Two Decades of Multiagent Teamwork Research: Past, Present, and Future

Matthew E. Taylor[1], Manish Jain[2], Christopher Kiekintveld[3], Jun-young Kwak[2], Rong Yang[2], Zhengyu Yin[2], and Milind Tambe[2]

[1] Lafayette College, Easton, PA
[2] University of Southern California, Los Angeles, CA
[3] University of Texas at El Paso, El Paso, TX

Abstract. This paper discusses some of the recent cooperative multiagent systems work in the TEAMCORE lab at the University of Southern California. Based in part on an invited talk at the CARE 2010 workshop, we highlight how and why execution-time reasoning has been supplementing, or replacing, planning-time reasoning in such systems.

1 Introduction

There have been over two decades of work in computer science focusing on cooperative multiagent systems and teamwork [6, 15, 28], much of it in the context of planning algorithms. In addition to the problems encountered in single-agent scenarios, multi-agent problems have a number of significant additional challenges, such as how agents should share knowledge, assist each other, coordinate their actions, etc. These extra considerations often make multi-agent problems exponentially more difficult, relative to single agent tasks, in terms of both the computational complexity and the amount of memory required for a planner.

As discussed in the following section, the BDI (Belief-Desire-Intention) framework was one of the first to directly addresses multi-agent problems with significant theoretical and experimental success. In addition to formalizing teamwork relationships, BDI became popular because of its ability to reduce computation. Rather than requiring agents to laboriously plan for all possible outcomes, or expect a centralized planner to account for a state space exponential in the number of agents, the BDI approach allowed agents to reason about their plans at execution-time and adapt to information gathered about the environment and teammates.

Later techniques focused more on preemptive planning, requiring a computationally intensive planning phase up front, but allowed the agents to execute their joint plan with few requirements at execution time. Two particular approaches, DCOPs and DEC-POMDPs, will be discussed in later sections of this chapter. The DCOP framework allows agents to explicitly reason about their coordination in a network structure in order to achieve a joint goal. DEC-POMDPs use centralized planning to reasoning about uncertainty, both in the sensors and actuators of the agents, producing provably (near-) optimal plans for the multi-agent team.

While the current generation of multi-agent techniques, including DCOPs and DEC-POMDPs, have been successful in a number of impressive contexts, they often fail to scale up to large numbers of agents. More important, they ignore the power of execution-time reasoning and focus on planning-time reasoning. In this chapter, we argue that the multi-agent community would do well to focus on incorporating more execution-time reasoning, possibly inspired by past BDI methods, in order to 1) reduce planning time, 2) reduce the amount of required coordination, and/or 3) allow agents to gracefully handle unforeseen circumstances. After we give a brief introduction to the BDI framework, we will discuss some of our own work in the DCOP and DEC-POMDP frameworks, highlighting the benefits of integrating execution-time and planning-time reasoning.

2 The Past: BDI

The Belief-Desires-Intention (BDI) formalism was the dominant approach to multiagent teamwork in the mid-90's, spurred on in large measure from the work on *SharedPlans* [6] and *joint-intentions* [15]. The key idea behind BDI was to capture some of the "common-sense" ideas of teamwork and address questions like: "why does communication arise in teamwork," "why do teammates help each other," and "how can a teammate best help another teammate?" Answers to these questions were captured in a logic-based domain-independent form, allowing for the same types of team-level reasoning in disparate domains (e.g., a team of airplane pilots or a team of personal office assistants).

One important contribution of BDI was that this domain independence allowed programmers to reason about teams at very high levels of abstraction. BDI teamwork libraries could be responsible for the low-level control of coordinating the team, handling failures, assigning agent roles, etc., allowing the programmer to instead focus on coding at the *team level* of abstraction. BDI proved useful in (at least) three distinct ways:

1. through direct implementation of the logic as agent decision-making code,
2. as inspiration for operationalization in other languages, and
3. for the rational reconstruction of implemented systems.

Benefit #2 in particular has been useful in that it has allowed for the development and deployment of large-scale teams (c.f., [9, 28]).

A second important contribution of BDI was to focus on execution time reasoning. As discussed in the previous section, a set of pre-defined rules could be used at execution time, allowing agents to react to their environment without needing to plan for all possible team contingencies ahead of time.

3 DCOPs

This section briefly introduces the DCOP framework and then discusses recent advancements in multi-agent asynchronous reasoning and multi-agent exploration.

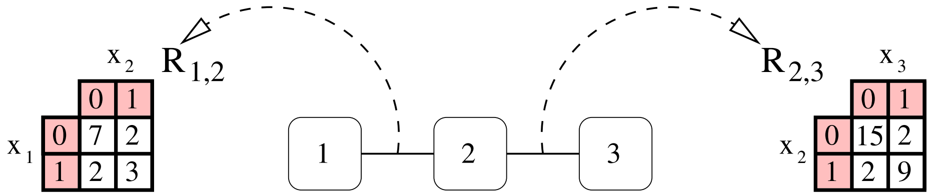


Fig. 1. This figure depicts a three agent DCOP.

3.1 Background

Distributed Constraint Optimization Problems (DCOP) [18, 19] are cooperative multi-agent problems where all agents are part of a single team; they share a common reward function. DCOPs have emerged as a key technique for distributed reasoning in multiagent domains, given their ability to optimize over a set of distributed constraints, while keeping agents' information private. They have been used for meeting scheduling problems [17], for allocating tasks (e.g., allocating sensors to targets [14]) and for coordinating teams of agents (e.g., coordinating unmanned vehicles [27] and coordinating low-power embedded devices [5]).

Formally, a DCOP 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 variable's assignment. 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 binary constraints and associated payoff or reward functions, $f_{ij} : D_i \times D_j \rightarrow N$, is maximized. More specifically, find an assignment, A, s.t. F(A) is maximized: $F(A) = \sum_{x_i, x_j \in V} f_{ij}(d_i, d_j)$, where $d_i \in D_i, d_j \in D_j$ and $x_i \leftarrow d_i, x_j \leftarrow d_j \in A$. 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 as shown. If agents 2 and 3 choose the value 1, the agent pair gets a reward of 9. If agent 1 now chooses value 1 as well, the total solution quality of this complete 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. The agents in a DCOP are traditionally assumed to have *a priori* knowledge of the corresponding reward functions.

3.2 k-OPT and t-OPT: Algorithms and Results

When moving to large-scale applications, it is critical to have algorithms that scale well. This is a significant challenge for DCOP, since the problem is known to be NP-hard. Recent work has focused on incomplete algorithms that do not guarantee optimal solutions, but require dramatically less computation and communication to achieve good solutions. Most of the incomplete algorithms in the literature provide no guarantees on solution quality, but two new methods based on local optimality criteria, k-size optimality [23] and t-distance optimality [11], offer both fast solutions and bounds on solution quality.

The key idea of k-size optimality is to define optimality based on a local criteria: if no subset of k agents can improve the solution by changing their assignment, an assignment is said to be k-size optimal. Using a larger group size gives better solutions (and bounds), but requires additional computational effort. A variation on this idea, t-distance optimality, uses distance in the graph from a central node to define the groups of agents that can change assignment. Formally, we define these optimality conditions as follows.

Definition 1. *Let $D(A, A')$ denote the set of nodes with a different assignment in A and A'. A DCOP assignment A is* k-size optimal *if $R(A) \geq R(A')$ for all A' for which $|D(A, A')| \leq k$.*

Consider the DCOP in Figure 1. The assignment $\{1, 1, 1\}$ is a k-size optimal solution for $k = 1$ (with reward of 12), but not $k = 2$ and $k = 3$. It is 1-size optimal because the reward is reduced if any single variable changes assignment. However, if x_2 and x_3 change to 0 the reward increases to 17 from 12, so $\{1, 1, 1\}$ is not 2-size optimal.

Definition 2. *Let $T(v_i, v_j)$ be the distance between two variables in the constraint graph. We denote by $\Omega_t(v) = \{u|T(u, v) \leq t\}$ the t-group centered on v. A DCOP assignment A is* t-distance optimal *if $R(A) \geq R(A')$ for all A', where $D(A, A') \subseteq \Omega_t(v)$ for some $v \in V$.*

There are at most n distinct t-groups in the constraint graph, centered on the n variables. There may be fewer than n distinct groups if some $\Omega_t(v)$ comprise identical sets of nodes. Consider again the DCOP in Figure 1. Assignment $\{1, 1, 1\}$ is 0-distance optimal, because each t-group contains a single node, equivalent to $k = 1$. However, $\{1, 1, 1\}$ is not 1-distance optimal. The $t = 1$ group for x_2 includes both other variables, so all three can change to assignment 0 and improve the reward to 22.

Both k-size optimal solution and t-distance optimal solution have proven quality bounds that improve with larger value of k or t. However, there is a distinct tradeoff between k-size and t-distance optimality. In k-size optimality, the number of nodes in each individual group is strictly bounded, but the number of distinct k-groups may be very large, especially in dense graphs. For t-distance optimality the situation is reversed; the number of groups is bounded by the number of variables, but the size of an individual t-group is unbounded and may be large in dense graphs. Empirically, this has significant implications for the speed of solution algorithms for computing the two types of local optima.

Algorithms One advantage of k-size and t-distance optimality is that they can be computed using local search methods. *DALO (Distributed Asynchronous Local Optimization)* (DALO) is an algorithmic framework for computing either k-size or t-distance optimal solutions for any setting of k or t. DALO is fast, and supports anytime, asynchronous execution. This makes it ideal for dynamic environments that require significant execution-time reasoning. At a high level, DALO executes in three phases:[4]

1. **Initialization** Agents send initialization messages to nearby agents, which are used to find all of the k or t groups in the constraint graph and assign each group a unique leader.

[4] More details about the algorithm can be found elsewhere [11].

2. **Optimization** Each group leader computes a new optimal assignment for the group, assuming that all *fringe nodes* maintain their current assignment, where *fringe nodes* of a group are directly connected to a group member, but are not members themselves.
3. **Implementation** The group leader implements the new assignment if it is an improvement, using an asynchronous locking/commitment protocol.

DALO is particularly useful in execution time reasoning of large agent teams for the following reasons. First, DALO allows agents to reason and act asynchronously by following the locking/commitment protocol, avoiding expensive global synchronization in execution. Second, as a locally optimal algorithm, DALO requires much smaller amount of computation and communication on each individual agent as opposed to a globally optimal algorithm, leading to efficient execution in dynamic environments. Third, as verified by our simulations, the convergence speed of DALO remains almost constant with increasing number of agents, demonstrating its high scalability.

Experimental Evaluation Here, we present an abbreviated set of results showing some of the advantages of local optimality criteria and the DALO algorithm. We test k-size optimality and t-distance optimality using a novel asynchronous testbed and performance metrics.[5] In our experiments, we vary the setting of computation / communication ratio (CCR) to test algorithms across a broad range of possible settings with different relative cost for sending messages and computation. Katagishi and Pearce's KOPT [10], the only existing algorithm for computing k-size optima for arbitrary k, is used as a benchmark algorithm. In addition, we examine tradeoffs between k-size and t-distance optimality.

We show results for: 1) random graphs where nodes have similar degrees, and 2) NLPA (Nonlinear preferential attachment) graphs in which there are large hub nodes. Figure 2 shows a few experimental results. As shown in Figures 2, both DALO-k and DALO-t substantially outperform KOPT, converging both more quickly and to a higher final solution quality.[6] In general, DALO-t converges to a higher final solution quality, though in some cases, the difference is small. Convergence speed depends on both the graph properties and the CCR setting. DALO-k tends to converge faster in random graphs (Figure 2(a)) while DALO-t converges faster in NLPA graphs (Figure 2(b)). Figure 2(c) shows the scalability of DALO-t and DALO-k as we increase the number of nodes tenfold from 100 to 1000 for random graphs. The time necessary for both DALO-k and DALO-t to converge is nearly constant across this range of problem size, demonstrating the high scalability of local optimal algorithms.

The asynchronous DALO algorithm provides a general framework for computing both k-size and t-distance optimality, significantly outperforming the best existing algo-

[5] Code for the DALO algorithm, the testbed framework, and random problem instance generators are posted online in the USC DCOP repository at http://teamcore.usc.edu/dcop.

[6] The settings t=1 and k=3 are the most closely comparable; they are identical in some special cases (e.g., ring graphs), and require the same maximum communication distance between nodes. Empirically, these settings are also the most comparable in terms of the tradeoff between solution quality and computational effort.

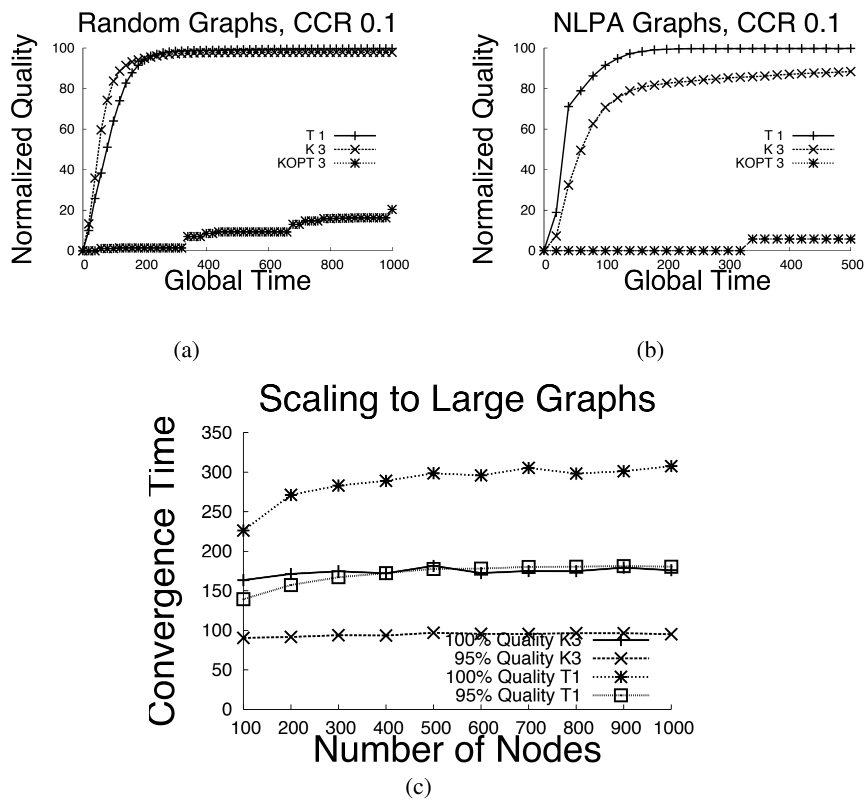


Fig. 2. Experimental results comparing DALO-k, DALO-t, and KOPT

rithm, KOPT, in our experiments and making applications of high values of t and k viable. DALO allows us to investigate tradeoffs: DALO-t consistently converges to better solutions in practice than DALO-k. DALO-t also converges more quickly that DALO-k in many settings, particularly when computation is costly and the constraint graph has large hub nodes. However, DALO-k converges more quickly on random graphs with low computation costs. Investigating additional criteria for group selection (e.g., hybrids of k-size and t-distance) can be a key avenue for future work.

3.3 DCEE: Algorithms and the Team Uncertainty Penalty

Three novel challenges must be addressed while applying DCOPs to many real-world scenarios. First, agents in these domains may not know the initial payoff matrix and must explore the environment to determine rewards associated with different variable settings. All payoffs are dependent on agents' joint actions, requiring them to coordinate in their exploration. Second, the agents may need to maximize the total accumulated reward rather than the instantaneous reward at the end of the run. Third, agents could face a limited task-time horizon, requiring efficient exploration. These challenges disallow direct application of current DCOP algorithms which implicitly assume that all agents have knowledge of the full payoff matrix. Furthermore, we assume that agents cannot fully explore their environment to learn the full payoff matrices due to the task-time horizon, preventing an agent from simply exploring and then using a globally optimal algorithm. Indeed, interleaving an exploration and exploitation phase may improve accumulated reward during exploration.

Such problems are referred to as DCEE (Distributed Coordination or Exploration and Exploitation) [7], since these algorithms must simultaneously explore the domain and exploit the explored information. An example of such a domain would be a mobile sensor network where each agent (mobile sensor) would explore new values (move to new locations) with the objective of maximizing the overall cumulative reward (link quality, as measured through signal strength) within a given amount of time (e.g., within 30 minutes).

We here discuss both k=1 and k=2 based solution techniques for DCEE problems. Most previous work in teamwork, including previous results in k-optimal algorithms, caused us to expect that increasing the level of teamwork in decision making would lead to improved final solution quality in our results. In direct contradiction with these expectations, we show that blindly increasing the level of teamwork may actually decrease the final solution quality in DCEE problems. We call this phenomenon the *teamwork uncertainty penalty* [29], and isolate situations where this phenomenon occurs. We also introduce two extensions of DCEE algorithms to help ameliorate this penalty: the first improves performance by disallowing teamwork in certain settings, and the second by *discounting* actions that have uncertainty.

Solution Techniques This section describes the DCEE algorithms. Given the inapplicability of globally optimal algorithms, these algorithms build on locally optimal DCOP algorithms. While all the algorithms presented are in the framework of MGM [22], a k-optimal algorithm for a fixed k, the key ideas can be embedded in any locally optimal DCOP framework.

In k=1 algorithms, every agent on every round: (1) communicates its current value to all its neighbors, (2) calculates and communicates its *bid* (the maximum gain in its local reward if it is allowed to change values) to all its neighbors, and (3) changes its value (if allowed). An agent is allowed to `move` its value if its *bid* is larger than all the bids it receives from its neighbors. At quiescence, no single agent will attempt to move as it does not expect to increase the net reward.

k=2 algorithms are "natural extensions" of k=1 algorithms. In these algorithms, each agent on each round: (1) selects a neighbor and sends an *Offer* for a joint variable change, based on its estimate of the maximal gain from a joint action with this neighbor; (2) for each offer, sends an *Accept* or *Reject* message reflecting the agent's decision to pair with the offering agent. Agents accept the offer with the maximum gain. (3) calculates the *bid* or the *joint gain* of the pair if an offer is accepted, and otherwise calculates the *bid* of an individual change (i.e. reverts to k=1 if its offer is rejected). (4) If the bid of the agent is highest in the agent's neighborhood, a confirmation message is sent to the partnering agent in case of joint move, following which (5) the joint / individual variable change is executed. The computation of the offer per agent in a k=2

DCEE algorithms is as in k=1, the offer for a team of two agents is the sum of individual offers for the two agents without double counting the gain on the shared constraint. k=2 algorithms require more communication than k=1 variants, however, have been shown to reach higher or similar solution quality in traditional DCOP domains [16].

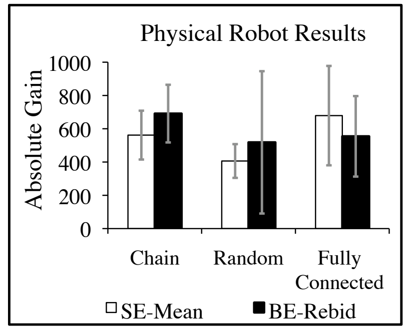
Static Estimation (SE) algorithms calculate an estimate of the reward that would be obtained if the agent explored a new value. **SE-Optimistic** assumes the maximum reward on each constraint for all unexplored values for agents. Thus, in the mobile sensor network domain, it assumes that if it moved to a new location, the signal strength between a mobile sensor and every neighbor would be maximized. On every round, each agent bids its expected gain: *NumberLinks* $\times$ *MaximumReward* $- R_c$ where R_c is the current reward. The algorithm then proceeds as a normal k=1 algorithm, as discussed above. SE-Optimistic is similar to a 1-step greedy approach where agents with the lowest rewards have the highest bid and are allowed to `move`. Agents typically explore on every round for the entire experiment. On the other hand, **SE-Mean** assumes that visiting an unexplored value will result in the average reward to all neighbors (denoted μ) instead of the maximum. Agents have an expected gain of: *NumberLinks* $\times \mu - R_c$, causing the agents to greedily explore until they achieve the average reward (averaged over all neighbors), allowing them to converge on an assignment. Thus, SE-Mean does not explore as many values as SE-Optimistic, and is thus more conservative.

Similarly, *Balanced Exploration* (BE) algorithms allow agents to estimate the maximum expected utility of exploration given a time horizon by executing `move`, as well as precisely when to stop exploring within this time horizon. The utility of exploration is compared with the utility of returning to a previous variable setting (by executing `backtrack`) or by keeping the current variable setting (executing `stay`). The gain for the action with the highest expected reward is bid to neighbors. This gain from exploration depends on: (1) the number of timesteps T left in the trial, (2) the distribution of rewards, and (3) the current reward R_c of the agent, or the best explored reward R_b if the agent can backtrack to a previously explored state. The agent with the highest bid (gain) per neighborhood wins the ability to move. **BE-Rebid** computes this expected utility of `move` given that an agent can, at any time, `backtrack` to the best explored value, R_b, in the future. On the other hand, **BE-Stay** assumes that an agent is not allowed to backtrack, and thus decides between to `move` to a new value or `stay` in the current value until the end of the experiment. Thus, BE-Stay is more conservative than BE-Rebid and explores fewer values.

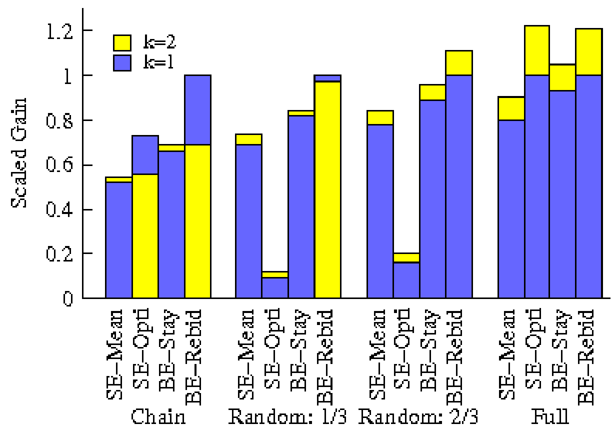
Results The DCEE algorithms were tested on physical robots and in simulation.[7] A set of Creates (mobile robots from iRobot, shown in Figure 3(a)) were used. Each Create has a wireless CenGen radio card, also shown in the inset in Figure 3(a). These robots relied on odometry to localize their locations. Three topologies were tested with physical robots: 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 were repeated 5 times with a time horizon of 20 rounds.

[7] The simulator and algorithm implementations may be found at http://teamcore.usc.edu/dcop/.

(a) iRobot Create



(b) Physical Robot Results



(c) Simulation Results

Fig. 3. Experimental results for DCEE algorithms on robots and in simulation

Figure 3(b) shows the results of running BE-Rebid and SE-Mean on the robots. SE-Mean and BE-Rebid were chosen because they were empirically found out to be the best algorithms for settings with few agents. The y-axis shows the actual gain achieved by the algorithm over the 20 rounds over no optimization. The values are signal strengths in decibels (dB). BE-Rebid performs better than SE-Mean in the chain and random graphs, but worse than SE-Mean 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.

Figure 3(c) compares the performance of the k=1 variants with the k=2 variants. The y-axis is the scaled gain, where 0 corresponds to no optimization and 1 corresponds to the gain of BE-Rebid-1. The x-axis shows the four different topologies that were used for the experiments. The different topologies varied the graph density from chain to fully connected with random $\frac{1}{3}$ and $\frac{2}{3}$ representing graphs where roughly $\frac{1}{3}$ and $\frac{2}{3}$ of number of links in a fully connected graph are randomly added to the network respectively. 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 (chain and random $\frac{1}{3}$). For instance, SE-Optimistic-1 and BE-Rebid-1 outperform their k=2 counterparts on chain graphs (paired t-tests, $p < 5.3 \times 10^{-7}$), and BE-Rebid-1 outperforms BE-Rebid-2 on Random graphs with $\frac{1}{3}$ of their links (although not statistically significant). This reduction in performance in k=2 algorithms is known as the *team uncertainty penalty*.

Understanding Team Uncertainty That k=2 does not dominate k=1 is a particularly surprising result precisely because previous DCOP work showed that k=2 algorithms reached higher final rewards [16, 23]. This phenomenon is solely an observation of the total reward accrued and does not consider any penalty from increased communication or computational complexity. Supplemental experiments that vary the number of agents on different topologies and vary the experiment lengths all show that the factor most critical to relative performance of k=1 versus k=2 is the graph topology. Additionally, other experiments on robots (not shown) also show the team uncertainty penalty — this surprising behavior is not limited to simulation.

Two key insights used to mitigate team uncertainty penalty are: (1) k=2 variants change more constraints, because pairs of agents coordinate joint moves. Given k=2 changes more constraints, its changes could be less "valuable." (2) k=2 variants of BE-Rebid and SE-Optimistic algorithms can be overly aggressive, and prohibiting them from changing constraints that have relatively low bids may increase their achieved gain (just like the conservative algorithms, BE-Stay-2 and SE-Mean-2, outperform their k=1 counterparts, as shown in Figure 3(c)). Indeed, algorithms have been proposed that discourage joint actions with low bids, and/or discount the gains for exploration in the presence of uncertainty and have been shown to successfully lessen the team uncertainty penalty [29].

4 DEC-POMDPs

This section provides a brief introduction to DEC-POMDPs and then highlights a method that combines planning- and execution-time reasoning.

4.1 Background

The Partially Observable Markov Decision Problem (POMDP) [8] is an extension of the Markov Decision Problem (MDP), which provides a mathematical framework for modeling sequential decision-making under uncertainty. POMDPs model real world decision making process in that they allow uncertainty in the agents' observations in addition to the agents' actions. Agents must therefore maintain a probability distribution over the set of possible states, based on a set of observations and observation probabilities. POMDPs are used to model many real world applications including robot navigation [4, 12] and machine maintenance [24]. Decentralized POMDPs (DEC-POMDPs) model sequential decision making processes in multiagent systems. In DEC-POMDPs, multiple agents interact with the environment and the state transition depends on the behavior of all the agents.

4.2 Scaling-up DEC-POMDPs

In general, the multiple agents in DEC-POMDPs have only limited communication abilities, complicating the coordination of teamwork between agents. Unfortunately, as shown by Bernstein et al. [3], finding the optimal joint policy for general DEC-POMDPs is NEXP-complete. There have been proposed solutions to this problem which typically fall into two categories. The first group consists of approaches for finding approximated solution using efficient algorithms [2, 20, 30]; the second group of approaches has focused on finding the global optimal solution by identifying useful subclasses of DEC-POMDPs [1, 21]. The limitation of first category of work is the lack of guarantee on the quality of the solution, while the second category of approaches sacrifices expressiveness.

4.3 Execution-time Reasoning in DEC-POMDPs

Although DEC-POMDPs have emerged as an expressive planning framework, in many domains agents will have an erroneous world model due to *model uncertainty*. Under such uncertainty, inspired by BDI teamwork, we question the wisdom of paying a high computational cost for a promised high-quality DEC-POMDP policy — which may not be realized in practice because of inaccuracies in the problem model. This work focuses on finding an approximate but efficient solution built upon the first category as discussed earlier to achieve effective teamwork via execution-centric framework [26, 31, 32], which simplifies planning by shifting coordination (i.e., communication) reasoning from planning time to execution time. Execution-centric frameworks have been considered as a promising technique as they significantly reduce the worst-case planning complexity by collapsing the multiagent problem to a single-agent POMDP at plan-time [25, 26]. They avoid paying unwanted planning costs for a "high-quality" DEC-POMDP policy by postponing coordination reasoning to execution-time.

The presence of model uncertainty exposes three key weaknesses in past execution-centric approaches. They: (i) rely on complete but erroneous model for precise online planning; (ii) can be computationally inefficient at execution-time because they plan for joint actions and communication at every time step; and (iii) do not explicitly consider the effect caused by given uncertainty while reasoning about communication, leading to a significant degradation of the overall performance.

MODERN (MOdel uncertainty in Dec-pomdp Execution-time ReasoNing) is a new execution-time algorithm that addresses model uncertainties via execution-time communication. MODERN provides three major contributions to execution-time reasoning in DEC-POMDPs that overcome limitations in previous work. First, MODERN maintains an approximate model rather than a complete model of other agents' beliefs, leading to space costs exponentially smaller than previous approaches. Second, MODERN selectively reasons about communication at critical time steps, which are heuristically chosen by *trigger points* motivated by BDI theories. Third, MODERN simplifies its decision-theoretic reasoning to overcome model uncertainty by boosting communication rather than relying on a precise local computation over erroneous models.

We now introduce the key concepts of *Individual estimate of joint Beliefs (IB)* and *Trigger Points*. IB^t is the set of nodes of the possible belief trees of depth t, which is used in MODERN to decide whether or not communication would be beneficial and to choose a joint action when not communicating. IB can be conceptualized as a subset of team beliefs that depends on an agent's local history, leading to an exponential reduction in belief space compared to past work [26, 31]. The definition of trigger points is formally defined as follows:

Definition 3. *Time step t is a* trigger point *for agent i if either of the following conditions are satisfied.*
Asking *In order to form a joint commitment, an agent requests others to commit to its goal, P. Time step t is an* Asking trigger point *for agent i if its action changes based on response from the other agent.*
Telling *Once jointly committed to P, if an agent privately comes to believe that P is achieved, unachievable, or irrelevant, it communicates this to its teammates. Time step t is a* Telling trigger point *for agent i if the other agent's action changes due to the communication.*

Empirical Validation: The MODERN algorithm first takes a joint policy for the team of agents from an offline planner as input. As an agent interacts with the environment, each node in IB is expanded using possible observations and joint actions from the given policy, and then MODERN detects trigger points based on the belief tree. Once an agent detects a trigger point, it reasons about whether or not communication would be beneficial using cost-utility analysis. MODERN's reasoning about communication is governed by the following formula: $U_{\mathrm{C}}(i) - U_{\mathrm{NC}}(i) > \sigma$. $U_{\mathrm{C}}(i)$ is the expected utility of agent i if agents were to communicate and synchronize their beliefs. $U_{\mathrm{NC}}(i)$ is the expected utility of agent i when it does not communicate, and σ is a given communication cost. $U_{\mathrm{NC}}(i)$ is computed based on the individual evaluation of heuristically estimated actions of other agents. If agents do not detect trigger points, this implies there is little chance of miscoordination, and they take individual actions as per the given policy.

We first compare the performance of MODERN for four different levels of model uncertainty (α) in the 1-by-5 and 2-by-3 grid domains with two previous techniques: ACE-PJB-COMM (APC) [26] and MAOP-COMM (MAOP) [31] as shown in Table 1. In both domains, there are two agents trying to perform a joint task. The 1-by-5 grid domain is defined to have 50 joint states, 9 joint actions, and 4 joint observations. In the 2-by-3 grid, there are 72 joint states, 25 joint actions, and 4 joint observations. In both tasks, each movement action incurs a small penalty. The joint task requires that both agents perform the task together at a pre-specified location. If the joint task is successfully performed, a high reward is obtained. If the agents do not both attempt to perform the joint task at the same time in the correct location, a large penalty is assessed to the team[8]. The communication cost is 50% of the expected value of the policies. The time horizon (i.e., the deadline to finish the given task) is set to 3 in this set of experiments. In Table 1, α in column 1 represents the level of model error. Error increases (i.e., the agents' model of the world becomes less correct, relative to the ground truth) as α decreases. Columns 2–4 display the average reward achieved by each algorithm in the 1-by-5 grid domain. Columns 5–7 show the results in the 2-by-3

[8] More detailed domain descriptions and comparisons are available elsewhere [13].

Table 1. Comparison MODERN with APC and MAOP: Average Performance

	1x5 Grid			2x3 Grid		
α	MODERN	APC	MAOP	MODERN	APC	MAOP
10	3.38	-1.20	-1.90	3.30	-1.20	-3.69
50	3.26	-1.20	-2.15	3.30	-1.20	-3.80
100	3.18	-1.20	-2.12	3.04	-1.20	-3.79
10000	2.48	-1.20	-2.61	2.64	-1.20	-4.01

grid domain. We performed experiments with a belief bound of 10 per time-step for our algorithm.

Table 1 shows that MODERN (columns 2 and 5) significantly outperformed APC (columns 3 and 6) and MAOP (columns 4 and 7). MODERN received statistically significant improvements (via t-tests), relative to other algorithms. MAOP showed the worst results regardless of α.

Another trend in Table 1 is that the solution quality generally increases as α decreases. When model uncertainty is high, the true transition and observation probabilities in the world have larger differences from the values in the given model. If the true probabilities are lower than the given model values, communication helps agents avoid miscoordination so that they can avoid a huge penalty. If the true values are higher, agents have more opportunity to successfully perform joint actions leading to a higher solution quality. When model uncertainty is low, the true probability values in the world are similar or the same as the given model values. Thus, agents mostly get an average value (i.e., the same as the expected reward). Thus, as model error increases, the average reward could increase.

We then measured runtime of each algorithm in the same domain settings. *Note that the planning time for all algorithms is identical and thus we only measure the average execution-reasoning time per agent.* In both grid domains, MODERN and APC showed similar runtime (i.e., the runtime difference between two algorithms was not statistically significant). MAOP took more time than MODERN and APC by about 80% in the 1-by-5 grid domain and about 30% in the 2-by-3 grid domain. Then, we further make the 2-by-3 grid domain complex to test the scalability of each algorithm. Two individual tasks are added to the grid, which require only one agent to perform. In this new domain, the number of joint states is 288, the number of joint actions is 49, and the number of joint observations is 9. If any agent performs the individual task action at the correct location, the team receives a small amount of reward. If an agent attempts to perform the individual task in a location where the action is inappropriate, a small penalty will be assessed. If an agent chooses the action `wait`, there will be no penalty or reward. In this domain, while APC or MAOP could not solve the problem within the time limit (i.e., 1800 seconds), MODERN only took about 120 seconds to get the solution. These results experimentally show that MODERN is substantially faster than previous approaches while achieving significantly higher reward.

One of our key design decisions in MODERN is to use trigger points to reason about communication. In these experiments, we show how significant the benefits of selective reasoning are. We used the same scaled-up 2-by-3 grid domain that was used for runtime comparisons. Figure 4 shows runtime in seconds on the y-axis and the time horizon on the x-axis. Time horizon was varied from 3 to 8.

The communication cost was set to 5% of the expected utility of the given policy. As shown in the figure, MODERN can speedup runtime by over 300% using trigger points. In particular, the average number of trigger points when T=8 was about 2.6. This means MODERN only reasons about communication for about 1/3 of the total time steps, which leads to roughly three-fold improvement in runtime.

MODERN thus represents a significant step forward because it allows agents to efficiently reason about communication at execution-time, as well as to be more robust to errors in the model than other DEC-POMDP methods.

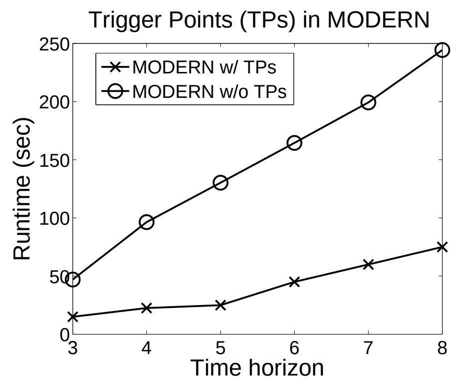


Fig. 4. Selective reasoning in MODERN

5 Conclusion

The multi-agent community was started with a BDI mindset, emphasizing execution-item reasoning. In recent years, however, much of the work has shifted to planning-time reasoning. The primary argument in this chapter is that we believe execution-time reasoning to be a critical component to multi-agent systems and that it must be robustly combined with planning-time computation. We have presented recent techniques in the cooperative multi-agent domains of DCOPs and DEC-POMDPs, emphasizing asynchronous reasoning, run-time exploration, and execution-time communication reasoning. Our hope is that as methods combining planning- and execution-time reasoning become more common, the capability of large teams of complex agents will continue to improve and deployments of such teams in real-world problems will become increasingly common.

References

1. Becker, R., Zilberstein, S., Lesser, V., Goldman, C.V.: Solving Transition Independent Decentralized Markov Decision Processes. Journal of Artifical Intelligence Research (2004)
2. Bernstein, D.S., Hansen, E., Zilberstein, S.: Bounded policy iteration for decentralized POMDPs. In: IJCAI (2005)
3. Bernstein, D.S., Givan, R., Immerman, N., Zilberstein, S.: The complexity of decentralized control of MDPs. In: UAI (2000)
4. Cassandra, A., Kaelbling, L., Kurien, J.: Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In: IROS (1996)
5. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: AAMAS (2008)
6. Grosz, B.J., Sidner, C.L.: Plans for discourse. In: Cogent, P.R., Morgan, J., Pollack, M. (eds.) Intentions in Communication. MIT Press (1990)
7. Jain, M., Taylor, M.E., Yokoo, M., Tambe, M.: DCOPs meet the real world: Exploring unknown reward matrices with applications to mobile sensor networks. In: IJCAI (2009)

8. Kaelbling, L., Littman, M., Cassandra, A.: Planning and acting in partially observable stochastic domains. Artificial Intelligence, 101 (1998)
9. Kaminka, G.A., Tambe, M.: Robust multi-agent teams via socially attentive monitoring. Journal of Artificial Intelligence Research 12, 105–147 (2000)
10. Katagishi, H., Pearce, J.P.: KOPT: Distributed DCOP algorithm for arbitrary k-optima with monotonically increasing utility. In: Ninth DCR Workshop (2007)
11. Kiekintveld, C., Yin, Z., Kumar, A., Tambe, M.: Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In: AAMAS (2010)
12. Koenig, S., Simmons, R.: Unsupervised learning of probabilistic models for robot navigation. In: ICRA (1996)
13. Kwak, J., Yang, R., Yin, Z., Taylor, M.E., Tambe, M.: Teamwork and coordination under model uncertainty in DEC-POMDPs. In: AAAI Workshop on Interactive Decision Theory and Game Theory (2010)
14. Lesser, V., Ortiz, C., Tambe, M.: Distributed sensor nets: A multiagent perspective. Kluwer Academic Publishers (2003)
15. Levesque, H.J., Cohen, P.R., Nunes, J.H.T.: On acting together. In: AAAI (1990)
16. Maheswaran, R.T., Pearce, J.P., Tambe, M.: Distributed algorithms for DCOP: A graphical-game-based approach. In: PDCS (2004)
17. Maheswaran, R.T., Tambe, M., Bowring, E., Pearce, J.P., Varakantham, P.: Taking DCOP to the real world: efficient complete solutions for distributed multi-event scheduling. In: AAMAS (2004)
18. Mailler, R., Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation. In: AAMAS (2004)
19. Modi, P.J., Shen, W., Tambe, M., Yokoo, M.: ADOPT: Asynchronous distributed constraint optimization with quality guarantees. AIJ 161, 149–180 (2005)
20. Nair, R., Pynadath, D., Yokoo, M., Tambe, M., Marsella, S.: Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In: IJCAI (2003)
21. Nair, R., Varakantham, P., Tambe, M., Yokoo., M.: Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In: AAAI (2005)
22. Pearce, J., Tambe, M.: Quality guarantees on k-optimal solutions for distributed constraint optimization. In: IJCAI (2007)
23. Pearce, J.P., Tambe, M., Maheswaran, R.T.: Solving multiagent networks using distributed constraint optimization. AI Magazine 29(3) (2008)
24. Pierskalla, W., Voelker, J.: A survey of maintenance models: The control and surveillance of deteriorating systems. Naval Research Logistics Quarterly 23, 353–388 (1976)
25. Pynadath, D.V., Tambe, M.: The communicative multiagent team decision problem: Analyzing teamwork theories and models. JAIR (2002)
26. Roth, M., Simmons, R., Veloso, M.: Reasoning about joint beliefs for execution-time communication decisions. In: AAMAS (2005)
27. Schurr, N., Okamoto, S., Maheswaran, R.T., Scerri, P., Tambe, M.: Evolution of a teamwork model. In: Cognition and multi-agent interaction: From cognitive modeling to social simulation. pp. 307–327. Cambridge University Press (2005)
28. Tambe, M.: Towards flexible teamwork. JAIR, Volume 7, pages 83-124 (1997)
29. Taylor, M.E., Jain, M., Jin, Y., Yooko, M., , Tambe, M.: When should there be a me in team? distributed multi-agent optimization under uncertainty. In: AAMAS (2010)
30. Varakantham, P., Kwak, J., Taylor, M.E., Marecki, J., Scerri, P., Tambe, M.: Exploiting coordination locales in distributed POMDPs via social model shaping. In: ICAPS (2009)
31. Wu, F., Zilberstein, S., Chen, X.: Multi-agent online planning with communication. In: ICAPS (2009)
32. Xuan, P., Lesser, V.: Multi-agent policies: from centralized ones to decentralized ones. In: AAMAS (2002)