Exploiting Coordination Locales in Distributed POMDPs via Social Model Shaping

Pradeep Varakantham[*], **Jun-young Kwak, Matthew Taylor, Janusz Marecki**[†], **Paul Scerri**[‡], **Milind Tambe**
University of Southern California, Los Angeles, CA, 90089
[*]Singapore Management University, Singapore, 207855
[†]IBM T.J. Watson Research Center, Yorktown Heights, NY 10598
[‡]Carnegie Mellon University, Pittsburgh, PA, 15213
pradeepv@smu.edu.sg,{junyounk,taylorm,tambe}@usc.edu,marecki@us.ibm.com,scerri@cs.cmu.edu

Abstract

Distributed POMDPs provide an expressive framework for modeling multiagent collaboration problems, but NEXP-Complete complexity hinders their scalability and application in real-world domains. This paper introduces a subclass of distributed POMDPs, and TREMOR, an algorithm to solve such distributed POMDPs. The primary novelty of TREMOR is that agents plan individually with a single agent POMDP solver and use *social model shaping* to implicitly coordinate with other agents. Experiments demonstrate that TREMOR can provide solutions orders of magnitude faster than existing algorithms while achieving comparable, or even superior, solution quality.

Introduction

The excitement about Distributed Partially Observable Markov Decision Problems (DEC-POMDPs) is due to their ability to tackle real-world multi-agent collaborative planning problems under transition and observation uncertainty (Bernstein, Zilberstein, & Immerman 2000; Becker *et al.* 2004; Seuken & Zilberstein 2007; Marecki *et al.* 2008). Given the NEXP-Complete complexity of DEC-POMDPs (Bernstein, Zilberstein, & Immerman 2000), however, the emerging consensus is to pursue approximate solutions (Nair *et al.* 2003; Seuken & Zilberstein 2007) and sacrifice expressiveness by identifying useful subclasses of DEC-POMDPs (e.g., transition-independent DEC-MDPs (Becker *et al.* 2004) and event-driven DEC-MDPs (Becker, Zilberstein, & Lesser 2004; Marecki & Tambe 2007)). Such algorithms, through finding non-optimal joint policies or exploiting the structure of a subclass, are able to significantly reduce planning time.

In this continuing quest for efficiency, our research identifies a subclass of DEC-POMDPs that allows for significant speedups in computing joint policies. This paper provides two key contributions. The first is a new subclass: *Distributed POMDPs with Coordination Locales* (DPCL). DPCL is motivated by domains, including those found in DEC-POMDP literature, where multiple collaborative agents must perform multiple tasks. Agents are typically able to act independently, except in certain coordination locales. These coordination locales are identified as a set of states where agents may be required to coordinate — just in case their policies require them to interact — so as to avoid interfering or facilitating other agents' performance. For example, in disaster rescue, multiple robots may act to save multiple injured civilians. While mostly independent, the robots could end up colliding in building's narrow corridors if their policies simultaneously use the same corridor, and could clear up debris along the hallway to assist other robots. DPCL's expressiveness allows it to model domains not captured in previous work: it does not require transition independence (Becker *et al.* 2004), nor does it require that the agents' task allocation and coordination relationships be known in advance (Becker, Zilberstein, & Lesser 2004; Marecki & Tambe 2007), but does account for local observational uncertainty.

Our second contribution is a novel approach to exploit the decoupling present in DPCLs: *TREMOR* (Team's REshaping of MOdels for Rapid execution), an efficient algorithm for finding joint policies in DPCLs. TREMOR's primary novelty is that: (i) it plans for individual agents using single-agent POMDP solvers, effectively harnessing the most efficient POMDP solution approaches; (ii) it then manages inter-agent coordination via *social model shaping* — changing the transition functions and reward functions of the individual agents at *coordination locales*. TREMOR avoids the overhead of searching the entire space of POMDP joint policies, concentrating on interactions only in coordination locales. We show that even in the presence of significant agent interactions, TREMOR can run orders of magnitude faster than state-of-the-art algorithms such as MBDP (Seuken & Zilberstein 2007) and provide higher solution quality; however, its solution quality does suffer in domains with extreme coordination requirements.

Motivating Domains

This work is motivated by cooperative multiagent domains where agents are assigned to different tasks as part of a joint plan. In addition to task allocation, there are positive and negative interactions between agents when performing tasks (Scerri *et al.* 2005; Wurman, D'Andrea, & Mountz 2007). As tasks are initially unassigned, agent interactions are initially unknown, but are limited to set regions of the state space. Examples include disaster response – fire-engines and ambulances must be assigned to fight fires and save civilians (Nair & Tambe 2005), wilderness search and rescue (Cooper & Goodrich 2008), and space exploration.

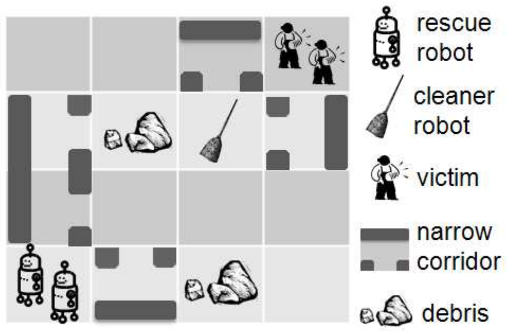

Figure 1: A 4×4 Rescue Domain where two rescue robots plan to reach two victims (1089 joint states).

As an example, consider a specific Rescue Domain. After a disaster, a group of heterogeneous robots may need to save civilians trapped in a building with debris impeding their progress. We use two types of robots, each of which has to deal with sensing and acting uncertainty. *Rescue* robots provide medical attention to victims. *Cleaner* robots remove potentially dangerous debris from building corridors and walkways. Saving victims provides a high reward, while cleaning up debris yields a lower reward.

We model this as a discrete grid (Figure 1), where grid squares may be "safe" or "unsafe." Each agent begins with a health value, which is reduced by 1 if it enters an unsafe square. An agent is disabled if its health falls to zero. Collisions may occur in narrow hallways if two robots try to pass through simultaneously, resulting in minor damage (cost) and causing one of the robots (chosen at random) to move back to its previous state. If a rescue robot attempts to traverse a "debris grid," it will get delayed by one time unit with high probability. A cleaner robot will instantly remove debris from the grid square it is in.[1]

Each agent has eight actions: move in the four cardinal directions and observe in each of the four cardinal directions. A movement action may succeed or fail, and observational uncertainty may lead to inaccurate information about movement success or safety of a location. Every action has a small cost and a rescue robot receives a high reward for being co-located with a victim, ending its involvement in the task. When modeling this domain as a DEC-POMDP, the goal of the planner is to obtain a reward-maximizing joint policy, where each policy assigns a rescue robot to a victim, and which debris (if any) each cleaner robot will clean.

The DPCL Model

In a DPCL, a team of N agents is required to perform a set of M tasks, one agent per task but potentially many tasks per agent, in the presence of transitional and observational uncertainty. Like a DEC-POMDP, DPCL is also a tuple $\langle S, A, P, R, \Omega, O, b\rangle$, where S, A, and Ω and the sets of joint states, actions and observations; $P : S \times A \times S \rightarrow [0, 1]$, $R : S \times A \times S \rightarrow \Re$, and $O : S \times A \times \Omega \rightarrow [0, 1]$ are the joint transition, reward, and observation functions respectively; and $b = [b(s)]_{s\in S}$ is a starting belief state such that $b(s) > 0, s \in S$ and $\sum_{s\in S} b(s) = 1$. However, DPCL is distinguished from DEC-POMDPs in that they assume $S := S_g \times S_1 \times \ldots \times S_N$ where S_n is a set of local states of agent n for $1 \leq n \leq N$ and $S_g = (E \times S_t)$ is a set of global states where $E = \{e_1, \ldots, e_H\}$ is the set of decision epochs and S_t is a set of task states s_t that keep track of the execution of tasks. Precisely, $s_t = (s_{t,m})_{1\leq m\leq M}$ where $s_{t,m} \in \{\textit{Done, NotDone}\}$ is the status of task m.[2]

Finding optimal joint policies to DEC-POMDPs is NEXP-Complete due to joint P, R and O functions. Since, DPCL is designed specifically for domains where the interactions among agents are limited, the transition, observation, and reward functions are defined for each agent separately. TREMOR thus avoids the computational complexity of solving general DEC-POMDPs. Let $\mathcal{P}_n : (S_g \times S_n) \times A_n \times (S_g \times S_n) \rightarrow [0, 1]$, $\mathcal{R}_n : (S_g \times S_n) \times A_n \times (S_g \times S_n) \rightarrow \Re$, and $\mathcal{O}_n : (S_g \times S_n) \times A_n \times \Omega_n \rightarrow [0, 1]$ denote agent local transition, reward and observation functions respectively. DPCL restricts a DEC-POMDP in that it assumes agent observations are fully independent, i.e., $O((s_g, s_1, \ldots, s_N), (a_1, \ldots a_N), (\omega_1, \ldots \omega_N)) = \prod_{1\leq n\leq N} \mathcal{O}_n((s_g, s_n), a_n, \omega_n)$, and that agent transitions and rewards are partially independent. Precisely, DPCL identifies situations where agent coordination is necessary, so that, with the exception of these situations, P and R naturally decompose into $\{\mathcal{P}_n\}_{1\leq n\leq N}$ and $\{\mathcal{R}_n\}_{1\leq n\leq N}$. These exceptional situations, referred to as *coordination locales* (CLs), are assumed in DPCL to be either Same-time (CL_s) or Future-time (CL_f) coordination locales.[3]

Same-Time Coordination Locales STCLs identify situations where state or reward resulting from the simultaneous execution of actions by a subset of agents cannot be described by the local transition and reward functions of these agents. Formally, a STCL for a group of agents $(n_k)_{k=1}^K$ is a tuple $cl_s = \langle(s_g, s_{n_1}, \ldots, s_{n_K}), (a_{n_1}, \ldots, a_{n_K})\rangle$ where s_g is the current global state and $(a_{n_k})_{k=1}^K$ are the actions that agents $(n_k)_{k=1}^K$ execute in their current local states $(s_{n_k})_{k=1}^K$. For cl_s to qualify as a STCL, there must exist joint states $s = (s_g, s_1, \ldots s_N), s' = (s'_g, s'_1, \ldots s'_N) \in S$ and a joint action $a = (a_n)_{n=1}^N \in A$ where $(s_{n_k})_{k=1}^K$ and $(a_{n_k})_{k=1}^K$ are specified in cl_s, such that the joint transition or reward function is non-decomposable, i.e., $P(s, a, s') \neq \prod_{1\leq n\leq N} \mathcal{P}_n((s_g, s_n), a_n, (s'_g, s'_n))$ or $R(s, a, s') \neq \sum_{1\leq n\leq N} \mathcal{R}_n((s_g, s_n), a_n, (s'_g, s'_n))$. The set of all STCLs is denoted as CL_s.

Example: Consider a robot in the Rescue Domain entering a narrow corridor. If another robot were to attempt to enter the same narrow corridor simultaneously, the robots would collide and one of them would be forced to transition back to its starting state.

Future-Time Coordination Locales FTCLs identify situations where actions of one agent impact actions of others in the future. Informally, because agents modify the global state $s_g = (e, s_t)$ as they execute their tasks, they can have a future impact on other agents' transitions and rewards since both $\mathcal{P}_n$ and $\mathcal{R}_n$ depend on s_g. Formally, a FTCL for a

[1]More details of the experimental domain and all DPCLs are shown in *http://teamcore.usc.edu/dpomdp/TREMOR/*.

[2]Task statuses need not be binary.

[3]If agent interactions are limited, $|CL_s| + |CL_f| \ll |dom(P)|$ and DPCLs are easier to specify than equivalent DEC-POMDPs.

group of agents $(n_k)_{k=1}^K$ is a tuple $\langle m, (s_{n_k})_{k=1}^K, (a_{n_k})_{k=1}^K \rangle$ where m is a task number and $(a_{n_k})_{k=1}^K$ are the actions that agents $(n_k)_{k=1}^K$ execute in their current local states $(s_{n_k})_{k=1}^K$. For cl_f to qualify as a FTCL, the actual rewards or transitions of agents $(n_k)_{k=1}^K$ caused by the simultaneous execution of actions $(a_{n_k})_{k=1}^K$ from states $(s_{n_k})_{k=1}^K$ must be different for $s_{t,m} =$ *Done* and *NotDone* for some global state $s_g = (e, s_t) \in S_g$.

Precisely, there must exist: (i) starting joint states $s = (s_g, s_1, \ldots s_N)$, $\overline{s} = (\overline{s}_g, s_1, \ldots s_N) \in S$ where $(s_{n_k})_{k=1}^K$ are specified in cl_f and $s_g = (e, s_t)$ differs from $\overline{s}_g = (e, \overline{s}_t)$ only on $s_{t,m} \neq \overline{s}_{t,m}$, (ii) a joint action $a = (a_n)_{n=1}^N \in A$ where $(a_{n_k})_{k=1}^K$ are specified in cl_f and (iii) ending joint states $s' = (s'_g, s'_1, \ldots s'_N), \overline{s}' = (\overline{s}'_g, s'_1, \ldots s'_N) \in S$ where $s'_g = (e', s'_t)$ differs from $\overline{s}'_g = (e', \overline{s}'_t)$ only on $s'_{t,m} \neq \overline{s}'_{t,m}$, such that either $P(s, a, s') \neq P(\overline{s}, a, \overline{s}')$ or $R(s, a, s') \neq R(\overline{s}, a, \overline{s}')$. The set of all FTCLs is denoted as CL_f.

Example: In our domain, a corridor with debris can be traversed quickly by a rescue robot only if a cleaner robot has first completed the task of removing these debris.

Solving DPCLs with TREMOR

We are interested in providing scalable solutions to problems represented using the DPCL model. To that end, we introduce TREMOR, an approximate algorithm that optimizes expected joint reward while exploiting limited coordination locales. TREMOR computes an assignment of tasks to agents and a policy to each agent (for completing the tasks assigned) using a two stage algorithm: (1) A branch and bound technique to efficiently search through the space of possible task assignments. (2) Evaluating task assignments (for step (1) above) in the presence of uncertainty (transitional and observational) and coordination locales by computing locally optimal joint policies.

Branch and Bound Search

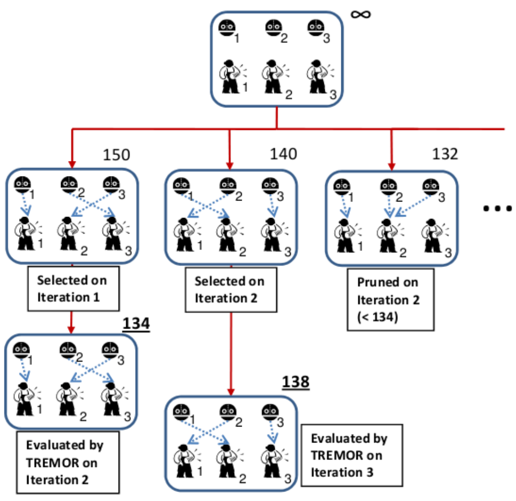
∞
150
140
132
Selected on Iteration 1
Selected on Iteration 2
Pruned on Iteration 2 (< 134)
134
138
Evaluated by TREMOR on Iteration 2
Evaluated by TREMOR on Iteration 3

Figure 2: Branch and Bound search in TREMOR

Multiagent planning problems often have a large number of possible task assignments precluding exhaustive evaluation. TREMOR incorporates a *Breadth-first Branch and Bound* search algorithm to exploit task decomposition among a team, significantly pruning the search space. To aid the search, we compute upper bounds on the expected value of joint policy using a heuristic that solves the decision problems of agents as MDPs (ignoring the observational uncertainty). Search begins with computation of upper-bounds for all task assignments and evaluation of the task assignment with highest upper-bound using TREMOR. Any assignment with an upper-bound that is lower than a complete evaluation calculated by TREMOR is pruned. Task assignments[4], with the highest heuristic evaluations are repeatedly evaluated until all remaining allocations are evaluated or pruned.

Algorithm 1 TREMOR-EvalTaskAssign(*Agents*, *Tasks*)

```
1:  for agent $n = 1, \ldots, N$ do
2:      $\mathcal{POMDP}_n \leftarrow$ CONSTRUCTPOMDP $(n, Tasks[n])$
3:      $\pi_n \leftarrow$ SOLVEPOMDP$(\mathcal{POMDP}_n)$
4:  repeat
5:      $ordering \leftarrow$ GETRANDOMORDER$(1, \ldots, N)$
6:      for $1 \leq k \leq N$ do
7:          $n \leftarrow ordering[k]$
8:          $I_n \leftarrow$ HANDLESTCLS$(n, CL_s)$
9:          $I_n \overset{+}{\leftarrow}$ HANDLEFTCLS$(n, CL_f)$
10:         for all $i \in I_n$ do
11:             $\pi_i \leftarrow$ SOLVEPOMDP$(\mathcal{POMDP}_i)$
12: until $\cup_{1 \leq n \leq N} I_n = \emptyset$ or maximum number of iterations has
    been reached
```

Task Assignment Evaluation

In this section, we introduce the algorithm that is used to evaluate task-assignments provided by the branch and bound step. The goal is to compute an optimal joint policy consistent with the task assignment. To avoid the significant computational complexity involved in searching through the entire joint space of policies, TREMOR's approach is to search for a locally optimal joint policy (see Algorithm 1). To that end, TREMOR performs two steps at every iteration: (a) Computing a joint policy assuming that agents are not interacting, i.e., by (near) optimally solving updated individual agent POMDPs (lines 1–3 and 10–11 of Algorithm 1) from step (b) below; (b) Identifying the interactions, both STCLs and FTCLs, caused due to the joint policy (computed in (a)) at coordination locales and shaping the models (transition and reward functions) of individual agents (lines 6–9) to account for the interactions. As there can be interactions due to both reward and transition function at any coordination locale, we provide detailed expressions to update both those functions for individual agents below. Steps (a) and (b) are performed until no agent policies can be changed or a maximum number of iterations is reached.

At each iteration of Algorithm 1, we re-compute policies π_i for all agents which are part of the sets I_n, where $1 \leq n \leq N$. This set (I_n) includes agents whose local transition, $\mathcal{P}_i$, and reward functions, $\mathcal{R}_i$, have been changed due to interactions with agent n. TREMOR considers interactions due to STCLs and FTCLs in Algorithm 2 and 3

[4]Note that TREMOR allows the number of agents and tasks to be different, and allow one agent to be assigned to multiple tasks.

Algorithm 2 HANDLESTCLS(n, CL_s)

```
1:  $I_n \leftarrow \phi$
2:  for $cl_s = \langle (s_g, (s_{n_k})_{k=1}^K), (a_{n_k})_{k=1}^K \rangle \in CL_s$ do
3:      $\hat{c} \leftarrow$ COMPCLPROB$(cl_s, \pi)$
4:      $\mathcal{P}'_n \leftarrow$ SHAPESTCLTRANSITIONS$(n, cl_s, \mathcal{P}_n, \hat{c}, P)$
5:      $\mathcal{R}'_n \leftarrow$ SHAPESTCLREWARDS$(n, cl_s, \mathcal{R}_n, \hat{c}, R)$
6:      $V \leftarrow EU(\pi_n \mid \mathcal{P}_n, \mathcal{R}_n)$
7:      $V' \leftarrow EU(\pi_n \mid \mathcal{P}'_n, \mathcal{R}'_n)$
8:      $V^\Delta \leftarrow (V' - V)$
9:      if $V^\Delta > 0$ then
10:         $I_n \leftarrow I_n \cup \{n_k\}_{1 \le k \le K}$
11:         for agent $i \in \{n_k\}_{1 \le k \le K}$ and $(s'_g, s'_i) \in S_g \times S_i$ do
12:             $\mathcal{R}_i((s_g, s_i), a_i, (s'_g, s'_i)) \overset{+}{\leftarrow} V^\Delta / K$
13:             $\mathcal{P}_i \leftarrow$ SHAPESTCLTRANSITIONS$(i, cl_s, \mathcal{P}_i, \hat{c}, P)$
14:     else if $V^\Delta < 0$ then
15:         $I_n \leftarrow I_n \cup (\{n_k\}_{1 \le k \le K} \setminus \{n\})$
16:         for agent $i \in \{n_k\}_{1 \le k \le K} \setminus \{n\}$ and $(s'_g, s'_i) \in S_g \times S_i$
            do
17:             $\mathcal{R}_i((s_g, s_i), a_i, (s'_g, s'_i)) \overset{+}{\leftarrow} V^\Delta / (K - 1)$
18: return $I_n$
```

respectively.

Upon verifying that a STCL cl_s: $\langle (s_g, s_{n_1}, \ldots, s_{n_K}), (a_{n_1}, \ldots, a_{n_K}) \rangle$, involves agent n (line 2), Algorithm 2 computes the probability, $\hat{c}$ that cl_s will occur given the current joint policy, π (computation of $\hat{c}$ is explained at the end of the section). Lines 4 and 5 of Algorithm 2 compute temporary transition and reward models for agent n based on $\hat{c}$. We then compute the difference $V^\Delta = V' - V$ in the expected utility, $EU(\pi_n)$ for agent n's policy, given that new models of transition, $\mathcal{P}'_n$ and reward functions, $\mathcal{R}'_n$ of agent n are employed for state action pairs in cl_s (lines 6–8). The algorithm behaves differently, depending on whether cl_s is beneficial to agent n.

If the shaping reward, V^Δ is positive (beneficial to agent n; lines 9–13), agents are encouraged to follow policies that induce cl_s. Each agent involved in the coordination locale cl_s is influenced by adding a fraction V^Δ/K of the shaping reward to local reward function $\mathcal{R}_i$ of each agent involved in the coordination locale. To ensure coherent dynamics models for the agents after interaction, local transition models of agents are redefined (line 13) by using the global transition function P (for local state-action pairs resulting in cl_s) in the SHAPESTCLTRANSITIONS() function; such a redefinition takes into account the probability of coordination locales. Let $e, e' \in E$ be the starting and ending decision epochs for the transition of interest, $a_i \in A_i$ be the agent's action, $s_i, s'_i \in S_i$ be the starting and ending local agent states, $s_t, s'_t \in S_t$ be the starting and ending task states. The local transition probability of agent i assuming a STCL cl_s does not occur, $\mathcal{P}_{i,\neg cl_s}(((e, s_t), s_i), a_i, ((e', s'_t), s'_i))$, is given as a domain input. We derive the local transition probability of agent i assuming cl_s occurs, $P_{i,cl_s}(((e, s_t), s_i), a_i, ((e', s'_t), s'_i))$, from the joint transition function, P. This is accomplished by summing the probabilities of transitions where the initial joint state and joint action are determined by cl_s and the destination state belongs to all the joint states s' where agent i is in state s'_i, as shown in Equation 1. Then, in Equation 2, we compute the new transition probability $\mathcal{P}'_i$ by taking a weighted average of $\mathcal{P}_{i,cl_s}$ and $\mathcal{P}_{i,\neg cl_s}$.

$$\mathcal{P}_{i,cl_s}(((e, s_t), s_i), a_i, ((e', s'_t), s'_i)) \leftarrow \sum_{s' \in S : s'_{n_i} = s'_i} P(((e, s_t), (s_{n_k})_{k=1}^K), (a_{n_k})_{k=1}^K, ((e', s'_t), (s'_{n_1}, \ldots, s'_i, \ldots))) \tag{1}$$

$$\mathcal{P}'_i \leftarrow \hat{c} \cdot \mathcal{P}_{i,cl_s} + (1 - \hat{c}) \cdot \mathcal{P}_{i,\neg cl_s} \tag{2}$$

To account for the impact of cl_s on the local reward function $\mathcal{R}_i$ of agent i, we compute new reward models of individual agents on line 5 of Algorithm 2. The local reward for agent i assuming STCL occurs, $\mathcal{R}_{i,cl_s}$ is computed in Equation 3. Since, the state transitions of agent i corresponding to the occurrence of cl_s are already considered in the computation of $\mathcal{R}_{i,cl_s}$, we include a normalization factor β_i to avoid multiplying the reward twice. The local reward for agent i assuming STCL, cl_s, does not occur, i.e., $\mathcal{R}_{i,\neg cl_s}$ is the original reward function for agent i, $\mathcal{R}_i$. Given these, the new reward function, $\mathcal{R}'_i$ is the weighted average of rewards when cl_s occurs and when it does not.

$$\begin{aligned}\mathcal{R}_{i,cl_s}(((e, s_t), s_i), a_i, ((e', s'_t), s'_i)) \leftarrow \\ \sum_{s' \in S : s'_{n_i} = s'_i} P(((e, s_t), (s_{n_k})_{k=1}^K), (a_{n_k})_{k=1}^K, \\ ((e', s'_t), (s'_{n_1}, \ldots, s'_i, \ldots, s'_{n_K}))) \cdot \\ R(((e, s_t), (s_{n_k})_{k=1}^K), (a_{n_k})_{k=1}^K, \\ ((e', s'_t), (s'_{n_1}, \ldots, s'_i, \ldots, s'_{n_K})))\end{aligned} \tag{3}$$

$$\mathcal{R}'_i \leftarrow \hat{c} \cdot \frac{\mathcal{R}_{i,cl_s}}{\beta_i} + (1 - \hat{c}) \cdot \mathcal{R}_{i,\neg cl_s}, \tag{4}$$

$$\beta_i \leftarrow \mathcal{P}_{i,cl_s}(((e, s_t), s_i), a_i, ((e', s'_t), s'_i))$$

In contrast, if the shaping reward is negative (not beneficial to agent n; lines 14–17) agents in coordination locale cl_s are discouraged from executing policies that induce cl_s, except for agent n which is given no incentive to modify its behavior. As cl_s will not occur in this interaction, there is no need to redefine the agent local transition functions in terms of the joint transition function P. To illustrate the algorithm execution, we provide an example from our motivating domain. We are initially given the transition and reward functions for individual agents traveling through the building. When two robots' policies lead to a determination that an STCL (agents i and j bump into each other in a narrow corridor) has $\hat{c} > 0$, we first check if the STCL is beneficial or not. If it is not beneficial, we provide a negative reward to one of the robots (robot j) to encourage it to avoid the narrow corridor; the robots' transition functions are not modified since this STCL will not occur. On the other hand, for a beneficial STCL (i.e., agents gain by bumping into each other) we provide a positive shaping reward, so that agents can bump into each other with higher probability. To have a coherent model of the dynamics after bumping of agents, we update the transition functions of both the robots using the above formulae.

TREMOR then considers all FTCLs: $cl_f \in CL_f$, $\langle m, (s_{n_k})_{k=1}^K, (a_{n_k})_{k=1}^K \rangle$, involving agent n in Algorithm 3.

Algorithm 3 HandleFTCLs (n, CL_f)

```
1:  $I_n \leftarrow \phi$
2:  for $cl_f = \langle m, (s_{n_k})_{k=1}^K, (a_{n_k})_{k=1}^K \rangle \in CL_f$: $m \in Tasks[n]$ do
3:      for all $e \in E$ do
4:          $P_\pi^{e,s_{t,m}^+} \leftarrow \sum_{s \in S: s_g=(e,s_{t,m}^+); a \in A}$ COMPCLPROB$(\langle s, a \rangle, \pi)$
5:      for all $k \leq K$ do
6:          $\mathcal{P}'_{n_k} \leftarrow$ SHAPEFTCLTRANSITIONS$(\mathcal{P}_{n_k}, \{P_\pi^{e,s_{t,m}^+}\}_{e \in E})$
7:          $V' \xleftarrow{+} EU(\pi_{n_k} \mid \mathcal{P}'_{n_k})$
8:          $V \xleftarrow{+} EU(\pi_{n_k} \mid \mathcal{P}_{n_k})$
9:      $V^\Delta \leftarrow (V' - V)$
10:     if $V^\Delta > 0$ then
11:         $I_n \leftarrow I_n \cup \{n_k\}_{1 \leq k \leq K}$
12:         $\mathcal{P}_{n_k} \leftarrow \mathcal{P}'_{n_k}, 1 \leq k \leq K, n_k \neq n$
13:     else if $V^\Delta < 0$ then
14:         /*Do Nothing*/
15:     for all $((s_g, s_n), a_n, (s'_g, s'_n)) \in (S_g \times S_k) \times A_n \times (S_g \times S_k)$ : $s_t$ differs from $s'_t$ on $s_{t,m} \neq s'_{t,m}$ do
16:         $\mathcal{R}_n((s_t, s_n), a_n, (s'_t, s'_n)) \xleftarrow{+} R^\Delta$
17: $I_n \leftarrow I_n \cup n$
18: return $I_n$
```

To that end, it computes probabilities, $\{P_\pi^{e,s_{t,m}^+}\}_{e \in E}$, that task m is completed at decision epoch e by agent n, when the joint policy π is executed (line 4 of Algorithm 3). These probabilities are used to change the transition functions of all agents n_k at each decision epoch $e \in E$ (line 6). We then compute the difference in expected utility, V^Δ (= $V' - V$), for the current policies of the agents on the updated (V') and original (V) POMDP models. This difference is also the *shaping reward*. Similar to STCLs, depending on the sign of the shaping reward, the behavior of the algorithm differs.

When the shaping reward is positive (coordination locale cl_f is beneficial, lines 10-12), agents participating in cl_f will have their local transition functions $\mathcal{P}_i$ modified using heuristics so that the execution status of task m can change from $NotDone$ to $Done$, with probability $P_\pi^{e,s_{t,m}^+}$. The function SHAPEFTCLTRANSITIONS() (line 6) is used for changing transition functions in FTCLs. Let $s_t \in S_t$ be the starting task state where task m (that some other agent j executes) is not yet completed, i.e., $s_{t,m} = NotDone$ and $s_t'^+, s_t'^- \in S_t$ be two possible ending task states that differ on the status of execution of task m, i.e., $s_{t,m}'^+ = Done$ and $s_{t,m}'^- = NotDone$. $\mathcal{P}_i(((e, s_t), s_i), a_i, ((e', s'_t), s'_i)))$ is required to be updated based on the probability that task m will now be completed in decision epoch e with probability $P_\pi^{e,s_{t,m}^+}$. The new transition probability is computed for all agents i, where $m \notin tasks[i]$ as follows:

$$\mathcal{P}'_i(((e, s_t), s_i), a_i, ((e', s_t'^+), s'_i)) \leftarrow P_\pi^{e,s_{t,m}^+} \cdot \mathcal{P}_i(((e, s_t), s_i), a_i, ((e', s_t'^-), s'_i))$$

$$\mathcal{P}'_i(((e, s_t), s_i), a_i, ((e', s_t'^-), s'_i)) \leftarrow (1 - P_\pi^{e,s_{t,m}^+}) \cdot \mathcal{P}_i(((e, s_t), s_i), a_i, ((e', s_t'^-), s'_i))$$

In contrast, if the shaping reward is not beneficial (lines 13–14), agents will not have to do anything because task m will not move from $NotDone$ to $Done$ at any decision epoch. The current shaping reward V^Δ is added to n's local reward function $\mathcal{R}_n$, to either encourage (if $V^\Delta > 0$) or discourage (if $V^\Delta < 0$) agent n from executing task m. Finally, n is added to I_n to indicate that n's model has changed and needs to be solved again.

Computation of $\hat{c}$ We now describe the COMPCLPROB() function, which is used to compute $\hat{c}$ (line 3 of Algorithm 2) for an STCL, $cl_s = \langle((e, s_t), s_{n_1}, \ldots, s_{n_K}), (a_{n_1}, \ldots, a_{n_K})\rangle$. The same function is also used in computation of $P_\pi^{e,s_{t,m}^+}$ for an FTCL (line 4 of Algorithm 3). We first calculate the probability $\hat{c}_i$ that agent i will engage in the coordination locale cl_s at time T. Let $\pi_i : B_i \times A_i \rightarrow \{0, 1\}$ be a deterministic policy of agent i and $b_i^{(0)}$ be its starting belief state. Let $\hat{c}_i^{(t)}(b)$ be the probability that agent i will execute at time T action a_{n_i} from state s_{n_i}, if it starts the execution at time $0 \leq t \leq T$ in a belief state b, and G_i be the belief state transition operator for agent i. We calculate $\hat{c}_i^{(t)}(b)$ using the following recursive scheme: $\hat{c}_i^{(t)}(b) = \begin{cases} b(s_{n_i})\pi_i(b, a_{n_i}) & t = T \\ \sum_{a \in A_i} \pi_i(b, a) \sum_{\omega \in \Omega_i} P_i^a(\omega|b) \cdot \hat{c}_i^{(t+1)}(G_i^{a,\omega}(b)) & t \leq T \end{cases}$,

$$\begin{aligned} G^{a,\omega}(b)(s') &= \frac{1}{P^a(\omega|b)} O^a(\omega|s') \sum_{s \in S} P^a(s'|s) b(s), \\ P^a(\omega|b) &= \sum_{s' \in S} [O^a(\omega|s') \sum_{s \in S} P^a(s'|s) b(s)] \end{aligned}$$

represent the equations for computing belief state transitions (Littman, Cassandra, & Kaelbling 1995). The probability $\hat{c}_i$ is then $\hat{c}_i^{(0)}(b_i^{(0)})$. Finally, since agents coordinate only at coordination locales, cl_s, we have $\hat{c} = \prod_i \hat{c}_i$.

Empirical Results

This section demonstrates that TREMOR can successfully solve DPCL problems in time orders of magnitude faster than existing locally optimal algorithms, while discovering policies of higher or comparable value. To that end, we evaluate TREMOR's performance on a set of disaster rescue tasks (described in the Motivating Domain section) with three existing planning approaches.

Experimental Setup TREMOR employs EVA (Varakantham *et al.* 2007) as the single agent POMDP solver. We compare against JESP (Joint Equilibrium-based Search for Policies) (Nair *et al.* 2003) and MBDP (Memory-Bounded Dynamic Programming for DEC-POMDPs) (Seuken & Zilberstein 2007), two of the leading algorithms for approximately solving DEC-POMDPs. Lastly, we consider a planner that ignores interactions between agents (constructed using TREMOR's algorithms without any of the model-shaping), henceforth referred to as Independent POMDPs.

The maximum number of iterations in TREMOR is set to 50 and ϵ for EVA is 2.0. MBDP experiments used the parameters suggested by the authors: type of algorithm = *approximate*, max trees = 3, max observations = 2, depth of recursion = 2, and backup type = *Improved Memory-Bounded Dynamic Programming*. Experiments were run on quad-core Intel 3.2GHz processors with 8GB of RAM. All techniques

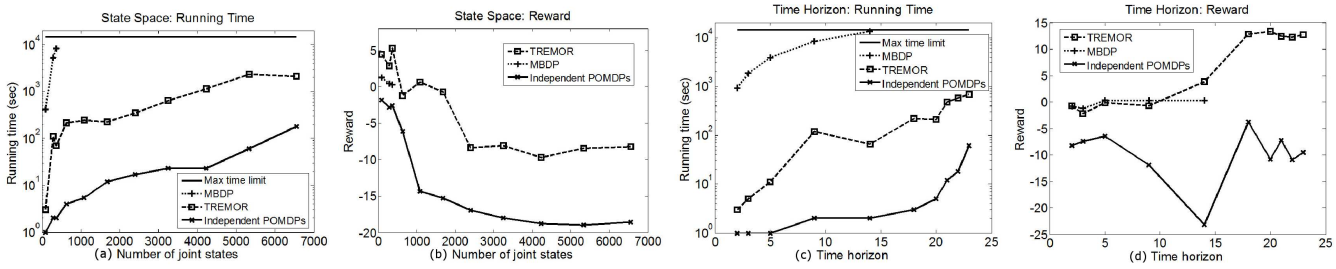

Figure 3: Comparison with MBDP and Independent POMDPs: State Space and Time Horizon Scale-Up.

were run 20 times on each DPCL and we report the average wall-clock time. All planners were given a maximum wall-clock time of 4 hours. Each joint policy's expected value is measured as the average value of 500 executions.

State Space This set of experiments show that TREMOR scales to problems with large state spaces. Every problem has one cleaner robot, two rescue robots, and a time horizon of 10. The state space changes from 81 to 6561 joint states (2×2 to 4×10 grids). Figure 3a shows TREMOR's runtime with respect to the size of state space. The x-axis shows the number of joint states and the y-axis shows the runtime in seconds on a log scale. MBDP is only able to solve tasks of up to 361 joint states within the time limit and requires 1.6–2.1 orders of magnitude more time than TREMOR. Independent POMDPs plan faster than TREMOR as they disregard all inter-agent interactions.

TREMOR's runtime does not increase monotonically with the size of the state or horizon, as shown in Figure 3a. It depends on (i) the time taken to resolve interactions at each iteration (lines 5–11 in Algorithm 1), (ii) the maximum number of such iterations, both of which depend on the specifics of the problem. JESP was unable to solve any task within the time limit and thus is not shown. For illustrative purposes, we ran JESP on a 81 joint state problem with T=2 (reduced from T=10). It finished executing in 228 seconds, yielding a reward of 12.47, while TREMOR required only 1 second and received a reward of 11.13.

Figure 3b displays the average reward accrued by polices on the y-axis over the same set of tasks as in 3a. TREMOR outperforms MBDP, even though MBDP plans with joint models. In addition, TREMOR also achieved the statistically significant result (via t-tests) of outperforming Independent POMDPs with respect to average reward.

Time Horizon In the second set of experiments, we varied the time horizon from 2–23 as shown in Figures 3c and 3d. These results show that increased time horizon leads to higher runtimes and that TREMOR can generate deep joint-policy trees. We considered problems with two rescue robots, one cleaning robot and 361 joint states. MBDP is able to solve tasks up through T=14, but takes 1.8–2.6 orders of magnitude more time than TREMOR, while its final policies' rewards are dominated in most of the cases. TREMOR requires 0.3–1.7 orders of magnitude more time than Independent POMDPs, but produces policies that accrue significantly more reward.

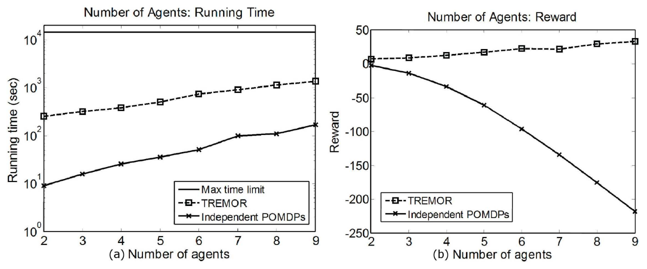

Figure 4: Agents and Tasks Scale-Up

Number of Agents and Tasks In the third set of experiments, we keep the state space and time horizon constant (1089 joint states and T=10) and show that TREMOR scales well with the number of agents, relative to Independent POMDPs. Figures 4a and 4b show the runtime and reward accrued on tasks with one cleaning robot and 2–9 rescue robots (the number of victims and rescue robots are equal).

As shown in Figure 4b, TREMOR and the Independent POMDPs' rewards diverge as the number of agents (and tasks) is increased due to increasing numbers of possible collisions. Increased numbers of interactions leads to a higher runtime for TREMOR, but also higher rewards. In contrast, the runtime of Independent POMDPs do not increase as dramatically, but rewards suffer as they are increasingly penalized for their lack of coordination. MBDP fails to solve any case here within the time limit.

Number of CLs The last set of experimental results show how TREMOR performs when the number of CLs changes: more CLs imply more inter-agent interactions, increasing TREMOR's overhead and reducing its benefit relative to MBDP. All experiments have 361 joint states, T=10, two rescue robots, and one cleaning robot; these settings were chosen explicitly so that MBDP could complete the task within the cutoff time. Figures 5a and 5b show the runtime and reward for different number of CLs. The performance of TREMOR depends on the number of CLs and the maximum number of model refinement iterations. As we discussed in the previous section, TREMOR is well-suited for domains which require limited coordination. These results demonstrate that the runtime increases and reward decreases when more coordination is required. It should be noted that TREMOR can trade off time and quality by tuning the maximum number of model refinement iterations.

The runtime of MBDP and Independent POMDPs do not change with the number of CLs, as they are not explic-

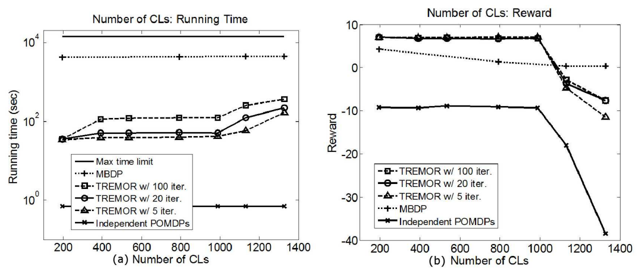

Figure 5: Scale-Up with number of CLs

itly considered. MBDP computes a joint policy superior to TREMOR in problems with very large numbers of CLs ($>$ 1100), although it requires more time to compute the joint policy. This is an expected result, because in domains with large number of CLs, the agents are tightly coupled and hence an approach like TREMOR, which relies on decoupling agent interactions using shaping heuristics, will not be as effective. To further analyze this result, we consider two instances of a specific problem with a time horizon of 5. We consider the problem with two agents, 361 joint states, and the number of CLs at 544 and 1328.

Specifically, we compare three approaches: MBDP, TREMOR, and TREMOR+. TREMOR+ is TREMOR with a problem-specific hand crafted heuristic for selecting the order of agents (on line 5 of Algorithm 1). As shown in Figure 6a, when we have very large number of CLs (1328), MBDP outperforms TREMOR in terms of expected reward (y-axis). However, when we use a pre-specified order of agents (TREMOR+) instead of the random order used in TREMOR, it outperforms MBDP in terms of solution quality. This result indicates that TREMOR's current heuristic to select agents for model shaping is not ideal for all situations and is an issue for future work. On the other hand, in domains with a relatively small number of CLs, TREMOR beats MBDP, and the reward difference between TREMOR and TREMOR+ is much smaller. The key conclusion from this experiment is that in tightly coordinated domains, MBDP provides better performance than TREMOR. However, with better agent ordering heuristics, TREMOR can provide considerable improvement in performance. Figure 6b shows the runtime comparison among MBDP, TREMOR, and TREMOR+ on the 1328 CL problem. The y-axis shows runtime on log scale.

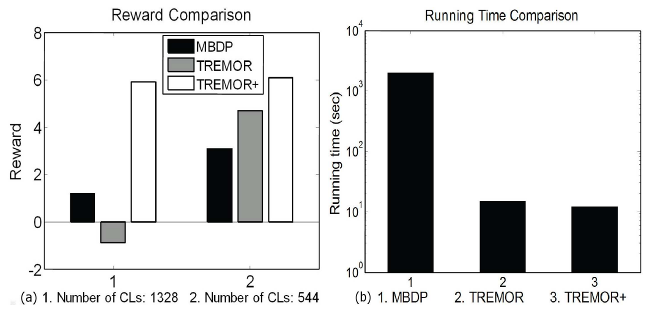

Figure 6: Comparison in tightly coordinated domains

We have shown TREMOR's superior scalability with respect to state space, time horizon, agents, tasks, and coordination locales. Furthermore, TREMOR provided solutions of comparable, or even superior, quality to those found by existing DEC-POMDP solvers.

Related Work

There are three categories of research that are related to DPCL and TREMOR:

DEC-MDPs: These problems rely on individual full observability of local state and also collective full observability of global state (Spaan & Melo 2008). Due to this assumption, approaches employed for solving DEC-MDPs are not applicable to solving DPCL problems, where there is individual partial observability of local state and collective partial observability of global state. However, we include them for completeness, and to show how ideas from DEC-MDP research may influence DEC-POMDP work.

Roth et al. (2007) examine factored DEC-MDPs and focus on exploiting limited agent interactions to speed up centralized policy generation. While their method is able to identify where agents can ignore other agents, they focus on DEC-MDPs which have context-specific independence and require communication between agents. Interaction-Driven Markov Games (Spaan & Melo 2008) focus on DEC-MDPs where agents interact in only a subset of states. When not in these states the agents are independent, but when in one of the interaction states the agents must communicate (to coordinate about the interaction state). In addition to the differences mentioned between DPCL and DEC-MDP models, TREMOR employs a fundamentally different mechanism of shaping transition and reward functions to solve DPCL problems, which does not rely on communication.

A sub-category of the research in DEC-MDPs focuses explicitly on evaluating **task allocation**. ED-DEC-MDPs (Becker, Zilberstein, & Lesser 2004; Marecki & Tambe 2007) and OC-DEC-MDPs (Beynier & Mouaddib 2006) leverage pre-specified task allocation and dependencies to reduce the search space, while assuming the presence of only task performing actions. Approaches used to solve these models have been shown to scale up to domains with hundreds of tasks. When DPCL is compared against ED-DEC-MDPs and OC-DEC-MDPs, there are two key differentiating factors: (a) While task allocations and dependencies are assumed to be part of the input in ED-DEC-MDP and OC-DEC-MDP, they are computed as part of the output solution to DPCL; and (b) Whereas ED-DEC-MDP and OC-DEC-MDP provide solutions assuming the presence of only task performing actions, DPCL models domains where there are other actions as well. For instance, in the Rescue Domain, there are path planning actions in addition to the actions associated with rescuing civilians (performing tasks). Due to the presence of general actions, the complexity of DPCL problems cannot be measured solely by the number of tasks.

DEC-POMDPs: Becker et al (2004), defined transition independence, initiating research into restricted DEC-POMDP classes for efficient solutions. ND-POMDPs (Marecki *et al.* 2008) build on transition-independence and add network structure interactions. Though DPCL assumes observation independence, it differs due to transition dependence (captured using coordination locales), thus focusing on a broad new class of multiagent applications. Oliehoek et al. (2008) provide efficient algorithms for factored DEC-POMDPs by considering agents that interact with only few other agents, while also assuming that interactions are statically inferred from the domain. While interactions in DPCL are confined to CLs, these interactions influence agents' policies only if the policies are observed to interact. Also, in our experiments, all agents can interact with all other agents.

JESP (Nair *et al.* 2003) has a similar approach to TREMOR in that it solves single agent POMDPs and updates the individual models to manage inter-agent interactions. However, the model updating in JESP leads to an exponential increase in state space and hence cannot scale to larger problems (as shown in the experimental results). Finally, Nair and Tambe (2005) exploit problem structure for efficient solutions to DEC-POMDPs that require role allocation; however, they crucially require a human-specified abstract policy, which TREMOR does not require.

Model Shaping: TREMOR is similar to Guestrin and Gordon (2002), where subsystems can plan separately and then iteratively re-plan if the subsystems interact unfavorably. However, the use of POMDPs and social model shaping sets our work apart. Other recent work has also considered using reward shaping in DEC-POMDPs (Williamson, Gerding, & Jennings 2009). In this work, the authors use reward shaping, in conjunction with an estimate of team cohesion, to help reduce the amount of communication necessary when reasoning in a fully distributed fashion.

Conclusion

This paper has introduced TREMOR, a fundamentally different approach to solve distributed POMDPs. TREMOR is an approximate algorithm and it does not apply to general DEC-POMDPs. However, it is extremely efficient for solving DPCLs, an important subclass of distributed POMDPs. This subclass includes a range of real-world domains where positive or negative agent interactions occur in a relatively small part of the overall state space. By iteratively discovering interactions and using shaping of models to influence efficient individual POMDPs, TREMOR enables a team of agents to act effectively and cohesively in environments with action and observation uncertainty. The main insight behind TREMOR is using social reward and transition shaping allows a DEC-POMDP to be approximated by a set of single-agent POMDPs. TREMOR can thus exploit advances in single-agent POMDP solvers. Extensive experimental results show that TREMOR provides dramatic speedups over previous distributed POMDP approaches; TREMOR is mostly able to deliver these speedups without degrading solution quality. However, TREMOR's solution quality suffers in tightly coupled domains characterized by a very large number of coordination locales.

Acknowledgements

We thank the anonymous reviewers for their comments and suggestions, and Alan Carlin for providing us with the source code for MBDP. This work was supported in part by US. Army SBIR contract number W15P7T-09-C-S601, DARPA SBIR contract number W31P4Q-06-0286, and Perceptronics Solutions, Inc.

References

Becker, R.; Zilberstein, S.; Lesser, V.; and Goldman, C. V. 2004. Solving Transition Independent Decentralized Markov Decision Processes. *JAIR* 22.

Becker, R.; Zilberstein, S.; and Lesser, V. 2004. Decentralized Markov Decision Processes with Event-Driven Interactions. In *AAMAS*.

Bernstein, D. S.; Zilberstein, S.; and Immerman, N. 2000. The complexity of decentralized control of markov decision processes. In *UAI*.

Beynier, A., and Mouaddib, A.-I. 2006. An iterative algorithm for solving constrained decentralized markov decision processes. In *AAAI*.

Cooper, J., and Goodrich, M. 2008. Towards combining UAV and sensor operator roles in UAV-enabled visual search. In *HRI*.

Guestrin, C., and Gordon, G. 2002. Distributed planning in hierarchical factored MDPs. In *UAI*.

Littman, M. L.; Cassandra, A. R.; and Kaelbling, L. P. 1995. Learning policies for partially observable environments: Scaling up. In *ICML*.

Marecki, J., and Tambe, M. 2007. On opportunistic techniques for solving decentralized MDPs with temporal constraints. In *AAMAS*.

Marecki, J.; Gupta, T.; Varakantham, P.; Tambe, M.; and Yokoo, M. 2008. Not all agents are equal: Scaling up distributed pomdps for agent networks. In *AAMAS*.

Nair, R., and Tambe, M. 2005. Hybrid BDI-POMDP framework for multiagent teaming. *JAIR* 23.

Nair, R.; Pynadath, D.; Yokoo, M.; Tambe, M.; and Marsella, S. 2003. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI*.

Roth, M.; Simmons, R.; and Veloso, M. 2007. Exploiting factored representations for decentralized execution in multiagent teams. In *AAMAS*.

Scerri, P.; Farinelli, A.; Okamoto, S.; and Tambe, M. 2005. Allocating tasks in extreme teams. In *AAMAS*.

Seuken, S., and Zilberstein, S. 2007. Improved memory-bounded dynamic programming for decentralized POMDPs. In *UAI*.

Spaan, M. T. J., and Melo, F. S. 2008. Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *AAMAS*.

Varakantham, P.; Maheswaran, R. T.; Gupta, T.; and Tambe, M. 2007. Towards efficient computation of error bounded solutions in POMDPs: Expected value approximation and dynamic disjunctive beliefs. In *IJCAI*.

Williamson, S. A.; Gerding, E. H.; and Jennings, N. R. 2009. Reward shaping for valuing communications during multi-agent coordination. In *AAMAS*.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2007. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *AAAI*.