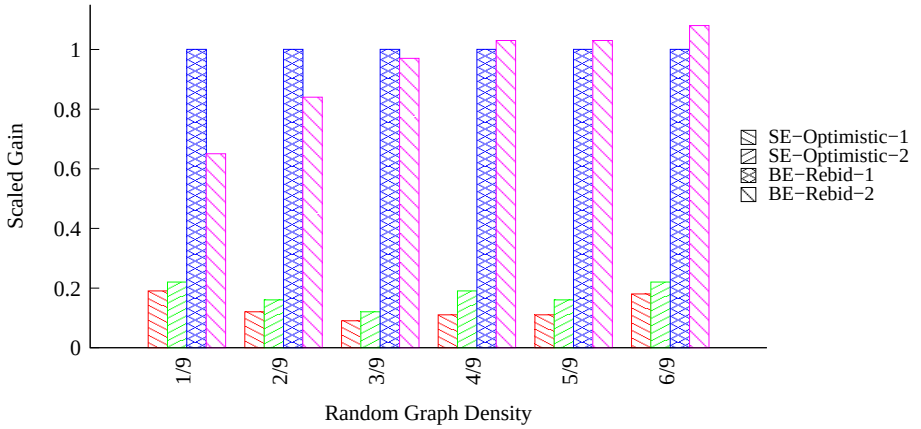


Fig. 8: Six types of random graph topologies are considered, where $\frac{1}{9}$ to $\frac{6}{9}$ of the number of constraints in a fully connected graph is added. BE-Rebid-2 outperforms BE-Rebid-1 as the density in randomly connected graphs increases.

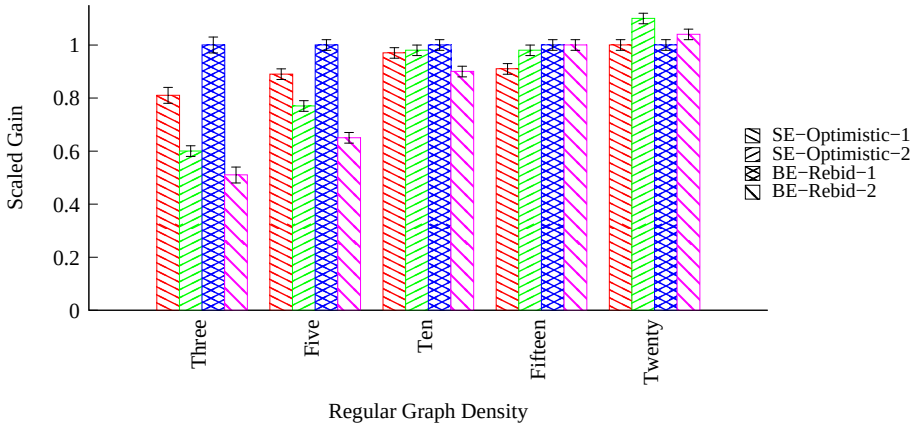


Fig. 9: This graph shows the relative performance of k=1 and k=2 change with the number of neighbors where the labels on the y-axis show the number of neighbors each agent has in the regular graph topology.

statistically significant gain differences.

Most previous work in teamwork and joint actions, including previous results in k-optimal algorithms, led us to expect that increasing the level of teamwork in decision making would lead to improved total reward 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. (Recall that k-movement was defined in Section 2.2, where k refers to the maximum number of agents that can perform a joint action.) We have been able to isolate situations where this phenomenon oc-

curs — in graphs with low density, k=2 algorithms can perform worse than k=1 algorithms. Section 5 explains the origin of this team uncertainty penalty with the ultimate goal of improving our algorithms so that increasing teamwork does not decrease performance.

4.3. *Physical Robots Results*

To test our DCEE algorithms we produce two implementations, both for the Create robot. The first used a wireless network card by CenGen and a proprietary API, whereas the second used a small computer with a mini PCI card.[k]

In the mobile ad-hoc wireless network domain, each variable setting corresponds to a different physical location. In order to ensure signal strengths in different locations are uncorrelated, agents explore by moving forward to a location that is at least $\frac{1}{2}$ of a wavelength (i.e., $\geq$ 2.5 cm.) away from the current location.

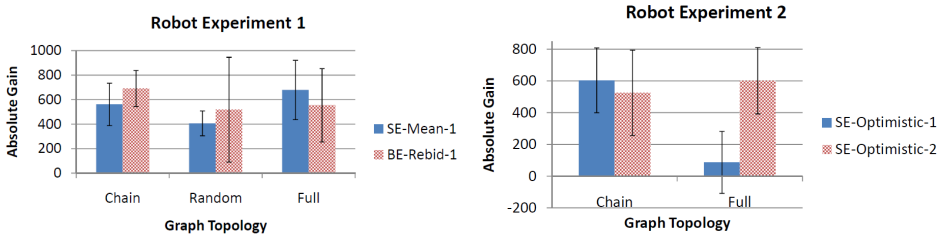
The length of a round in this distributed setting is dominated by agent movement, which takes longer than either the computation or communication for the algorithms presented. This is true both because of how slowly robots move (relative to communication signals) and because the robots must take several readings of signal strength after moving to calculate a reliable average signal strength. If agents converge upon a final configuration before the test ends, no agents will move within a round. However, the length of the round does not vary: the length of a round determines how continuous time is discretized to measure signal strength in experiments but the discretization is not critical for measuring the relative performance of algorithms.

The first set of experiments tested three different topologies: chain, random, and fully connected. In the random topology tests, the robots were randomly placed and the CenGen API automatically defined the neighbors, whereas the robots had a fixed set of neighbors over all trials in the chain and fully connected tests. Each of the three experiments used four robots and was repeated five times for 20 rounds each.

Figure 10(a) shows the results of running BE-Rebid-1 and SE-Mean-1 on the robots. The gain on the y-axis has not been normalized. RSSI (received signal strength indication) values are reported in decibels (dB). BE-Rebid-1 performs better than SE-Mean-1 in the chain and random graphs, but loses in the fully connected graph. While too few trials were conducted for statistical significance, it is important to note that in all cases there is an improvement over the initial configuration of the robots. Additionally, because decibels are a log-scale metric, the gains are *even more significant* than one may think on first glance.

The second set of tests corroborates the team uncertainty penalty found in simulation. These experiments used five Create robots with mini PCI cards. Figure 10(b)

[k]Details of the hardware setup may be found at `http://enl.usc.edu/projects/peg/platform.html` and source code for running DCEE algorithms on the Create robot is at `http:/teamcore.usc.edu/dcop`.



(a) (b)

Fig. 10: (a) shows the total gain received by agents in the first set of experiments. Both SE-Mean and BE-Rebid successfully improve the team's reward. In figure (b), SE-Optimistic-1 is compared to SE-Optimistic-2 on chain and full graphs. In the chain graph, SE-Optimistic-1 outperforms SE-Optimistic-2, and the reverse is true for full graphs.

shows the average gain per round SE-Optimistic-1 and SE-Optimistic-2 on chain and full graphs, with 20 rounds each. Reward for this setup is again measured in terms of RSSI in decibels. However, results are not directly comparable because different numbers of robots are used in the different sets of experiments and different wireless vendors have different implementations of RSSI measurement utilities. The plots average ten trials and error bars show the standard error. In all cases, the algorithms improve the reward of the team, although SE-Optimistic-1 on the complete graph improves much more slowly. One possible reason for this discrepancy is that the gain achieved by the agents is dependent on the starting configuration — the worse the starting configuration, the more latitude exists for achievement. Unfortunately, the physical agents cannot be returned to exactly the same start state, and thus different trials have different initial signal strengths. The average team initial reward for the complete graphs when run using SE-Optimistic-1 was 574 $\pm$ 55, whereas the average for SE-Optimistic-2 was 506 $\pm$ 31, thus SE-Optimistic-1 has a relatively harder time improving the team's reward.

Figure 10(b) displays the total gain of robots in the second experiment, and Figure 11 shows the gain per round (the total gain is the area under the curve in Figure 11). We again see that SE-Optimistic-1 on a complete graph performs worse than all other algorithms. Most important is that these results confirm the team uncertainty penalty. In a chain graph, k=1 outperforms k=2, and in a complete graph, k=2 outperforms k=1. As such, these experiments on five robots confirm trends predicted in simulations of 10–50 virtual agents.

5. Understanding the Team Uncertainty Penalty

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

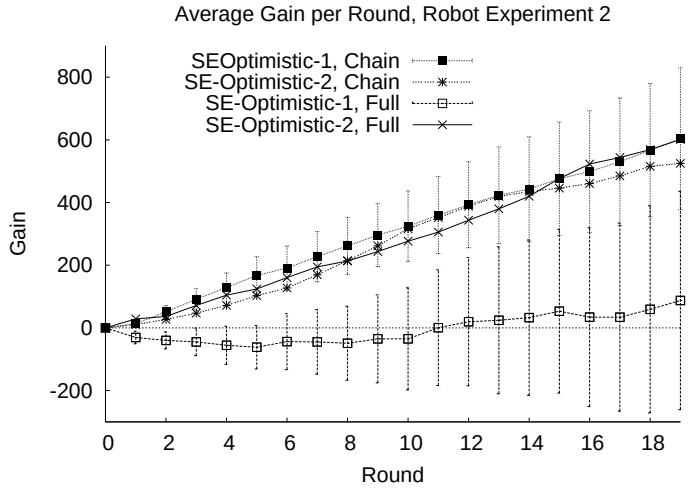


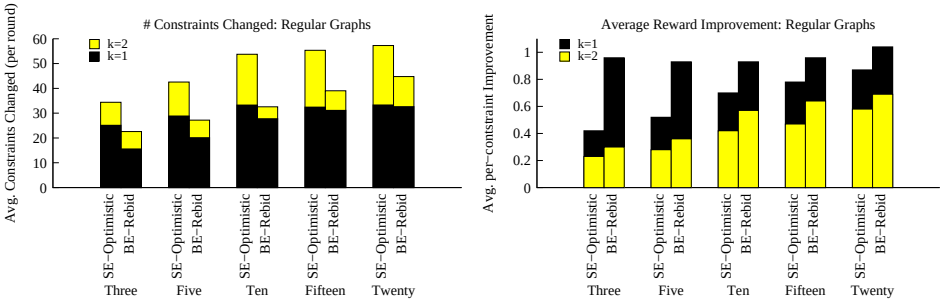
Fig. 11: Both SE-Mean-1 and BE-Rebid-1 successfully improve the team's performance over their initial configurations. Error bars, shown only on SE-Optimistic-1 for readability, display the standard error.

5.1. *Relative Performance of k=1 and k=2*

As discussed earlier, 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) and therefore achieve higher performance. 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 12(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. Data was averaged over 10 trials and each trial used 40 agents running for 100 rounds. This graph shows that k=2 does perform as expected, consistently 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 12(b) displays data from the same trials, but now the y-axis shows the average immediate reward improvement for an agent that changes variables during the experiment, normalized by the number of constraints. For example, agents using the SE-Optimistic-1 algorithm on density three graphs improved their reward by 0.41 per constraint. This equates to an improvement in reward of $0.41 \times 3 = 1.23$ per agent when an agent changes its variable setting.

Figures 12(a) and 12(b) 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 change fewer con-



(a) Avg. Constraints Changed (b) Avg. Reward Improvement

Fig. 12: The average number of constraints changed by an agent in a single round in (a) show that in regular graphs, k=2 algorithms always change more constraints. (b) shows average per-constraint improvement for different regular graphs: k=1 algorithms have higher improvement.

straints than k=2, on average. As the graph density increases, the ratio between the number of constraints changing in k=1 and k=2 *increases* (Figure 12(a), which we will refer to as *Trend #1*). In contrast, the ratio in the per-constraint improvement in k=1 and k=2 *decreases* as graph density increases (Figure 12(b), *Trend #2*). Thus, as the graph density increases, k=2 changes relatively more constraints than k=1 and the k=2 changes are relatively more effective than k=1. The following section further investigates these graphs and trends to provide additional insight. However, these two plots summarize the fundamental cause of the team uncertainty penalty.

Trend # 1

- In a regular graph of density three, SE-Optimistic-2 changes 30% more constraints per round then SE-Optimistic-1.
- In the regular graph with density twenty, SE-Optimistic-2 improves relative to SE-Optimistic-1, as now SE-Optimistic-2 changes 55% more constraints per round then SE-Optimistic-1.

Trend # 2

- In a regular graph of density three, SE-Optimistic-2 achieves an average gain per constraint that is 59% of the gain per constraint for SE-Optimistic-1.
- In the regular graph with density twenty, SE-Optimistic-2 improves relative to SE-Optimistic-1, as now SE-Optimistic-2 achieves 71% of the gain per constraint that SE-Optimistic-1 achieves.

5.2. *SE-Optimistic-1 vs. SE-Optimistic-2*

This section provides additional empirical analysis, providing insight to Figures 12(a) and 12(b). In particular, we address the following questions for SE-

Optimistic:

(1) In Figure 12(a), why does the average number of constraints changed increase as density increases for both SE-Optimistic-1 and SE-Optimistic-2?
(2) In Figure 12(a), why does the ratio between the number of constraints changed in k=1 and k=2 increase as the density increases (Trend #1)?
(3) In Figure 12(b), why does the average gain per variable of both SE-Optimistic-1 and SE-Optimistic-2 increase as graph density increases?
(4) In Figure 12(b), why does the ratio between the per-constraint improvement in k=1 and k=2 decrease as density increases (Trend #2)?

In Section 5.3, the performance of SE-Optimistic-1 and SE-Optimistic-2 will be contrasted with the performance of SE-Mean-1 and SE-Mean-2, as the SE-Mean algorithms do not suffer from the team uncertainty penalty.

5.2.1. *Number Constraints Changed Increases as Density Increases: Figure 12(a)*

Consider that there are two separate effects as graph density increases. (1) as the graph density increases, an agent that changes its variable will change more constraints, but (2) it will prevent more neighbors from changing their constraints (unless part of a joint move). To understand the interplay of these two effects, consider Table 3. The table reports experimental results on regular graphs with density 3, 20 and 39. An agent that changes its variable setting will change 3, 20, and 39 constraints, respectively. As the graph density increases, the fifth column shows that the number of agents moving decreases, but the eighth column shows that the total number of constraints increases. The net result is that as the graph density increases, the number of constraints changed will increase for both SE-Optimistic-1 and SE-Optimistic-2. In all cases, SE-Optimistic-2 allows more constraints to be changed than SE-Optimistic-1.

5.2.2. *k=2 Changes an Increasing Number of Constraints, Relative to k=1, as Density Increases: Figure 12(a)*

Trend #1 shown in Figure 12(a) can be understood by considering how the performance of SE-Optimistic-1 and SE-Optimistic-2 change in Table 3. In SE-Optimistic-2, roughly twice the number of agents move as in SE-Optimistic-1 for a given graph type (column 5). However, *moving twice the number of agents does not equate to changing twice the number of constraints.* For instance, in a regular 3 graph, moving one agent results in 3 constraints changing, but moving a pair of agents results in 5 constraints changing because the shared constraint is not double-counted. In the regular-39 graph, a single agent changes 39 constraints while a pair of agents changes 79 constraints.

As discussed above, as the graph density increases, both algorithms change more

Number of Agents and Constraints Changing per Round

Algorithm	Density	Single Agents	Pairs of Agents	Total # Agents	Single Agent Constraints	Pairs of Agents Constraints	Total Constraints
SE-Optimistic-1	3	8.4	0	8.4	25.2	0	25.2
SE-Optimistic-2	3	0.7	7.0	14.7	2.1	35	37.1
SE-Optimistic-1	20	1.7	0	1.7	34	0	34
SE-Optimistic-2	20	0.1	1.5	3.1	2.0	58.5	60.5
SE-Optimistic-1	39 (full)	1	0	1	39	0	39
SE-Optimistic-2	39 (full)	0	1	2	0	79	79

Table 3: Columns 3–6 of this table shows the average number of single agents, pairs of agents, the total number of agents that change variable settings per round. Columns 7–9 show the number of constraints that are changed by single agents, pairs of agents, and in total. All numbers are averaged over 150 trials with 40 agents. Recall that in k=2 algorithms, agents that do not form a pair may bid to change values on their own, which is why some single agents optimize in SE-Optimistic-2.

constraints. However, in low density graphs, SE-Optimistic-2 changes fewer constraints relative to SE-Optimistic-1 than in high density graphs. For instance, in regular-3 graphs, when a pair of agents acts, they change 5 constraints, whereas a single agent changes 3 constraints — the pair changes $\frac{5}{3}$ times as many constraints. In a full (regular-39) graph, a moving pair changes 79 constraints, while a single agent changes 39 — the pair now changes $\frac{79}{39}$ times as many constraints.

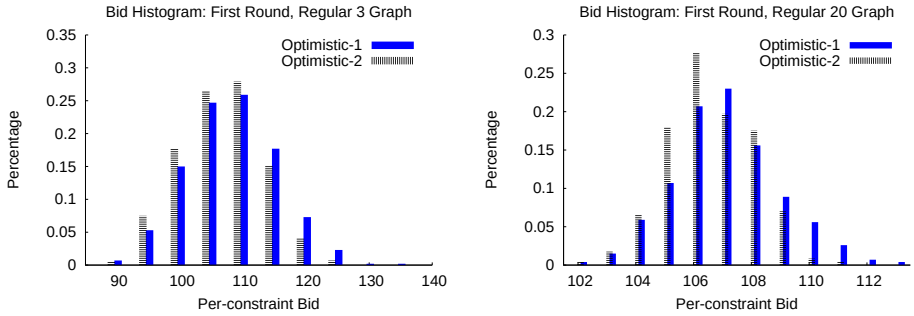
In summary, on different density graphs, SE-Optimistic-2 allows roughly twice as many agents to move as SE-Optimistic-1. However, as the graph density increases, moving pairs of agents changes relatively more constraints than moving single agents (i.e., from $\frac{5}{3}$ to $\frac{79}{39}$ in our example).

5.2.3. *Average Gain per Constraint Increases as Graph Density Increases: Figure 12(b)*

To explain why the average gain per variable change in both SE-Optimistic-1 and SE-Optimistic-2 increase as graph density increases, we first note that it is most instructive to consider the earliest rounds of a trial. As the number of rounds increases, the average bid approaches the mean and the average gain approaches zero. As seen earlier in Figure 4, Optimistic algorithms never stabilize, but will approach a constant team reward.

Figures 13(a) and 13(b) show histograms for the first round.[1] The x-axis shows the average bid (per constraint) of agents that are allowed to change variables, and the y-axis shows the percentage of constraints that have such a bid.

[1]If one instead considers such a histogram averaged over all rounds, as the total number of rounds in a trial increases, the distribution of bids shifts so that it approaches a Gaussian distribution around the mean of the per-constraint reward, which is 100 (not shown).



(a) SE-Optimistic, Density 3 (b) SE-Optimistic, Density 20

Fig. 13: These two histograms show the percentage of constraints that were changed on the first round in regular graphs of density 3 and 20, using algorithms SE-Optimistic-1 and SE-Optimistic-2. The results are averaged over 150 independent trials. In higher density graphs, no agents won the bid to move if their average per-constraint bid was less than 100 (i.e., the agent had a reward less than the average).

In the higher density graph, both SE-Optimistic-1 and SE-Optimistic-2 have a narrower distribution of bids (relative to the lower density graph), and *all* successful bids will result in a positive gain, on average. In higher density graphs there is more connectivity between agents and in order to win the bid to change variables and agent must "beat out" many competitors. In higher density graphs, agents must thus have a relatively high bid to change variables. Put differently, in *higher density graphs, more information is shared between agents and the team is better able to select the best agent to change variables* (i.e., the agent with the lowest reward).

Figure 14 shows the average bid (per constraint) on the x-axis and the average resulting gain (per constraint) on the y-axis for SE-Optimistic-1 and SE-Optimistic-2 agents. The linear relationship between the two shows that a high bid corresponds to a high gain in both algorithms. As graph density increases, both optimistic algorithms increase their average gain per constraint change as they are able to better limit movement to agents which have the most to gain.

5.2.4. *Average gain per k=2 constraint increases relative to k=1 as graph density increases: Figure 12(b)*

Finally, this section explains Trend #2, where the average gain per constraint for SE-Optimistic-2 increases relative to SE-Optimistic-1 as the graph density increases. As discussed above, both algorithms receive higher gain per constraint as the density increases. However, SE-Optimistic-2 improves more, as it has more room for improvement.

An alternate way of evaluating the algorithms is to consider the number of agents that make a "poor" move on the first round. For instance, consider the number of agents that change their variable when their average reward is greater

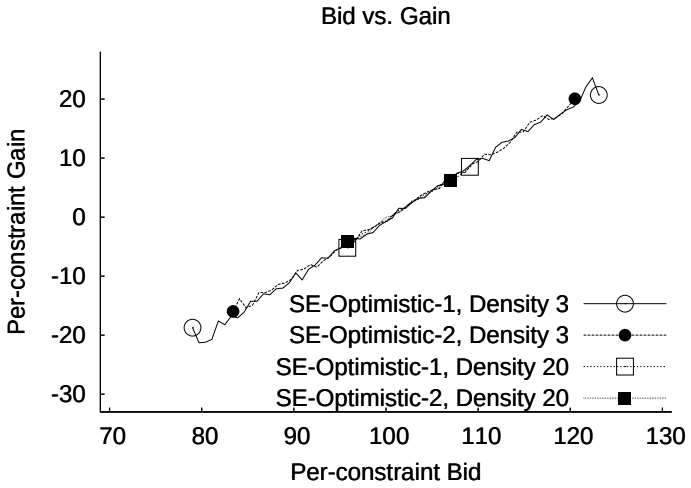
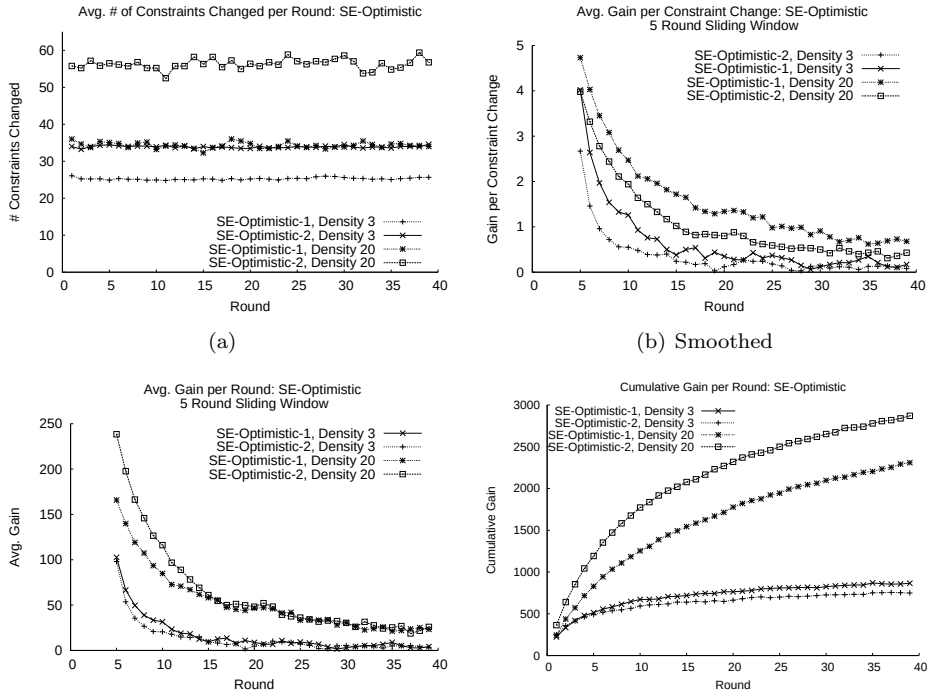


Fig. 14: This figure shows the average per constraint bid and corresponding per constraint gains for Optimistic algorithms. In all cases, the gain received is proportional to the bid, where a bid of 100 (the mean) will result in an average immediate improvement of zero. On average, a high bid will result in a high gain.

Algorithm	Density	# Agents changing when $r \geq \mu$
SE-Optimistic-1	3	0.82 ± 0.08
SE-Optimistic-2	3	2.57 ± 0.12
SE-Optimistic-1	10	0 ± 0
SE-Optimistic-2	10	0.04 ± 0.02
SE-Optimistic-1	20	0 ± 0
SE-Optimistic-2	20	0 ± 0

Table 4: This table reports the number of agents that move when their reward per constraint is not below the mean times the number of constraints (i.e., the agent does not have a positive expected value for moving). Data was collected from the first round of 150 trials using 40 agents each; reported errors are the standard error of averages.

than the mean per constraint (i.e., the agent does not have a positive expected gain). Table 4 shows that in density 3 graphs, both SE-Optimistic-1 and SE-Optimistic-2 allow agents to change variables when they do not have a positive expected gain (column 3). SE-Optimistic-2 allows many more agents to move than SE-Optimistic-1 in density 3 graphs. However, in density 10 graphs, SE-Optimistic-2 allows very few such moves, and in density 20 graphs, it allows none. Thus, at higher densities (where the agents receive information from their many neighbors), although SE-Optimistic-2 does not receive as high a gain as SE-Optimistic-1, it does not cause "poor" agents to move, which does happen at lower densities (where agents receive relatively little information from their neighbors).



(a)

(b) Smoothed

(c) Gain Per Round - smoothed

(d) Cumulative Gain

Fig. 15: (a) shows the number of constraints changed by SE-Optimistic algorithms on different graph densities. Results are averaged over 150 trials and each trial uses 40 agents. Recall that SE-Optimistic algorithms continually optimize, and thus the number of constraints changing is roughly constant throughout the trial. (b) shows the average change in reward per constraint, for every constraint that was changed, on each round in a trial. A 5-round sliding window is used to make the trends more apparent. On average, SE-Optimistic-1 has a higher reward improvement per constraint than SE-Optimistic-2 for both types of regular graph. (c) shows the average gain per round for the two SE-Optimistic algorithms on regular graphs of density 3 and 20. (d) shows the total gain per round for the same trials. Although the difference in gains between algorithms may be small, they compound over time to produce significantly different cumulative gains (and, thus, total rewards).

5.2.5. *Summary*

As discussed at the end of Section 5.1, two trends in Figures 12(a) and 12(b) combine to produce the team uncertainty penalty in SE-Optimistic: as graph density increases, the difference between the number of constraints changing in k=1 and k=2 *increases* while the difference in the per-constraint improvement in k=1 and k=2 *decreases*. The previous sections have discussed these two trends, as well as other underlying behavior. In particular,

- Moving twice the number of agents does not equate to changing twice the

number of constraints — it approaches a factor of two in high density graphs, but it is lower for lower density graphs. This helps explain why k=2 changes an increasing number of constraints, relative to k=1.

- In higher density graphs, more information in shared between agents and the team is better able to select the best agent to change variables. Agents in low density graphs are less knowledgeable and there should be extra restrictions on their movement.

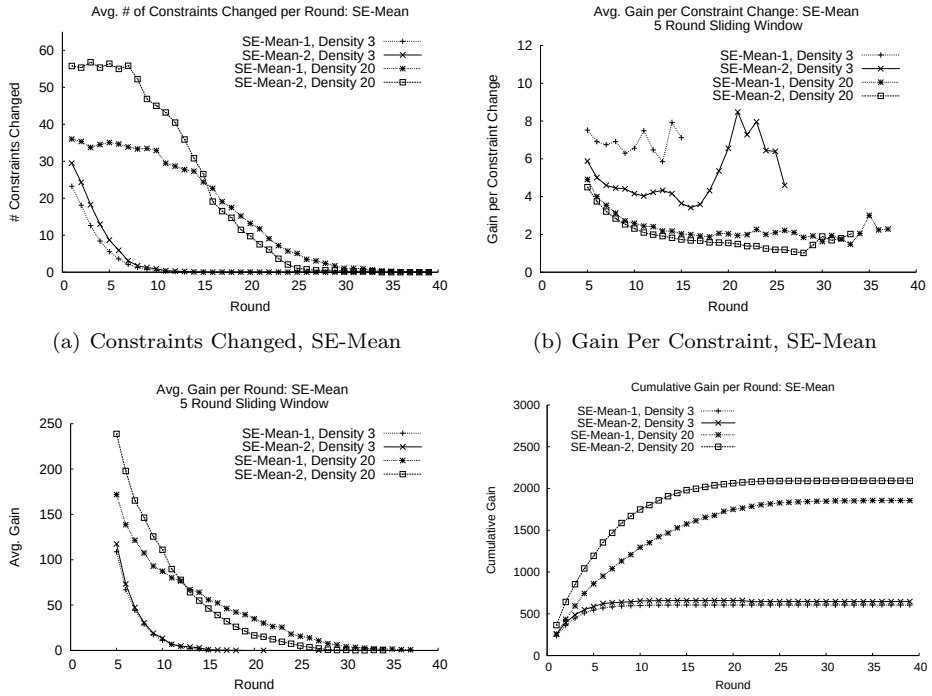
Figures 15(a)–15(d) summarize the performance of SE-Optimistic-1 and SE-Optimistic-2. Figure 15(a) reports the number of constraints changed per round. Figure 15(b) shows the gain per constraint change, smoothed with a 5-round sliding window. Figure 15(c) reports the average gain per round, again with a 5-round sliding window. Finally, Figure 15(d) reports the total gain per round. Recall that the goal of DCEE algorithms is to maximize the on-line reward, corresponding to maximizing the area under this curve.

In the following section, we discuss SE-Mean using figures analogous to Figures 15(a)–15(d) and highlight the differences which allow SE-Mean-2 to always outperform SE-Mean-1, avoiding the team uncertainty penalty.

5.3. *In Contrast: SE-Mean*

Recall that in Figure 7(a) SE-Mean-2 always outperforms SE-Mean-1 — it does not exhibit the team uncertainty penalty. The primary reason is that the algorithm is very *conservative* in its bidding. SE-Mean agents assume that a changed variable will only return the mean link value, causing it to drastically lower its bids relative to SE-Optimistic. However, avoiding the penalty comes at a cost: recall that in Figure 7(a) both SE-Mean algorithms underperform the SE-Optimistic algorithms (with the exception of graphs with random topology, which the SE-Optimistic algorithms do not handle well).

Figure 16(a) demonstrates the "conservative" nature of SE-Mean-1 and SE-Mean-2 on density 3 and density 20 graphs: the number of constraints changed quickly converges to zero. In contrast, recall that SE-Optimistic continues to optimize throughout the trial (Figure 15(a)). Similar to SE-Optimistic, SE-Mean-1 generally has a higher per-constraint gain then SE-Mean-2 (Figure 16(b)). Unlike SE-Optimistic-1 and SE-Optimistic-2, SE-Mean-1 and SE-Mean-2 have relatively high gains per constraint in density three graphs. In low density graphs, both SE-Mean algorithms quickly stop allowing most agents to move and only the agents with very low rewards (and thus very high gains) are allowed to move. This behavior also explains why the lines in Figure 16(b) are truncated as the number of agents moving reaches zero. Although SE-Mean receives higher reward per constraint change than SE-Optimistic, it changes many fewer constraints per round, and the algorithm quickly converges. Thus the total gain from the SE-Mean algorithms is lower than that of the SE-Optimistic algorithms (compare Figures 15(d) and 16(d)).



(a) Constraints Changed, SE-Mean

(b) Gain Per Constraint, SE-Mean

(c) Gain Per Round, SE-Mean

(d) Cumulative Gain Per Round, SE-Mean

Fig. 16: These figures report the number of constraints changed, the gain per constraint change, the gain per round, and the cumulative gain for SE-Mean-1 and SE-Mean-2 on regular three and regular twenty graphs. Each line averages 150 independent trials.

Most relevant to the team uncertainty penalty, the average gain per round of SE-Mean-2 does not decrease faster than SE-Mean-1 in the density 3 graph, resulting in SE-Mean-2 outperforming SE-Mean-1 in terms of both the gain per round (Figure 16(c)) and the cumulative reward improvement (Figure 16(d)), while the reverse is true for SE-Optimistic-1 and SE-Optimistic-2 (Figures 15(c) and 15(d)).

Recall that the goal of the algorithms is to maximize the on-line reward, which equates to maximizing the area under the cumulative gain curve. SE-Mean-2 does not suffer from the team uncertainty penalty because it changes many fewer constraints, focusing on those agents which have the highest expected gain from moving. Agents never move if their average per-constraint reward is already greater than the mean (a table showing the number of agents that changed variables when $r \geq \mu$, as in Table 4, would always show zero agents changing), which avoids the team uncertainty penalty at the cost of overall performance.

5.4. *BE-Rebid-1 vs. BE-Rebid-2*

Similar to the SE-Optimistic-2 algorithm, BE-Rebid-2 can underperform BE-Rebid-1 on sparse graphs (see Figure 7(b)). As discussed in Section 5.1, the two trends in Figures 12(a) and 12(b) combine to explain the team uncertainty penalty. Sections 5.2.1–5.2.4 explain these trends and graphs for both the SE-Optimistic and SE-Rebid algorithms, with one exception, described next.

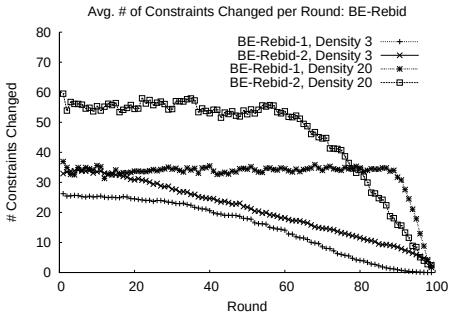
In Figure 12(b), both SE-Optimistic-1 and SE-Optimistic-2 receive increased benefit to changing constraints as the graph density increases. BE-Rebid-2 follows this pattern, but BE-Rebid-1 does not — the gain from changing a constraint is roughly constant in BE-Rebid-1 over different graph topologies. However, Trend #2 (the average gain per constraint for SE-Optimistic-2 increases relative to SE-Optimistic-1 as the graph density increases) holds because BE-Rebid-2's improvement per-constraint change increases with increasing graph density.

The main change in behavior of the BE-Rebid algorithms from the SE-Optimistic algorithms can be seen by comparing Figures 15(a) and 15(b) with Figures 17(a) and 17(b). SE-Optimistic-1 and SE-Optimistic-2 greedily attempt to improve the team reward over time and thus the number of constraints changed remains roughly constant, whereas BE-Rebid-1 and BE-Rebid-2 are both aware of the number of rounds remaining. This extra knowledge allows them to calculate the value of exploration and exploitation, resulting in fewer agents changing values over time, except when there is a very high chance of a large improvement (e.g., very few agents move near the end of a trial, but those that do receive very high gains on average). Despite these differences, the gain per round of BE-Rebid-2 quickly drops below that of BE-Rebid-1 on the density three graph, as seen in Figures 17(c) and 17(d), leading to BE-Rebid-1 outperforming BE-Rebid-2 on low density graphs.

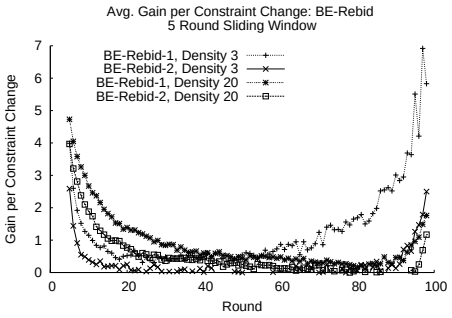
5.5. *BE-Stay-1 and BE-Stay-2*

As was the case with SE-Mean relative to SE-Optimistic, BE-Stay is more conservative than BE-Rebid. BE-Stay agents cannot backtrack and therefore down-weight exploration because they are unable to exploit previously discovered constraint settings. When comparing the performance BE-Stay (Figures 17(e)–17(h)) to SE-Rebid (Figures 17(a)–17(d)), one can see that BE-Stay changes relatively fewer constraints. BE-Stay-2 always outperforms BE-Stay-1, as in the case of SE-Mean.

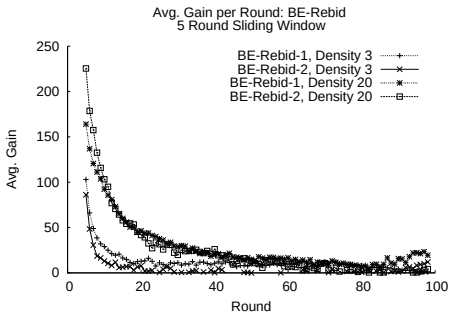
Again similar to SE-Optimistic and SE-Mean, the BE-Rebid algorithms change almost an order of magnitude more constraints than do the BE-Stay algorithms (i.e., the area under the curve in Figure 17(a) is roughly ten times that under the curve in Figure 17(e)). Thus, even in low density graphs, agents only change values when there is a very good chance of increasing the team's reward, resulting in a higher gain per constraint change in the first half of the trial of the BE-Stay algorithms (Figure 17(f)) compared to the BE-Rebid algorithms (Figure 17(b)). Because relatively few agents move in BE-Stay algorithms, relative to BE-Rebid, the gain per round (Figure 17(g)) and total gain (Figure 17(h)) are lower than in



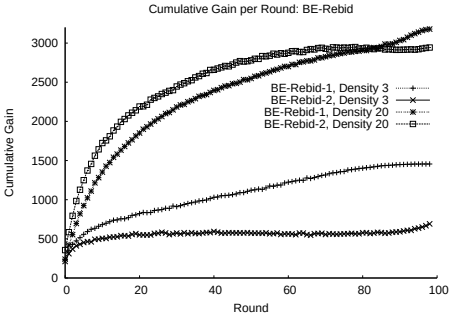
(a) Constraints Changed, BE-Rebid



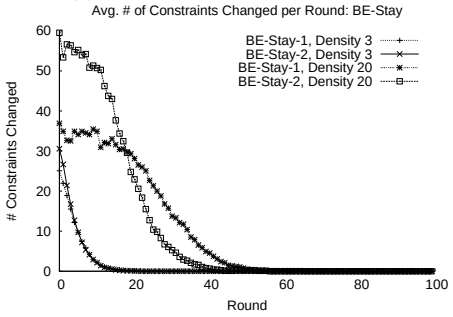
(b) Gain Per Constraint, BE-Rebid



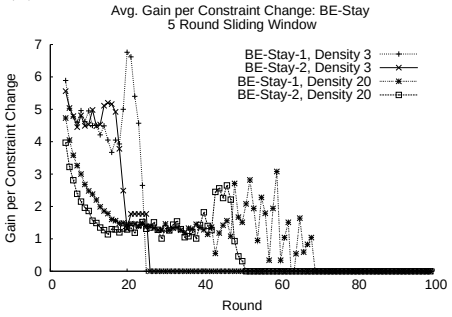
(c) Gain Per Round, BE-Rebid



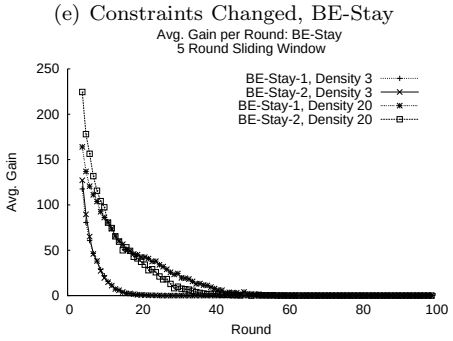
(d) Cumulative Gain Per Round, BE-Rebid



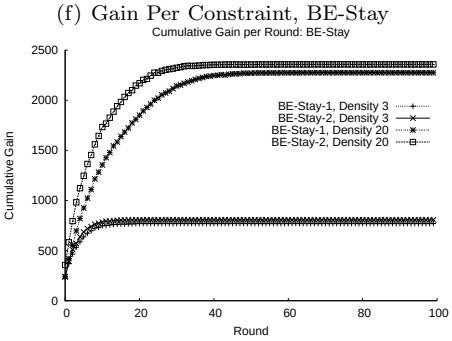
(e) Constraints Changed, BE-Stay



(f) Gain Per Constraint, BE-Stay



(g) Gain Per Round, BE-Stay



(h) Cumulative Gain Per Round, BE-Stay

Fig. 17: (a)–(d) summarize the behavior of BE-Rebid-1 and BE-Rebid-2 over time and can be contrasted with BE-Stay-1 and BE-Stay-2 in (e)–(h).

BE-Rebid. Although BE-Stay does not suffer from the team uncertainty penalty, one or both of BE-Rebid-1 and BE-Rebid-2 will outperform BE-Stay-1 and BE-Stay-2 (Figure 7(b)).

6. DCEE Algorithm Extensions: Avoiding the Team Uncertainty Penalty

This section revisits the SE-Optimistic-2 and BE-Rebid-2 algorithms and proposes two extensions that help ameliorate the team uncertainty penalty. In principle, one could evaluate a graph and 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.*

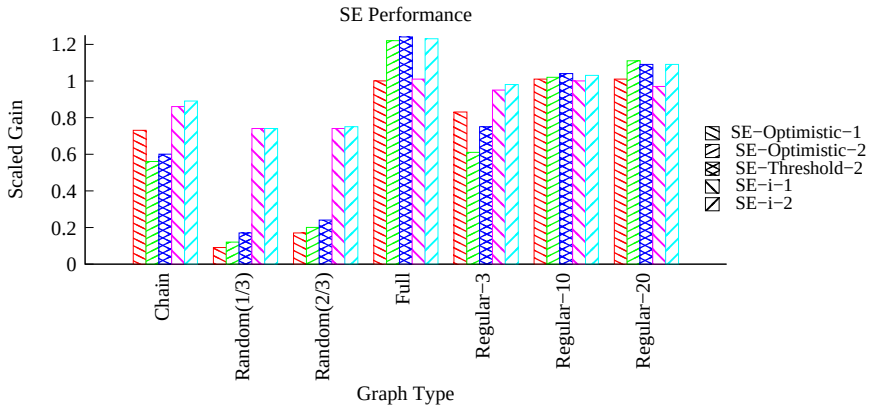
6.1. *Discouraging Joint Actions*

The first approach taken to decrease the team uncertainty penalty is to discourage agents from executing joint actions with low bids. In DCEE algorithms, pairs of agents that have relatively high rewards will make relatively low bids. As one may expect (c.f., Figure 14), agents with low bids (which 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 τ units of reward per constraint in the next round, the algorithm disallows the joint action and reverts to k=1. Said differently, this method parameterizes the decision of when to use teamwork, allowing agents to form a team only if they expect to receive a high reward improvement (and thus are less likely to receive a low, or negative, change in reward).

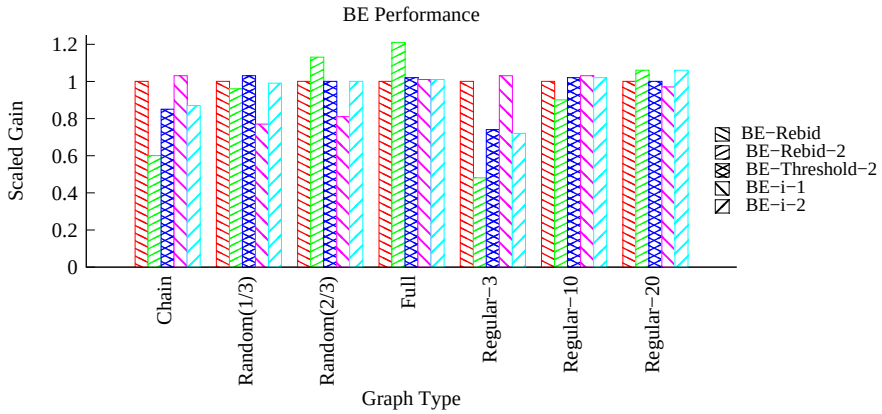
SE-Threshold-2 is identical to SE-Optimistic-2 except that agents are allowed to form a pair only if their bid will be at least $\tau\times$*(number of agents that neighbor the pair).* **BE-Threshold-2** is similar: pairs may form only if the pair has a bid above $\tau\times$*(number of agents that neighbor the pair).* This change is enacted by adding an extra conditional to Algorithm 2, line 9. Agents only execute joint actions when there should be a significant advantage to the joint move and otherwise "play it safe" with k=1.

Figures 18(a) and 18(b) shows the performance of SE-Threshold-2 and BE-Threshold-2, respectively. In these two graphs, both algorithms use a single value of τ for all trials, 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, $p < 0.05$, but underperforms SE-Optimistic-1, $p < 0.05$), and dominates SE-Optimistic-1 on high density graphs. Likewise, BE-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) Comparing SE-Optimistic-1 and SE-Optimistic-2 with SE-Threshold-2, SE-i-1, and SE-i-2



(b) Comparing BE-Rebid-1 and BE-Rebid-2 with BE-Threshold-2, BE-i-1, and BE-i-2

Fig. 18: (a) shows the scaled gain for SE-Threshold-2 with $\tau = 100$, SE-i-1 and SE-i-2 with $i = 110$, and compares it to SE-Optimistic-1 and SE-Optimistic-2. Similarly, (b) shows the performance of BE-Threshold-2 using $\tau = 15$, BE-i-1 and BE-i-2 using $i = 4$, and compares to BE-Rebid-1 and BE-Rebid-2.

A second set of results tested how the algorithms performed on a single graph type with different parameter settings. Figure 19(a) shows the results of BE-Threshold-2 on regular graphs of varying densities. To tune our algorithms, we tested roughly ten different values of τ for both SE-Threshold-2 and BE-Rebid-2,

and selected the best parameter value, per graph type, for each algorithm. Tuning the threshold parameter for SE-Threshold-2 produces qualitatively similar behavior. 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 τ (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.

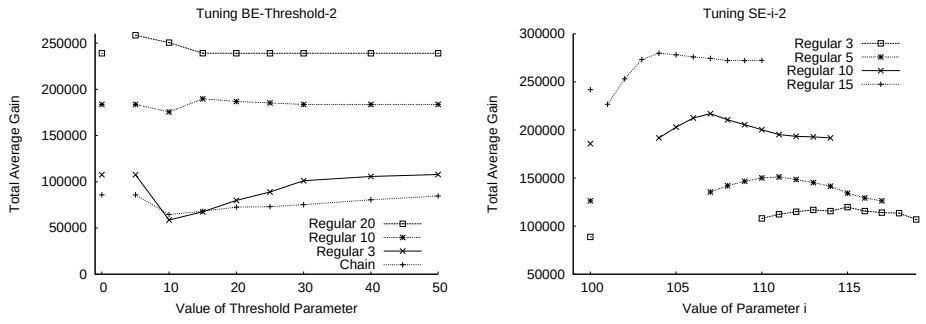
6.2. *Discounting All Bids Under Uncertainty*

The second approach taken to decrease the team uncertainty penalty is to discount *all* bids. Both SE-Optimistic-2 and BE-Rebid-2 receive negative average reward when they have 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.[m]

SE-i-1 and **SE-i-2** are similar to SE-Mean-1 and SE-Mean-2 in that all unexplored variable assignments are assumed to receive a reward per link of i (where $i{=}\mu$ for SE-Mean), effectively determining the calculated utility of exploration. While SE-Optimistic agents change their values throughout all trials, SE-i agents will stop optimizing once all agents have a reward of at least $i{\times}NumberLinks$. **BE-i-1** and **BE-i-2** generalize the BE-Rebid algorithms so that all utilities for the explore action are discounted, proportional to the number of constraints that would be changed: the value of exploration is discounted by $i{\times}NumberLinks$. This shift in utility discourages agents from being overly optimistic with their bids and encourages agents to exploit (i.e., backtrack), rather than explore.

Figure 18(a) 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 other static estimation methods. The discount factor i allows agents with many neighbors to more easily decline to move so that they do not dominate their neighbors (unlike SE-Optimistic, as discussed earlier in Section 4.2). SE-i-2 is the highest performing static estimation algorithm. Additionally, SE-i-2 also outperforms SE-i-1 — SE-i-2 does not suffer

[m]Although BE-Rebid-2 calculates the value of exploration using decision-theoretic reasoning, our results have shown that this estimate can be incorrect, and is more incorrect for $k{=}2$ in low density graphs. Were the decision-theoretic calculations flawless, such a correction would not be needed. However, the gain calculations in all algorithms are only approximations, although the BE-Rebid calculations are substantially more sophisticated than in SE-Optimistic.



(a) (b)

Fig. 19: The performance of (a) BE-Threshold-2 and (b) SE-i-2 changes as their parameters are tuned. In (a), the x-axis show the value of τ, the threshold parameter, and τ=0 is equivalent to BE-Rebid-2. In (b), the x-axis show the value of i, and i=100 is equivalent to SE-Mean-2 (SE-Optimistic-2 would be equivalent to i=200, which is not shown).

from the team uncertainty penalty. SE-i-2 is the highest performing static estimation algorithm, given a reasonable value of i.

The performance of BE-i-2 in Figure 18(b) 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 balanced exploration algorithm, in that it more often outperforms BE-i-2 than not, and reduces the team uncertainty penalty (relative to BE-Rebid-2). 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 when 40 agents are fully connected.

Figure 19(b) is analogous to Figure 19(a): it shows how the performance of SE-i-2 changes on a single graph type as the parameter i is varied (tuning BE-i-2 over a range of parameters produces qualitatively similar results). 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, dominating all other SE algorithms. 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.

6.3. *Analysis: Reducing the Team Uncertainty Penalty*

SE-Threshold-2 does perform better than the worst of SE-Optimistic-1 and SE-Optimistic-2, and sometimes outperforms both. Put differently, if one does not know the topology of the DCEE in advance, SE-Threshold-2 may be a safer choice than either of the SE-Optimistic algorithms because its performance does not suffer on low density graphs (like SE-Optimistic-2) and still performs well on high density graphs. SE-Threshold-2 also outperforms SE-Mean-1 and SE-Mean-2 on all graphs except random topologies (see Section 4.2 for a discussion of why SE-Optimistic, and by extension SE-Threshold-2, suffers on random graphs).

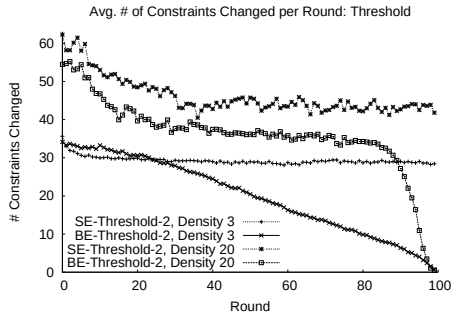
SE-Threshold-2 is able to successfully avoid the team uncertainty penalty by only allowing agents to execute joint moves when their expected gain is above some threshold. This behavior is evidenced by SE-Threshold-2 changing more variables over the course of a trial than SE-Mean-2 (compare Figures 20(a) and 16(a)), but fewer variables than SE-Optimistic-2 (compare Figures 20(a) and 15(a)). The remainder of the SE-Threshold-2 figures (Figures 20(b)–20(d)) follow from the SE-Threshold-2 algorithm being more conservative than SE-Optimistic-2, but more aggressive than SE-Mean-2.

Similar to SE-Threshold-2, BE-Threshold-2 does at least as well as the worst of BE-Rebid-1 and BE-Rebid-2. BE-Threshold-2 also outperforms BE-Stay-1 and BE-Stay-2, with the exception of full graphs. BE-Threshold-2 changes more constraints per trial than BE-Stay-2 (compare Figure 20(a) with 17(e)), but fewer variables than BE-Rebid-2 (compare Figures 20(a) and 17(a)). The remainder of the BE-Threshold-2 graphs follow from this change in behavior.

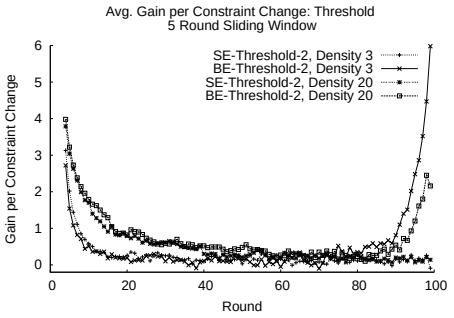
SE-i-2 is the most successful static estimation algorithm and should generally be preferred — it performs at least as well as the best SE-Optimistic algorithm, except for regular twenty graphs, and thus is a very safe choice. SE-i-2 also outperforms SE-i-1 on all tested DCEE topologies. SE-i-2 does not suffer from the team uncertainty penalty. SE-i-2 always performs better than SE-Mean-1 and SE-Mean-2. The performance of SE-i-1 and SE-i-2 is summarized in Figures 21(a)–21(d). Similar to SE-Threshold-2, SE-i-2 changes more variables per trial than SE-Mean-2 but fewer variables per trial than SE-Optimistic-2 (see Figure 21(a)). Most important, because the gain per round of SE-i-2 is higher than SE-i-1, even for density 3 graphs (see Figure 21(c)), SE-i-2 does not suffer from the team uncertainty penalty.

BE-i-2 is not as successful as SE-i-2, but it does perform at least as well as the worst of BE-Rebid-1 and BE-Rebid-2, making it a safe choice if the DCEE topology is unknown. The analysis of BE-i-1 and BE-i-2 is summarized in Figures 21(e)–21(h). BE-i-2 does suffer from the team uncertainty penalty — BE-i-1 outperforms BE-i-2 on low density graphs (see Figure 21(g)). BE-i-2 outperforms BE-Stay-1 and BE-Stay-2, with the exception of a fully connected graph. As in SE-i-2, BE-i-2 changes more constraints (Figure 21(e)) than BE-Stay-2 but fewer than BE-Rebid-2.

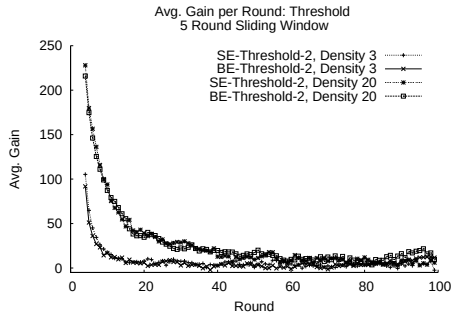
Taken together, the results in this section present encouraging results — the team uncertainty penalty can be reduced or even avoided. However, both types of



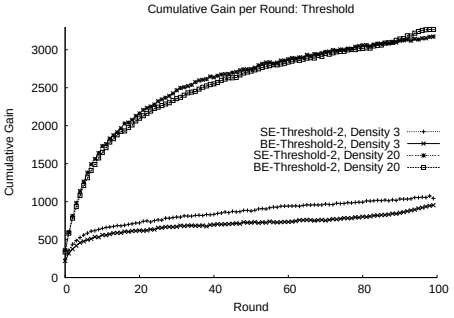
(a) Constraints Changed, Threshold Methods



(b) Gain Per Constraint, Threshold Methods



(c) Gain Per Round, Threshold Methods



(d) Cumulative Gain Per Round, Threshold Methods

Fig. 20: These figures show the behavior of SE-Threshold-2 and BE-Threshold-2 over time. All results are averaged over 150 trials.

algorithm introduced in this section require tuning an extra parameter.

7. Related Work

This article focuses on a class of problems that traditional DCOP algorithms could not address. We show that such real world domains raise new challenges: (1) agents do not know the initial payoff matrices, (2) the goal is to maximize the total reward instead of the final reward, and (3) agents have insufficient time to fully explore the environment. These challenges open up a new area for DCOP research, as current DCOP algorithms cannot be directly applied. This section discusses related work and approaches to such distributed problems.

Related work in DCOPs has been discussed in earlier sections. While there has been significant previous work in sensor networks [12, 23, 52, 41], none of it uses distributed constraint reasoning and handle unknown rewards.

A number of other works on mobile ad-hoc networks for communications (c.f., Cheng et al. [6], Marden et al. [29], and Correll et al. [7]) are based on other techniques (e.g., swarm intelligence, potential games, or other robotic approaches).

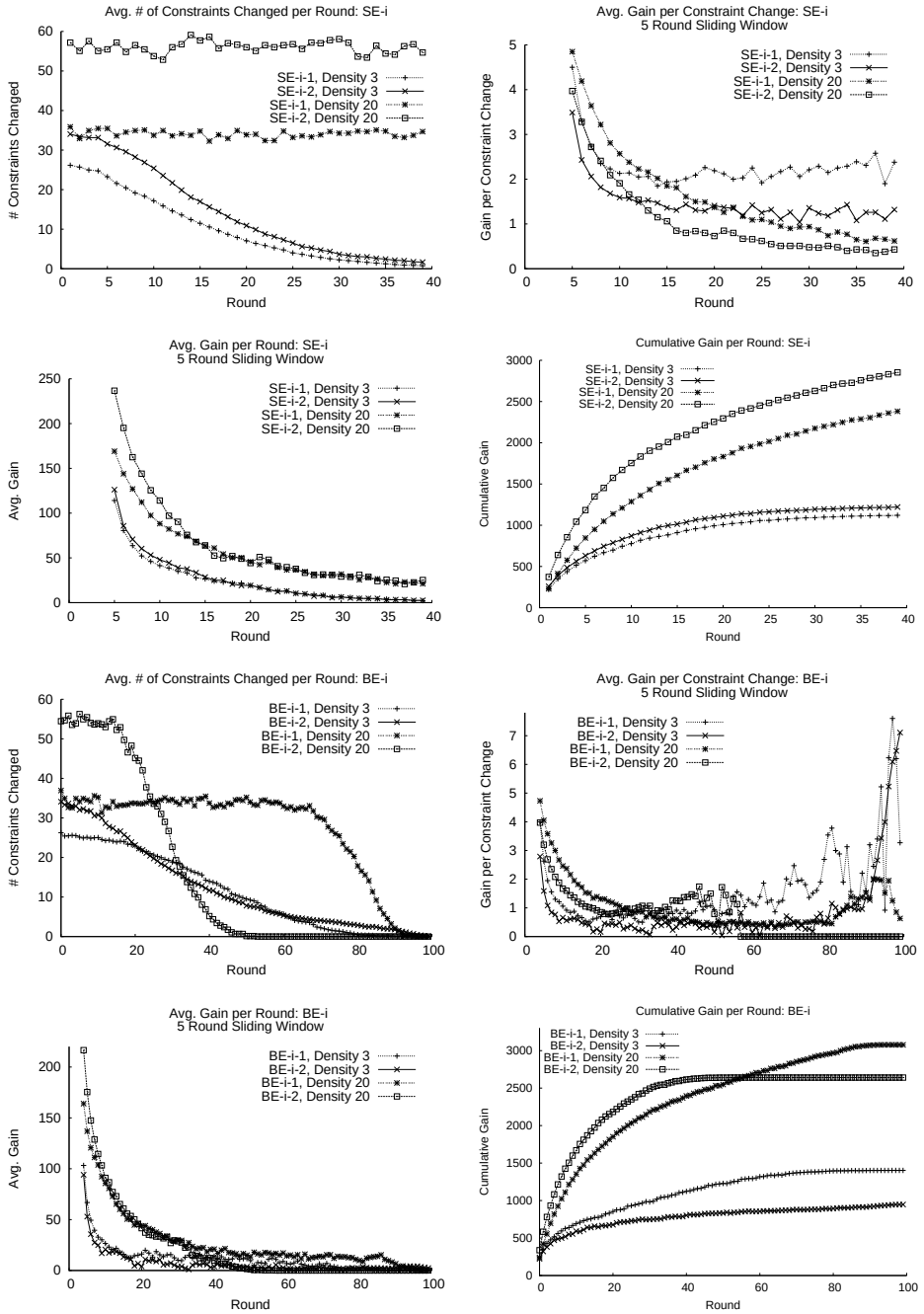


Fig. 21: These figures show the behavior of SE-i-1, SE-i-2, BE-i-1, and BE-i-2 over time. All results are averaged over 150 trials.

Instead, we extended DCOPs as they can scale to large tasks using local interactions. Other researchers have examined lower-level concerns in mobile wireless networks (c.f., the LANdroids,[n] GUARDIANS,[o] and Space-Time Processing for Tactical Mobile Ad-Hoc Networks projects[p]) rather than explicit multi-agent coordination issues. Farinelli et al. [11] also perform decentralized coordination on physical hardware using factor graphs, however, rewards are known and cumulative reward is not considered.

In multi-agent work, teamwork is typically considered beneficial, although if too much information is shared, computational requirements will explode [9]. 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 [19] has focused on "level of cooperativeness" in *distributed constraint satisfaction problems* (DisCSPs), as well as on the size of a mediation group to select asynchronous partial overlay DisCSP algorithms [2]. 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. [53] analyze the distributed stochastic search algorithm (DSA) and show that it often performs better than the distributed breakout algorithm (DBA) along multiple dimensions, encouraging many researchers to extend the DSA framework (c.f., Zivan [54]). 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 in Section Appendix C showed that our MGM-based algorithms substantially outperformed DSA algorithms if DSA's free parameter was not carefully tuned.

While the majority of DCOP and DisCSP work has assumed all rewards are known, a few works have addressed uncertainty in the environment in settings different from DCEE. For instance, Lass et al. [36] consider the case where a DCOP's reward matrix can change over time in an unknown way. Their solution focuses on solving the new problem instance quickly without accounting for any exploration. Brito et al. [5] consider that an agent may only know part of the environment, e.g., the agents are self-interested and do not share all their knowledge. Lastly, *open constraint programming* [10] explicitly reasons about exploration. In their work, centralized algorithms try to solve the satisfaction problem, and if no solution exists, agents explore — there is no notion of trading off exploration with exploitation to maximize the on-line reward.

Optimal stopping problems are a well-studied class of problems where one must decide when to cease drawing random numbers in order to maximize some objective.

[n] `http://www.darpa.mil/ipto/programs/ld/ld.asp`
[o] `http://vision.eng.shu.ac.uk/mmvlwiki/index.php/GUARDIANS`
[p] `http://zeidler.ucsd.edu/muri/pages/index.php`

One popular instance of such problems is the *secretary problem* [14]. While many different formulations of the original problem exist, to the best of our knowledge, all solved problems of this type focus on the final rank of the selected instance rather than any on-line metrics. Furthermore, the mobile ad-hoc network problem is multiagent, while the optimal stopping problems are all single agent.

An alternate way of describing the DCEE problem is to think of it as an extension to the *multi-armed bandit* [37] problem. In this single agent problem, an agent has a number of possible actions available and it does not know the payoff of each action. The goal of such algorithms is typically to balance exploration with exploitation so that the agent's on-line reward approaches the optimal reward (i.e., minimizes the agent's *regret* relative to always choosing the best action). A DCEE is significantly harder because it is a distributed multi-agent multi-armed bandit. Not only does the team's reward depend on the combination of actions that are executed by the different agents, but we assume there are (in effect) an infinite number of actions.

Reinforcement learning (RL) [42] is a popular approach to multiagent learning for sequential decision tasks with limited feedback that use a Markov Decision Process (MDP) formulation. When using RL in multiagent domains, there are three common approaches:

(1) Every agent learns separately and treats other agents as part of the (non-stationary) environment (c.f., Stone et al. [40]). While this approach enables agents to use unmodified single-agent RL algorithms, learning is likely to converge with poor performance, or fail to converge to a stable policy.
(2) Agents collaborate to find optimal behaviors using a construct like a *coordination graph* [17] (which applies to factored MDPs). For instance, Kok and Vlassis [21] use a reinforcement learning approach that applies to multi-agent tasks with coordination graphs. Their model allows agents to learn a value for each variable setting but uses only a single state, where agents take actions to select a particular variable setting and then return to the same state (similar to learning $Q(\cdot,a)$ values). Additionally, they do not explicitly reason about the length of the experiment, while our analysis of the BE algorithms have shown that changing the exploration/exploitation trade-off based on the amount of time remaining in an experiment can be beneficial.
(3) If the task is fully observable, a centralized approach like MMDP [4], can successfully treat the entire system as a single agent and reason over the entire joint action space. While this approach is P-complete, in many real-world domains every agent cannot observe the entire state space. If the task is fully observable only if agents pool their state information, a *DEC-MDP* approach [3] may be used. However, this approach is NEXP-complete and the computational requirements balloon with the number of agents in the system. While the MMDP and DEC-MDP formulations allow for stochastic transitions, DCEE focuses on problems with deterministic transitions.

8. Conclusion and Future Work

This article focuses on a class of problems that DCOPs could not address before. This is the first application of DCEE that demonstrates improvement in performance in a physical problem with uncertainty. We show that such real world domains raise new challenges: (1) agents do not know the initial payoff matrices, (2) the goal is to maximize the total reward instead of the final reward, and (3) agents have insufficient time to fully explore the environment. These challenges open up a new area for DCOP research, as current DCOP algorithms cannot be directly applied. We present and empirically compare two classes of novel DCEE algorithms addressing these challenges. We also present results from the DCEE algorithms implemented on physical robots. Our results show significant improvement in the reward function in mobile ad-hoc networks. Our experiments demonstrate the superiority of decision theoretic approaches, but static estimation strategies perform well on fully connected graphs or when task time horizon is small.

An additional contribution of this article is establishing that increased teamwork under uncertainty may sometimes degrade performance. We have 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. While both approaches help alleviate this penalty in static estimation algorithms, they are less effective in the balanced exploration algorithms. Additional research is needed before we know how often the team uncertainty penalty appears in practice, or if it is possible to use our knowledge of the penalty to design algorithms which do not experience the team uncertainty penalty.

The DCEE framework and algorithms presented in this article lead to a number of open questions. In the future, we are interested in exploring problems where there is more prior knowledge available to agents about the reward distribution. For instance, when deploying agents in a mobile ad-hoc network, an engineer may have some idea about what would be good positions for the agents. In this case, the agents should be able to robustly use this prior information, allowing them to improve their reward much faster, while limiting the penalty to the team's performance if the prior information is incorrect. Other exciting directions include applying DCEE algorithms to different problems, such as channel allocation, and examining more topologies, such as scale-free networks.

We are also interested in making the DCEE framework more flexible. For instance, there may be problems in which

- Reward distributions may be different for different agents or constraints.
- Reward distributions may be time dependent.
- The topology may change over time.
- The experiment runs infinitely long.
- etc.

We believe that DCEE algorithms could be adapted to address such scenarios, but these explorations are left for future work.

An important open question is whether the parameters i and τ in the SE-i/BE-i and SE-Threshold-2/BE-Threshold-2 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.

The investigation of the team uncertainty penalty in this article does not make it clear which reward distributions or domains will suffer from the penalty. The experiments that compare DCEE algorithms to the Omniscient algorithms suggest that uncertainty in the reward function is necessary for the penalty to manifest, but it is unclear what conditions are sufficient for the penalty to affect team reward. Our suspicion is that there will be many other settings where such a penalty exists, such as in multi-agent learning.

Finally, additional experiments on robots would be necessary before DCEE could be deployed on a fully autonomous mobile ad-hoc network. For instance, tests with many different robots interacting simultaneously would be useful to verify the simulator results with large numbers of agents. In principle, other reward metrics like maximizing throughput or battery life, and other distributions of rewards (e.g., uniform) should be easy to incorporate into the DCEE framework, but it is possible that additional unexpected and interesting phenomena will be discovered.

Appendix A. k=3 Algorithms

The DCEE algorithms presented in this article could be extended to arbitrary levels of k, but our code base currently requires each algorithm to be re-implemented for every value of k desired.[q] This section explains the two k=3 algorithms used in experiments.

Omniscient-3 is similar to Omniscient-2, but now up to three agents can make a joint move. The primary change is that agents first attempt to form a triple (Algorithm 3, lines 8–14). If that fails, it attempts to form a pairs (lines 18–24). Finally, if it is unable to form a pair, agents will attempt to change variables alone (line 28). Because of the number of possible teams considered, this algorithm requires a significant increase in the number of messages compared to Omniscient-2, particularly when the agents have many neighbors, as *getMaxGainAndAssignmentForTriple()* must consider all possible groups three neighboring agents (that include itself). For each group of three agents, it computes the optimal assignment for each of the three agents, assuming none of the neighbors of the three agents change their variables. This added complexity guarantees 3-optimal solutions, which generally outperform 2-optimal solutions.

SE-Optimistic-3 extends SE-Optimistic-2 so that sets of three agents can make joint moves. It differs from Omniscient-3 in how the functions *getMaxGainAndAs-*

[q]A general-k-optimal DCOP algorithm has only been recently introduced [20] but such an algorithm could be used to create a general-k-movement DCEE algorithm in the future.

Algorithm 3 PSEUDOCODE FOR k=3 ALGORITHMS

for each neighbor i **do**
  Send variable assignment and reward matrices to i
  Receive variable assignment and reward matrices from i
Aggregate all information
for each neighbor i **do**
  Send all aggregated information to i
  Receive aggregate information from i
Find the triple with the maximum gain, g, with agents p and p', and in which my variable assignment is a:
$g, p, p', a \leftarrow$ *getMaxGainAndAssignmentForTriple()*
AcceptCount $\leftarrow 0$
Send `OfferTriple` to form triple to agents p and p'
for all `OfferTriple` messages received **do**
  if agent requesting to form triplet $\in \{p, p'\}$ **then**
    Send `Accept` to offering agent
    AcceptCount $\leftarrow$ *AcceptCount* + 1
Receive responses from neighbors, if any
if (Did not receive `Accept` messages from both from agent p and agent p') or (*AcceptCount* $\neq 2$) **then**
  $p, p' \leftarrow \emptyset, \emptyset$
  Find maximum gain, neighbor to pair with, and variable assignment:
  $g, p, a \leftarrow$ *getMaxGainAndAssignmentForPair()*
  Send `OfferPair` to p
  AcceptCount $\leftarrow 0$
  for all `OfferPair` messages received **do**
    if agent requesting to form pair is p **then**
      Send `Accept` to p
      AcceptCount $\leftarrow$ *AcceptCount* + 1
  Receive responses from neighbors, if any
  if (Did not receive `Accept` from agent p) or (*AcceptCount* $\neq 1$) **then**
    $p \leftarrow \emptyset$
    Find max gain and preferred assignment (individual update):
    $g, a \leftarrow$ *getMaxGainAndAssignment()*
Send `Bid` (g) to all neighbors
Receive n `Bids` from all neighbors, ignoring message from p
$G \leftarrow \max_n$ `Bids`$_n$
if $g > G$ **then**
  bChanging$\leftarrow$ True
else
  bChanging$\leftarrow$ False
  if $p' \neq \emptyset$ **then**
    Send `ProhibitVariableChange` to agent p'
  if $p \neq \emptyset$ **then**
    Send `ProhibitVariableChange` to agent p
Receive any messages sent by neighbors
if (*bChanging*) and ($p \neq \emptyset$) **then**
  if Received `ProhibitVariableChange` from agent p **then**
    bChanging$\leftarrow$ False
if (*bChanging*) and ($p' \neq \emptyset$) **then**
  if Received `ProhibitVariableChange` from agent p' **then**
    bChanging$\leftarrow$ False
if *bChanging* **then**
  UpdateAssignment(a)

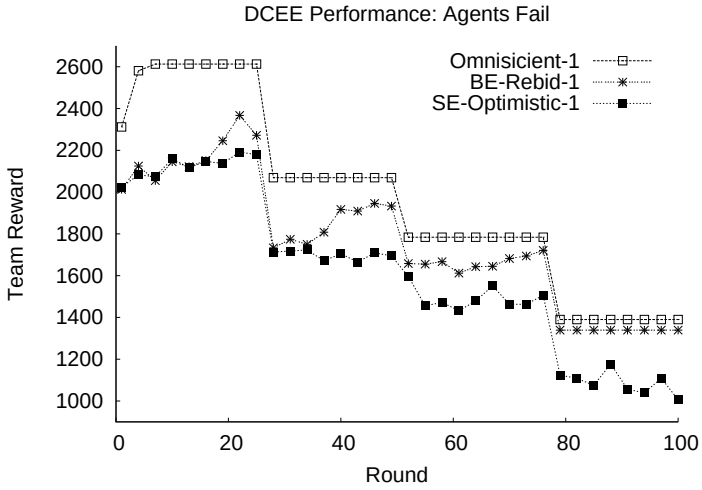


Fig. 22: This graph shows how agents can automatically recover after a failure of 2 agents after every 25 rounds.

signmentForTriple(), *getMaxGainAndAssignmentForPair()*, and *getMaxGainAndAssignment()* calculate utilities. SE-Optimistic-3 requires significantly more messages when compared to SE-Optimistic-2, similar to how Omniscient-3 requires more than Omniscient-2.

Appendix B. Dynamic Changes in Graph Topology

Although this article does not focus on the robustness of DCEE algorithms, this section presents a result suggesting that DCEE algorithms are indeed robust to agent failures. The agents calculate their gains with the current set of neighbors on every round and act accordingly. Even if the algorithm has converged, it will begin optimizing again if needed. DCEE algorithms automatically handle situations where agents and/or constraints are added/removed from the system. While there are likely enhancements that could be made to the algorithms to increase their resilience, such investigations are left to future work.

Figure 22 shows a representative learning curves for two k=1 DCEE algorithms and compares to the Omniscient-1 algorithm. The x-axis shows the round number (from 1–100) and the y-axis shows the team's performance. There are 20 agents forming a chain topology. Every 25 rounds, two agents are disabled at random. Disabled agents are removed from the DCEE: all constraints connected to disabled agents are removed from the graph and the disable agents cannot contribute to the team reward. The graph shows that agents resume their optimization after such failures, and that BE-Rebid-1 outperforms SE-Optimistic-1.

Appendix C. DSA Results

This section presents results from using algorithms based on DSA [13], while all other results presented in this paper use MGM as a framework for the DCEE algorithms. The distributed stochastic search algorithm (DSA) is popular both for its simplicity and numerous successes (c.f., Zhang et al. [52] and Zivan [54]). In the DSA algorithm for DCOPs, every agent draws a random number. If the agent's number exceeds some pre-defined threshold, then it has the ability to move on the current round.

We define a **DSA-Omniscient** algorithm, where an agent that wins the ability to move will change its value to maximize its reward, assuming all neighbors do not move. In **DSA-Optimistic**, an agent that wins the ability to move will always move, as it assumes that executing the explore action will return the maximum reward. In **DSA-Mean**, an agent that wins the ability to move will only do so if its current reward is less than the mean reward multiplied by its number of neighbors.

Figure 23 shows that DSA-based algorithms can outperform MGM-based algorithms. DSA-Omniscient and DSA-Mean outperform their counterparts. However, DSA-Optimistic significantly underperforms SE-Optimistic. These results report the average of thirty trials each on the four topologies. It is worth noting, however, that the DSA algorithms were each tuned separately, as an appropriate value of p, the parameterized threshold, was critical to obtaining high performance. Figures 24(a) and 24(b) show how the performance of DSA is affected by its parameter setting. Results show the average total gain of 10 trials, each using 20 agents running for 50 rounds. The best value of p, per DSA algorithm, was used to produce the values in Figure 23.

Acknowledgements

The majority of the research in this paper occurred at the University of Southern California in the TEAMCORE research lab. We thank Lockheed Martin and Perceptronics Inc. for support during the early versions of this work. This research was supported in part by the United States Department of Homeland Security through the Center for Risk and Economic Analysis of Terrorism Events (CREATE).

References

[1] Agmon, N., Kraus, S., and Kaminka, G. A., Multi-robot perimeter patrol in adversarial settings, in *Proceedings of the International Conference on Robotics and Automation (ICRA)* (2008).

[2] Benisch, M. and Sadeh, N., Examining DCSP coordination tradeoffs, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2006).

[3] Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S., The complexity of decentralized control of markov decision processes, *Mathematics of Operations Research* **27** (2002).

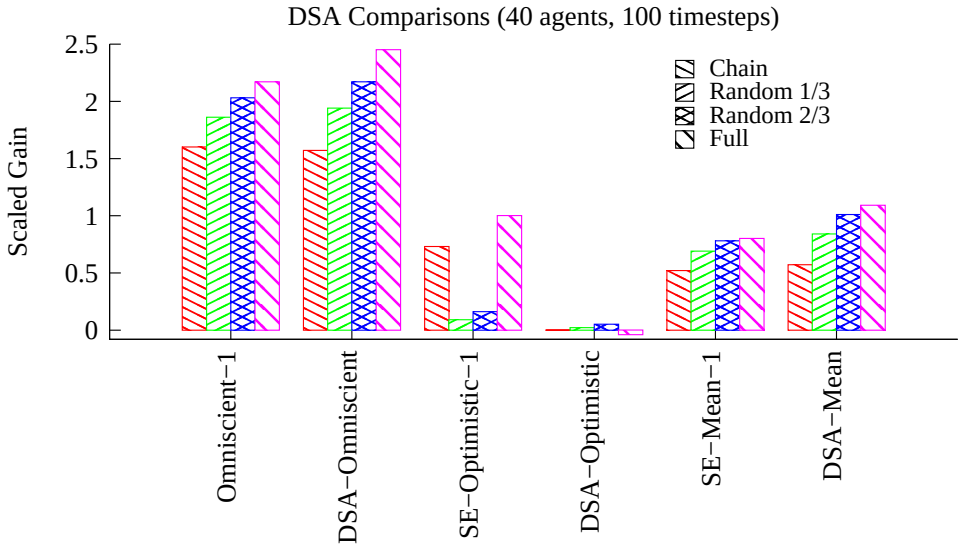
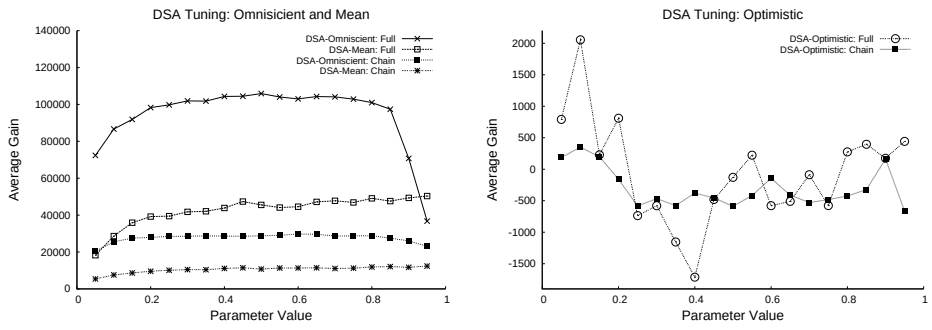


Fig. 23: DSA-Omniscient can outperform the Omniscient-1 algorithm, and DSA-Mean can outperform SE-Mean-1. However, DSA-Optimistic significantly underperforms SE-Optimistic-1.



(a) (b)

Fig. 24: Tuning the value of p, the parameter controlling the behavior of DSA, can be difficult. For instance, in (a), a mid-range value of p maximizes the performance of the DSA-Omniscient algorithms, while the highest value of p maximizes the performance of DSA-Mean. (b) suggests that there is little regularity to the performance of DSA-Optimistic with respect to the value of p.

[4] Boutilier, C., Sequential optimality and coordination in multiagent systems, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (1999).

[5] Brito, I., Meisels, A., Meseguer, P., and Zivan, R., Distributed constraint satisfaction with partially known constraints, *Constraints* **14** (2009) 199–234.

[6] Cheng, J., Cheng, W., and Nagpal, R., Robust and self-repairing formation control

for swarms of mobile agents, in *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (2005).

[7] Correll, N., Bachrach, J., Vickery, D., and Rus, D., Ad-hoc wireless network coverage with networked robots that cannot localize, in *Proceedings of the International Conference on Robotics and Automation (ICRA)* (2009).

[8] Cox, J., Durfee, E., and Bartold, T., A distributed framework for solving the multiagent plan coordination problem, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2005).

[9] Durfee, E. H., Blissful ignorance: Knowing just enough to coordinate well, in *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (1995).

[10] Faltings, B. and Macho-Gonzalez, S., Open constraint programming, *Artificial Intelligence Journal* **161** (2005) 181–208.

[11] Farinelli, A., Rogers, A., Petcu, A., and Jennings, N., Decentralized coordination of low-power embedded devices using the max-sum algorithm, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2008).

[12] Fernàndez, C., Béjar, R., Krishnamachari, B., and Gomes, C. P., Communication and computation in distributed CSP algorithms, in *Proceedings of the International Conference on the Principles and Practice of Constraint Programming (CP)* (2002).

[13] Fitzpatrick, S. and Meertens, L., Distributed coordination through anarchic optimization, in *Distributed Sensor Networks*, eds. Lesser, V., Ortiz, C. L., and Tambe, M. (Kluwer Academic Publishers, 2003).

[14] Freeman, P. R., The secretary problem and its extensions: A review, *International Statistical Review* **51** (1983).

[15] Greenstadt, R., Grosz, B., and Smith, M. D., SSDPOP: Improving the privacy of DCOP with secret sharing (short paper), in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2007).

[16] Grosz, B. J. and Sidner, C. L., Plans for discourse, in *Intentions in Communication*, eds. Cogent, P. R., Morgan, J., and Pollack, M. (MIT Press, 1990).

[17] Guestrin, C., Koller, D., and Parr, R., Multiagent planning with factored MDPs, in *Proceedings of the Neural Information Processing Systems Conference (NIPS)* (2001).

[18] Jain, M., Taylor, M. E., Yokoo, M., and Tambe, M., DCOPs meet the real world: Exploring unknown reward matrices with applications to mobile sensor networks, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (2009).

[19] Jung, H. and Tambe, M., Argumentation as distributed constraint satisfaction: Applications and results, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2001).

[20] Kiekintveld, C., Yin, Z., Kumar, A., and Tambe, M., Asynchronous algorithms for approximate distributed constraint optimization with quality bounds, in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2010).

[21] Kok, J. R. and Vlassis, N., Collaborative multiagent reinforcement learning by payoff propagation, *Journal of Machine Learning Research* **7** (2006) 1789–1828.

[22] Kozono, S., Received signal-level characteristics in a wide-band mobile radio channel, in *IEEE Transactions on Vehicular Technology* (1994).

[23] Lesser, V., Ortiz, C., and Tambe, M., *Distributed sensor nets: A multiagent perspective* (Kluwer Academic Publishers, 2003).

[24] Levesque, H. J., Cohen, P. R., and Nunes, J., On acting together, in *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (1990).

[25] Maheswaran, R. T., Pearce, J. P., and Tambe, M., Distributed algorithms for DCOP:

A graphical-game-based approach, in *Proceedings of the International Conference on Parallel and Distributed Computing and Systems (PDCS)* (2004).

[26] Mailler, R. and Lesser, V., Solving distributed constraint optimization problems using cooperative mediation, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2004).

[27] Mailler, R. and Lesser, V., Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems, *Journal of Artificial Intelligence Research* **25** (2006) 529–576.

[28] Maio, D., Rizzi, S., and Risorgimento, V., Unsupervised multi-agent exploration of structured environments, in *Proceedings of the International Conference on Multi-Agent Systems (ICMAS)* (1995).

[29] Marden, J., Arslan, G., and Shamma, J., Connections between cooperative control and potential games illustrated on the consensus problem, in *Proceedings of the European Control Conference (ECC)* (2007).

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

[31] Molisch, A. F., *Wireless Communications* (IEEE Press, 2005).

[32] Montemerlo, R., Gordon, G., Schneider, J., and Thrun, S., Approximate solutions for partially observable stochastic games with common payoffs, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2004).

[33] Pearce, J. and Tambe, M., Quality guarantees on k-optimal solutions for distributed constraint optimization, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (2007).

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

[35] Petcu, A. and Faltings, B., A scalable method for multiagent constraint optimization, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (2005).

[36] R. Lass, E. S. and Regli, W., Dynamic distributed constraint reasoning, in *Proceedings of the Conference on Artificial Intelligence (AAAI)* (2008).

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

[38] Santana, H., Ramalho, G., Corruble, V., and Ratitch, B., Multi-agent patrolling with reinforcement learning, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2004).

[39] Scerri, P., Pynadath, D., Johnson, L., Rosenbloom, P., Si, M., Schurr, N., and Tambe, M., A prototype infrastructure for distributed robot-agent-person teams, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2003).

[40] Stone, P., Kuhlmann, G., Taylor, M. E., and Liu, Y., Keepaway soccer: From machine learning testbed to benchmark, in *RoboCup-2005: Robot Soccer World Cup IX* (2006).

[41] Stranders, R., Farinelli, A., Rogers, A., and Jennings, N. R., Decentralized coordination of mobile sensors using the max-sum algorithm, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (2009).

[42] Sutton, R. S. and Barto, A. G., *Introduction to Reinforcement Learning* (MIT Press, 1998).

[43] Tambe, M., Towards flexible teamwork, *Journal of Artificial Intelligence Research* **7** (1997) 83–124.

[44] Tambe, M., Bowring, E., Jung, H., Kaminka, G., Maheswaran, R., Marecki, J., Modi, P. J., Nair, R., Okamoto, S., Pearce, J. P., Paruchuri, P., Pynadath, D., Scerri, P., Schurr, N., and Varakantham, P., Conflicts in teamwork: Hybrids to the rescue, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2005).

[45] Taylor, M. E., Jain, M., Jin, Y., Yooko, M., and Tambe, M., When should there be a "me" in "team"? distributed multi-agent optimization under uncertainty, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2010).

[46] Verbeeck, K., Nowé, A., and Tuyls, K., Coordinated exploration in multi-agent reinforcement learning: an application to load-balancing, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2005).

[47] Vinyals, M., Pujol, M., Rodriguez-Aguilar, J. A., and Cerquides, J., Divide-and-coordinate: DCOPs by agreement, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2010).

[48] Xu, Y., Scerri, P., Xu, B., Okamato, S., Lewis, M., and Sycara, K., An integrated token-based algorithm for scalable coordination, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2005).

[49] Yokoo, M., Durfee, E. H., Ishida, T., and Ukwabara, K., The distributed constraint satisfaction problem: formalization and algorithms, *IEEE Transaction on Knowledge and Data Engineering* **10** (1998).

[50] Yokoo, M. and Hirayama, K., Algorithms for distributed constraint satisfaction: A review, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2000).

[51] Zafar, H., Lesser, V., Corkill, D., and Ganesan, D., Using organization knowledge to improve routing performance in wireless multi-agent networks, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2008).

[52] Zhang, W., Xing, Z., Wang, G., and Wittenburg, L., An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks, in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2003).

[53] Zhang, Y., Bellingham, J. G., Davis, R. E., and Chao, Y., Optimizing autonomous underwater vehicles' survey for reconstruction of an ocean field that varies in space and time, in *Proceedings of the Fall Meeting of the American Geophysical Union (AGU)* (2005).

[54] Zivan, R., Anytime local search for distributed constraint optimization, in *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (2008).