5-2011

Distributed Model Shaping for Scaling to Decentralized POMDPs with hundreds of agents

Prasanna VELAGAPUDI
Carnegie Mellon University

Pradeep Reddy VARAKANTHAM
Singapore Management University, pradeepv@smu.edu.sg

Katia Sycara
Carnegie Mellon University

Paul Scerri
Carnegie Mellon University

Distributed Model Shaping for Scaling to Decentralized POMDPs with Hundreds of Agents

Prasanna Velagapudi
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15217, USA
pkv@cs.cmu.edu

Pradeep Varakantham
Singapore Management Univ.
80 Stamford Road
Singapore 178902
pradeepv@smu.edu.sg

Katia Sycara
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15217, USA
katia@cs.cmu.edu

Paul Scerri
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15217, USA
pscerri@cs.cmu.edu

ABSTRACT

The use of distributed POMDPs for cooperative teams has been severely limited by the incredibly large joint policy-space that results from combining the policy-spaces of the individual agents. However, much of the computational cost of exploring the entire joint policy space can be avoided by observing that in many domains important interactions between agents occur in a relatively small set of scenarios, previously defined as *coordination locales* (CLs) [11]. Moreover, even when numerous interactions *might* occur, given a set of individual policies there are relatively few *actual* interactions. Exploiting this observation and building on an existing model shaping algorithm, this paper presents D-TREMOR, an algorithm in which cooperative agents iteratively generate individual policies, identify and communicate possible interactions between their policies, shape their models based on this information and generate new policies. D-TREMOR has three properties that jointly distinguish it from previous DEC-POMDP work: (1) it is completely distributed; (2) it is scalable (allowing 100 agents to compute a "good" joint policy in under 6 hours) and (3) it has low communication overhead. D-TREMOR complements these traits with the following key contributions, which ensure improved scalability and solution quality: (a) techniques to ensure convergence; (b) faster approaches to detect and evaluate CLs; (c) heuristics to capture dependencies between CLs; and (d) novel shaping heuristics to aggregate effects of CLs. While the resulting policies are not globally optimal, empirical results show that agents have policies that effectively manage uncertainty and the joint policy is better than policies generated by independent solvers.

Categories and Subject Descriptors

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

Cite as: Distributed Model Shaping for Scaling to Decentralized POMDPs with hundreds of agents, Prasanna Velagapudi, Pradeep Varakantham, Katia Sycara and Paul Scerri, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. XXX-XXX.

General Terms

Algorithms; Experimentation

Keywords

DEC-POMDP, Uncertainty, Multi-agent systems

1. INTRODUCTION

Cooperative decision making in the presence of uncertainty is a problem that is encountered in many domains such as sensor networks and disaster rescue [6, 11]. Given the desire for representational accuracy of uncertainty in these domains, rich models such as Decentralized Partially Observable MDPs (DEC-POMDPs) are imperative. However, the NEXP complexity of solving DEC-POMDPs [2] limits their application to problems with two or three agents.

Recently, a model shaping approach called TREMOR was proposed to solve a sub-class of DEC-POMDPs [11]. It exploits dynamic locality, in which interactions are assumed to happen primarily in certain "coordination locales" (CLs), to scale to problems with ten agents. For example, two robots might be able to move freely across an open room, but interact by colliding in a narrow corridor. TREMOR computes the joint policy by iterating between (a) shaping of individual agent models to account for the active coordination locales ; and (b) resolving the models to obtain new policies. In addition to solving of POMDPs, the computation of active coordination locales and their value to the team (used for shaping) are both computationally expensive operations, which preclude its scalability to larger problems.

In this paper, we present the Distributed TREMOR (D-TREMOR) algorithm, a distributed planning approach that focuses computation on the most valuable interactions, to allow scale-up to hundreds of agents. The key to distributing the planning effort is being able to compute interaction values, without having to perform the exponential operation of comparing individual agent policies. In D-TREMOR, after computing an individual local policy, each agent creates a list of the CLs that have non-zero probability of occurrence and orders that list by the expected reward (or cost) of another agent being in that CL. For example, if an agent's local policy took it into a narrow corridor with high probability and another agent being there at the same time would

lead to a dramatic drop in its expected utility, that CL will appear near the top of the list. The highest value CLs are communicated to other agents who compare them against their own policy to find CLs with high value (or cost) interactions. Those are communicated back to the sending agent which uses them to shape rewards and recompute, similar to the shaping mechanism used in TREMOR. Notice that this mechanism differs conceptually from TREMOR because instead of comparing whole policies for interactions it focuses the search towards more likely and more important interactions. While this potentially reduces solution quality by a small amount, it leads to dramatic computational and communication savings.

However, distributed computation alone is not sufficient to reach good solutions, as the concurrent computation of policies can lead to impractical amounts of information exchange between agents, undesirable dynamics such as oscillations, and complexities in the dependencies between interactions. Thus, in combination with the distributed computation of important CLs, we introduce the following techniques which significantly improve the performance (both run-time and solution quality) of D-TREMOR. Firstly, we propose intelligent communication heuristics to reduce overhead. Secondly, unlike TREMOR, D-TREMOR employs heuristics – agent prioritization and probabilistic shaping – to ensure convergence, a property imperative for avoiding oscillations in distributed algorithms. In fact, for certain classes of CLs, D-TREMOR is proven to converge in a number of iterations equal to the size of the team. Thirdly, apart from being distributed, the algorithm employed for detecting and evaluating CLs uses a sampling technique to improve run-time without sacrificing quality. Next, in domains where there are a large number of CLs, shaping corresponding to a CL can have non-trivial effects on occurrence probability and value of other CLs, which in turn can affect the computation of the final joint policy. We provide mechanisms to capture these dependencies in computing improved policies. Finally, in TREMOR, the model shaping heuristics employed to capture the effects of one CL can potentially overwrite shaping performed for another. To address this, we introduce new shaping heuristics in D-TREMOR that aggregate the probabilities and values of multiple CLs that may occur when an agent has a particular local state and action.

D-TREMOR is evaluated on a simulated search and rescue task in which agents must work together to rescue victims while avoiding interfering with each other. Experiments are performed to measure performance as the size of the team and the number of potential interactions in the team are increased and the number of communications is decreased. Results show that D-TREMOR is able to find effective solutions in problems of up to 100 agents while remaining computationally tractable.

2. ILLUSTRATIVE RESCUE DOMAIN

We employ an illustrative disaster rescue problem similar to the one introduced in [11]. In this problem, a team of heterogeneous robots need to save victims trapped in a building where debris impedes robot movement. There are two types of robots available: (a) *rescue* robots provide medical attention to victims; while (b) *cleaner* robots remove debris from building corridors to allow easy passage for *rescue* robots. All robots must reason about uncertainty in their actual positions, slippages (action failures) when moving to locations and incomplete knowledge about the safety of locations.

The building is modeled as a discrete grid with narrow corridors and debris in certain grid cells (examples can be seen in Figure 3 in Section 5). The goal of the robots is to save as many victims as possible within the time available. Narrow corridors allow for only one robot to pass through; when multiple robots try to pass through, a collision (modeled as negative reward and the failure of one of the robots to enter the cell) occurs. On the other hand, cells containing debris let *rescue* robots pass through with only low probability. When a *cleaner* robot passes through, the debris is removed with certainty. If a robot passes through an unsafe cell, it incurs damage (modeled as negative reward). This creates a rich environment of conflicting positive and negative interactions and situations where modeling uncertainty is critical to team performance, making this a challenging problem in which to test decision-making. Interestingly, in our experiments we find that this simplification of modeling collisions and unsafe cells as negative rewards means that when these rewards are sufficiently large enough to impact policies, it is possible for policies that avoid risk to achieve higher values than policies that successfully rescue many victims, leading to unintuitive rankings of solutions.

3. BACKGROUND

In this section, we briefly describe the DPCL model and the TREMOR algorithm.

DPCL: We employ the Distributed POMDPs with Coordination Locales, DPCL model introduced in [11] to represent the problems of interest in this paper. DPCL is similar to the DEC-POMDP model and it is represented using the tuple of $\langle \mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}, \Omega, \mathbb{O} \rangle$, where $\mathbb{S}, \mathbb{A}, \Omega$ are the joint states, actions and observations over all the agents and $\mathbb{P}, \mathbb{R}, \mathbb{O}$ are the joint transition, reward and observation functions respectively.

DPCL differs from DEC-POMDPs in two aspects:

(a) The state space in DPCL consists of global states and local states for the individual agents, with global states representing the status of tasks.

(b) The interactions among agents are limited and in this regard, DPCL assumes that there can exist two types of interaction between agents:

(i) Same Time Coordination Locales (STCLs): STCLs represent situations where the effect of simultaneous execution of actions by a subset of agents cannot be described by the local transition and reward functions of these agents. *Example*: In the illustrative Rescue domain mentioned earlier, if two robots 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.

(ii) Future Time Coordination Locales (FTCLs): FTCLs represent situations where actions of one agent impact actions of others in the future. Informally, because agents modify the global state s_g 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.

A CL is defined as the tuple of $\langle t, s_g, \{s_i\}_1^m, \{a_i\}_1^m, \Gamma \rangle$, where t is the decision epoch, s_g and s_i are global and local states of agent i respectively, a_i is the action taken by agent i and Γ is the type of the coordination locale (either STCL or FTCL). The set of coordination locales is computed from the joint transition and reward functions. This can be performed automatically as described in [11]. Informally, a CL is "active" for an agent when it has a significant probability of entering the states and actions described by the CL.

TREMOR: We now describe the TREMOR (Teams RE-

shaping of MOdels for Rapid execution) algorithm [11]. The goal in TREMOR is to find an optimal task allocation, and provide a policy for each of the agents to accomplish their tasks. TREMOR performs an approximate branch and bound search over the set of all task allocations using MDP-based heuristics. The actual value of a specific task allocation is computed by solving the DPCL model for that allocation (Algorithm 1). In Algorithm 1, firstly, the poli-

Algorithm 1 ComputeValueofAllocation()

```
1: π* ← SolveIndividualPOMDPs({P_i}_{i≤N})
2: π ← φ
3: iter ← 0
4: while π ≠ π* && iter < MAX_ITERATIONS do
5:    ActiveCLs ← ComputeActiveCLs({P_i}_{i≤N}, AllCLs)
6:    for all cl ∈ ActiveCLs do
7:       {val_a}_{a∈cl.agents} ← EvaluateCL(cl)
8:       {P_a} ← ShapeModels(cl, ⟨{val_a}, {P_a}⟩_{a∈cl.agents})
9:    π* ← π
10:   π ← SolveIndividualPOMDPs({P_i}_{i≤N})
11:   iter ← iter + 1
```

cies are computed for individual agents assuming no other agents exist in the environment (line 1). Given the policies, the probability of occurrence of coordination locales is determined by propagating beliefs for the individual POMDPs and only the ones that are "active" (having a probability of occurrence $> \epsilon$) are considered for next stages in the algorithm (line 5). All the active CLs are evaluated for every agent involved in those CLs and these valuations along with the probability of occurrence of CLs are used to shape the POMDP models for the individual agents (lines 6-8). The updated models are solved to obtain new policies for the agents (line 10) and these steps are continued until convergence or for a maximum number of iterations (line 4).

The shaping of models in TREMOR is done in two steps: (a) Firstly, the individual transition and reward functions are modified in such a way that the joint policy evaluation is equal (or nearly equal) to the sum of individual policy evaluations; and (b) Secondly, incentives or hindrances are introduced in the individual agent models based on whether a CL accrues extra reward or is a cost to the team members. This incentive/hindrance is the difference in policy value for the team with the Coordination Locale.

By starting from individual POMDPs and incrementally modifying the model to accommodate most likely interactions, TREMOR was able to scale to problems that were not feasible with earlier approaches for Distributed POMDPs. However, the centralized detection and evaluation of interactions with all other agents limits the scalability of TREMOR. Towards addressing this problem with TREMOR, we introduce D-TREMOR.

Several other approaches exploit problem structures similar to DPCL to improve planning efficiency. Becker et al. [1] provided approaches for solving transition independent DEC-POMDPs, while ND-POMDPs [5] extend this transition-independence with network structure interactions. Oliehoek et al. [7] provide efficient algorithms for factored DEC-POMDPs assuming static interactions between agents. Seuken et al. [10] provide memory bounded dynamic programming (MBDP) approaches for solving general DEC-POMDPs. While MBDP and its variants solve considerably higher horizon problems, they have been primarily limited to two agent problems. There exist numerous other relevant approaches for solving DEC-MDPs/DEC-POMDPs, however, we differ from those through the distributed planning and the scale of problems solved by D-TREMOR.

4. DISTRIBUTED TREMOR

In this paper, our focus is primarily on the computation of a joint policy given an allocation of tasks to agents. Any existing distributed role allocation algorithm [9, 8, 3] can be used to compute the allocation of tasks to agents in DPCL. Distributed TREMOR (D-TREMOR) avoids the scalability problems inherent in TREMOR and other approaches for solving DEC-POMDPs by distributing the planning effort between agents and employing heuristics in CL communication and model shaping. We describe the basic distributed planning algorithm of D-TREMOR and then detail the various heuristics employed to improve its performance.

In D-TREMOR, each agent after initializing to a starting policy iterates over the following two steps until convergence (or a maximum number of iterations):

Step 1: Exchange messages with other agents indicating relative impact of coordination locales given the current individual policies.

Step 2: Use received messages to shape individual models and re-compute policies.

Algorithm 2 provides the pseudo code executed at each agent in performing these two steps. In **Step 1**, each agent

Algorithm 2 D-TREMOR(Agent i)

```
1: π_i ← ObtainInitialPolicy(M_i, allCLs)
2: iter ← 0
3: while iter < MAX_ITERATIONS do
4:    αCLs ← ComputeActiveCLs(M_i, allCLs, π_i)
5:    for all cl ∈ αCLs do
6:       val_{i,cl} ← EvaluateCL(cl, M_i, π_i)
7:       CommunicateCL(i, cl, pr_{i,cl}, val_{i,cl})
8:    recCLs ← ReceiveCLs()
9:    M_i ← ShapeModel(recCLs, M_i)
10:   π_i ← SolveIndividualPOMDP(M_i)
11:   iter ← iter + 1
```

computes the set of CLs which could be active given its own policy, i.e., $\alpha CLs = \{cl | cl = \langle t, s_g, \{s_i\}_1^m, \{a_i\}_1^m, \Gamma \rangle, Pr_{\pi_i}((s_g, s_i), a_i) > \epsilon\}$. For ease of explanation, we will refer to $Pr_{\pi_i}((s_g, s_i), a_i)$ as Pr_{cl_i}. Since the interaction between agents is determined by the CLs active for all the agents concerned, each agent communicates its set of active *CL Messages* to all the relevant agents.

A *CL Message* is defined as the tuple: $\langle id, cl, Pr_{cl_i}, V_{cl_i} \rangle$. It contains the agent ID, the coordination locale (which also contains the time of interaction), probability of occurrence of the coordination locale for the agent, and the value associated by the agent for the coordination locale. For a CL between two agents, given a particular pair of messages, it is thus possible to approximate the utility and probability of the event occurring. Given $\langle id_i, cl_i, Pr_{cl_i}, V_{cl_i} \rangle$ and $\langle id_j, cl_j, Pr_{cl_j}, V_{cl_j} \rangle$, the joint utility of the action can be estimated as $V_{cl_i} + V_{cl_j}$, while the probability of the event is $Pr_{cl_i} * Pr_{cl_j}$. From this, the expected joint utility can be computed to be $Pr_{cl_i} \cdot Pr_{cl_j} \cdot (V_{cl_i} + V_{cl_j})$.

In **Step 2**, each agent shapes the transition and reward function of its individual model upon receiving CL messages from other agents. Each agent i that receives a CL message from j computes the probability of occurrence of cl, Pr_{cl_i} and the value of the CL, V_{cl_i}. The probability of occurrence

of a coordination locale with respect to both agents is then computed, i.e. $\hat{c}_{cl} = Pr_{cl_i} * Pr_{cl_j}$.

In TREMOR, the new transition probability $\mathcal{P}'^e_i$ at decision epoch e for STCLs is computed by using a shaping heuristic. According to this heuristic, we take the weighted average of $\mathcal{P}^e_{i,cl_s}$ and $\mathcal{P}^e_{i,\neg cl_s}$. $\mathcal{P}^e_{i,\neg cl_s}$ is the transition probability without any interactions, i.e. $\mathcal{P}_i$. In D-TREMOR, we provide a new improved heuristic as described in Section 4.6. While the expressions below are for STCLs, the expressions for FTCLs are similar as explained in [11].

$$\mathcal{P}^e_{i,cl}((s_g, s_i), a_i, (s'_g, s'_i)) \leftarrow \sum_{s' \in S : s' = (s'_i, s'_j)} P((s_g, s_i, s_j), (a_i, a_j), (s'_g, s'_i, s'_j)) \quad (1)$$

$$\mathcal{P}'^e_i \leftarrow \hat{c}_{cl} \cdot \mathcal{P}^e_{i,cl} + (1 - \hat{c}_{cl}) \cdot \mathcal{P}^e_{i,\neg cl} \quad (2)$$

$$\mathcal{R}^e_{i,cl}((s_g, s_i), a_i, (s'_g, s'_i)) \leftarrow \sum_{s' \in S : s' = (s'_i, s'_j)} R((s_g, s_i, s_j), (a_i, a_j), (s'_g, s'_i, s'_j)) \quad (3)$$

$$\mathcal{R}'^e_i \leftarrow \hat{c}_{cl} \cdot \mathcal{R}^e_{i,cl} + (1 - \hat{c}_{cl}) \cdot \mathcal{R}^e_{i,\neg cl} \quad (4)$$

We now explain the key contributions made by the D-TREMOR algorithm, which considerably improve its performance over existing algorithms. As we will show in the experimental results, the combination of these ideas helps D-TREMOR scale to hundred agent DPCL problems, at least an order of magnitude larger than the scale of problems solved previously.

4.1 Distributed computation

As with all distributed algorithms, there needs to be parallelism in computation to get improved performance. In D-TREMOR, we ensure that this parallelism is exploited in all the key bottleneck computations:

(a) Computing Pr_{cl_i}: Every agent i only needs to compute the probability of all distinct $(e, (s_g, s_i), a_i)$ pairs (given its current policy) out of all possible CLs. Thus for a cl : $\langle (e, (s_g, s_i, s_j), (a_i, a_j)) \rangle$, agent i computes the probability for $(e, (s_g, s_i), a_i)$ given its policy π_i and agent j computes the probability for $(e, (s_g, s_j), a_j)$ given its policy π_j. Therefore, there is independence (or parallelism) in this computation of probability of CL occurrence or Pr_{cl_i}.

(b) Evaluation of CLs: As with probability of occurrence of CLs, the value of a CL for that agent can also be computed independent of other agents, thus allowing parallelism.

(c) Solving individual POMDPs: After the shaping of models is performed corresponding to the received messages, the individual POMDP models are solved. Since there is no dependence between agents in solving these models, parallelism is exploited. Specifically, as the complexity of the individual model increases (i.e. more states, actions, observations), run-time benefits due to distributed computation also increase.

4.2 Communication heuristics

In its simplest form, D-TREMOR completely communicates CL messages across the team. That is, every active CL can be converted into a CL message and sent to every team member. This ensures that every agent is aware of any teammate it might interact with, but also means that agents send n messages for every active CL, quickly leading to thousands of messages being exchanged over the team. It is possible that not all of the messages need to be exchanged, as many of them may describe interactions that are of little value or unlikely to actually occur.

One approximation of the usefulness of a CL message is its local expected value. This is the product $Pr_{cl} \cdot V_{cl}$. Figure 1 shows a distribution of these values compiled from D-TREMOR runs on the scaling dataset described in Section 5. It appears that a majority of CLs have relatively low value, and a small number have very high value. It therefore seems that communication could be made more efficient by prioritizing the delivery of high-valued CL messages while dropping some lower-valued messages. A *best-first* commu-

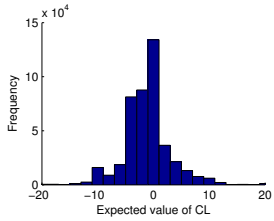


Figure 1: Distribution of expected CL value over scaling dataset

nication heuristic, in which agents order CL messages by absolute local expected value, can be applied to this task. Each agent selects up to the top k messages from their ordered list and sends these CLs to the team. Under this scheme, the CLs that have highest potential impacts on the value of the team should be sent first, but overall, communications should be reduced. While intuitive, experimental results with this heuristic reveal the sensitivity of D-TREMOR to communications loss.

4.3 Convergence heuristics

As it involves multiple agents concurrently planning, D-TREMOR faces the challenge of avoiding oscillations that can occur when multiple agents simultaneously correct for a common interaction. These oscillations delay the exploration of policy space, and in the worst case, can prevent the discovery of other solutions altogether. Though not theoretically guaranteed for all cases, empirically (as we show in our experimental results) D-TREMOR is typically able to break out of oscillations and converge to a solution. This is obtained by using a combination of two heuristics:

(a) Probabilistic model shaping: This heuristic is inspired by the approach adopted by the Distributed Stochastic Algorithm (DSA) for solving Distributed Constraint Satisfaction Problems [13]. It is governed by a parameter δ, which represents the probability that an agent will shape its model given messages from other agents. Upon receiving messages from other agents at each iteration of D-TREMOR, an agent generates a random number (between 0 and 1) and only if the generated random number is greater than δ, that agent shapes its model to account for the received CL messages.

(b) Agent prioritization: This heuristic is specifically designed to handle negative interactions (i.e. CLs with negative expected value). In negative interactions, the penalty is avoided if all agents except one avoid the interaction. For instance, in the example problem of Section 2, an interaction where two robots collide in a corridor, it is sufficient if we allow only one agent to pass through the corridor. As part of this heuristic, each agent is initially (before start of the algorithm) assigned a priority value randomly and an agents' model is shaped corresponding to a negative CL message unless it has the highest priority of all the agents involved.

PROPOSITION 1. *D-TREMOR will converge within n (number of agents) iterations for any DPCL problem with only*

negative coordination locales if the agent prioritization heuristic is employed.

PROOF. Without loss of generality let us assume a DPCL problem with n agents and priorities, $\{r_i\}_1^n$, such that $r_1 > r_2 > r_3 ... > r_n$. At the first iteration of D-TREMOR, all the agents would compute their individual policies. According to the agent prioritization heuristic, agent 1 would continue its course (i.e. not shape its model) irrespective of any CL messages it would have received from other agents. Thus, agent 1 would not change from its initial policy and consequently, communicates the same set of CL messages to other agents in all the iterations.

Agent 2 only needs to shape its model corresponding to CL messages from agent 1. Therefore it would have a new policy in iteration 2. Since it receives the same set of messages from agent 1, agent 2 would not have to change its policy after iteration 2. Therefore, agent 2 communicates the same set of CL messages to other agents after iteration 2.

Continuing this reasoning, agent 3 would not have to modify its policy in iteration 3 and so on. Therefore, the D-TREMOR algorithm will converge within n number of iterations with agent prioritization heuristic. □

In the motivating domain of Section 2, collisions in narrow corridors represent negative coordination locales primarily because (a) There is a cost to collision of robots; and (b) collisions cause robots to return to their original position with certain probability; Thus from the above proposition, D-TREMOR with agent prioritization converges for problems where there are only narrow corridors.

4.4 Computing Pr_{cl_i} and V_{cl_i} efficiently

While parallelism in the computation of Pr_{cl_i} and V_{cl_i} improves performance significantly, the exponential computational complexity involved in computing Pr_{cl_i} and V_{cl_i} is still a bottleneck at each agent. This is because an exact computation of Pr_{cl_i} and V_{cl_i} requires evaluation over all possible combinations of the occurrence of previous CLs. To improve the efficiency of these computations, we provide an approach inspired from the sampling approach developed for solving large Markov Decision Processes by Kearns *et al* [4]. The main idea is that in problems where there exists a generative model, the value function can be computed efficiently by using a set of samples generated with the generative model. Algorithm 3 provides the sampling method to compute the probability of a CL for an agent i. In this approach, we generate execution samples corresponding to the current policy and agent model. Finally, we obtain the average number of times the coordination locale is active over the total number of execution samples. Depending on the time horizon and the desired accuracy of Pr_{cl_i}, the total number of samples can be modified. A similar algorithm is used for computing V_{cl_i}. We also provide a preprocessing step to detect CLs which can be completely eliminated from consideration at future iterations of the algorithm. For instance, a robot on the first floor of a building should not have to worry about the robots on the 10th floor if the time horizon is small. For each agent, the part of interest in a CL is its state, s and action, a which can lead to an interaction with other agents. The key idea here is to solve maximization and minimization problems on the belief update expressions and eliminate the consideration of CLs where the state s (of the agent in consideration) is unreachable, i.e. $b_s < \epsilon$ (where ϵ is close to zero) given the time horizon. Given an action a and observation ω, the maximization problem for belief probability of state s_t (state s at decision epoch t) is given by:

$$\max_{b_{t-1} \in B_{t-1}} \frac{O_t(s_t, a, \omega) \Sigma_{s_{t-1}} P_{t-1}(s_{t-1}, a, s_t) b_{t-1}(s_{t-1})}{\sum_{s_t} O_t(s_t, a, \omega) \Sigma_{s_{t-1}} P_{t-1}(s_{t-1}, a, s_t) b_{t-1}(s_{t-1})}$$

This is solved in polynomial time using the lagrangian techniques presented in [12].

Algorithm 3 COMPUTEPRCL$(i, cl, \hat{pi}_i, b^0)$

```
iter ← 0
val = 0
while iter < NUM − SAMPLES do
    π_i ← π̂_i; s ← GETSIMSTATE(b^0); τ ← 0
    while τ < cl.t do
        act ← π_i.a
        s' ← GETSIMFUTURESTATE(s, act)
        ω ← GETSIMOBS(s', act)
        π_i ← π_i(ω); s ← s'
    if s = cl.s_i and act = cl.a_i then
        val ← val + 1
return val / (NUM−SAMPLES)
```

4.5 Capturing dependencies between CLs

In TREMOR, each CL is treated independently of others, i.e. assuming that model shaping corresponding to one CL does not affect any other CL. In weakly coupled domains, i.e., ones with few CLs, such an assumption is perfectly reasonable. However in tightly coupled domains, these dependencies are non-trivial. To obtain better coordination between agents, it is imperative that such dependencies are accounted for. However, capturing dependencies between all CLs would entail searching for an optimal policy in the joint policy space and hence would be prohibitively expensive.

Therefore, we are interested in capturing dependencies between CLs which improve performance without incurring a significant computational cost. One such set of dependencies are the ones between CLs occurring at different decision epochs. In our rescue domain, for example, there may be a case where having a collision in one epoch (an STCL) might prevent a cleaner robot from clearing some debris in a later epoch (an FTCL). In order to capture these dependencies over decision epochs, we make the following modifications: Firstly, we sort the received set of messages with respect to the decision epoch, $cl.e$. Secondly, while computing Pr_{cl_i} and V_{cl_i}, we consider the modifications made to the model for CLs with decision epochs, $cl'.e < cl.e$. Using such an approach, we are able to capture dependencies between CLs and obtain accurate estimates of Pr_{cl_i} and V_{cl_i}, while not sacrificing efficiency. Such accurate estimates of Pr_{cl_i} and V_{cl_i} essentially reduce the difference between the shaped models and the joint model and hence provide improved solutions.

4.6 Shaping Heuristics

In the context of the expressions in Equation 2 and Equation 4, consider a scenario where two CLs, $cl1$ and $cl2$ have the same e, s_i and a_i (but different s_g, s_j and a_j). If the model for agent i is updated corresponding to $cl1$ first and $cl2$ next, it should be noted that the model update corresponding to $cl1$ could potentially be overwritten by model update due to $cl2$. To address such inconsistencies in model updates, we propose new model shaping heuristics. We use the set $CL^i_{s,a}$ to correspond to all CLs which have the same state s and same action a corresponding to agent i. Instead of considering the occurrence and non occurrence of each CL separately, we aggregate corresponding to all CLs which have the same state and action pair for the agent. Therefore, the new heuristics for shaping of transition and reward functions are:

$$\mathcal{P}''^{e}_{i} \leftarrow \sum_{cl \in CL^{i}_{s,a}} \hat{c}_{cl} \cdot \mathcal{P}^{e}_{i,cl} + (1 - \sum_{cl \in CL^{i}_{s,a}} \hat{c}_{cl}) \cdot \mathcal{P}^{e}_{i,\neg cl} \quad (5)$$

$$\mathcal{R}''^{e}_{i} \leftarrow \sum_{cl \in CL^{i}_{s,a}} \hat{c}_{cl} \cdot \mathcal{R}^{e}_{i,cl_s} + (1 - \sum_{cl \in CL^{i}_{s,a}} \hat{c}_{cl}) \cdot \mathcal{R}^{e}_{i,\neg cl_s} \quad (6)$$

In these expressions, we compute new transition and reward values by accounting for effects of all the CLs at once and hence effects of a CL are not overwritten.

4.7 Policy Initialization

Given the local optimal moves made at each agent, the initial policy assumes significance in D-TREMOR. In TREMOR, the best local policy (obtained by solving the initial individual model) is the starting point for the algorithm. Due to local optimization, such a policy may not traverse states and actions where the joint rewards are higher than individual rewards. For instance, in the illustrative domain of Section 2, consider the example in Figure 2. If we assume there is no reward for the cleaner robot to clean the debris, the best policy for the cleaner robot is to stay in its cell, and for the rescue robot, it is to go around the debris. With such a starting policy, the CL corresponding to the debris would never be detected in TREMOR. To account

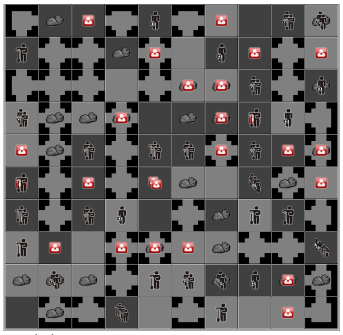
Figure 2: Policy initialization example.

for such positive interactions, we introduce an **optimistic** policy. We modify the model of each agent to account for the optimistic assumption, i.e. assuming that all positive reward CLs occur at every decision epoch. That is to say: For every agent i, $\forall cl \in CLs$, if $\mathbb{R}(s_g,(s_i,s_j),(a_i,a_j)) > \mathcal{R}_i(s_g,s_i,a_i) + \mathcal{R}_j(s_g,s_j,a_j)$, then $Pr^{e}_{i,cl} = 1$.

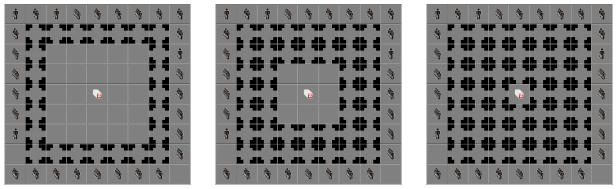
These updated models are solved to obtain the optimistic policy. While, it is not guaranteed to account for all possible interactions, empirically it is able to identify all the important interactions.

5. EVALUATION

Two datasets were created to test the performance of D-TREMOR under various conditions, a *scaling* dataset and a *density* dataset. In the scaling dataset, the total number of agents is varied from 10 to 100 agents. Maps are constructed randomly, with salient features fixed proportionally to the number of agents. Maps are square, with a ratio of approximately 2 map cells per agent. 35% of the cells are narrow and only 50% of the remaining are safe. The team is half rescue agents and half cleaner agents. Debris and victims are added to the map of the same numbers as cleaner and rescue agents, respectively. Figure 3(a) shows a sample of the maps generated for this dataset. The purpose of this dataset is to test the overall performance and scalability of D-TREMOR on complex environments with multiple types of interactions. However, due to the long computation time (up to 15 min. per iteration), only three randomly generated map sets could be evaluated. In this small of a dataset, some maps can have pathologically extreme interaction, sometimes never requiring agents to interact and sometimes requiring tremendous interaction in order to accomplish anything. Because this variation in maps translates to high variance in performance measures, we focus on qualitative overall trends in the data, rather than the quantitative values of individual data points.



(a) Scaling map, 50 agents



(b) Density, 1 ring (c) Density, 2 rings (d) Density, 3 rings

Figure 3: Examples of the maps generated for the scaling and density datasets.

In the density dataset, a square 9×9 map is constructed with 100 rescue agents located on the outer perimeter, and 100 victims located in the center of the map. As seen in Figures 3(b), 3(c) and 3(d), the victims are surrounded by 1, 2, or 3 rings of narrow corridors, forcing the agents to negotiate passage through an increasingly crowded map. The purpose of this dataset is to test D-TREMOR in handling increasingly dense STCL interactions.

Due to the large size of these state spaces, other state-of-the-art POMDP solvers cannot be used for comparison, including the original TREMOR algorithm (demonstrated only in problems of up to 10 agents [11]). D-TREMOR is thus compared against several heuristic strategies, *independent* planning, *optimistic* planning, a *do-nothing* policy, and a *random* policy. In independent planning, n independent POMDP solvers are executed in parallel, with no coordination between agents, and with each agent assuming that the environment will remain exactly as specified a priori. In optimistic planning, n independent planners are used again, but agents assume the optimistic policy introduced in Section 4.7. That is, rescue agents assume that all narrow corridors are unobstructed, and all debris will be cleared. Cleaner agents assume that all narrow corridors are unobstructed, and that any debris that is successfully cleared will allow a rescue agent to reach a victim, yielding a net reward exactly equal to the reward of rescuing the victim (i.e. ignoring the movement costs of a rescue robot, etc.). In the do-nothing policy, agents do not move from their original locations, and in the random policy, each agent independently selects their action uniformly randomly from the set of possible actions.

Several performance measures are taken from each run to study the performance of the algorithms. The policies generated by each agent are jointly simulated 2000 times to empirically compute an expected joint reward. This is used as the primary measure of task performance. Empirical averages of the numbers of collisions, victims saved and debris cleared are also recorded. The planning times and number of activated CLs for each agent are also totaled and averaged. As the D-TREMOR algorithm consists of multiple iterations (of message communication and shaping), these measures

can be computed at each iteration or averaged over entire runs. In these experiments, D-TREMOR performs a greedy role assignment in the first iteration, and communicate CLs fully during subsequent iterations. An iteration limit of 20 is used for all of the maps. All experiments were performed on a 104 CPU computing cluster, with each POMDP solver running as a single thread on an available CPU.

Because D-TREMOR has agents individually approximate the joint value at each iteration, it is possible for the team to find good solutions but not be able to detect it. Thus, it is sensible to provide two measures of the overall performance of the algorithm: (a) the value of the joint policy generated by D-TREMOR at the end of the last iteration (D-TREMOR); and (b) the highest joint-valued policy among all the D-TREMOR iterations (Max D-TREMOR). The latter requires some additional communication and computation overhead, as it necessitates exchanging policies and performing a joint evaluation every iteration, but this is relatively small compared to the cost of POMDP planning.

The results of the scaling dataset can be seen in Figure 4. Data are normalized to independent planning by subtracting its performance from that of the other algorithms. Figure 4(b), compares the average joint value of the various solution policies. The maximum iteration of D-TREMOR outperforms or matches the other techniques in every case. This establishes the ability of the algorithm to find good joint solutions in complex environments. However, the value of the last iteration of D-TREMOR underscores the fact that in its current form, it cannot always detect when it has reached a good solution. In a single run (Figure 4(i)), we see that overall, joint value trends upward, but over individual iterations joint value can decrease.

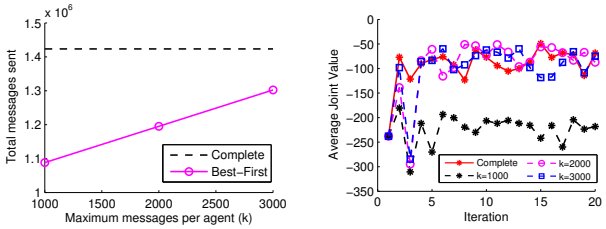
Examining the components of the value function, it is possible to determine how the D-TREMOR achieves its value. In looking at the number of victims rescued (Figure 4(c)), it is apparent that there is not much difference between the independent, optimistic, and D-TREMOR algorithms, while random and do-nothing policies manage very few rescues. In avoiding collisions (Figure 4(d)), however, D-TREMOR has fewer collisions than independent or optimistic, performing similarly to the random policy. Optimistic collides very frequently by comparison, while do-nothing avoids collisions trivially by never moving.

While cleaner robots clear many debris under the optimistic policies (Figure 4(f)), their number of rescue robots colliding with debris is higher than that of the independent policies (Figure 4(e)) as optimistic rescue robots assume debris is clear before it can be cleared. D-TREMOR is more targeted, clearing only a few more debris than the independent and random policies, which clear debris only when it is self-serving (independent), or by chance (random), while reducing the number of debris collisions to often be below that of the independent policies.

Next, we consider the time scalability of the algorithm. The computing cluster used in this experiment had over 1 virtual core per agent, making it is possible to directly compare the running times across the scaling dataset, as agents need not compete for CPU resources. Figure 4(h) shows a linear trend in average time per iteration. Deviations from this trend appear to correspond to maps that cause a large number of activated CLs (Figure 4(g)).

Results of the density dataset are seen in Figure 5. As expected, increasing the density of narrow corridors decreases the performance of all policies except the do-nothing policy (Figure 5(a)). The abundance of narrow corridors causes optimistic and independent policies to suffer a very high number of collisions (Figure 5(b)), dropping their overall value despite the fact that they manage to secure some victims (Figure 5(c)). The do-nothing and random policies do not rescue any victims, but have relatively few collisions, leaving them with high overall joint values. The random policy has only a one in eight chance of entering narrow corridors at all, while the do-nothing policy never attempt to, so their values differ by the expected penalty of the random policy causing a collision. D-TREMOR's policies, in value alone, straddle this region, but other measures suggest that it reaches this region through a vastly different behavior than the previous two policies. D-TREMOR rescues more victims than any of the other policies, and while it drastically reduces, it cannot eliminate collisions between agents. However, despite rescuing many more victims, the failure of D-TREMOR to resolve the remaining collisions leads to a poorer overall value than the do-nothing policy, a counter-intuitive effect of the reward/penalty functions constructed for this domain.

The number of CLs activated (Figure 5(d)) indicate that while there are many possible collisions in the map, relatively few must actually be resolved. Agents consider on average only 40 to 90 joint state-action pairs. Part of this, and the intuition behind the drop in CLs between 2 rings and 3 rings, is because in the initial few iterations, many agents realize that they cannot all fit through the narrow corridors, and decide to stay clear entirely, ceasing to generate CLs.



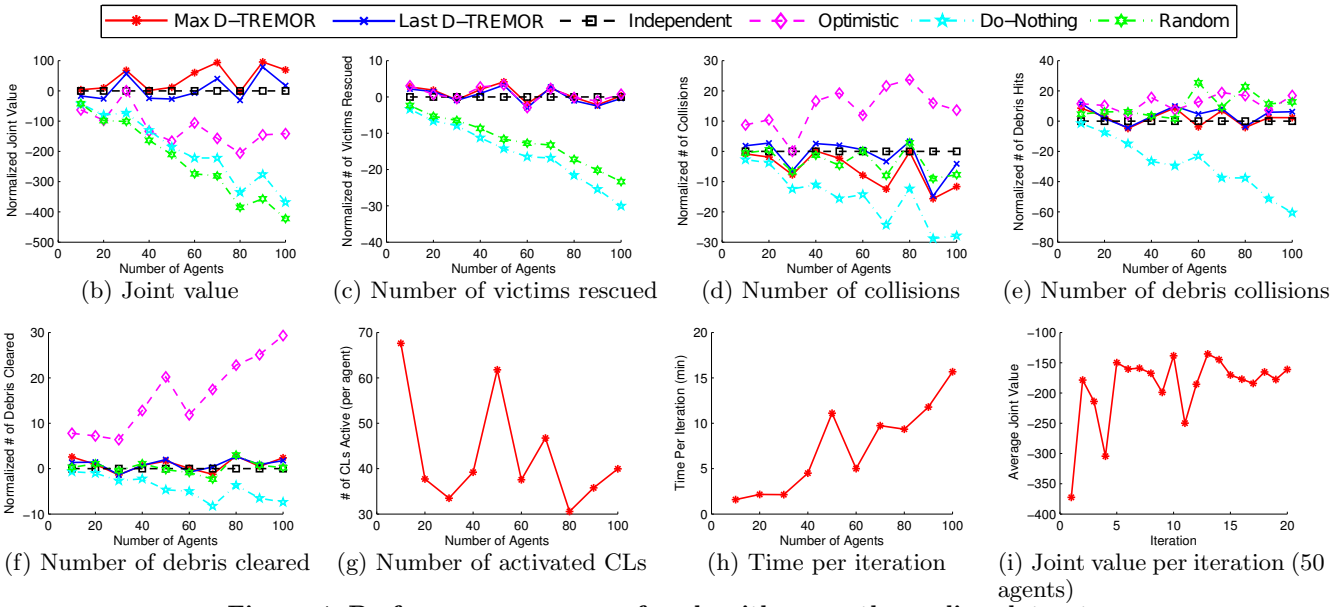
(a) Number of sent messages (b) Joint value

Figure 6: Results of *best-first* communications.

An experiment was performed to determine if using the *best-first* heuristic could reduce communications over the team without sacrificing performance. Using the heuristic, a maximum number of messages per agent, k, was set on a 50 agent map from the scaling dataset. Adjusting k led to a smooth reduction in total message exchange, as seen in Figure 6(a), while maintaining performance–up to a critical point. In Figure 6(b), the joint value of the D-TREMOR algorithm is plotted for various k. The change in performance is minimal for $k \geq 2000$, but between $k = 2000$ and $k = 1000$, the algorithm no longer converges anywhere near the complete communication solution. This suggests that the convergence of D-TREMOR is extremely sensitive to the message exchange between agents, and that selecting messages by local expected value, while effective in reducing communication, may not be a stable mechanism controlling message exchange across the team.

6. CONCLUSIONS AND FUTURE WORK

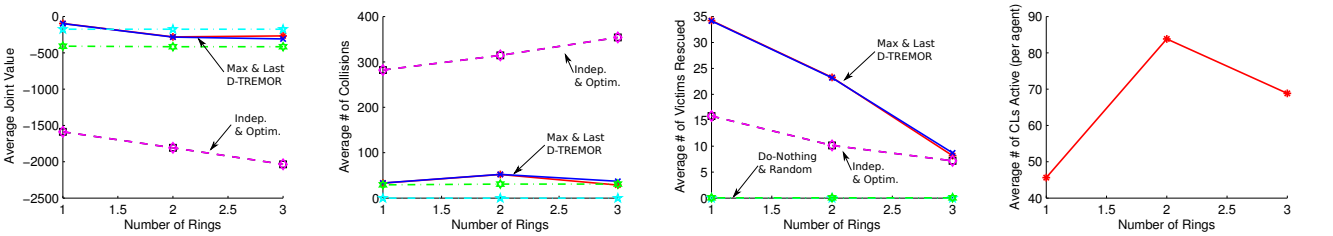
In this paper, we present D-TREMOR, a fully distributed DEC-POMDP algorithm capable of computing policies for 100 agents in around five hours. This represents a dramatic increase in the size of problem that can be solved. The algorithms gets its scalability by taking advantage of the fact that although agents might interact in a very large number



(b) Joint value (c) Number of victims rescued (d) Number of collisions (e) Number of debris collisions

(f) Number of debris cleared (g) Number of activated CLs (h) Time per iteration (i) Joint value per iteration (50 agents)

Figure 4: Performance measures for algorithms on the scaling dataset.



(a) Joint value (b) Number of collisions (c) Number of victims rescued (d) Number of CLs activated

Figure 5: Performance measures for algorithms on the density dataset.

of ways, for any particular choices of individual actions they will interact in relatively few *coordination locales*. Several additional techniques are applied to assure convergence and allow agents to discover high-quality solutions efficiently.

While this work represents a significant step towards making DEC-POMDPs a practically useful tool, much more work is required. Our immediate focus will be to find more effective ways of reaching convergence and reducing the message traffic of the algorithm. Since D-TREMOR uses an off-the-shelf POMDP solver, we can also exploit technical advances in POMDP-solving to further increase the size and complexity of the problems that can be addressed.

7. ACKNOWLEDGMENTS

This research has been funded in part by the AFOSR MURI grant FA9550-08-1-0356. This material is based upon work supported under a National Science Foundation Graduate Research Fellowship.

8. REFERENCES

[1] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Solving transition independent decentralized Markov decision processes. *JAIR*, 22:423–455, December 2004.

[2] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Math. Oper. Res.*, 27(4):819–840, 2002.

[3] B. Gerkey and M. Mataric. Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *ICRA*, 2003.

[4] M. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002.

[5] J. Marecki, T. Gupta, P. Varakantham, M. Yokoo, and M. Tambe. Exploiting coordination locales in distributed POMDPs via social model shaping. In *ICAPS*, 2009.

[6] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed pomdps: A synthesis of distributed constraint optimization and POMDPs. In *AAAI*, 2005.

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

[8] P. V. Sander, D. Peleshchuk, and B. J. Grosz. A scalable, distributed algorithm for efficient task allocation. In *AAMAS*, 2002.

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

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

[11] P. Varakantham, J. Y. Kwak, M. Taylor, J. Marecki, P. Scerri, and M. Tambe. Exploiting coordination locales in distributed POMDPs via social model shaping. In *ICAPS*, 2009.

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

[13] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1-2):55–87, 2005.