In *The 12th International workshop on Distributed Constraint Reasoning at AAMAS-10* (DCR-10), Toronto, Canada, May 2010.

Towards a Theoretic Understanding of DCEE

Scott Alfeld, Matthew E. Taylor, Prateek Tandon, and Milind Tambe
http://teamcore.usc.edu/

The University of Southern California

Abstract. Common wisdom says that the greater the level of teamwork, the higher the performance of the team. In teams of cooperative autonomous agents, working together rather than independently can increase the team reward. However, recent results show that in uncertain environments, increasing the level of teamwork can actually decrease overall performance. Coined the *team uncertainty penalty*, this phenomenon has been shown empirically in simulation, but the underlying mathematics are not yet understood. By understanding the mathematics, we could develop algorithms that reduce or eliminate this penalty of increased teamwork.
In this paper we investigate the team uncertainty penalty on two fronts. First, we provide results of robots exhibiting the same behavior seen in simulations. Second, we present a mathematical foundation by which to analyze the phenomenon. Using this model, we present findings indicating that the team uncertainty penalty is inherent to the level of teamwork allowed, rather than to specific algorithms.

1 Introduction

Recently there has been a rise in autonomous teams of agents working cooperatively. Teams of artificial agents, whether software or robotic are being deployed today in a variety of scenarios, including mobile sensor networks (Cheng et al., 2005; Marden et al., 2007; Jain et al., 2009), and teams of underwater autonomous scouts (Zhang et al., 2005). Many problems are universal across multiagent systems. Even in fully observable environments, centralized solutions scale poorly with the number of agents. Often centralization is not an option, and agents must behave partially independently while still working with each other to facilitate cooperation. The underlying understanding of these issues is that the higher the level of teamwork[1] the better the outcome.

Recently, however, the question of *if* agents should work together has arisen. As such, we do not concern ourselves with issues of communication in this paper, but instead focus on when and if the level of teamwork amongst agents should be increased. We recently introduced the notion of the *team uncertainty penalty* (Taylor et al., 2010). Put informally, this result shows that agents should sometimes act alone, rather than attempt to coordinate, even if communication is free. Previous work points to the density of a graph as the leading culprit, but the underlying cause remains a mystery.

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

Empirical evidence has shown manifestations of the team uncertainty penalty in simulation (Taylor et al., 2010). While analysis of these results suggest when the phenomenon will be observed, there has not yet been a way to determine *a priori* when the team reward will or will not suffer from increasing agent cooperation. Outside of simulation, the extent of the team uncertainty has not yet been studied and it is unknown how debilitating it can be in robotic applications. Moreover, as of yet there is no mathematical framework under which the specifics of the phenomenon can be analyzed.

This paper provides experimental evidence of robot teams exhibiting similar results as those found in simulation. We also provide a mathematical analysis used to help pinpoint the contributing factors of this counter-intuitive phenomenon. Additionally, in Section 5.3, we provide evidence showing that the team uncertainty penalty is a part of the DCEE framework rather than an artifact of specific algorithms. We introduce the notion of L-Movement and analyze that ramifications of restricting the allowed total movement of an agent team. We also introduce the *configuration hypercube* for specific problem formulations and propose that analysis on the probabilistic structure of this hypercube will lend key insights into the behavior of previously explored algorithms. In particular, we suggest that analysis of the hypercube will provide a heuristic by which one can predict the effects of increasing the level of teamwork for a given problem.

2 Background

The *Distributed Constraint Satisfaction Problem* (DCOP) (Mailler & Lesser, 2004; Modi et al., 2005; Petcu & Faltings, 2005) framework is becoming common for modeling cooperative multiagent scenarios. Because of its ability to model the need for cooperation via the notion of joint rewards, the formulation is being used in several problems, such modeling sensor networks and for multiagent plan coordination (Cox et al., 2005). Solving a DCOP (determining a globally optimal configuration) is NP-Hard (Modi et al., 2005), and thus there is substantial work towards finding fast, locally optimal algorithms (Maheswaran et al., 2004; Pearce et al., 2008; Stranders et al., 2009).

In real world applications, the rewards are not known *a priori*, and discovering them requires exploration. Moreover, agents are concerned with a total, online reward achievable in a limited time frame. We present one example of such an application in Section 4. When working in such environments, agents must strike a balance between exploiting their current knowledge and exploring their environment. While exploration vs. exploitation is a common problem for single agents (c.f., reinforcement learning (Sutton & Barto, 1998) and multi-armed bandits (Robbins, 1952)), having a multiagent system where rewards are determined by the coordination of agents adds a level of complexity. The DCOP framework, however, assumes that all aspects of the environment are known. Allowing uncertainty to be modeled, the notion of a *Distributed Coordination of Exploration and Exploitation* (DCEE) problem was recently introduced (Jain et al., 2009). This extension of DCOP to real world environments accounts for the uncertainty of rewards, as well incorporating the notion of a limited time horizon. For reference, we describe DCEE here.

A DCEE domain is much like a DCOP, with several key differences. Specifically, the rewards in the constraint matrices are not known until they are explored. That is to say that the reward achieved by agents A_i and A_j when they assign their variables to

λ_m and λ_n respectively is not known until a point that A_i assigns value λ_m and A_j simultaneously assigns value λ_n. What is known *a priori*, however, is something about the distribution over rewards for each constraint. In this paper we assume the underlying distribution of rewards is known.

A DCEE consists of a set Q of n variables, $\{x_1, x_2, \ldots, x_n\}$, assigned to a set of agents, where each agent controls one (or more) variable's assignment. For this paper we concern ourselves with the case when every agent has only one variable. Agents have at most T *rounds* to modify their variables x_i, which can take on any value from the finite domain D_i. The goal of such a problem is for agents to choose values for the variables such that the cumulative sum over a set of binary constraints and associated payoff or reward functions, $f_{ij} : D_i \times D_j \rightarrow \Re$, is maximized over time horizon $T \in \mathbb{N}$. More specifically, the agents attempt to pick a set of assignments (one per time step: $\mathcal{A}_0, \ldots, \mathcal{A}_T$) such that the total reward (the *return*) is maximized: $R = \sum_{t=0}^{T} \sum_{x_i, x_j \in V} f_{ij}(d_i, d_j)$, where $d_i \in D_i, d_j \in D_j$ and $x_i \leftarrow d_i, x_j \leftarrow d_j \in A_k$.

The following is a list of select properties that DCEEs have, but DCOPS do not: (1) agents initially know the constraint graph but only discover rewards through exploration (i.e., a pair of agents set their values to explicitly discover a reward), (2) problems last a set amount of time, (3) there are more combinations of domain values than can be explored within this time (disallowing exhaustive exploration), and (4) we seek to maximize the online global reward for the team over this time horizon T.

A DCEE can be thought of as being defined in part by a graph $G = \{V, E\}$. In this framework, every variable corresponds to a vertex in V, and every constraint is an edge in E. Every edge is thus a two dimensional matrix where every element has been drawn i.i.d. from a known distribution. Throughout this paper we refer to an agent changing its variable's value as that agent *moving*.

2.1 *k*-Optimal

Centralized algorithms for DCOPs scale poorly, as finding a globally optimal reward is NP-Hard (Modi et al., 2005). Because of this, approximate DCOP algorithms are heavily worked investigated (Zhang et al., 2003; Pearce & Tambe, 2007). The notion of k-optimal configurations expresses the level of a locally optimal solution. The lower the value of k, the more local the k-optimal solution.

Algorithms previously presented (Taylor et al., 2010) come in different variants based on the value of k. This value indicates the maximum size of a coalition of agents that can move at each step. In a DCOP, a k-optimal configuration is defined as one where no coalition of up to k agents would benefit by changing their variable. The notion of k-optimal does not extend directly to DCEE, as we discuss in Section 5.1.

2.2 SE-Optimistic

The DCEE algorithm SE-Optimistic-1, where $k = 1$, will behave as follows. A group of agents is considered to be in a neighborhood if the subgraph created by those agents form a connected graph. In each round, every agent calculate their potential gain (how much they could improve their reward) obtained by moving under the assumption that all other agents in their neighborhood will not move. Each agent also queries each of its neighbors for their potential gain. If an agent has a higher potential gain than any of its neighbors in a given round, it will move so as to obtain its maximum gain. In

SE-Optimistic-2, where $k = 2$, a pair of agents (in the same neighborhood) will move if their total potential gain is higher than any other pair of agents in their neighborhood.

Intuitively, these algorithms perform a greedy search. In each round, the agents who would most benefit by moving alone do so, while others in the same neighborhood do not move. $k = 2$ algorithms allow more agents to attempt to improve their individual rewards in each around, while in $k = 1$ algorithms only one agent per neighborhood may move per round.

2.3 MGM-Omniscient

Omniscient algorithms (Taylor et al., 2010) are artificially provided the values of all rewards of joint actions and thus do not need to explore. This information, causing the algorithms to be "omniscient" in regards to the rewards, modifies a DCEE into a DCOP. The omniscient DCEE algorithms referred to in this paper, namely omni-1 and omni-2, are previously introduced algorithms run on this resulting DCOP. In omni-1, in each round every agent calculates how much it could improve its own reward by being the one in its neighborhood to move. After agents communicate these values amongst others in their neighborhood, the agent who can increase their reward the most moves, and others in the same neighborhood do not. Omni-2 is similar, but allows for agents to form coalitions of two agents in the same neighborhood, and both may move in a single round. These algorithms, called *Maximum Gain Message* (MGM), are described in detail elsewhere (Pearce & Tambe, 2007).

3 Team Uncertainty Penalty

Intuitively, it seems that the more teamwork amongst truthful, cooperative agents the better the overall performance of the team. Increasing the level of teamwork among agents will add to communication overheads and computational costs, but in this work, we ignore both communication and computational costs. Instead, the team uncertainty penalty has to do with decreased total reward (in some circumstances) when teamwork is increased (Taylor et al., 2010).

The prevalence of the team uncertainty penalty has been shown empirically. It has been observed in varying graph topology, and across several algorithms. In Section 5.3 we present experimental results of a more mathematical nature that suggest the team uncertainty penalty in even more general, being an artifact of the level of teamwork rather than the specifics of the algorithms.

Before discussing the penalty in more depth, we first outline the domain used and then discuss results in the domain exhibiting this counter-intuitive behavior.

3.1 Simulator Domain

The DCEE *mobile wireless network* problem was first introduced elsewhere (Jain et al., 2009). To explore this problem, we have built and released a simulator[2] that models a set of mobile robots with wi-fi antennas that must optimize the sum of the inter-agent signal strengths. Value settings correspond to agent (robot) locations, constraints are determined by the network topology, and rewards are based on link quality between robots. As in prior work, we assume that the topology is fixed, that small changes to

[2] Available at `http://teamcore.usc.edu/dcop`.

agent locations causes signal strengths for that agent to become uncorrelated with their previous values (i.e., small scale fading dominates (Molisch, 2005)), and that signal strengths between agents are symmetric.

3.2 Simulator Results

Previous results (Jain et al., 2009; Taylor et al., 2010) demonstrated that multiple DCEE algorithms were effective at improving the online reward in the simulated domain over multiple numbers of agents, experiment lengths, and network topologies. To quantify performance, we consider the *total gain* of different algorithms, where the gain on round n is defined as the difference between the team's reward on round n and the team's reward on round 0.

Figure 1 shows the total gains of algorithms on graphs of 10 agents run for 50 rounds each (averaged over 10 independent trials). First, consider the Omniscient algorithms (Omni, Omni-2, and Omni-3). In the left half of the graph, we see that as the amount of teamwork (i.e., k) in the Omniscient algorithm increases, the total reward during experiments on chain graphs increases. The right half of the graph shows the same trend for complete graphs: as k increases, the Omniscient algorithm receives more reward.

Results of the Optimistic algorithms in Figure 1 tell a different story. While increasing k again improves the gain for complete graphs, higher values of k *decrease* the reward gained in chain graphs (ring graphs with one fewer edge). This is precisely what we mean by the team uncertainty penalty: increasing levels of cooperation may decrease the team reward (relative to lower levels of teamwork). This phenomena is further analyzed and discussed elsewhere (Taylor et al., 2010), but strictly on an empirical basis.

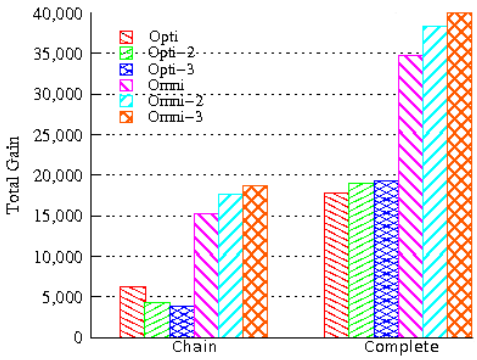


Fig. 1. In most cases, increasing the amount of teamwork improves the team reward of agents. However, as teamwork increases in SE-Optimistic in a ring graph, team performance decreases. The y-axis shows the "total gain," which is a measure of the total on-line improvement over the length of the experiment, relative to no optimization.

4 Robot Experiments

This section provides novel experimental results on physical robots, corroborating the results from simulation discussed in the previous section. Previous results (Taylor et al., 2010) come from simulations of DCEE. Here we provide evidence that the DCEE formulation holds true to real world scenarios, specifically in the case of mobile robots connected via a wireless network. Moreover, we demonstrate the team uncertainty penalty manifesting on actual hardware.

4.1 Problem Setup

Robots in this section run DCEE algorithms in order to maximize the signal strength between wireless network receivers, analogous to the simulation described in Section 3.

An agent can measure the wireless signal strength between itself and each neighbor, corresponding to measuring the reward for the pair of agent assignments. Agents select from a set of possible physical locations (i.e., a variable assignment). Because the time for movement in physical robots dominates communication and calculation time (Jain et al., 2009), we measure experiment length by the number of *rounds*, defined by the period in which every agent may decide to move to a new position and then reach that position. All agents may choose to either `stay` in their current position or `explore`.

The Create, made by iRobot, is used as the platform for our experiments (see Figure 2). Additionally, an ebox-3854 is used as an on-board computer and the EMP-8602 mini PCI card is used for 802.11 b communication between the robots. More details can be found at `http://enl.usc.edu/projects/peg/platform.html`.

4.2 Results

This section discusses the results of executing the SE-Optimistic-1 and SE-Optimistic-2 algorithms on physical robots. To corroborate the results in simulation, the two algorithms are tested on both chain and complete graphs of five agents each. Figure 3 shows the average gain per round of the four different setups. 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 ± 55, whereas the average for SE-Optimistic-2 was 506 ± 31, thus SE-Optimistic-1 has a relatively harder time improving the team's reward.

Fig. 2. Three Creates with additional hardware.

Figure 4(a) displays the total gain (i.e., the area under the curves in Figure 3). We again see that SE-Optimistic-1 on a complete graph performs worse than all other algorithms. Most important is that the trends from section 3 are seen again. 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 Analysis of the Team Uncertainty Penalty

This section provides a mathematical foundation for analyzing the team uncertainty penalty. We extend the notion of a k-optimal configuration in a DCOP to the DCEE formulation. We also introduce the notion of L-movement which expresses the flexibility

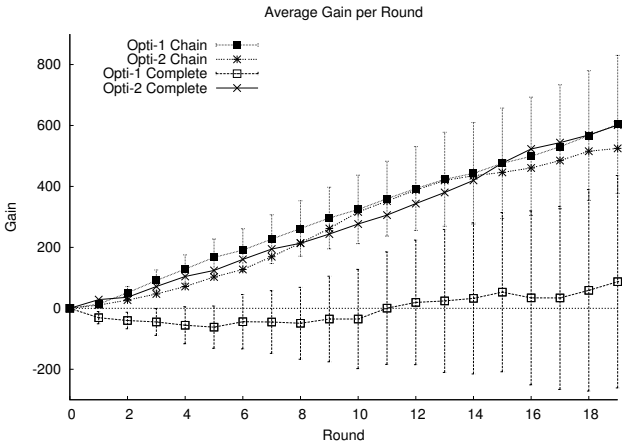


Fig. 3. This plot shows performance of SE-Optimistic on two different graph topologies for $k = 1$ and $k = 2$. The x-axis shows the round number and the y-axis shows the gain. Results are averaged over 10 trials, each using 5 agents. Standard errors are shown only on $k = 1$ plots for readability.

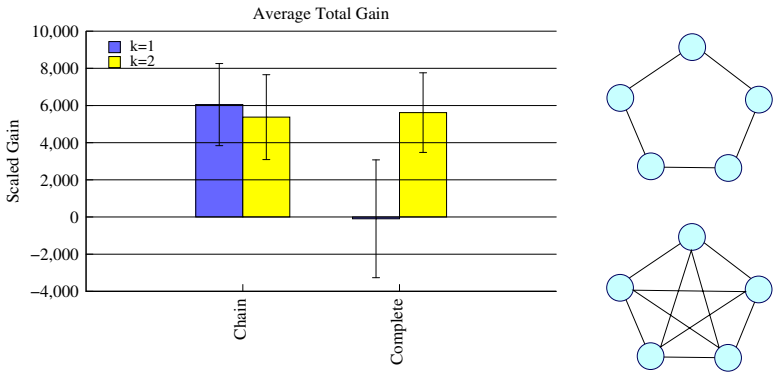
in exploration of a given DCEE algorithm. Finally, we present a mathematical model of DCEE problems based on the interplay of random variables.

Previously introduced algorithms such as SE-Optimistic-1 and SE-Optimistic-2 have behavior defined by the value k, controlling the number of agents that cooperate in each round. We refer to this class of algorithms as *k-dependent*. Previous work points to graph density as a strong contributing factor (Taylor et al., 2010), and we extend this analysis by introducing the notion of L-movement, which ties directly to graph density. In addition, we provide simulation results showing the effects of changing the value of k in k-dependent algorithms.

We hope that the DCEE formulation will eventually be understood fully on a theoretic level. This section provides a foundation for such an understanding, as well as some initial results from the proposed mathematical model. We show that the team uncertainty penalty may not be the fault of the individual algorithms, and provide evidence that the phenomenon is an inexorably tied to DCEE formulation and algorithms' dependence on k.

5.1 Extending k-Optimality and Introducing L-Movement

In a DCOP, a configuration is defined as k-optimal if no connected group of at most k agents would benefit by jointly moving (Maheswaran et al., 2004; Pearce et al., 2008). In a DCEE, however, the rewards are not known, and thus an agent (or group of agents) cannot know whether or not they would increase their reward by changing their variable(s). Therefore the notion of k-optimality does not extend directly from DCOP to DCEE. However, if the reward distribution(s) are known in a DCEE, a natural extension of the k-optimal definition is that a DCEE configuration is k-optimal if and only if no connected group of up to k agents has a positive expected gain by choosing an entirely new configuration. However, this does not allow for the possibility of agents reverting to a previously seen (partial-)configuration.



(a) Robot Results (b) Topologies

Fig. 4. The chart in (a) shows that the total gain in robot tests follow the trends from simulation. SE-Optimistic-1 outperforms SE-Optimistic-2 on chain graphs, but underperforms it on complete graphs. Standard error over the 10 trials are shown for each setting. (b) depicts two topologies used throughout the paper: a five node ring graph (top), and a five node complete graph (bottom).

In a starting configuration of a DCEE, where only one configuration (the current one) has been explored, this definition is reasonable, and connects well with the notion of k-optimality of a DCOP configuration. After even one round, however, groups of agents have the option to jointly move back to a configuration where some or all of the constraint reward values have been observed. As such, the k-optimality of a configuration would depend on the set of configurations already observed. We therefore provide a definition of k-optimal in DCEE as follows.

Definition 1. *A DCEE configuration, combined with the history of explored configurations, is k-optimal if and only if no group of up to k agents has a positive expected gain by selecting another configuration with zero or more constraint values already explored.*

This says a configuration is only k-optimal if no coalition of up to k agents could move so as to increase their total reward by each agent either (i) selecting previously explored values and obtaining the known rewards, or (ii) selecting unexplored actions leading to a positive expected gain in total reward. Because of the dependence on explored and unexplored configurations, the set of potentially k-optimal configurations of a DCEE is extremely large. Consider the following example.

Example 1. A DCEE with two agents A_1 and A_2 and one constraint. When each variable takes on the first value, the first reward is known – suppose it is 120. A_1 then moves, selecting a new value, and the constraint yields a reward of 110. At this point two rewards are known. Suppose either A_1 or A_2 can select an action that will yield a reward of 110 with probability .5, and a reward of 100 with probability 5. Selecting a new action would yield an expected gain of $110 - 105 = -5$. However, if A_1 reverts to its previous action it can increase the total reward by $120 - 110 = 10$. Thus this configuration, combined with the history of explored configurations, is not 1-optimal.

We now focus on a different aspect of k-dependent algorithms. First we introduce *L-Movement*, a value we will use heavily in our analysis.

Definition 2. *The L-movement, denoted as L, of a DCEE algorithm on a graph G is the maximum number of agents that can move in any one round.*

There is no restriction inherent to the DCEE formulation on the number of agents that can move at each step. For example a valid, albeit naïve, algorithm would move every agent at every step. In SE-Optimistic-1 and SE-Optimistic-2, as well as other k-dependent algorithms, only a limited subset of agents can move at each step.

This restriction of movement is caused by the fact that an agent cannot move if more than $k-1$ of its neighbors also move. For a given graph, we denote the maximum number of agents that can move in each round of a k-dependent algorithm as L. We that note L is dependent on the graph topology, for example in $k = 1$ algorithms L is the size of the largest maximal independent set of the graph, a value known to be NP-hard to calculate for a general graph. L-movement is central to our analysis presented in 5.3 where we show the change in L caused by changing k determines the change in team performance. Consider a ring graph and a complete graph, both with 5 vertices (see Figure 4(b)). In the ring graph, the size of the maximal independent set is 2, whereas in the complete graph it is only 1. In general, the size of the maximal independent set of a ring graph is $\lfloor \frac{|V|}{2} \rfloor$, and is 1 for a complete graph.

5.2 The Configuration Hypercube

For a DCEE problem, we use the following notation:

$G = \{V, E\}$	The graph of the DCEE.
$A = \{A_1, \ldots A_{\|V\|}\}$	The set agents.
$R = \{R_1, \ldots, R_{\|E\|}\}$	The set of constraints.
r_i	The distribution from which the rewards of R_i are drawn.
T	The number of rounds for which the algorithms are run.
x_i	The variable of the agent A_i.
D_i	The domain of x_i.
$\mathcal{C}$	The configuration hypercube (defined below).
λ_i	The value of the i-th coordinate of a location in $\mathcal{C}$.

We consider the set of all DCEE algorithms where:

- The starting configuration is initialized randomly.
- The algorithm is run for s steps, ending with some configuration.
- That ending configuration is then chosen for every step for the rest of time.

This is to say that s steps are allotted to exploration, after which the algorithm only exploits (keeps the same configuration) for the remaining $T - s$ rounds. We analyze the reward achieved after the s steps of exploration.

To analyze such algorithms we define a $|V|$-dimensional hypercube $\mathcal{C}$ as the *Configuration Hypercube*. Each dimension of $\mathcal{C}$ corresponds to an agent, with the location within that dimension defined by the assignment of the agent's variable. We let $\mathcal{C}[\lambda_1, \lambda_2, \ldots, \lambda_{|V|}]$ be the total reward when agent A_i takes value λ_i.

We note that, for a given agent, no value for its variable is believed *a priori* to be more beneficial than any other. Therefore, without loss of generality we say that each agent A_i has an ordered list of values $\alpha = \alpha_1, \ldots, \alpha_{|D_i|}$, and that at each round of a DCEE algorithm, an agent has only two choices: (1) select an value previously

chosen, or (2), select the first value (the value with the lowest index) that has not yet been assigned. Further, for any DCEE algorithm, we assume (again without loss of generality) that all agents begin with their first variable setting, α_1.

At each round of an algorithm, zero or more agents move. Note that at any location $\lambda = (\lambda_1 \dots \lambda_{|V|})$ in $\mathcal{C}$, the achievable locations after one step are all locations adjacent to λ (l_∞ distance to λ is 1). That is to say that λ is reachable if and only if $\max_i(\lambda_i) \leq s$. Further, for a general DCEE algorithm, the locations reachable from the starting location in s steps form an $(s+1)$-sized $|V|$-dimensional hypercube $\mathcal{C}_s$ where all locations have l_∞ distance to the starting location less than or equal to s. The expected maximum value of $\mathcal{C}_s$ is an upper bound for the total reward achievable by any DCEE algorithm (under conditions described above) running for s rounds.

However, computing the expected maximum element of $\mathcal{C}_s$ is non-trivial. Every entry in $\mathcal{C}_s$ is the sum of $|E|$ values (one from each constraint), and thus the entries are highly dependent. Furthermore, while each constraint contributes s^2 values to $\mathcal{C}_s$ and the expected maximum of each of these can easily be computed, $\mathcal{C}_s$ contains only a subset of the possible sums of constraint values. The expected maximum is not simply the sum of the expected maximums of the constraint matrices. Moreover, this subset is a function of the graph structure and finding the expected maximum can be difficult.

We note that the expected maximum of $\mathcal{C}_s$ is the expected maximum reward obtainable by an algorithm that does need to explore. By this we mean an algorithm with perfect knowledge of the values of all locations in C_s, but with the limitation that configurations outside of C_s are unreachable. As such, it is not a tight bound for DCEE algorithms. We illustrate this fact with the following example.

Example 2. Consider a DCEE with two agents A_1 and A_2, and one constraint between them. Let $M_d(i)$ denote the expected maximum of i samples drawn independently from distribution d. The specific form of $M_d(i)$ is unimportant for our results, but the function can be computed for the distributions we use. The 2-dimensional cube $\mathcal{C}_s$ would have $(s+1)^2$ values, with the expected maximum being $M_d((s+1)^2)$. An algorithm that knows all values in $\mathcal{C}_s$ could assign values for both agents' variables such that this reward is obtained. An algorithm without access to the individual values, however, has four choices at each step – change A_1's variable, change A_2's variable, change both agents' variables, or leave the configuration unchanged. When either or both of the agents change their variable the (only) constraint is re-sampled. Thus the best an algorithm could hope to achieve is the expected maximum of $s+1$ samples, i.e., $M_d(s+1)$.

Achieving the maximum value of $\mathcal{C}_s$ requires the algorithms to know more about the reward matrices than is allowed in the DCEE framework. We also note an additional relaxation. In k-dependent algorithms, not all k-sized subsets of agents can move in one round. For example, running a $k = 1$ algorithm on a five vertex ring graph has a value of $L = 2$. Our experiments take the maximum value of C_s reachable by changing any two (or fewer) agents in each round. A k-dependent algorithm, however, could not have two adjacent agents change their variable's value in a single round when $k = 1$.

Even with these relaxations, insights can be gained by observing what happens when we consider the portion of $\mathcal{C}_s$ reachable when we restrict the number of agents that can move at each step. Consider the example with two agents described above as we discuss the locations in the configuration hypercube that are reachable. If zero, one,

or both agents can move at every step ($L = 2$), then all $(s + 1)^2$ locations in $\mathcal{C}_s$ are reachable. However, if at most one agent can move in each round ($L = 1$), at every step only one (or neither) agent can move, then only $\frac{(s+1)^2}{2} + s$ locations can be reached. See Figure 5.

For a DCEE, recall that $\mathcal{C}_s$ is defined as all locations in $\mathcal{C}$ with l_∞ distance to the starting point less than or equal to s. If only one agent can move per step, the reachable locations of $\mathcal{C}_s$ instead are all locations $\lambda = (\lambda_1 \dots \lambda_{|V|})$ in $\mathcal{C}$ with l_1 distance less than or equal to s (i.e., $\sum_i(\lambda_i) \leq s$). When two or more agents can move at each step, the set of reachable locations in $\mathcal{C}$ becomes less intuitive. We now define such sets.

In general a location $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_{|V|})$ in $\mathcal{C}_s$ is reachable if and only if

$$\sum_{i=0}^{|V|} \lambda_i \leq Ls$$

where L is the maximum number of agents that can move at a single step, as defined above. To understand this inequality, imagine having n bins corresponding to the n agents (see Figure 6). In each round of the algorithm, you can place up L balls into these bins, with the restriction that you cannot place more than one ball in any one bin in a single round. Placing a ball in the i-th bin on the j-th round corresponds to the agent A_i changing its variable on the j-th round. After s rounds, there are two restrictions of the ending quantities in the bins: (1) the most full bin can contain at most s balls, and (2) the total number of balls must be less than or equal to Ls. Restriction 1 says the location must be within the confines of C_s, and the above inequality upholds restriction 2.

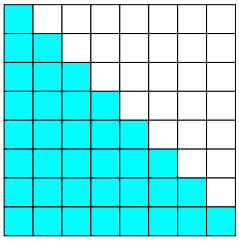


Fig. 5. C_7 for a two vertex, single edge graph, where the initial configuration corresponds to the lower left corner and shaded cells are reachable by an algorithm with $L = 1$.

5.3 *L*-Movement Experimentation

We constructed $\mathcal{C}_s$ for $s = 1 \dots 30$ for a complete graph and a ring graph (both containing 5 nodes). Every constraint has its rewards drawn from a Gaussian distribution with mean 100 and variance 16. For $L = 1 \dots 5$, we calculated the maximum value of the achievable locations. Values are averaged over 50 trails.

Note that the plots for Ring graphs and Complete graphs are exceptionally similar (see Figure 7). The difference between $L = i$ and $L = i + 1$ decreases as i increases in both figures – the $L = 4$ curve overlaps almost entirely with the $L = 5$ curve. Specifically, we note that the $L = 1$ curve is drastically different from the $L = 2$ curve. This indicates that L is a considerable contributing factor to the team uncertainty penalty. We ran experiments on other graphs as well, and using different distributions for reward values. All results showed similar behavior in regards to changing L. See Figure 8.

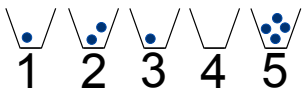


Fig. 6. For a five agent graph and a $L = 2$ algorithm, up to two of the five bins could receive a ball in each round. A configuration (such as the one shown above) would indicate that agent A_i takes value λ_j where there are j balls in the i-th bin. The configuration shown above is reachable by a $L = 2$ algorithm in 4 rounds, but not in 3 rounds – bin 5 contains 4 balls.

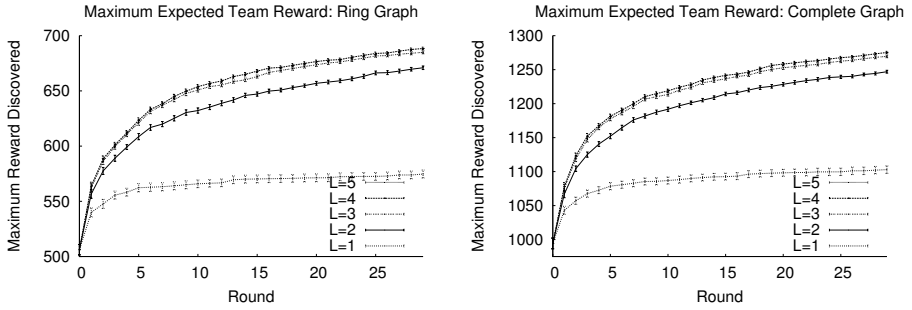


Fig. 7. These plots show the best team reward found in each round while exploring five-agent ring and complete graph DCEEs where all rewards are drawn from a Gaussian distribution with mean 100 and standard deviation 16 for $L = 1 \ldots 5$. Note that the performance difference between $L = 1$ and $L = 2$ is larger than between $L = 2$ and $L = 3$. This relative performance difference decreases as L increases. Values are averaged over 50 runs, and error bars show standard error of the mean.

Let us consider the specific case of SE-Optimistic-1 and SE-Optimistic-2 running on DCEEs defined in part by these graphs (see Figure 9). In the complete graph, $L = 1$ for $k = 1$ and $L = 2$ for $k = 2$. Thus we would imagine that SE-Optimistic-2 would outperform SE-Optimistic-1 by a fair margin. In the ring graph, on the other hand, $L = 2$ for $k = 1$ and $L = 3$ for $k = 2$. This means that in the complete graph, we are potentially obtaining a large benefit from changing $k = 1$ to $k = 2$ because doing so means we go from $L = 1$ to $L = 2$. In the ring graph, however, the same change in k yields a change from $L = 2$ to $L = 3$ which has a much smaller potential performance gain. Therefore we would expect to see a smaller improvement, if any, of SE-Optimistic-2 over SE-Optimistic-1. Empirically, this is precisely what is exhibited both in simulation and on robots.

These results indicate that the L-movement, determined by the value of k, may contribute to the team uncertainty penalty rather than the operations of the specific algorithms. A general algorithm with $L = 2$ will have the ability to explore more of $\mathcal{C}_s$ than an algorithm with $L = 1$. Note that L is strictly determined by the graph topology and k, thus any k-dependent algorithm will experience diminishing returns as k increases. It is worth noting that the L-movement difference between $k = 1$ algorithms and $k = 2$ algorithms will only be higher in larger ring and complete graphs. Recall that for a general ring graph the maximum independent set is of size $\lfloor \frac{|V|}{2} \rfloor$, and is always 1 for a complete graph. This means that the L-movement for a $k = 1$ algorithm is always $L = 1$ for a complete graph, but increases with the number of vertices in a ring graph.

5.4 Predictions

We consider how L varies with k. For these graphs we have the following table:

5 Agents	Ring Graph	Complete Graph
$k = 1:$	$L = 2$	$L = 1$
$k = 2:$	$L = 3$	$L = 2$
$k = 3:$	$L = 3$	$L = 3$
$k = 4:$	$L = 4$	$L = 4$
$k = 5:$	$L = 5$	$L = 5$

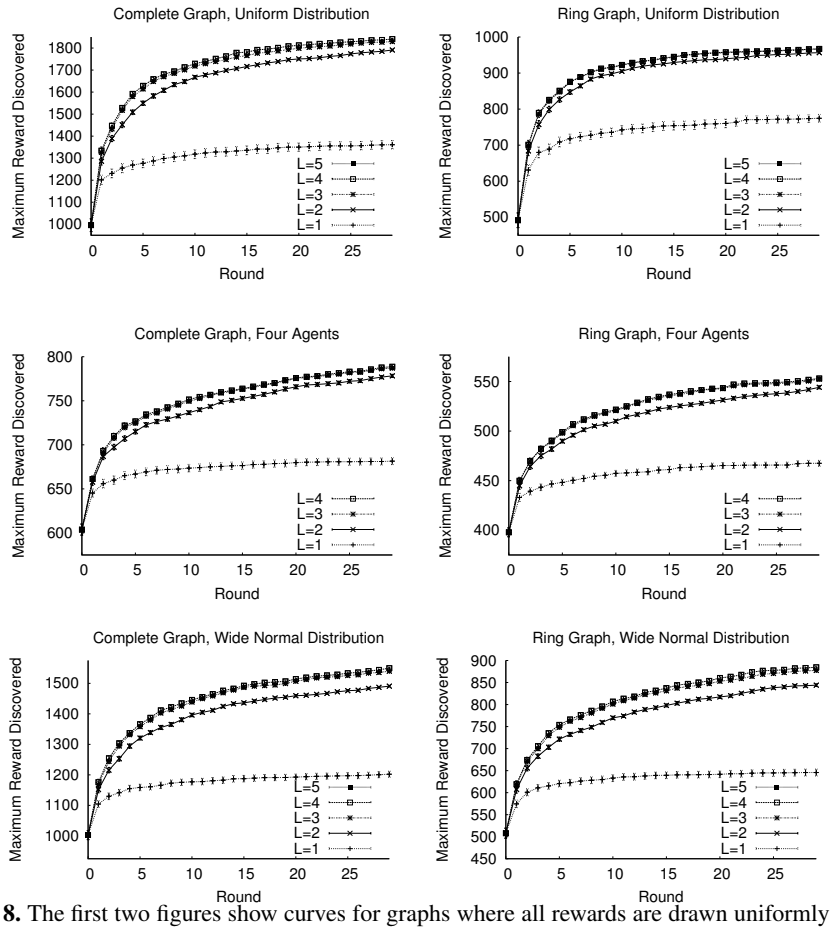


Fig. 8. The first two figures show curves for graphs where all rewards are drawn uniformly from the range $0 \ldots 200$. The next two figures show curves generated from four agent graphs, with rewards drawn from a Gaussian with $\mu = 100$ and $\sigma = 16$. The final two figures again have five agents, but rewards were drawn from a Gaussian with $\mu = 100$ and $\sigma = 32$.

This indicates that the difference between a $k = 1$ algorithm and a $k = 2$ algorithm is much greater in a complete graph than in a ring graph. Moreover, for a given graph and a given k, there is a unique L. Thus this analysis provides a heuristic by which one can *a priori* determine if a team will suffer from the team uncertainty penalty by estimating the potential benefit of a higher k.

Previous results both from simulation (Taylor et al., 2010) and from on robots we see that $k = 1$ optimistic algorithms out perform $k = 2$ algorithms on ring graphs, but underperform on complete graphs. Our experimental analysis of L-movement shows that when increasing the L-movement of an algorithm from $L = 1$ to $L = 2$, a much greater gain is expected than when increasing the L-movement from $L = 2$ to $L = 3$. In the case of complete versus ring graphs, changing an algorithm from $k = 1$ to $k = 2$ corresponds exactly with changing L from 1 to 2 (on a complete graph), and from 2 to 3 (on a ring graph). Our results also predict that a $k = 3$ algorithm on a ring graph

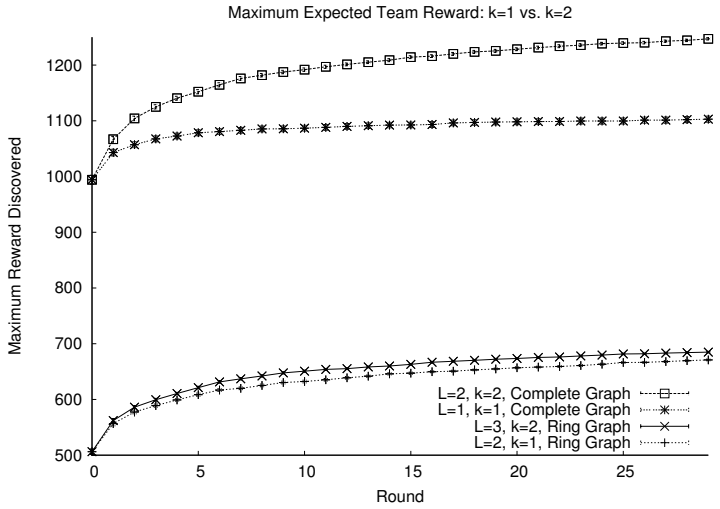


Fig. 9. The difference between $k = 1$ and $k = 2$ for a complete graph is far greater than for a Ring graph. In the ring graph, the increase from $k = 1$ to $k = 2$ results in an increase from $L = 2$ to $L = 3$. In the complete graph, however, we instead have $L = 1$ and $L = 2$.

would perform very similarly to a $k = 2$ algorithm, because $L = 3$ for both graphs. On a complete graph we would predict that a $k = 3$ algorithm would see a gain in total reward over a $k = 2$ algorithm, albeit it a smaller increase than that from $k = 1$ to $k = 2$. This is observed both in simulation and on robots.

6 Conclusions and Future Work

The DCEE framework is a recent extension of DCOPs enabling agents to cope with uncertainty in the environment and problems in which online reward is critical. While investigating the effect of teamwork in DCEEs, our earlier work (Taylor et al., 2010) showed the counter-intuitive team uncertainty penalty in simulation. This paper as further strengthened the claim that the team uncertainty penalty is an important phenomenon worth studying by confirming its existence on physical agents.

By understanding the team uncertainty penalty and what causes it, we will better be able to design algorithms to avoid or less its impact. In order to attempt to better understand this phenomenon, this paper has made a number of contributions. First, we have introduced the notion of L-movement in the context of a DCEE, an extension of k-optimality from the DCOP literature. Second, we show how L-movement and k-optimality are related in a graph-dependant manner, and suggest that the uncertainty penalty may be due to this relation. Third, we have introduced the notion of a configuration hypercube in a DCEE, which we will leverage in the future to theoretically analyze different classes of DCEE algorithms.

This paper has presented evidence that the team uncertainty penalty is possibly an intrinsic phenomenon of the L-movement algorithms, which is determined not by the specifics of the algorithms, but the level of k. In the future, we will continue analysis of the configuration hypercube to develop bounds on L-movement algorithms, design

algorithms to maximize the expected reward found on a given configuration hypercube, and to fully explain the team uncertainty principle.

7 Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments and suggestions. This work was supported in part by a fellowship provided by the National Center for Risk and Economic Analysis of Terrorism Events (CREATE).

Bibliography

Cheng, J., Cheng, W., & Nagpal, R. (2005). Robust and self-repairing formation control for swarms of mobile agents. *AAAI*.

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

Jain, M., Taylor, M. E., Yokoo, M., & Tambe, M. (2009). DCOPs meet the real world: Exploring unknown reward matrices with applications to mobile sensor networks. *IJCAI*.

Maheswaran, R. T., Pearce, J. P., & Tambe, M. (2004). Distributed algorithms for DCOP: A graphical-game-based approach. *PDCS*.

Mailler, R., & Lesser, V. (2004). Solving distributed constraint optimization problems using cooperative mediation. *AAMAS*.

Marden, J., Arslan, G., & Shamma, J. (2007). Connections between cooperative control and potential games illustrated on the consensus problem. *European Control Conference*.

Modi, P. J., Shen, W., Tambe, M., & Yokoo, M. (2005). ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *AIJ*, *161*, 149–180.

Molisch, A. F. (2005). *Wireless communications*. IEEE Press.

Pearce, J., & Tambe, M. (2007). Quality guarantees on k-optimal solutions for distributed constraint optimization. *IJCAI*.

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

Petcu, A., & Faltings, B. (2005). A scalable method for multiagent constraint optimization. *IJCAI*.

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

Stranders, R., Farinelli, A., Rogers, A., & Jennings, N. R. (2009). Decentralized coordination of mobile sensors using the max-sum algorithm. *IJCAI*.

Sutton, R. S., & Barto, A. G. (1998). *Introduction to reinforcement learning*. MIT Press.

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

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

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