In *The AAMAS workshop on Multi-agent Sequential Decision-Making in Uncertain Domains* (MSDM-09), Budapest, Hungary, May 2009.

Exploiting Coordination Locales in Distributed POMDPs via Social Model Shaping

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

ABSTRACT

While distributed POMDPs provide an expressive framework for modeling multiagent collaboration problems, NEXP-Complete complexity hinders their scalability and application in real-world domains. This paper introduces a subclass of distributed POMDPs, and TREMOR, a novel algorithm to solve such distributed POMDPs. Two major novelties in TREMOR are (i) use of social model shaping to coordinate agents, (ii) harnessing efficient single agent-POMDP solvers. Experimental results demonstrate that TREMOR may provide solutions orders of magnitude faster than existing algorithms while achieving comparable, or even superior, solution quality.

1. INTRODUCTION

The excitement of Distributed Partially Observable Markov Decision Problems (DEC-POMDPs) flows from their ability to tackle real-world multi-agent collaborative planning, under transition and observation uncertainty [2, 3, 9, 17]. Given the NEXP-Complete complexity of DEC-POMDPs [3], however, the emerging consensus is to pursue approximate solutions [12, 17] and sacrifice expressivity by identifying useful subclasses of DEC-POMDPs (e.g., transition-independent DEC-MDPs [2, 15], and event-driven DEC-MDPs [1, 4, 10]). 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 distributed POMDPs that allows for significant speedups in computing joint policies. We thus provide two key contributions. The first is a new subclass: *Distributed POMDPs with Coordination Locales* (DPCL). DPCL is motivated by the many domains, including those found in distributed POMDP literature, where multiple collaborative agents must perform multiple tasks. The agents can usually act independently, but they interact in certain *coordination locales*, identified as a set of states and times where agents could potentially need to coordinate, such as to avoid interfering with each other's task performance or to facilitate other agents' task performance. For example, in disaster rescue [8], multiple robots may act to save multiple injured civilians. While often acting independently, the robots should avoid colliding with other robots in a building's narrow corridors, and could clear up debris along the hallway to assist other robots. DPCL's expressivity allows it to model domains not captured in previous work: it does not require transition independence [2], nor does it require that agents' task allocation and coordination relationships be known in advance [1, 10], but does account for local observational uncertainty.

Our second contribution is a novel approach to solving 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, thus 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 coordinating agents. While TREMOR is an approximate approach and it will not apply to the most general DEC-POMDPs, it does open a new line of attack on a large subclass of problems. 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 [17] and provides higher solution quality.

2. MOTIVATING DOMAINS

Our work is motivated by cooperative multiagent domains where agents must be assigned to different tasks. There are positive and negative interactions when agents perform these tasks [16, 20]. Since tasks are initially unassigned, agent interactions are initially unknown, but are limited to certain regions of the state space. Examples include disaster response [13] where fire-engines must be assigned to fight fires and ambulances to save civilians, wilderness search and rescue [5], and space exploration [6].

This paper focuses on urban disaster response where multiple robots must save civilians trapped in a building following a disaster. We use two types of robots, each of which must deal with sensing and action uncertainty. *Rescue* robots provide medical attention to victims. *Cleaner* robots remove potentially dangerous debris from building corridors, lobbies, and walkways. Saving victims provides a high reward, where the amount of reward depends on the victim's health status; cleaning up debris yields a lower reward (see Figure 1).

We model this as a discrete grid, where grid squares may be "safe" or "unsafe." Each agent begins with a health value of 3, 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 a grid it is in and receive a small positive reward. [1]

AAMAS 2009 Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains, May 11, 2009, Budapest, Hungary.

[1]More details of the experimental domain and all DPCLs are shown in Appendix A & B.

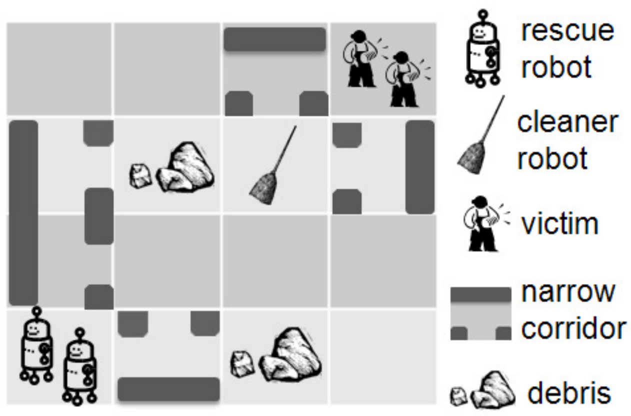

Figure 1: This figure shows a 4×4 domain (with 1089 joint states). Two rescue robots plan to reach two victims. The rescue robots may collide in narrow corridors; a cleaner robot can remove debris to assist the rescue robots. Safeness of a grid cell (not shown in figure) is only known with a certain degree of certainty.

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.

3. 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 DEC-POMDPs, DPCL too is 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 $b = \Delta S$ is a starting belief region. However, DPCL specializes from DEC-POMDPs in that it assumes $S := S_g \times S_1 \times \ldots \times S_N$ 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 execution of task m.

Finding optimal joint policies to DEC-POMDPs is NEXP-Complete because the functions P, R and O are defined jointly, even if agent interactions are limited — DPCL is designed specifically to overcome this limitation. 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 DEC-POMDPs in that it assumes that 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 situations, referred to as *coordination locales* (CLs), are assumed in DPCL to be either same- or future-time. [2]

3.1 Same-time coordination locales (STCLs)

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.

3.2 Future-time coordination locales (FTCLs)

FTCLs identify situations an action impacts actions in the future. Informally, because agents modify the current global state $s_g = (e, s_t)$ as they execute their tasks, they can have a future impact on agents' transitions and rewards since both $\mathcal{P}_n$ and $\mathcal{R}_n$ depend on s_g. Formally, a FTCL for a 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 states $(s_{n_k})_{k=1}^K$ must be different for $s_{t,m} = \textit{Done}$ and *NotDone* for some global state $s_g = (e, s_t) \in S_g$. Precisely, there must exist: (i) starting joint states $s = (s_g, s_1, \ldots s_N), \bar{s} = (\bar{s}_g, s_1, \ldots s_N) \in S$ where $(s_{n_k})_{k=1}^K$ are specified in cl_f and $s_g = (e, s_t)$ differs from $\bar{s}_g = (e, \bar{s}_t)$ only on $s_{t,m} \neq \bar{s}_{t,m}$; (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 $s' = (s'_g, s'_1, \ldots s'_N), \bar{s}' = (\bar{s}'_g, s'_1, \ldots s'_N) \in S$ where $s'_g = (e', s'_t)$ differs from $\bar{s}'_g = (e', \bar{s}'_t)$ only on $s'_{t,m} \neq \bar{s}'_{t,m}$ such that either $P(s, a, s') \neq P(\bar{s}, a, \bar{s}')$ or $R(s, a, s') \neq R(\bar{s}, a, \bar{s}')$. The set of all FTCLs is denoted as CL_f.

Example: Consider a rescue robot from the domain in Section 2, entering a narrow corridor. If another robot were to attempt to enter the same narrow corridor simultaneously, one of them would transition back to starting state and the robots would damage each other (STCL). If the narrow corridor had debris and a cleaner robot completed the task of removing this debris, the rescue robot would traverse the corridor faster (FTCL).

4. SOLVING DPCLS WITH TREMOR

We are interested in providing scalable solutions to problems represented using the DPCL model. To this end, we provide TREMOR, an approximate algorithm that optimizes expected joint reward while exploiting coordination regions between agents. TREMOR accounts for the coordination locales, 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.

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

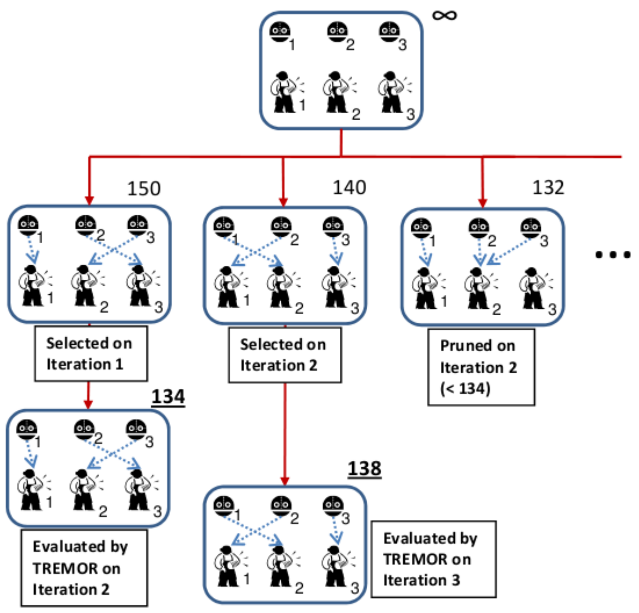
∞
150
140
132
Selected on Iteration 1
Selected on Iteration 2
Pruned on Iteration 2 (< 134)
134
138
Evaluated by TREMOR on Iteration 2
Evaluated by TREMOR on Iteration 3

Figure 2: This diagram depicts the branch and bound search.

4.1 Branch and Bound Search

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. In order 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 lower than a complete evaluation calculated by TREMOR is pruned. Task assignments [3] with the highest heuristic evaluations are repeatedly evaluated until all remaining allocations are evaluated or pruned (see Figure 2).

4.2 Task Assignment Evaluation

At this point of algorithmic execution, agents have been assigned their tasks and an optimal joint policy consistent with this assignment has to be found. Since the problem is still NEXP-Complete, TREMOR's approach in evaluating the current task assignment is to search for a locally optimal joint policy (see Algorithm 1). To that end, TREMOR initially finds the optimal joint policy assuming that agents are not interacting, i.e., by solving individual agent POMDPs (lines 1–3). Note that we can employ state of the art POMDP solvers to solve a POMDP in SOLVEPOMDP() (line 3). We then try to improve the joint policy (lines 5–41) until no agent policies can be changed.

At each iteration, 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 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 (lines 6–21) and FTCLs (lines 22–38) separately.

[3]Note that TREMOR allows the number of agents and tasks to be unequal, as well as allowing an agent to be assigned to multiple tasks.

Upon verifying that a STCL c, $\langle(s_t, s_{n_1}, \ldots, s_{n_K}), (a_{n_1}, \ldots, a_{n_K})\rangle$, involves agent n (line 8), the algorithm computes the difference $R^+ - R^-$ in the expected utility, $EU(\pi_n)$ for agent n's policy, given that the transition, $\mathcal{P}_n$ and reward functions, $\mathcal{R}_n$ of agent n are updated for state action pairs in c. TREMOR then computes the probability, $\hat{c}$, that c will occur given the current joint policy, π, and uses $\hat{c}$ to determine the *shaping reward* R^{Δ}. Depending on whether c is beneficial to agent n or not, the algorithm behaves differently.

If the shaping reward is positive (beneficial to agent n; lines 15–17), agents are encouraged to follow policies that induce c. The agent is influenced by adding a fraction R^{Δ}/K of the shaping reward to local reward function $\mathcal{R}_i$ of each agent. To ensure a coherent dynamic model for the agents after interaction, local transition models of agents are then redefined by using the global transition function P (for local state-action pairs resulting in c); such redefinition could potentially take into account the probability of coordination locales (although not used in our implementation). To calculate the old transition functions of each agent, P_i, we first "extract" the old probability of transitioning from one agent state to another given its action and a status of task, s_t. Let $e, e' \in E$ be the starting and ending decision epochs for that transition, $a_i \in A_i$ 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 c does not occur, $P_{i,\neg c}(((e, s_t), s_i), a_i, ((e', s_t'), s_i'))$, is given as a domain input. We derive the local transition probability of agent i assuming c occurs, $P_{i,c}(((e, s_t), s_i), a_i, ((e', s_t'), s_i'))$, from a given joint transition function, and update P_i using $\hat{c}$ and derived probabilities. Formally:

$$P_{i,c}(((e, s_t), s_i), a_i, ((e', s_t'), s_i')) \leftarrow$$

$$\sum_{\{\forall s'_{n_k} \in S, k \neq i\}} P((s_{n_1}, \ldots, s_{n_K}), (a_{n_1}, \ldots, a_{n_K}), (s'_{n_1}, \ldots, s'_{n_i}, \ldots, s'_{n_K})),$$

$$P_i \leftarrow \hat{c} \times P_{i,c}(((e, s_t), s_i), a_i, ((e', s_t'), s_i'))$$

$$+(1 - \hat{c}) \times P_{i,\neg c}(((e, s_t), s_i), a_i, ((e', s_t'), s_i'))$$

In contrast, if the shaping reward is negative (not beneficial to agent n; lines 18–21) agents in coordination locale c are discouraged from policies that induce c, except for agent n which is given no incentive to modify its behavior. As c 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 update the reward and transition functions in an STCL, consider the following example from our domain. A STCL occurs when two robots, i and j, bump into each other in a narrow corridor. We are initially given the transition probability of an individual robot i's traveling through the corridor alone (as input). When two robots' policies create an STCL (agents i and j bump into each other in the narrow corridor), we first check if the STCL is beneficial or not. If it is non-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. Although this would not happen in our example domain, a beneficial STCL would need a positive shaping reward. We then update the transition function P_i of robot i, using the transition probabilities P_i when bumping occurs and when it does not occur, using the updating formula above.

TREMOR then considers all FTCLs:
$c \in CL_f$, $\langle m, (s_{n_k})_{k=1}^K, (a_{n_k})_{k=1}^K\rangle$, involving agent n (lines 22–38). To that end, it computes probabilities, $P_{\pi}^{e,s_{t,m}^+}$, that a task

is completed by a decision epoch, e, when the joint policy π is executed. These probabilities are used to determine the sum of expected utilities for the current policies of the agents, R^+ and R^-, when agent n completes task m and when task m was never completed respectively. As in STCLs, TREMOR computes the *shaping reward* $R^\Delta = (R^+ - R^-) \cdot P_\pi^{e,s_{t,m}^+}$. When the shaping reward is positive (coordination locale c is beneficial, lines 29–32), agents participating in coordination locale c will have their transition functions, P_i, modified using heuristics to reflect that in each decision epoch $e \in E$, task m can be moved to $Done$ state from $NotDone$, with probability $P_\pi^{e,s_{t,m}^+}$.

For FTCLs, a heuristic similar to that used for STCLs is applied, updating the local transition functions of each agent. First we "extract" the old probability of transitioning given its action and a status of specific task m that will be done by another agent j from P_i. 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 $s_{t,m}'^- = NotDone$. According to the old function P_i, agent i "believed" with a certain probability that task m will be completed by agent j in decision epoch e. Initially, $P_i(((e, s_{t,m}), s_i), a_i, ((e', s_{t,m}'^+), s_i')))$ and $P_i(((e, s_{t,m}), s_i), a_i, ((e', s_{t,m}'^-), s_i')))$ are given as a domain input, and used for updating P_i. Now, agent j is shaping the transition function of agent i and it makes agent i "believe" that task m will be completed decision in epoch e with a different probability. Agent j's commitment to the completion of its task m has changed, i.e., task m will now be completed in decision epoch e with probability $P_\pi^{e,s_{t,m}^+}$ – agent i's new transition function P_i should then be updated with this new information. For a given task m, we typically have the explicit task status pair, $(s_{t,m}, s_{t,m}')$. We calculate $P_\pi^{e,s_{t,m}^+}$ for each $(s_{t,m}, s_{t,m}')$ separately and keep updating P_i iteratively for all tasks. Formally:

$$P_i \leftarrow P_\pi^{e,s_{t,m}^+} \times P_i(((e, s_{t,m}), s_i), a_i, ((e', s_{t,m}'^+), s_i')))$$

$$+(1 - P_\pi^{e,s_{t,m}^+}) \times P_i(((e, s_{t,m}), s_i), a_i, ((e', s_{t,m}'^-), s_i')))$$

We could further generalize this updating step by summing over all current and future task states, however, that would increase the complexity of transition function shaping.

In contrast, if the shaping reward is not beneficial (lines 33–36) agents will have their transition functions $\mathcal{P}_i$ modified to reflect that $s_{t,m}$ cannot change from $NotDone$ to $Done$ in any decision epoch. At last, whenever agent n can execute task m, the current shaping reward R^Δ is added to its local reward function $\mathcal{R}_n$, to either encourage (if $R^\Delta > 0$) or discourage (if $R^\Delta < 0$) agent n from executing task m. The algorithm terminates the task assignment evaluation if the current joint policy cannot be improved, i.e., all the sets I_n for $1 \leq n \leq N$ are empty or the number of model refinements is greater than maximum number of iterations.

5. EMPIRICAL RESULTS

This section demonstrates that TREMOR can successfully solve DPCL problems orders of magnitude faster than required by existing locally optimal algorithms, while still discovering policies of comparable value. To that end, we evaluate TREMOR's performance on a set of disaster rescue tasks (described in Section 2) by comparing its planning time and solution value with three existing planning approaches.

Algorithm 1 TREMOR-EvalTaskAssignment(*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:
 5: repeat
 6:    for agent $n = 1, \ldots, N$ do
 7:       $I_n \leftarrow \emptyset$
 8:       for $c = \langle(s_g, s_{n_1}, ..., s_{n_K})(a_{n_1}, ..., a_{n_K})\rangle \in CL_s$ such that
          $n \in \{n_k\}_{1\leq k\leq K}$ do
 9:          $R^- \leftarrow EU(\pi_n)$
10:          $R^+ \leftarrow EU(\pi_n$ when $\mathcal{P}_n$ and $\mathcal{R}_n$ are redefined
             in terms of joint functions $P$ and $R$ for arguments
             $((s_g, s_n), a_n, (s_g', s_n'))$ for all $(s_g', s_n') \in S_g \times S_n)$
11:          $\hat{c} \leftarrow P_\pi((s_g, s_{n_1}, ..., s_{n_K})(a_{n_1}, ..., a_{n_K}))$
12:          $R^\Delta \leftarrow (R^+ - R^-) \cdot \hat{c}$
13:          if $R^\Delta > 0$ then
14:             $I_n \leftarrow I_n \cup \{n_k\}_{1\leq k\leq K}$
15:             for agent $i \in \{n_k\}_{1\leq k\leq K}$ and $(s_g', s_i') \in S_g \times S_i$ do
16:                $\mathcal{R}_i((s_g, s_i), a_i, (s_g', s_i')) \xleftarrow{+} R^\Delta/K$
17:                $\mathcal{P}_i \leftarrow \mathcal{P}_i$ redefined in terms of $P$ for arguments
                   $((s_g, s_i), a_n, (s_g', s_i'))$ for all $(s_g', s_i') \in S_g \times S_i$
18:          else if $R^\Delta < 0$ then
19:             $I_n \leftarrow I_n \cup (\{n_k\}_{1\leq k\leq K} \setminus \{n\})$
20:             for agent $i \in \{n_k\}_{1\leq k\leq K} \setminus \{n\}$ and $(s_g', s_i') \in S_g \times S_i$
                do
21:                $\mathcal{R}_i((s_g, s_i), a_i, (s_g', s_i')) \xleftarrow{+} R^\Delta/(K-1)$
22:       for $c = \langle(m, (s_{n_1}, ..., s_{n_K})(a_{n_1}, ..., a_{n_K}))\rangle \in CL_f$ such
          that $m \in Tasks[n]$ do
23:          for all $e \in E$ do
24:             $P_\pi^{e,s_{t,m}^+} \leftarrow \sum_{((e,s_t),s_1,\ldots s_N)\in S : s_{t,m}=Done; a\in A} P_\pi(s, a)$
25:          $R^+ \leftarrow \sum_{k=1}^K EU(\pi_{n_k}$ given that task $m$ will be completed
             in epoch $e \in E$ with probability $P_\pi^{e,s_{t,m}^+})$
26:          $R^- \leftarrow \sum_{k=1}^K EU(\pi_{n_k}$ if task $m$ not completed)
27:          $\hat{c} \leftarrow \sum_{s_g\in S_g} P_\pi((s_g, s_{n_1}, ..., s_{n_K})(a_{n_1}, ..., a_{n_K}))$
28:          $R^\Delta \leftarrow (R^+ - R^-) \cdot \hat{c}$
29:          if $R^\Delta > 0$ then
30:             $I_n \leftarrow I_n \cup \{n_k\}_{1\leq k\leq K} \cup \{n\}$
31:             for agent $i \in \{n_k\}_{1\leq k\leq K}$ and $e \in E$ do
32:                Modify $\mathcal{P}_i$ knowing that in each epoch $e \in E$, $s_{t,m}$
                   can change from $NotDone$ to $Done$ with probability
                   $P_\pi^{e,s_{t,m}^+})$
33:          else if $R^\Delta < 0$ then
34:             $I_n \leftarrow I_n \cup \{n_k\}_{1\leq k\leq K} \cup \{n\}$
35:             for agent $i \in \{n_k\}_{1\leq k\leq K}$ do
36:                Modify $\mathcal{P}_i$ knowing that in each epoch $e \in E$, $s_{t,m}$
                   cannot change from $NotDone$ to $Done$
37:          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
38:             $\mathcal{R}_n((s_t, s_n), a_n, (s_t', s_n') \xleftarrow{+} R^\Delta$
39:    for all $i \in I_n$ do
40:       $\pi_i \leftarrow$ SolvePomdp($\mathcal{POMDP}_i$)
41: until $\cup_{1\leq n\leq N} I_n = \emptyset$ or maximum iterations
```

5.1 Experimental Setup

TREMOR employs EVA [18, 19] as the single agent POMDP solver. We compare against JESP (Joint Equilibrium-based Search for Policies) [12] and MBDP (Memory-Bounded Dynamic Programming for DEC-POMDPs) [17], two of the leading approximate algorithms for solving DEC-POMDPs. Lastly, we consider a planner that ignores interactions between agents, i.e. TREMOR without any coordination locales (call independent POMDPs). All planners are given a maximum wall-clock time of 4 hours.

TREMOR and EVA's parameters were set as follows: maximum iterations of TREMOR= 50, and ϵ = 5.0. MBDP experiments used the parameters suggested by the authors: type of algorithm = *approximate*, max. number of trees = 3, max. number of observations for the improved MBDP algorithm = 2, depth of recursion = 2, and backup type = *Improved Memory-Bounded Dynamic Programming*. JESP has no tunable parameters.

Experiments were run on quad-core Intel 3.2GHz processors with 8GB of RAM. Each approach was run 20 times on each DPCL and we report the average wall-clock time. For computing expected value of a joint policy, we averaged over 500 runs.

5.2 State Space

This set of experiments show that TREMOR can handle large state spaces, unlike existing algorithms. Every experiment has a time horizon of 10, one cleaner robot, and two rescue robots. The state space changes from 81 to 6561 joint states (2×2 to 4×10 grids). Figure 3a shows scaling of TREMOR's runtime with respect to the size of state space. The x-axis shows the number of joint states in the problem and the y-axis shows $\log$ (plan time in sec). MBDP is only able to solve tasks of up to 361 joint states within the time limit and requires 1.5–2.9 orders of magnitude more time than TREMOR. Independent POMDPs plan faster than TREMOR as they disregard all inter-agent interactions.

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 is an algorithm that plans on the joint models and we expected it to account for interactions better. In addition, TREMOR also achieved the statistically significant result of outperforming independent POMDPs with respect to average reward, although using up to 1.6 orders of magnitude more time ($p < 1.5 \times 10^{-9}$).

TREMOR's runtime does not increase monotonically with the size of the state or horizon as shown in Figure 3. It depends on (i) the time it takes to resolve interactions for each resolution iteration (lines 6–40 in Algorithm 1), (ii) the maximum number of such iterations, both of which change depending on the details of each DPCL.

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.

5.3 Time Horizon

The second set of experiments consider an increasing time horizon from T=2–23, shown in Figures 3c and 3d. These experiments show increased episode lengths lead to higher planning times, but 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 at least 2.6 orders of magnitude more time than TREMOR, while its final policies' rewards are dominated by TREMOR's policies. TREMOR requires at most 1.1 orders of magnitude more time than independent POMDPs, but produces policies that accrue significantly more reward.

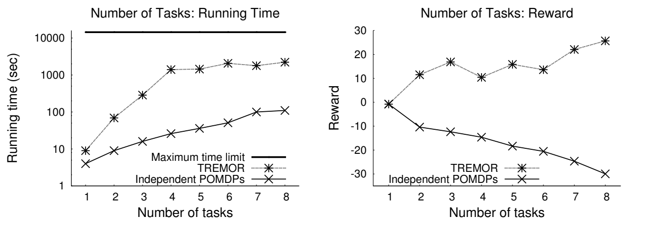

Figure 4: Agents and Tasks Scale-Up

5.4 Number of Agents and Tasks

The third set of experiments 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. In fact, TREMOR's improvement over independent POMDPs increases with the number of agents. Figure 4a and 4b show the running time and reward accrued on tasks with one cleaning robot and 1–8 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) are increased due to increasing numbers of CLs. Increased number 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 with two or more tasks within the time limit. [4] TREMOR requires between 0.35 and 1.73 orders of magnitude more time than independent POMDPs, but produces policies that accrue significantly more reward.

5.5 Number of CLs

The last set of experiments 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. Figure 5a and 5b show the running time and reward with various number of CLs. The performance of TREMOR depends on the number of CLs and maximum number of resolution interactions. As we discussed in the previous section, TREMOR is well-suited for domains which require limited coordination. These results demonstrate that the running time 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.

MBDP is able to discover a joint policy superior to TREMOR for very large numbers of CLs. For the problem with the number of CLs = 1368, MBDP received a higher reward than TREMOR and independent POMDPs, although it continues to require more time.

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

[4]We did not try MBDP with one task because there are no interesting same-time coordination locales.

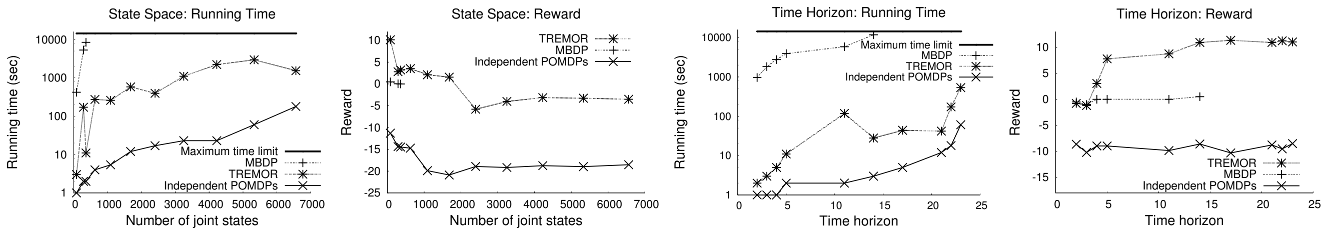

Figure 3: Comparison with MBDP and Independent POMDPs: State Space and Time Horizon Scale-Up.

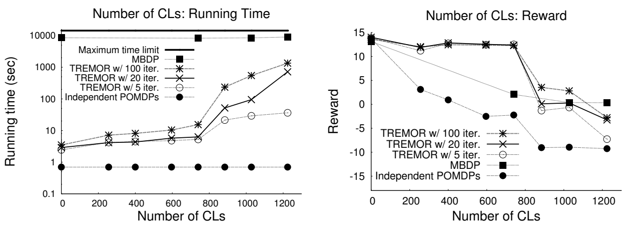

Figure 5: CLs Scale-Up

6. RELATED WORK AND CONCLUSIONS

As mentioned earlier, others have done significant work to identify classes of DEC-POMDPs that may be solved efficiently. For example, Becker *et al.* [2] assume an individually observable domain where agents are transition independent. ND-POMDPs build on transition-independence and add network structure interactions [9]. Though DPCL assumes individual observability, it differs due to transition dependence (captured using coordination locales), thus focusing on a broad new class of multiagent applications. Task-based ED-DEC-MDPs [1, 4, 10] leverage pre-specified task allocation and dependencies to reduce the search space. This is another key differentiating factor in DPCL, where task allocations and dependencies are not part of the model.

Others have also examined how to combine role allocation with distributed POMDP solvers [13], exploiting problem structure to speed up policy search. Oliehoek *et al.* [14] also exploit problem structure — factored DEC-POMDPs — but assume observation-dependence. TREMOR differs from these and other DEC-POMDP algorithms in its fundamental approach by employing single-agent POMDPs and exploiting social model shaping to manage inter-agent interactions. In this sense, TREMOR shares some similarity with other MDP-related work [7] where subsystems can plan separately, but can iteratively re-plan if the subsystems interact unfavorably. However, the use of POMDPs and social model shaping sets our work apart. Lastly, shaping rewards have been previously used in multi-agent contexts (c.f., Matarić [11]), but are typically present to assist agents via human-specified rewards. In TREMOR, shaping rewards are used to allow coordination between agents without explicit multi-agent planning and are determined autonomously.

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 also exploit advances in single-agent POMDP solvers. Extensive experimental results show how TREMOR provides dramatic speedups over previous distributed POMDP approaches without sacrificing expected reward.

7. ACKNOWLEDGMENTS

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.

8. REFERENCES

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

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

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

[4] A. Beynier and A. Mouaddib. A polynomial algorithm for Decentralized Markov Decision Processes with temporal constraints. In *AAMAS*, 2005.

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

[6] D. Goldberg, V. Cicirello, and M. B. Dias. A distributed layerd architecture for mobile robot coordination: Application to space exploration. In *NASA Workshop on Planning and Scheduling for Space*, 2002.

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

[8] H. Kitano and S. Tadokoro. RoboCup Rescue: A Grand Challenge for Multiagent and Intelligent Systems. *AI Magazine*, March 2001.

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

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

[11] M. J. Matarić. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4, 1997.

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

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

[14] F. A. Oliehoek, M. T. J. Spaan, S. Whiteson, and N. Vlassis. Exploiting locality of interaction in factored DEC-POMDPs. In *AAMAS*, 2008.

[15] M. Petrik and S. Zilberstein. Anytime coordination using separable bilinear programs. In *AAAI*, 2007.

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

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

[18] P. Varakantham, R. Maheswaran, and M. Tambe. Exploiting belief bounds: Practical POMDPs for personal assistant agents. In *AAMAS*, 2005.

[19] P. Varakantham, R. T. Maheswaran, T. Gupta, and M. Tambe. Towards efficient computation of error bounded solutions in POMDPs: Expected Value Approximation and Dynamic Disjunctive Beliefs. In *IJCAI*, 2007.

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

Table 1: State transition function

State	State1	Action	Probability
Safe	Safe	Success	$P_{safety} \times (1 - P_{actionFailure})$
		Failure	$P_{safety} \times P_{actionFailure}$
	Unsafe	Success	$(1 - P_{safety}) \times (1 - P_{actionFailure})$
		Failure	$(1 - P_{safety}) \times P_{actionFailure}$
Unsafe	Safe	Success	$P_{safety} \times (1 - P_{actionFailure})$
		Failure	$P_{safety} \times P_{actionFailure}$
	Unsafe	Success	$(1 - P_{safety}) \times (1 - P_{actionFailure})$
		Failure	$(1 - P_{safety}) \times P_{actionFailure}$

Table 2: Reward function

Action	Reward
Saving the victim *(only rescue robots)*	+8.0
Cleaning debris *(only cleaning robots)*	+1.0
Moving and observing	-0.2
Collisions	-4.0
Dead	-10.0

APPENDIX

A. DPCL FOR TREMOR

$\langle S, A, P, R, \Omega, O, b \rangle$ with STCLs, FTCLs

(1) S: set of world states (row, column, health): {(0, 0, 0), (0, 1, 0), (0, 2, 0), ..., }

Row and Column: $0 - n$, Health value for each robot: $0 - m$.

(2) A: actions: A = {*move north, move east, move south, move west, observe north, observe east, observe south, observe west*}

(3) P: state transition function: Transition to a state based on how the health of the robot will be affected due to safety of destination cell (P_{safety}) and probability of action failure ($P_{actionFailure}$).

P_{safety}: assigned randomly, $P_{actionFailure}$: 0.2 (See Table 1).

In case of collisions between savers *(same time coordination locale, STCL)*, the transition probabilities of states are dependent on actions of other agents. For instance in a collision between two agents in a narrow corridor (x, y), an agent gets to that cell and the other agent goes back to the originating cell. If agents are starting from different cells and colliding in (x, y), this happens with 0.5 probability for each agent.

In case of coordination due to cleaning of debris *(future time coordination locale, FTCL)*, the debris is cleared by the cleaner robot and cleaning action is guaranteed to succeed all the time.

(4) R: reward function (See Table 2).

(5) O: observations. O = {*Success/Failure* for moving action, *Safe/Unsafe* for observing action} (See Table 3).

(6) STCLs (same-time coordination locales): situations where state or reward resulting from the simultaneous execution of ac-

Table 3: Observations

Action	State	Observation	Probability
Moving	Success	Success	0.8
		Failure	0.2
	Failure	Success	0.6
		Failure	0.4
Observing	Safe	Safe	0.8
		Unsafe	0.2
	Unsafe	Safe	0.6
		Unsafe	0.4

tions.

$cl_s = \langle (s_g, s_{n_1}, \ldots, s_{n_K}), (a_{n_1}, \ldots, a_{n_K}) \rangle$ 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 $(s_{n_k})_{k=1}^{K}$.

(7) FTCLs (future-time coordination locales): situations an action impacts actions in the future.

$cl_f = \langle m, (s_{n_k})_{k=1}^{K}, (a_{n_k})_{k=1}^{K} \rangle$ 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 $(s_{n_k})_{k=1}^{K}$.

B. EXPERIMENTAL DOMAIN

(1) State Space Scale-Up: 2×2 (# of joint states: 81) – 4×10 (# of joint states: 6561) (See Figure 6).

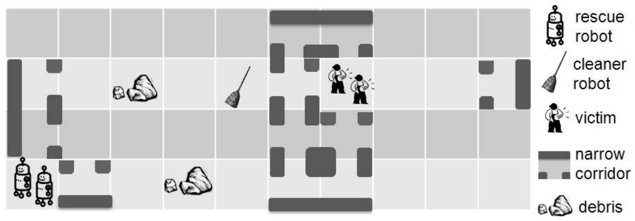

Figure 6: 4×10 (# of joint states: 6561), T=10: 2 rescue robots, 2 victims, 1 cleaning robot, 2 debris, & 11 narrow corridors.

One example case both CLs can happen:

$Saver_0$: (3, 0) → (2, 0) → (1, 0) → (1, 1) → (1, 2) → (1, 3) → (1, 4) → (1, 5) → (1, 6)

$Saver_1$: (3, 0) → (3, 1) → (3, 2) → (2, 2) → (2, 3) → (2, 4) → (2, 5) → (1, 5) → (1, 6)

$Cleaner_0$: (1, 4) → (1, 3) → (1, 2)

STCL happens between savers on (1, 5) at T=7:

$cl_{s7} : \langle (s_{g7}, s_{14}, s_{25}), (a_1, a_0) \rangle, where$

$s_{g7} : (NotDone_0, NotDone_1, 7)$

$s_{14} : (1, 4, 1)$

$s_{25} : (2, 5, 1)$

a_1 : *move east*

a_0 : *move north*

FTCL happens between $Saver_0$ and $Cleaner_0$ at $debris_0$'s location (1, 2):

$cl_{f1} : \langle m2, (s_{11}, s_{13}), (a_1, a_3) \rangle, where$

$m_2 : cleaningdebris_0 at(1, 2)$

$s_{11} : (1, 1, 1)$

$s_{13} : (1, 3, 1)$

a_1 : *move east*

a_3 : *move west*

(2) Time Horizon Scale-Up: T=2 – 23 (See Figure 7).

(3) Number of Agents and Tasks Scale-Up: 1–8 rescue robots, 1–8 victims (See Figure 8).

(4) Number of Coordination Locales Scale-Up: 0 narrow corridor (# of CLs: 0) – 7 narrow corridors (# of CLs: 1368) (See Figure 9).

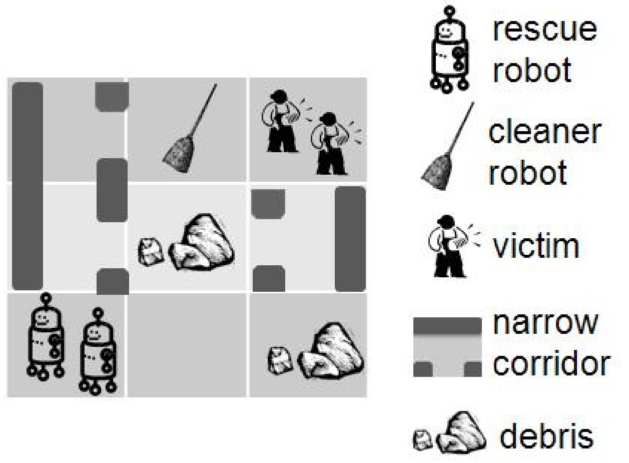

Figure 7: 3×3 (# of joint states: 361), T=2–23: 2 rescue robots, 2 victims, 1 cleaning robot, 2 debris, & 3 narrow corridors.

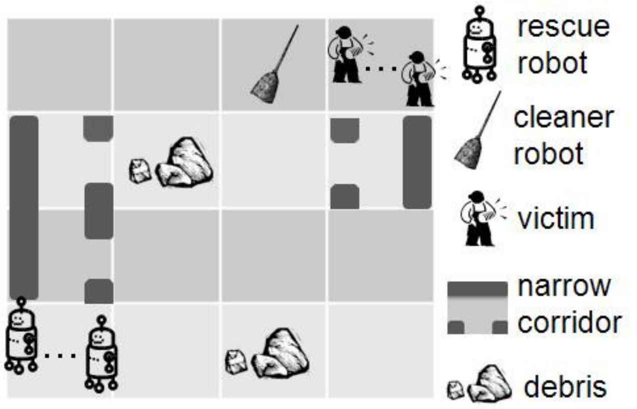

Figure 8: 4×4 (# of joint states: 1089), T=10: 1–8 rescue robots, 1–8 victims, 1 cleaning robot, 2 debris, & 3 narrow corridors.

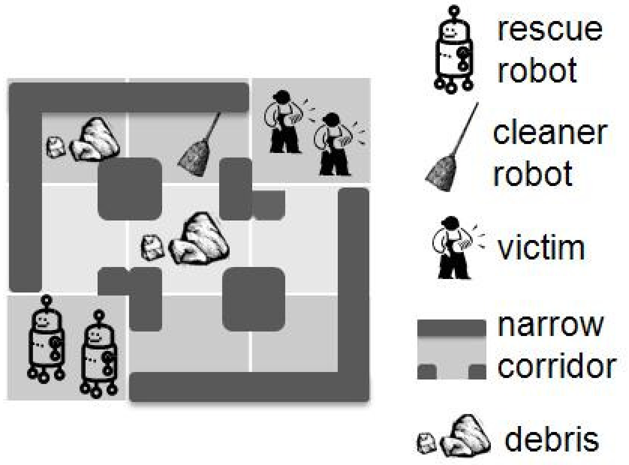

Figure 9: 3×3 (# of joint states: 361), T=10: 2 rescue robots, 2 victims, 1 cleaning robot, 2 debris, & 0–7 narrow corridors.