Reducing the Team Uncertainty Penalty: Empirical and Theoretical Approaches

Scott Alfeld[1], Kumera Berkele[2], Stephen DeSalvo[1], Tong Pham[2], Daniel Russo[3], Lisa Jing Yan[4], and Matthew E. Taylor[2]

1: University of Southern California, 2: Lafayette College, 3: University of Michigan, 4: York University,
Corresponding author: *taylorm@lafayette.edu*

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

Algorithms, Experimentation, Performance

Keywords

DCEE, Team Uncertainty Penalty, Multiagent Exploration

ABSTRACT

While a significant body of work has examined the exploration / exploitation tradeoff, little has focused on distributed settings. This paper introduces two novel distributed algorithms that outperform existing algorithms in this domain. Additionally, this paper provides a theoretical analysis of one of the algorithms and empirically validates the predictions, which is the first such analysis in this problem setting.

1. INTRODUCTION

The tradeoff between *exploration* and *exploitation* is a common topic of study in artificial intelligence, particularly in the context of reinforcement learning [9] or multi-armed bandit problems [1]. However, such studies have typically focused on single-agent settings. This paper instead examines the question in the context of the *distributed coordination of exploration and exploitation* [4] (DCEE) framework.

The DCEE framework is similar to that of *distributed constraint optimization problems* [6, 8] (DCOPs) in that a team of distributed, cooperative agents work together to maximize a team reward. However, in a DCEE, agents seek to maximize the team reward over time (rather than the final reward), and must explore (rather than having full knowledge provided). As in DCOPs, different amounts of teamwork can be incorporated into DCEE algorithms, where increased teamwork results in joint decisions between larger groups of agents. However, unlike in DCOP algorithms [7], increasing amounts of teamwork can actually decrease the team's reward, even without accounting for the increased communication and computation costs. This result was termed the *team uncertainty penalty* [10], where increasing the teamwork can decrease the team's total reward in certain circumstances.

The primary contributions of this paper are to 1) discuss an existing algorithm and show how it may be improved, effectively reducing the team uncertainty penalty; 2) introduce a novel algorithm, *SE-OptimisticPairs*, which outperforms existing methods; 3) analyze the SE-OptimisticPairs algorithm to predict the team performance in different situations (the first such analysis for a DCEE algorithm); 4) empirically show that the predictions are accurate; and 5) help avoid the uncertainty penalty via these predictions.

The Sixth Annual Workshop on Multiagent Sequential Decision-Making in Uncertain Domains (*MSDM-2011*), held in conjunction with *AAMAS-2011* on May 3, 2011 in Taipei, Taiwan.

2. BACKGROUND

In this section we introduce the DCEE framework, one type of DCEE solution algorithm, and the team uncertainty penalty.

2.1 DCEE Framework

In a DCEE, there is a set of variables that can take on a range of values, where each variable is controlled by a single agent. In an experiment, agents are allowed a set number of *rounds*, in which every agent can decide whether or not it should change the values of the variable(s) it controls. For simplicity, we will assume that an agent controls a single variable. There is a set of binary reward functions between pairs of agents, where the reward on the constraint between these two agents is determined by the agents' variable settings. On any given round, the team's reward is defined as the sum of all binary rewards. The agents initially do not know the matrices defining these different reward functions, but will receive a reward after setting their values (i.e., at the end of a round), and thus learn the reward functions over time. The goal of the team of agents is to maximize the on-line reward, such that the total reward received over all rounds is maximized, necessarily balancing exploring and exploiting the reward functions.

For example, consider Figure 1, which shows a three agent system with two constraints. Currently, each of the three agents have selected variable setting zero (i.e., $x_1 = 0, x_2 = 0, x_3 = 0$), which results in a team reward of 22. One or more of the agents may decide to change their variable setting: agent 1 may select a setting in $[0, k]$, agent 2 in $[0, m]$ and agent 3 in $[0, n]$. Rewards are assumed to be iid with a (possibly known) Gaussian distribution and thus agents have no reason to prefer one unexplored location over another (represented in the figure as a question mark). Without loss of generalization, an agent may exploit its past knowledge, or choose to explore by setting its variable to the next unexplored entry in the reward matrix.

More formally, 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

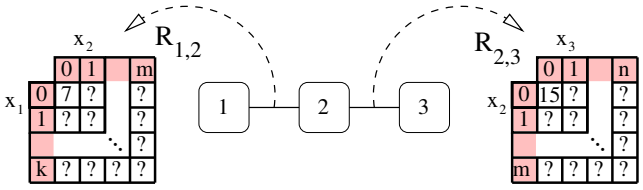

Figure 1: This figure shows an example 3-agent DCEE. Each agent controls one variable and the settings of these three variables determine the reward of the two constraints (and thus the total team reward).

$T \in \mathbb{N}$. 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:

$$R = \sum_{t=0}^{T} \sum_{x_i, x_j \in V} f_{i,j}(d_{i,t}, d_{j,t}),$$

where

$$d_{i,t} \in D_i, d_{j,t} \in D_j, \text{ and } (x_i \leftarrow d_{i,t}, x_j \leftarrow d_{j,t}) \in A_t.$$

Because we assume the agents are distributed, agents may only talk with their *neighbors*, or those agents with which they share a binary reward (we assume that agents know who their neighbors are, and thus who they share a constraint with). An agent knows the value of its own variable setting, and it knows the reward of any binary reward functions it is part of, but it must communicate with its neighbors to learn their variable settings and/or their (non-shared) rewards. Increasing amounts of teamwork in DCEE algorithms means that increasing numbers of agents agree to take a *joint move*, where the set of agents select new variables, and all agents that neighbor this set do not change variable settings. Such coordination is important — for instance, if every agent selected a new variable setting on every turn, the team's reward would not be expected to improve, even though the agents were gathering more and more information about the reward functions.

Experiments in this paper use the released DCEE simulator[1] for its experiments. This simulator models a mobile wireless network problem, where each agent has a wireless radio. The agents' physical location is their variable setting, binary rewards are defined as the signal strength between pairs of agents (some agents are unable to communicate directly), and the signal strengths can be modeled as being randomly drawn from a Gaussian distribution. The agents' goal is to maximize the signal strength of the network, balancing the need to maintain a high-quality network with the possibility of exploring and finding improved settings. Full details of the domain may be found elsewhere [4]. The details of the simulator are not important for this paper, but the fact that it is based on a physically motivated problem (and not written to fit the algorithms introduced in this paper) bolsters our claim that the algorithms we introduce and the conclusions we reach may be applicable in real-world scenarios.

2.2 SE-Optimistic

Static estimation (SE) DCEE algorithms are a class of approaches that have been successful in both simulation and on physical robots [4]. *SE-Optimistic-1* is a greedy approach where each agent assumes that if it were allowed to change its variable, it would maximize each of its binary rewards. Assuming all agents have the same number of binary reward functions, and that the maximum reward is constant for all of these functions, only the agent with the lowest total reward per neighborhood will be allowed to change its value. Agents exchange information with those in their neighborhood, and only a single agent per neighborhood is allowed to change its value.

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

Algorithm 1 SE-OPTIMISTIC-2

```
1:  for each neighbor i do
2:      Send variable assignment and reward matrices to i
3:  Find maximum gain, g, the corresponding neighbor to pair
    with, p, and the variable assignment, a:
    g, p, a ← getMaxGainAndAssignmentForPair()
4:  Send OfferPair to agent p
5:  doPair ← False
6:  for all OfferPair messages received do
7:      if agent requesting to pair is p then
8:          Send Accept to agent p
9:          doPair ← True
10: if (Did not received Accept from p) or (not doPair) then
11:     p ← ∅
12:     Find max gain and preferred assignment:
        g, a ← getMaxGainAndAssignment()
13: Send Bid (g, p) to all neighbors
14: Receive n Bids from all neighbors, ignoring message from p
15: G ← max_n Bids_n
16: if g > G then
17:     UpdateAssignment(a)
```

SE-Optimistic-2 [10] is an extension of SE-Optimistic-1: two agents can change variables as a pair and no agents that neighbor the pair change values. The hope is that when agents exchange more information they will be able to make larger joint moves and achieve a higher team reward faster than SE-Optimistic-1. These algorithms are reminiscent of the *k-optimal* algorithms in DCOPs, where k agents per neighborhood can change values and higher values of k result in improved team rewards [7].

The psudocode for SE-Optimistic-2 is shown in Algorithm 1, where the methods *getMaxGainAndAssignmentForPair*() and *getMaxGainAndAssignment*() use the optimistic assumption to estimate what their improvement would be if allowed to move. If the algorithm is amended so that lines 1–9 are removed and *doPair* = False, the psudocode is equivalent to SE-Optimistic-1.

2.3 Team Uncertainty Penalty

In DCOP problems, increasing the amount of teamwork would typically lead to improved team reward, at the expense of additional messages sent between agents and increased amounts of computational time. The *team uncertainty penalty* [10] was found to occur in DCEE settings, where increasing the amount of teamwork would sometimes *decrease* the team's reward, even when not accounting for the number of messages sent, nor the increased computation time. The surprising phenomenon means that it may be very difficult to decide what algorithm will perform better in a given situation without empirically testing all candidate algorithms. While some initial algorithms were introduced to reduce the affect of the penalty, they all suffered from additional parameters, further increasing the difficulty of using such algorithms in real-world situations. The penalty seemed to depend most heavily on the number of binary rewards each agent was connected to: in graphs with few links, low-teamwork algorithms outperform higher teamwork algorithms. In graphs with many shared binary rewards, the penalty did not surface and increased teamwork resulted in increased team reward.

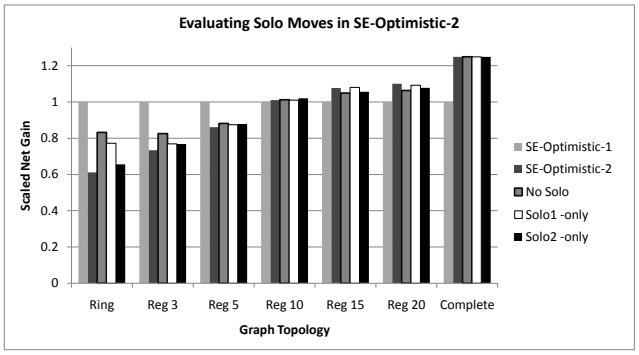

Figure 2: The y-axis shows the total net gain for each algorithm (higher is better), scaled so that SE-Optimistic-1 is 1.0.

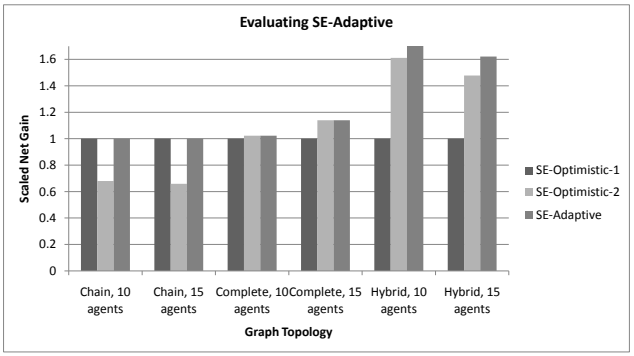

Figure 3: The scaled gain of the SE-Adaptive algorithm can match, or outperform, both SE-Optimistic-1 and SE-Optimistic-2.

3. SOLO MOVES IN SE-OPTIMISTIC-2

In DCOP algorithms, increasing the amount of teamwork (i.e., increasing k) generally increases the value of the solution found. If a solution is 2-optimal, it is by definition also 1-optimal. Similarly, DCEE algorithms were designed so that they were able to use different values of k [4]. In Algorithm 1, line 12, agents can bid to change their variable without a teammate. If the single agent wins the bid to change its variable, we term this a *Type 1 Solo Move*. The other case in which we found an agent could change variables alone in a $k = 2$ algorithm is when two agents successfully pair (line 9) and one agent wins the bid within its neighborhood, but the other does not — when one agent in a pair moves but the other does not, we term this outcome a *Type 2 Solo Move*. In both cases, we empirically found that such variable changes happen less than 5% of the time, but they had a non-trivial effect on the algorithm's performance, as discussed next.

Figure 2 shows how teamwork and solo movements can affect the team's performance. Each bar in the graph summarizes results from 30 independent trials of 40 agents acting over 100 time steps. The y-axis shows the net gain of the algorithms, which summarizes the team's performance, after being scaled so that the net gain of SE-Optimistic-1 is 1.0. We examine seven different topologies from a ring graph (2 neighbors per agent) and a 3-regular graph (3 neighbors each) up through a complete graph (39 neighbors each).

First, that the team uncertainty penalty is clearly exhibited in the algorithm, consistent with previous work. As the number of neighbors per agent increases, the performance of SE-Optimistic-2 increases relative to SE-Optimistic-1. In a ring graph, SE-Optimistic-2 receives roughly 60% of the SE-Optimistic-1's net gain, while it receives roughly 120% in a complete graph. Second, this figure shows that solo moves can decrease the team performance, particularly at low graph densities. For instance, in a ring topology, disabling type 1 solo moves (labeled "Solo2-only") results in an improvement over SE-Optimistic-2. Disabling type 2 solo moves (labeled "Solo1-only") results in increased team rewards, and disabling both types of solo movements (labeled "No Solo") significantly improves the reward relative to SE-Optimistic-2. For graphs where there are fewer than 15 neighbors per agent, removing the solo moves improves the performance of SE-Optimistic-2. For regular-15 through complete graphs, removing the solo moves decreases team performance slightly, or has a negligible effect. Third, the performance of No Solo algorithm does not suffer as much from the team uncertainty penalty as SE-Optimistic-2. In particular, the No Solo algorithm continues to outperform SE-Optimistic-1 in high density graphs, but does not underperform SE-Optimistic-1 in low density graphs as much as SE-Optimistic-2 underperforms SE-Optimistic-1. Thus, No Solo may be a better choice when the graph density is initially unknown — without other information, it may be better to use the No Solo algorithm, rather than risk performing poorly with SE-Optimistic-1 or SE-Optimistic-2.

4. AN ADAPTIVE TEAMWORK ALGORITHM

Others [10] have designed algorithms to mediate the team uncertainty penalty. However, these algorithms have relied on a parameter that must be manually tuned per graph type, significantly limiting their usefulness. No Solo is an improvement over these previous methods because no additional parameter need be tuned. In this section we introduce SE-Adaptive, which can run either SE-Optimistic-1 or SE-Optimistic-2 *per agent*, improving the overall performance of the team.

An agent running SE-Adaptive can decide whether to use SE-Optimistic-1 or SE-Optimistic-2, depending on the number of neighbors it has. We use a heuristic based on a per-agent ratio, $r =$ *# neighbors / total # of agents*. If $r < \frac{1}{3}$, then the agent uses SE-Optimistic-1. Otherwise, the agent tries to run SE-Optimistic-2 with any neighbors able to reciprocate, and if that fails, fall back on SE-Optimistic-1.

Our hypothesis is the per-agent ratio, r, is an important potential factor in determining the level of teamwork required. We conducted two sets of experiments which include same number of rounds and variable settings: one set tested the full graph (where each agent has $n - 1$ connections), while the other testsed chain graphs. The number of agents ranged from 3–15. Comparing the performance of each test on SE-1 and SE-2, the threshold value of $\frac{1}{3}$ was found empirically.

Figure 3 shows the results of using different numbers of agents running on different graph topologies for 50 rounds. SE-Adaptive performs as well as the best of SE-Optimistic-1 and SE-Optimistic-2 on chain and complete graphs. "Hybrid" graphs are constructed such that $\lfloor \frac{1}{2} \rfloor$ of the agents form a connected clique, and the remainder of the agents form a chain connected to one agent in the clique — SE-Adaptive agents in the clique use SE-Optimistic-2 and SE-Adaptive agents in the chain use SE-Optimistic-1, outperforming both algorithms individually.

5. SE-OPTIMISTICPAIRS

The adaptive algorithm presented in the previous section outperforms both SE-Optimistic-1 and SE-Optimistic-2. However, it must be trained on values from a particular reward distribution —

Algorithm 2 SE-OptimisticPairs(a)

Require: a – The agent to check
1: $T_{\forall a} \leftarrow$ PairsOf(a)
2: **for** $T \in T_{\forall a}$ **do**
3: $N_T \leftarrow$ NeighborsOf(T)
4: WinsBid $\leftarrow$ True
5: **for** $n \in N_T$ **do**
6: $T_{\forall n} \leftarrow$ PairsOf(n)
7: **for** $T_n \in T_{\forall n}$ **do**
8: **if** Gain(T) $<$ Gain(T_n) **then**
9: WinsBid $\leftarrow$ False
10: **if** WinsBid **then**
11: **return** True
12: **return** False

if the distribution of rewards is not known ahead of time, such training data may be hard or impossible to come by. This section introduces the *SE-OptimisticPairs* algorithm, which constrains the SE-Optimistic-2 algorithm to dis-allow solo moves. Unlike the No Solo algorithm, SE-OptimisticPairs uses an extra layer of reasoning as *only* pairs of agents change values.

Most significantly, the algorithm SE-OptimisticPairs has been designed so that it can be theoretically analyzed. To this point, previous work in the DCEE domain has been strictly empirical. By providing an algorithmic analysis for SE-Optimistic-1 and SE-OptimisticPairs, we show that questions in DCEE domains can be addressed with theoretical, as well as empirical, techniques.

5.1 Pseudocode

SE-OptimisticPairs can be summarized as follows. A team of two agents will win the ability to change its values if none of its neighbors change values, which in turn happens if no neighbor is part of a team with a higher expected gain. In other words, any par of agents that has the lowest value in its neighborhood will win the bid to change its values.

The per-agent reasoning in SE-Optimistic-Pairs is summarized in Algorithm 2. The function PairsOf(a) returns a set of all possible pairings for agent a. NeighborsOf(T) returns the set of all agents that are neighbors with either agent in team T. Finally, Gain is calculated as in SE-Optimistic-2: the pair of agents (optimistically) assume that if they are allowed to change their values, all of their rewards will be set to the maximum value. Ties should be broken consistently (line 8) (our implementation uses unique agent identifiers).

5.2 Theoretical Analysis of SE-OptimisticPairs

To analyze the SE-OptimisticPairs algorithm, and its relationship to SE-Optimistic-1, we consider the first round of the DCEE algorithm SE-Optimistic-1 and SE-OptimisticPairs run on DCEE graphs. We assume that all rewards are drawn i.i.d. from a discrete distribution that is well approximated by a Gaussian with known mean μ and variance σ^2. We assume rewards to be continuous values rather than integers so as to remain general, and avoid tie-breaking issues. In order to conform to the floored Gaussian used in the simulator [4], we approximate a floored Gaussian with a mean μ as a continuous Gaussian with mean $\mu - 0.5$ [5].

Let A be the set of all agents and the random variable $r_{i,j}$ be the reward on the constraint between agents i and j (i.e., between A_i and A_j). R_i is the total reward of A_i:

$$R_i \equiv \sum_j r_{i,j}, \text{where } A_j \text{ is a neighbor of } A_i$$

We focus on analyzing the first round of the algorithms, when all rewards are drawn from the Gaussian and are i.i.d. By analyzing the first round, exact probabilities can be found for *any* graph. Subsequent rounds, however, will depend heavily on the graph structure, which makes any explicit analysis difficult in general. In this paper, we provide a method of analyzing the first-round behavior given an arbitrary graph. For demonstration, we focus specifically on Ring and Complete graphs, and note that analysis of of rounds beyond the first round is left as future work. We define $\mathcal{A}_i$ to be the event that A_i changes values on the first round. For example, in SE-Optimistic-1, $\mathcal{A}_i$ occurs if $R_i < R_j, \forall j \in$ Neighbors(A_i), where Neighbors(A_i) is the set of agents which neighbor A_i.[2]

We define an agent to be "good" if its current total reward is above some value g. That is to say A_i is "good" *iff* $R_i \geq g$. One natural choice of g arises when all agents have the same degree (number of neighbors).[3] Let $g = d\mu$, where d is the degree of agents; we say an agents is "good" if it is above the expected (random) reward.

Below we will consider multiple graph structures and values of g, finding $P(\mathcal{A}_i \wedge R_i \geq g)$. The notation $\mathcal{A}_i$ means that agent i is allowed to change its variable, meaning that the previous expression is the probability that agent A_i is both good and changes values. We will show that, in ring graphs, the expected number of good agents that change value (on the first round) is higher in SE-OptimisticPairs than in SE-Optimistic-1. This is significant because it helps explain the team uncertainty penalty — an increase in the size of the team (i.e., k) results in an increase of agents changing variables even when they are in a high-reward position. Further, we will show that this is not the case in complete graphs. Additionally, we show how to determine the probability both of an agent moving, and of that agent moving given that it's "good" for a general graph.

First, we find the probability that agent A_i changes values on the first round, which is used no find the expected number of agents that change value. We then find the probability that agent A_i both change values, and is good, as well as the expected number of such agents. We compare the results of SE-Optimistic-1 and SE-OptimisticPairs for several graph topologies and values of k.

5.3 Probability an Agent Changes Values

To determine the probability that agent A_i changes values, denoted $P(\mathcal{A}_i)$, we exploit the fact that this decision is made locally. When $k = 1$, A_i changes values if it could gain more total reward than any of its neighbors. For higher values of k, however, A_i will changes value if any of the connected k-tuples of agents containing A_i decide to change value.

Let T_1 and T_2 denote two teams of size k that both contain agent A_i. If the agents in T_1 are able to change values then T_2 cannot, due to the shared agent. Let $\mathcal{T}$ be the event that team T changes its value. We therefore observe that

$$P(\mathcal{A}_i) = \sum_{T \in \text{TeamsOf(i)}} P(\mathcal{T})$$

where TeamsOf(i) is the set of all k-sized connected teams containing A_i.

We now describe a process by which we calculate the probability that some team T changes values on the first round. Consider a ring graph (see Figure 4) with agents ordered A_1 through A_8 when the agents use SE-OptimisticPairs. Let $T = (A_i, A_j)$ be written as $T_{i,j}$ and the value of the constraint between them $r_{i,j}$. To find the probability that agents A_4 and A_5 change value, $P(\mathcal{T}_{4,5})$, we

[2]More formally, Neighbors(A_i) $= \{A_j \in A \mid \exists r_{i,j}\}$.
[3]With minor alterations, g can vary from agent to agent.

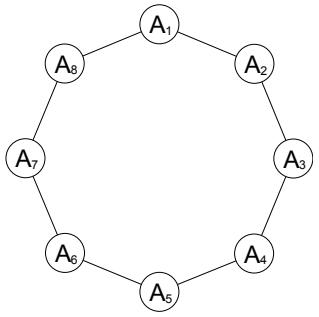

Figure 4: A ring graph with eight agents

first note that $\mathcal{T}_{4,5}$ occurs *iff* none of $\{\mathcal{T}_{2,3}, \mathcal{T}_{3,4}, \mathcal{T}_{5,6}, \mathcal{T}_{6,7}\}$ occur. $P(\mathcal{T}_{4,5}) =$

$$P\begin{pmatrix} r_{3,4}+r_{4,5}+r_{5,6} & < & r_{1,2}+r_{2,3}+r_{3,4} & \wedge \\ r_{3,4}+r_{4,5}+r_{5,6} & < & r_{2,3}+r_{3,4}+r_{4,5} & \wedge \\ r_{3,4}+r_{4,5}+r_{5,6} & < & r_{4,5}+r_{5,6}+r_{6,7} & \wedge \\ r_{3,4}+r_{4,5}+r_{5,6} & < & r_{5,6}+r_{6,7}+r_{7,8} & \end{pmatrix}$$

Simplifying, we obtain:

$$P(\mathcal{T}_{4,5}) = P\begin{pmatrix} -r_{4,5}-r_{5,6}+r_{1,2}+r_{2,3} & > & 0 & \wedge \\ -r_{5,6}+r_{2,3} & > & 0 & \wedge \\ -r_{3,4}+r_{6,7} & > & 0 & \wedge \\ -r_{3,4}-r_{4,5}+r_{6,7}+r_{7,8} & > & 0 & \end{pmatrix}$$

We then rewrite this probability as

$$P(\mathcal{T}_{4,5}) = P\begin{pmatrix} Y_1 & > & 0 & \wedge \\ Y_2 & > & 0 & \wedge \\ Y_3 & > & 0 & \wedge \\ Y_4 & > & 0 & \end{pmatrix}$$

for appropriate values of Y_i. Because each Y_i is a sum of independent Gaussians, they are themselves drawn from a Gaussian. However, the Y_i's are not independent due to the overlap of constraints. Thus: $Y_1 \sim N(0, 4\sigma^2), Y_2 \sim N(0, 2\sigma^2), Y_3 \sim N(0, 2\sigma^2), Y_4 \sim N(0, 4\sigma^2)$, and the covariance of the random variables are determined by the number of shared variables. Therefore we have a vector $\mathbb{Y} = (Y_1, Y_2, Y_3, Y_4)^\mathsf{T}$ drawn from a multivariate Gaussian with mean vector $\theta = (0, 0, 0, 0)$ and covariance matrix

$$\Sigma = \sigma^2 \begin{pmatrix} 4 & 2 & 0 & 1 \\ 2 & 2 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 1 & 0 & 2 & 4 \end{pmatrix}$$

In general, we calculate the mean vector by counting the number of constraints in each Y_i, and construct the covariance matrix by counting the number of constraints shared pairs of Y_i's. Algorithm 3 (in the Appendix) returns the covariance matrix for any graph, not just ring graphs. The following is the density function for the multivariate normal

$$f(\mathbb{Y}|\theta, \Sigma) = \frac{1}{2\pi^{\frac{\alpha}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(\frac{1}{2}(\mathbb{Y}-\theta)^\mathsf{T}\Sigma^{-1}(\mathbb{Y}-\theta)\right)$$

where α is the length of $\mathbb{Y}$ (in our example, $\alpha = 4$). Thus:

$$P(\mathcal{T}_{4,5}) = \int_{\mathbb{R}_+^\alpha} f(\mathbb{Y}|\theta, \Sigma)\mathrm{d}\mathbb{Y}$$

Plugging this into the previous equation, and letting $\mathbb{Y}_T, \theta_T$, and Σ_T denote $\mathbb{Y}, \theta$, and Σ created by a k-tuple T:

$$P(\mathcal{A}_i) = \sum_{T \in \text{TeamsOf}(i)} \int_{\mathbb{R}_+^\alpha} f(\mathbb{Y}_T|\theta_T, \Sigma_T)\mathrm{d}\mathbb{Y}_\mathbb{T}$$

where TeamsOf(i) is the set of all k-sized connected tuples that contain agent A_i. R_+^α is the region defined by $(0, \infty) \times \ldots (0, \infty)$, the cross product taken α times over the positive region of each dimension. The function f can be thought of as a multi-dimensional bell curve that is stretched and compressed in each dimension according to the entries in the covariance matrix.

5.4 Calculating the Number of Agents Updating

We now consider calculating the expected number of agents that change values on the first round (this method is summarized in Algorithm 4 in the appendix). We will use this value, along with the expected number of good agents that change values, to compute the proportion of agents that change values for various values of k and graph topologies. Let $1_{\mathcal{A}_i}$ be the indicator function of $\mathcal{A}_i$, that is to say that $1_{\mathcal{A}_i} = 1$ if $\mathcal{A}_i$ occurs and 0 otherwise. Let the random variable $|\mathcal{A}|$ be the number of agents that change value. We observe that

$$\mathbb{E}(|\mathcal{A}|) = \sum_i^n 1_{\mathcal{A}_i} = \sum_i^n \sum_{T \in \text{TeamsOf}(i)} \int_{\mathbb{R}_+^\alpha} f(\mathbb{Y}_T|\theta_T, \Sigma_T)\mathrm{d}\mathbb{Y}_\mathbb{T}$$

We now find the probability that an agent both changes values, and is in a good position before doing so. This will let us compute the proportion of moving agents that are good, which depends heavily on the value of k and the graph topology.

Consider the example from the previous subsection:

$$P(\mathcal{A}_4 \wedge R_4 \geq g) = P(\mathcal{T}_{4,5} \wedge R_4 \geq g) + P(\mathcal{T}_{3,4} \wedge R_4 \geq g)$$

In general we see that

$$P(\mathcal{A}_i \wedge R_i \geq g) = \sum_{T \in \text{TeamsOf}(i)} P(\mathcal{T} \wedge R_i \geq g)$$

which is calculated similar to $P(\mathcal{A}_i)$. In our example:

$$P(\mathcal{T}_{4,5} \wedge R_4 \geq g) = P\begin{pmatrix} r_{3,4}+r_{4,5} & \geq & g & \wedge \\ -r_{4,5}-r_{5,6}+r_{1,2}+r_{2,3} & > & 0 & \wedge \\ -r_{5,6}+r_{2,3} & > & 0 & \wedge \\ -r_{3,4}+r_{6,7} & > & 0 & \wedge \\ -r_{3,4}-r_{4,5}+r_{6,7}+r_{7,8} & > & 0 & \end{pmatrix}$$

Let $Y_0 = r_{3,4} + r_{4,5}$, or in general $Y_0 = R_i$. We note that $Y_0 \sim N(2\mu, 2\sigma^2)$, or in general R_i is drawn from $N(d\mu, d\sigma^2)$ where d is the degree of A_i. We then let $\mathbb{Y} = (Y_0, Y_1, Y_2, \ldots)^\mathsf{T}$, and construct Σ as we did previously. Again as before, $f(\mathbb{Y}|\theta, \Sigma)$ is defined as the multivariate Gaussian's density function.

The integration over the multivariate Gaussian differs slightly from the previous case because $R_i \geq g$ whereas all other $Y_i > 0$. We must integrate over the region of the multivariate Gaussian where $Y_0 \geq g$, and all other $Y_i > 0$. Therefore we let $\bar{\mathbb{Y}} = (Y_1, Y_2, \ldots)^\mathsf{T}$ be the mean vector without the Y_0 element, and obtain

$$P(\mathcal{T}_{4,5} \wedge R_4 \geq g) = \int_g^\infty \int_{\mathbb{R}_+^{\alpha-1}} f(\mathbb{Y}|\theta, \Sigma)\mathrm{d}\bar{\mathbb{Y}}\mathrm{d}Y_0$$

And in the general case we let $\mathbb{Y}_T$ denote the value $\mathbb{Y}$ created by the connected k-tuple T and obtain

$$P(\mathcal{A}_i \wedge R_i \geq g) = \sum_{T \in \mathcal{T}_i} \int_g^{\infty} \int_{\mathbb{R}_+^{\alpha_T - 1}} f(\mathbb{Y}_T | \theta_T, \Sigma_T) \mathrm{d}\bar{\mathbb{Y}}_T \mathrm{d}Y_{T0}$$

Similarly to how we calculated the expected number of agents that change value, we now compute the expected number of agents that both change values and are good (denoted $|\mathcal{A}_g|$):

$$\mathbb{E}(|\mathcal{A}_g|) = \sum_{i}^{n} \sum_{T \in \text{TeamsOf}(i)} \int_g^{\infty} \int_{\mathbb{R}_+^{\alpha_T - 1}} f(\mathbb{Y}_T | \theta_T, \Sigma_T) \mathrm{d}\bar{\mathbb{Y}}_T \mathrm{d}Y_{T0}$$

5.5 Results

We have described above how to obtain both the expected number of agents that change value, and the expected number of agents that both change value and are good. A procedural approach for obtaining these numbers is outlined in the Appendix (if accepted, we will additionally release the code used to calculate these numbers). We implemented this algorithm in Python, and used numerical integration techniques implemented using the R library mvtnorm [3] to determine the quantities for varying values of g.

Our implementation works for general graphs, and we obtained numbers for ring and complete graphs with various numbers of agents. We used the rewards distributions $N(0, 1)$ and $N(100, 16^2)$ to be consistent with past work in the DCEE domain [4]. It is worth noting that if X is a normal random variable drawn from $N(\mu, \sigma^2)$ then it can be rescaled arbitrarily ($(X - \mu)/\sigma$ is a random variable drawn from $N(0, 1)$). Further, we note that the expected number of agents that change values is independent of the distribution and its parameters, as no agent is more likely to change values than any other (assuming all agents have the same number of edges in the graph and all have the same reward distribution).

We first find $P(R_i \geq g, \mathcal{A}_i)$ for different numbers of agents and values of g. We normalize these values to obtain $P(R_i \geq g \mid \mathcal{A}_i)$. For ring graphs, the number of agents does not matter. This is because the decision of whether or not to change values is determined purely by a limited section of the graph, and thus the induced covariance matrix is of fixed size. In a complete graph, however, all agents interact when determining which change values, and the covariance matrix grows with the number of agents.

$\mathbb{E}(|\mathcal{A}|)$ and $\mathbb{E}(|\mathcal{A}_g|)$ for ring and complete graphs and various (k, g) were then calculated. Let $\bar{\sigma}$ be the variance of R_i and $\bar{\mu}$ its mean, determined by the graph. In a ring graph, $\bar{\mu}$ is 2μ as every agent is a part of two constraints, each with mean μ. Similarly, $\bar{\sigma}$ is $\sqrt{2}\sigma$. In a complete graph with n agents, $\bar{\mu}$ is $(n-1)\mu$, and $\bar{\sigma}$ is $\sqrt{(n-1)}\sigma$.

In all cases, more good agents will change value in ring graphs than in complete. However, more agents in total change value in *ring* graphs. In complete graphs, exactly k agents will change value in each round. Table 2 shows the expected number of agents changing values in ring graphs, and empirical results show that the predictions are accurate. Empirical results are averaged over 1,000 independent trials, and the standard error is shown.

Of the agents that change value, we then find the expected percentage of them that are good, shown in Table 1. In the same table, we show the results from running 1,000 simulations and recording the percentage of agents that moved for different values of g. The recorded mean and standard errors show that the theoretical calculations match the empirical results. These results help explain the team uncertainty penalty. In ring graphs, SE-Optimistic-1 will allow fewer agents to move that are in "good" positions, relative to SE-OptimisticPairs, while in complete graphs, the opposite is true. Further, in Figure 5, we see that SE-OptimisticPairs suffers less from the team uncertainty penalty than SE-Optimistic-2, and is competitive with No Solo, even though SE-OptimisticPairs has no tunable parameters.

Table 2: $|\mathcal{A}|$ for ring graphs, 40 agents

	$k = 1$	$k = 2$
Calculated	10.000	12.121
$n = 10$	2.500	3.032
Empirical	9.987 ± 0.037	12.240 ± 0.059

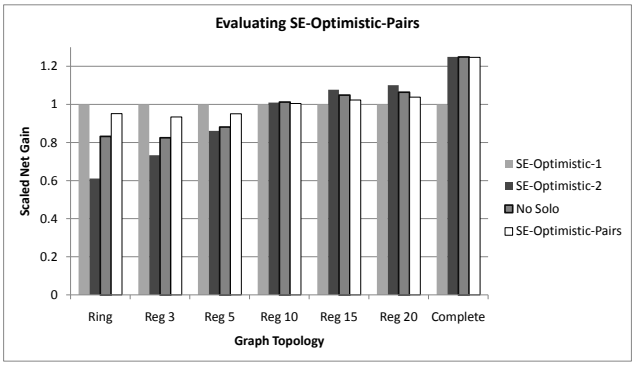

Figure 5: SE-OptimisticPairs outperforms SE-Optimistic-2 when agents have few neighbors, outperforms SE-Optimistic-1 when agents have many neighbors, and is competitive with No Solo.

6. CONCLUSION AND FUTURE WORK

This paper has introduced two novel DCEE algorithms which reduce the effect of the team uncertainty penalty. In SE-Adaptive, agents can individually decide on the appropriate level of teamwork, where the decision rule has been empirically pre-determined, based on the distribution of rewards. In SE-OptimisticPairs, the agents have no tunable parameters, which is a significant improvement over existing algorithms designed to avoid the penalty. More importantly, the SE-OptimisticPairs algorithm has been theoretically analyzed so that we may predict whether agents should pair or not, and these predictions are confirmed empirically.

In the future, we would like to extend our analysis so that it is easy to compute $\mathbb{E}(|\mathcal{A}_g|)$ for arbitrary graphs. We intend apply the insights regarding solo moves to a different type of DCEE algorithms, termed Balanced Exploration algorithms. This paper focused on different levels of teamwork, but did not explicitly analyze communication overheads — such an analysis may prove useful for better quantifying the benefits of different algorithms. Finally, it would be interesting to see if the insights about the amount of teamwork in DCEE problems can be applied to DisCSP (distributed constraint satisfaction problems), similar to DisCSPs work [2] that focused on algorithmic run time, rather than the performance.

Acknowledgements

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). The authors would like to thank Milind Tambe and Manish Jain for their help on previous DCEE research, and the anonymous reviewers for their comments and suggestions.

Table 1: $|\mathcal{A}_g|$: Rewards drawn from $N(100, 16^2)$ for 40 agents. Empirical results display the standard error.

			$g = \bar{\mu} - 3\bar{\sigma}$	$g = \bar{\mu} - 2\bar{\sigma}$	$g = \bar{\mu} - \bar{\sigma}$	$g = \bar{\mu}$	g = $\bar{\mu} + \bar{\sigma}$
$\mathbb{E}(\lvert\mathcal{A}_g\rvert)$	Ring	$k = 1$	0.994	0.917	0.583	0.153	0.011
		$k = 2$	0.995	0.927	0.606	0.161	0.010
	Complete	$k = 1$	0.907	0.266	0.001	4×10^{-11}	3×10^{-24}
		$k = 2$	0.945	0.461	0.005	2×10^{-8}	2×10^{-17}
Empirical	Ring	$k = 1$	0.993 ± 0.012	0.921 ± 0.016	0.580 ± 0.02	0.159 ± 0.013	0.012 ± 0.004
		$k = 2$	0.994 ± 0.016	0.931 ± 0.017	0.605 ± 0.022	0.170 ± 0.014	0.012 ± 0.004
	Complete	$k = 1$	0.904 ± 0.007	0.224 ± 0.011	0	0	0
		$k = 2$	0.949 ± 0.005	0.426 ± 0.012	0.003 ± 0.002	0	0

7. REFERENCES

[1] R. E. Bellman. A problem in the sequential design of experiments. *Sankhya*, 16:221–229, 1956.

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

[3] A. Genz and F. Bretz. *Computation of Multivariate Normal and t Probabilities*. Lecture Notes in Statistics. Springer-Verlag, 2009.

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

[5] A. A. Kulikova and Y. V. Prokhorov. Distribution of the fractional parts of random vectors: The Gaussian case I. *Theory of Probability and its Applications*, 48(2):355–359, 2004.

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

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

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

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

[10] M. E. Taylor, M. Jain, Y. Jin, M. Yooko, and M. Tambe. When should there be a "me" in "team"? distributed multi-agent optimization under uncertainty. In *AAMAS*, 2010.

APPENDIX

A. ADDITIONAL ALGORITHMS

Here we provide pseudo-code for the algorithms used to compute the values presented earlier in the paper. Note that these algorithms compute the number of agents moving, with no notion of g. To incorporate g, minor modifications must be made to the covariance matrix (Algorithm 3), and the integration in Algorithm 4.

Given a graph, Algorithm 4 computes the expected number of agents that will changes value on the first round. It loops through every agent, and for each agent it must calculate a covariance matrix, and then integrate over a multivariate Gaussian. Note that, for ring and complete graphs, the probability that an agent changes values is the same for all agents due to symmetry. Thus the algorithm can be optimized by calculating the probability that agent A_1 changes values, and multiplying it by n.

To construct the covariance matrix for a team T, we loop over all teams 'touching' T, and calculate the involved random variables for each. We will use this matrix to define a multi-variate Gaussian which we integrate over in Algorithm 4. The mean of the multivariate Gaussian is calculated with Algorithm 5. Note that there may be many teams that share an constraint with T, and we are looping over all of them.

Given a team and a graph, EdgesTouching(T, G) returns the set of constraints that would be resampled if T were to changes value. More formally, EdgesTouching(T, G) returns

$$\{e = (a, b) \in E \mid a \in T \vee b \in T\}$$

The function AllTeamsTouching(T, G) returns the set of all connected teams (of size k) which cannot change value if team T changes values. Note that in k-dependent DCEE algorithms, two teams cannot both change value if they share a constraint. More formally, AllTeamsTouching(T, G) returns

$$\{T' \subset V \mid |T'| = k \wedge (\text{EdgesTouching}(T', G) \cap \text{EdgesTouching}(T, G)) \neq \emptyset\}$$

Algorithm 3 CreateCovarianceMatrix(T, G)

Require: G = {V, E} - The input graph
Require: $T \subset V$ - The team to changes value ($|T| = k$)
1: $E_T \leftarrow$ EdgesTouching(T, G)
2: $A \leftarrow$ AllTeamsTouching(T, G)
3: **for** $i = 1 \ldots |A|$ **do**
4: **for** $j = 1 \ldots |A|$ **do**
5: $C[i][j] = |(\text{EdgesTouching}(A_i, G) \triangle E_T) \cap (\text{EdgesTouching}(A_j, G) \triangle E_T)|$
6: **return** C

Algorithm 4 GetExpectedNumberOfAgentsChanging(G)

1: $E \leftarrow 0$
2: **for** A_i in A **do**
3: $P \leftarrow 0$
4: **for** T in $\mathcal{T}_i$ **do**
5: $\Sigma \leftarrow$ CreateCovarianceMatrix(T, G)
6: $\theta \leftarrow$ GetMeanVector(T, G)
7: $P \leftarrow P + \int_{\mathbb{R}_+^\alpha} f(\mathbb{Y}_T|\theta, \Sigma) \mathrm{d}\mathbb{Y}_\mathbb{T}$
8: $E \leftarrow E + P$
9: **return** E

Algorithm 5 GetMeanVector(T, G)

1: O $\leftarrow$ AllTeamsTouching(T, G)
2: **for** $i = 1 \ldots |O|$ **do**
3: $\theta_i \leftarrow \mu(|\text{EdgesTouching}(O_i) \setminus \text{EdgesTouching}(T)| - |\text{EdgesTouching}(T) \setminus \text{EdgesTouching}(O_i)|)$
4: **return** θ