Thesis Proposal
Distributed Planning for Large Teams

Prasanna Velagapudi

November 2010

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Katia Sycara, Co-chair
Paul Scerri, Co-chair
J. Andrew Bagnell
Edmund H. Durfee, University of Michigan

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Copyright © 2010 Prasanna Velagapudi
Abstract

In many domains, teams of hundreds of agents must cooperatively plan to perform tasks in a complex, uncertain environment. Naively, this requires that each agent take into account every teammates’ state, observation, and choice of action when making decisions about its own actions. This results in a huge joint policy space over which it is computationally intractable to find solutions. In certain problems, however, searching this complete space may not be necessary. Specifically, there are problems in which individual agents usually act independently, but have a few combinations of states and actions in which they share a non-factorable transition, reward, or observation function with one or more teammates.

This thesis focuses on exploiting this structure, along with two other properties that are often present in these cases, to greatly improve planning efficiency. First, while there are a large number of possible interactions between agents, the number of interactions that actually occur in a particular solution instance is often quite small. It is therefore possible to disregard many irrelevant combinations of interactions by dynamically handling only those that arise during the planning process. Second, in the case of intelligent agents, computational power itself is often distributed across the team. Thus, distributed approaches have access to computational resources that grow linearly with team size, making it easier to scale to very large teams.

Taking advantage of these properties, we propose DIMS, a framework in which agents plan iteratively and concurrently over independent local models which are then shaped by the expected observations, movements and rewards of their teammates. By dynamically discovering relevant interactions and distributing computation, planning efficiency is greatly improved, allowing joint solutions to be computed for teams into the hundreds of agents.

Initial experiments have been conducted in a simplified urban search and rescue domain. These experiments demonstrate the promise of the approach. To complete the thesis work, we will extend these experiments to urban search and rescue planning and a humanitarian convoy planning task. In each domain, a low-fidelity, large scale model and a high-fidelity, real-time physical simulation model are constructed. Empirical results over these four conditions verify the solution quality, scalability and practicality of the framework.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong></td>
<td>Introduction</td>
<td>8</td>
</tr>
<tr>
<td>1.1</td>
<td>Thesis statement</td>
<td>10</td>
</tr>
<tr>
<td><strong>2</strong></td>
<td>Background</td>
<td>11</td>
</tr>
<tr>
<td>2.1</td>
<td>Path-planning</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Decision-theoretic planning</td>
<td>12</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Dec-MDPs</td>
<td>13</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Dec-POMDPs</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>Game-theoretic planning</td>
<td>14</td>
</tr>
<tr>
<td><strong>3</strong></td>
<td>Approach</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Dec-POMDPs</td>
<td>15</td>
</tr>
<tr>
<td>3.2</td>
<td>Sparsely Interacting Dec-POMDPs (SI-Dec-POMDPs)</td>
<td>16</td>
</tr>
<tr>
<td>3.3</td>
<td>Distributed Iterative Model-Shaping (DIMs)</td>
<td>17</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Role Allocation</td>
<td>19</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Independent Planning</td>
<td>20</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Interaction Exchange</td>
<td>20</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Model Shaping</td>
<td>20</td>
</tr>
<tr>
<td>3.4</td>
<td>Problem Domains</td>
<td>21</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Urban Search and Rescue</td>
<td>21</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Convoy Planning</td>
<td>22</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td>Preliminary Work</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Distributed Path Planning</td>
<td>23</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Distributed Prioritized Planning (DPP)</td>
<td>24</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Experimental results using DPP</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>Distributed POMDP Solving</td>
<td>30</td>
</tr>
<tr>
<td>4.2.1</td>
<td>DPCL</td>
<td>31</td>
</tr>
<tr>
<td>4.2.2</td>
<td>TREMOR</td>
<td>32</td>
</tr>
<tr>
<td>4.2.3</td>
<td>D-TREMOR</td>
<td>33</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Distributed computation</td>
<td>35</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Convergence heuristics</td>
<td>36</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Computing $P_{cl}$ and $V_{cl}$ efficiently</td>
<td>37</td>
</tr>
<tr>
<td>4.2.7</td>
<td>Capturing dependencies between CLs</td>
<td>38</td>
</tr>
<tr>
<td>4.2.8</td>
<td>Shaping Heuristics</td>
<td>38</td>
</tr>
<tr>
<td>4.2.9</td>
<td>Policy Initialization</td>
<td>38</td>
</tr>
<tr>
<td>4.3</td>
<td>Initial results</td>
<td>39</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Summary of results</td>
<td>44</td>
</tr>
<tr>
<td><strong>5</strong></td>
<td>Proposed Work</td>
<td>45</td>
</tr>
<tr>
<td>5.1</td>
<td>Interaction Taxonomy</td>
<td>46</td>
</tr>
<tr>
<td>5.2</td>
<td>Heuristic Development</td>
<td>46</td>
</tr>
</tbody>
</table>
5.3 Evaluation ............................................................... 47
5.4 Timeline ............................................................... 48
5.5 Risk Mitigation ....................................................... 50

6 Conclusion .................................................................. 50
6.1 Expected Contributions ............................................... 50
6.2 Significance and Future ............................................... 50

7 Bibliography ................................................................ 51
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Descriptions of the symbols used in the simple rescue domain.</td>
</tr>
<tr>
<td>2</td>
<td>A $5 \times 5$ rescue domain where a rescue robot plans to reach a victim, assisted by a cleaner robot.</td>
</tr>
<tr>
<td>3</td>
<td>Multiagent path planning describes a rescue problem where there is no action or observational uncertainty, role-allocation has been done a-priori, and every cell is a narrow corridor with an infinite collision penalty.</td>
</tr>
<tr>
<td>4</td>
<td>Two typical maps used in DPP experiments. Circles denote robot start positions while stars denote goals. In the smaller map, dotted lines connect associated start and goal positions.</td>
</tr>
<tr>
<td>5</td>
<td>Average results of centralized and distributed prioritized planners on the team-size dataset. Each point represents 15 runs.</td>
</tr>
<tr>
<td>6</td>
<td>Average results of centralized and distributed prioritized planners on the map-density datasets. Each point represents 15 runs.</td>
</tr>
<tr>
<td>7</td>
<td>Policy initialization example.</td>
</tr>
<tr>
<td>8</td>
<td>Examples of the maps generated for the scaling and density datasets.</td>
</tr>
<tr>
<td>9</td>
<td>Performance measures for algorithms on the scaling dataset.</td>
</tr>
<tr>
<td>10</td>
<td>D-TREMOR summary statistics for experiments on the scaling dataset.</td>
</tr>
<tr>
<td>11</td>
<td>Performance measures for algorithms on the density dataset.</td>
</tr>
<tr>
<td>12</td>
<td>Two high-fidelity simulation environments will be used to create and verify rescue and convoy domain models.</td>
</tr>
<tr>
<td>13</td>
<td>Detailed timeline of proposed research.</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Comparison of DIMS and decompositions of distributed problem solving in literature</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>Probabilistic cellular automaton used to generate path planning maps.</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>Comparison of TREMOR and D-TREMOR</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>Proposed domains and models</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>Timeline of proposed research</td>
<td>48</td>
</tr>
<tr>
<td>6</td>
<td>Risks and mitigation strategies</td>
<td>50</td>
</tr>
</tbody>
</table>
1 Introduction

Exciting current and proposed applications in humanitarian and military convoy automation and protection [58], disaster response [55], wilderness [23] and urban [31] search and rescue, unmanned surveillance [14, 24, 30] and tactical communications [35] envision large heterogeneous teams of hundreds to thousands of agents coordinating to accomplish joint tasks in complex environments.

In these domains, each agent has access to its own local computation and imperfect sensing, but does not know what other agents are computing or sensing. Agents are faced with the challenge of choosing sequences of actions to take based on this incomplete and uncertain knowledge of the environment, their teammates, and the outcomes of their own actions. To further compound the problem, the actions chosen by other agents can influence the outcome of an agent’s own actions, or the state of the environment. The success of the team is measured by its ability to achieve the high-level tasks of the application, not by the individual successes and failures of individual agents. Thus, agents must choose their sequences of actions in a coordinated manner in order to be effective.

Multiagent sequential decision-making (MSDM) is a class of multiagent problems that addresses this task of generating policies or strategies to select sequences of actions in a team of agents [12]. Specifically, in the taxonomy proposed by [44], our problem can be described as that of multiagent, fully-cooperative, sequential decision making. Given a number of independent mechanisms with partial knowledge performing actions in an environment, sharing a single measure of task success or utility as a result of their actions, we want each to select a sequence from a set of possible actions that, combined with the actions of other agents, yields some desired outcome.

Unfortunately, such MSDM problems have demonstrated themselves to be notoriously difficult to solve. Two of the most common and general problem formulations, Dec-POMDPs and Dec-MDPs, have been shown to have NEXP complexity with even two agents [9], and methods which search the full joint state-action space are typically unable to handle more than a few agents [10, 42, 55]. Because the full problem so quickly grows intractable, many approaches to solving MSDM problems focus on exploiting some specific structure in the underlying problem that can greatly reduce the state space that must be searched.

One particular structural property which has recently been exploited in various forms is that of sparsity of interaction. In full Dec-POMDPs and Dec-MDPs, it is assumed that the state, actions and observations of the agents can be interrelated in any way. While in the worst case, this may be true, it is often the case that in realistic problem settings, each agent may only affect and be affected by a small subset of agents at any time [60], or only be able to affect other agents in very specific ways [24]. It is therefore often possible to partially or wholly factor the full problem into regions where agents can act independently, and regions where their states, actions, or observations are somehow related, and solve this reduced problem. This can vastly improve the efficiency of finding a solution.

Various representations such as I-MDP [36], TD-POMDP [69], ND-POMDP [40], Factored MDPs [43], Transition-Independent Dec-MDPs [7], Event-Driven Dec-MDPs [6, 34] and EDICRs [57] take advantage of this type of problem structure, either by constructing problem models that enforce strict independence of the observation, reward, or transition functions, or by explic-
itly enumerating interactions and choosing problems where such enumerations remain compact. However, these approaches are typically either restrictive in their models or retain one or more exponentially scaling algorithmic steps that limit practical use to teams of fewer than 10 agents.

In our work, an alternate view is taken. A distributed planning framework is constructed with the constraint of remaining tractable in problems of at least hundreds of agents while fully representing a wide class of Dec-POMDP problems and ensuring linear scaling in complexity with team size. With scalability, tractability, and model generality held as fixed design requirements, various coordination and planning strategies are employed to determine what levels of performance are possible given these constraints.

Several key assumptions are made about the structure of the problems. As in previous work, agents actions are assumed to be selected independently, at each agent, and all potential interactions are assumed to be known and codified before planning occurs. In order to enable further problem-solving efficiency, our work introduces two additional assumptions. First, computational resources are assumed to be distributed evenly among all agents in the team. This is reasonable for the domains of interest, as agents are typically embodied in physical systems that contain significant computing payloads. Furthermore, while many potential interactions may exist in the problems, relatively few actually occur in any given evaluation of the problem. Taking these properties into account, the SI-Dec-POMDP formulation and the DIMS planning framework are introduced as a method of encoding and a solution framework that preserve these salient features and allow the construction of scalable solvers for these problems.

Sparsely-interacting decentralized partially-observable Markov decision processes (SI-Dec-POMDPs) are a subclass of the popular Decentralized POMDP (Dec-POMDP) formulation that explicitly factors out interactions between agents. It is based on both the Event-Driven Interaction with Complex Rewards (EDI-CR) model introduced by [37], and the Distributed POMDP with Coordination Locales (DPCL) model presented in [65], but extends these models in several key ways. First, unlike EDI-CR, it adds partial-observability to the model, moving from an underlying MDP to POMDP. Second, it allows the specification of observation dependencies between agents. Finally, it more specifically relates time to dependencies, by explicitly representing temporal relationships in dependencies rather than simply considering order of action execution. This is particularly useful when interactions are closely coupled or involve simultaneity, for example, if two agents can collide with one another.

The Distributed Iterative Reward-Shaping (DIMS) planning framework is a 4-stage iterative planning methodology adapted from common themes in the multiagent planning and learning literature[19, 44]. In it, agents iteratively solve an SI-Dec-POMDP planning problem by repeating a set of algorithmic steps. First, the problem is decomposed into smaller subproblems which are allocated among the agents. Next, the agents attempt to predict the actions of their teammates, and independently solve for the best local policies given these actions. Agents then exchange messages about these local policies to determine if and when conflicts arise, and use this information to improve their models of the actions of their teammates, at which point the process begins again.

This thesis will focus on two particular domains: the problems of planning for urban search and rescue, and planning for a humanitarian convoy. These domains are selected for their rich variety of agent interaction and access to existing large-scale high-fidelity simulation environments.
Autonomous agents have long been considered for dangerous jobs in urban disaster sites [17, 31, 53]. When urban disasters occur, rescuers are faced with searching potentially unstable, partially known environments as quickly as possible. Robots and software agents can perform a number of useful roles in these situations, providing situation awareness, victim search, and structure stabilization, all reducing or eliminating risk to human life. Delays in locating and extricating survivors can seriously impact their survivability, thus it is of great value for such agents to be able to coordinate their responses.

The second domain considered in this work is that of planning for a convoy. One of the more common logistical problems, this entails safely and reliably moving people and/or objects from one place to another. Planning for convoys of vehicles requires allocating limited vehicular resources to the movement of particular goods under uncertainty, then determining the best courses of action to maximize the quantity and utility of delivered goods despite possible delays and losses. A number of heterogeneous vehicles are usually available with varying specializations and tasks, but resources such as fuel and roads are shared, and unknown variables such as road and vehicle condition can hinder delivery of goods.

Using the combination of the SI-Dec-POMDP formulation and DIMS framework, this thesis will explore the performance and scalability characteristics of simple, iterative reasoning strategies on large, complex decision problems in these two domains. Preliminary work to date has applied the DIMS framework to two simpler subproblems of SI-Dec-POMDP, discrete multiagent path planning and Distributed POMDPs with Coordination Locales (DPCLs).

In the former case, with agents planning simultaneous paths over a grid structure, the result is Distributed Prioritized Planning (DPP), a simple variant of the sequential Prioritized Planning algorithm [20] that allows the exploration of key properties of the DIMS framework in a simplified environment. Results with DPP demonstrate that iterative planning in situations where interaction is sparse can produce efficient solutions in relatively few iterations with respect to team size. However, they also emphasize the importance of low variance in individual agent planning times in allowing distributed, iterative planning to be more effective than sequential, decoupled planning.

The latter subproblem, DPCL, is addressed by the more powerful D-TREMOR algorithm, a DIMS extension to the centralized, iterative TREMOR algorithm [65]. D-TREMOR significantly scales the TREMOR algorithm by replacing joint search and evaluation steps with fully-distributed heuristic approximations. Performance is demonstrated in solutions of DPCLs with over 100 agents in a simplified rescue domain. The results show the efficacy of prioritization and randomization in adjusting models of teammates’ actions for the interactions modeled in the rescue domain, but suggest that additional work is necessary to further improve performance and generalize D-TREMOR to other potential types of agent interactions.

1.1 Thesis statement

Agents in a large team with known sparse interactions can find computationally efficient high-quality solutions to planning problems through an iterative process of estimating the actions of teammates, locally planning based on these estimates, and refining their estimates by exchanging coordination messages.
2 Background

The problem of planning for multiple agents has been widely studied in the AI community. At its core lies the general problem of coming up with a set of actions for a set of agents that maximizes some reward or utility. However, there are some additional characteristics of the problems addressed in this work that narrow down the relevant related work. Here, the focus will be on approaches to solving problems at planning time involving multiple agents making sequences of decisions, with a specific emphasis on those techniques that exploit structural properties of the domain to improve computational efficiency.

Within this theme, the literature can be roughly categorized into three areas: path-planning, decision-theoretic approaches, and game-theoretic approaches. Path-planning approaches are heavily search-based, involving the direct exploration of joint state space to find a feasible solution. These are primarily used in simpler problems in which actions have certain outcomes and state is fully observable. Decision-theoretic strategies use optimization or learning methods to attempt to find local minima in agents’ belief space. Finally, game-theoretic strategies attempt to pose the multiagent decision-making problem as an extensive-form game being played between two or more agents, in which agents are “playing” sequences of actions for which the game reward is the result of a joint execution, of which they have only partial or imperfect knowledge. Once in this form, a variety of techniques can be used to search for interesting equilibria, which often correspond to local solution optima.

Furthermore, while execution of policies is by definition distributed, the planning approaches are further divisible into centralized and decoupled strategies. In centralized approaches, a single planning process attempts to determine the necessary policies for all agents. Note that while this process can contain subtasks that are distributable, it belongs to the class of centralized techniques as long as some part of the process requires a single, monolithic computation. In decoupled approaches, multiple processes are independently responsible for determining the necessary actions for subsets of individual agents, coordinating to avoid interference with one another.

2.1 Path-planning

One of the more widely studied subfields of the general multiagent planning problem is the case where multiple robots share a workspace and must avoid interfering with each other while potentially maintaining constraints such as convoying or formation-keeping. This is commonly referred to as multiagent path planning. In these problems, agents are typically assumed to have perfect certainty in their actions and the state of the environment, a reasonable assumption when agent perception and localization is very accurate, or replanning can be done relatively quickly if actions do lead to unexpected outcomes. This is the case in numerous domains such as planetary exploration, agriculture and industry, and warehouse management [45].

The problem of multiagent path planning has been extensively studied for a number of years. For a detailed summary of the literature, we refer readers to [45]. Centralized planners (e.g. [5, 46, 54, 59]) combine the DOFs of each robot into a single high-dimensional composite robot and plan in this joint space. These approaches can theoretically find optimal solutions for multirobot planning problems, but are restrictive in the number of robots for which they can plan, as the complexity of planning grows exponentially with number of robots. Thus, while they provide
the highest-quality solutions overall, they are generally intractable for large teams.

Two common decoupled approaches, on the other hand, are *path coordination* and *prioritized planning*. The former decomposes the planning problem into a spatial path planning problem and a temporal velocity planning problem [27]. Robots first plan paths in space that avoid static obstacles, but ignore dynamic constraints such as other robots. Then, coupled [3, 25, 41, 47], prioritized [1, 61], or reactive [56] methods are used on the simpler problem of solving for velocity profiles that allow the robots to avoid collisions.

Prioritized planning, introduced in [20], is another, often complementary approach to path coordination. In it, trajectories are generated sequentially according to some prioritization of the robots. Higher priority robots form obstacles in space-time for lower priority robots. This greedy and incomplete strategy is often effective in practice [21, 68]. Searching or optimizing over the prioritization can yield improved solutions [4, 8, 13], but often, simple heuristics perform quite well [61]. However, a major problem with prioritized planning is that paths must be computed in sequence, which scales poorly to large numbers of robots.

Several works have tried to address this by identifying simplified graph structures online for which prioritized planning can be done. In [62], robots are divided into small coupled cliques and plan to execute in the order of their inter-clique dependencies, with each group waiting until the last finishes. This allows solutions to be found in certain highly constrained settings, but can lead to long execution times as cliques must execute in sequence rather than simultaneously. In [50], the map itself is partitioned into subgraphs, simplifying the space over which prioritized planning takes place. In [16], dynamic networks are formed as robots move through the environment. When new members are added, trajectories are exchanged and checked for collisions. Each robot then executes a prioritized replan on its own probabilistic roadmap [28], with the best plan used by the entire group. In [15], robots locally sense conflicts, and use a prioritization scheme to determine whether to add the corresponding dynamic obstacles to their maps. The DPP algorithm described here in the preliminary work is similar to this latter technique, however it makes no assumptions between connectivity and spatial locality. Instead, it attempts to resolve all conflicts at planning time rather than during execution; it also addresses larger team sizes.

### 2.2 Decision-theoretic planning

Much existing work in the multiagent planning community is built around the decentralized Markov decision process (Dec-MDP) and partially-observable Markov decision process (Dec-POMDP) [9]. These elegant yet general models can represent a wide array of constraints, uncertainty, and dependencies in a discrete environment, but in their full form, they are exceedingly difficult to solve. Even with only two agents, the complexity of Dec-MDPs and Dec-POMDPs has been shown to be NEXP-complete [9]. Complete joint solvers for these problems, e.g. [10, 42, 55], are extremely limited in the sizes of the problems they can handle. Therefore, a number of recent approaches have focused on exploiting structure in various problem types in order to more efficiently find optimal and approximate solutions.
2.2.1 Dec-MDPs

In Dec-MDPs, the state space of the problem is factored into global and local state. Agents are assumed to have full observability of their local state and to collectively have full observability of the global state.

The transition-independent Dec-MDP (TI-Dec-MDP), proposed by [7], was one of the first forays into the exploitation of structure for efficient solving of decentralized MDPs. In this model, agents’ transitions and observations were solely dependent on local states, local actions and shared external features, properties which were defined as transition and observation independence, respectively. However, agents shared a partially-joint reward structure. Specifically, the reward function was split into two components, a set of additive, independent local reward functions, and a joint reward function with additional rewards based on combined actions. These joint rewards were based on events, tuples indicating that an agent had performed some particular activity. The Event-Driven Interaction Dec-MDP (EDI-Dec-MDP) model [6], which followed shortly thereafter, assumed instead that agents had independent, additive, reward functions, but partially dependent transition functions. In this case, agents’ transition functions were factored into subfunctions based on whether events had occurred or not.

The Event-Driven Interaction with Complex Rewards Dec-MDP (EDI-CR) [37] combined attributes from the TI-Dec-MDP and EDI-Dec-MDP, namely, the partial transition dependence of the EDI model and the complex reward interactions of the TI-Dec-MDP model, to form a more general representation than either parent model. In [37], a bilinear program formulation for EDI-CRs was constructed and demonstrated to solve larger problem instances than had previously been demonstrated for TI- or EDI-Dec-MDPs.

OC-Dec-MDPs [11] specifically dealt with problems involving a set of tasks with constraints in ordering, precedence, and resources. Here, the problem was formulated as a set of local MDPs, which were then solved using an iterative process of policy improvement. This refinement process allows the consideration of problems with hundreds of tasks and 15 agents within this fairly restrictive subclass.

Roth et al [49] exploited limited agent interaction to speed up centralized policy generation by determining when agents could ignore other agents. This was done within the model proposed in [48], in which Dec-MDPs had context specific independence and used execution-time communication to coordinate actions.

Spaan et al [57] proposed the I-MDP model, a Dec-MDP model in which agents only interact in a subset of states. Outside of these specified states, agents operate completely independently, while within the states, agents are assumed to communicate explicitly at execution time to coordinate their actions. This model, and the execution-time communication model in [48] were the basis for [36], in which a reinforcement learning approach was applied to allow agents to dynamically fit an I-MDP model that minimized necessary coordination. One interesting characteristic of this model is that agents were not assumed to share a reward function, allowing the possibility of modeling non-cooperative agent interactions.
2.2.2 Dec-POMDPs

In Dec-POMDPs, problems incorporate partial observability of both local and global state. This further increases the difficulty of solving multiagent problems, but also allows for a richer representation of uncertainty and agent interactions.

JESP [38], attempted to find solutions to Dec-POMDPs by solving individual single-agent POMDPs and then updating these individual models to reflect interactions between agents. However, the method of updating these models introduced an exponential increase in state space, severely limiting its scalability to more than a few agents.

In [40], a different sort of structure was exploited, a human-specified abstract policy, which allowed domain-specific knowledge to be exploited to accelerate Dec-POMDP solution.

In network-distributable-POMDPs (ND-POMDPs) [34, 39], transition independence was also assumed, but additionally, interactions between agents were modeled as a stationary graph. Factored Dec-POMDPs, as proposed by [43], presented a more general model, in which transitions, rewards, and observations could be factored into composable functions over subsets of agents. However, to construct this, agents were restricted to only interact with relatively few other agents. Thus, in essence, this extended the ND-POMDP model to allow the interaction graph to vary over decision epochs.

Transition decoupled POMDPs (TD-POMDPs) [69], allow for transition interaction of multiple agents through the use of non-local state features that can be modified by agents. Thus, agents can decouple interactions and plan joint behavior using abstractions of policies rather than the policies themselves. This allows reasoning over a far more compact influence space, allowing optimal solution of problems of several agents.

2.3 Game-theoretic planning

Another popular formulation of the MSDM problem is that of an \( n \)-player stochastic extensive-form game. In this formulation, interactions are explicitly represented as a sequence of actions with various probabilistic distributions of payoffs. Agents attempt to choose game actions that will maximize their payoff, corresponding to making plans that either do or do not elicit certain interaction results. These payoffs correspond to the various factorizations of the joint utility, or simply the joint utility itself, in the case of identical interest games. Interesting local optima in the decision-theoretic formulations often correspond to correlated equilibria in the corresponding extensive-form games (EFGs).

MAIDs [33], Succinct EFGs [18], and Temporal ACGs [26] are all various methods for compactly representing EFGs of MSDM problems. In fact, in [37], a method was presented for converting problems from the decision-theoretic EDI-CR to a stochastic game in both canonical EFG and MAID form. Once in this form, a variety of techniques exist for finding various correlated equilibria between agents.

In [43], factored Dec-MDPs were created by decomposing Dec-POMDPs in this way, converting each decision epoch into a collaborative Bayesian game between agents. From this, efficient methods for finding the Bayes-Nash equilibria can be applied, which [43] claims corresponds to local optima in the original Dec-MDP. In [37], once in stochastic game representation, an existing bilinear program formulation is adapted to take advantage of the “loosely-coupled”
structure of the problem, yielding an efficient solution.

Overall, the major limitations of prior works lie in scalability. Most algorithms suffer from exponential increases in complexity as the number of agents is increased, while the ones that do not (such as decoupled path planning approaches, TI-Dec-MDP and ND-POMDP), do so by categorically eliminating important problem characteristics such as action uncertainty or transition dependence.

3 Approach

In this section, the problem formulation, solution approach, and domains of interest are described in detail. The Sparsely-Interacting Decentralized POMDP (SI-Dec-POMDP) is introduced as a subclass of Dec-POMDP that explicitly factors agent interactions into independent and partially joint functions. The Distributed Iterative Model Shaping (DIMS) framework is laid out as an algorithmic framework from which to construct planners that can efficiently solve SI-Dec-POMDPs. Finally, the models that will be used to represent the urban search and rescue and convoy planning domains are described in detail.

In the domains of interest, there are several constraints to which it is necessary to adhere. There may be uncertainty in the results of agent actions. Agents will only have access to some partition of the full state space, and this portion of the state may additionally be only partially observable. Most importantly, most of the time, changes in agents’ local and environmental state will be describable by a factored model that is independent of the state, actions, and observations of other agents. We begin by reviewing the Dec-POMDP, a problem representation capable of handling this model complexity and uncertainty.

3.1 Dec-POMDPs

The decentralized partially-observable Markov decision process, or Dec-POMDP, is a general and widely-used problem formulation for multiagent planning under both action and observational uncertainty. It is also one of the most general, capable of expressing a rich variety of constraints, interactions, and dependencies between agents and the environment. Adapting the nomenclature from [2, 43], a Dec-POMDP problem can be described as a tuple \( \langle \mathcal{A}_g, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{O}, b_0, h \rangle \), where:

- \( \mathcal{A}_g = \{1, \ldots, n\} \) is the set of agents.
- \( \mathcal{S} \) is a finite set of states.
- \( \mathcal{A} = \times_i \mathcal{A}_i \) is the set of joint actions, where \( \mathcal{A}_i \) is the set of actions available to agent \( i \).
- \( \mathcal{O} = \times_i \mathcal{O}_i \) is the set of joint observations, where \( \mathcal{O}_i \) is the set of observations available to agent \( i \).
- \( \mathcal{P} := \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to (0, 1) \) is the transition function.
- \( \mathcal{R} := \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R} \) is the reward function.
- \( \mathcal{O} := \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{O} \to (0, 1) \) is the observation function.
$b_0 \in \mathcal{P}(\mathcal{S})$ is the initial belief distribution, where $\mathcal{P}(\mathcal{S})$ denotes the set of probability distributions over the finite set $\mathcal{S}$.

$h$ is the time horizon of the problem.

A Dec-POMDP describes a situation in which agents are placed in an environment with some initial state drawn from distribution $b_0$. At each time step $t \in 0, \ldots, h - 1$, each agent chooses an action. Depending on the joint action taken over all agents, the state of the world is changed probabilistically according to the transition function $\mathcal{P}$. In addition, each agent receives an observation sampled from $\Omega$ according to the observation function $\mathcal{O}$. Finally, the agents as a whole also receive a reward at each stage based on their current state, choice of action, and resulting next state, as defined in $\mathcal{R}$.

The objective of agents in a Dec-POMDP problem is to find a sequence of joint actions that maximizes expected reward. Thus, the output of a Dec-POMDP planner is a joint policy $\Pi = \langle \pi_1, \ldots, \pi_n \rangle$, where each policy $\pi_i$ provides a mapping from an agent’s action-observation history to an individual action for the next stage.

The objective of a Dec-POMDP planner is then to find a joint policy $\Pi$ that maximizes expected joint reward for the agents over all stages, given the uncertainty in the effects of agents’ actions and the state of the agents and environment.

Note that in the case of problems without uncertainty or partial-observability, such as Dec-MDPs or graph planning problems, the $\mathcal{P}$ and $\mathcal{O}$ terms can be simplified to identity matrices to represent the problem.

### 3.2 Sparsely Interacting Dec-POMDPs (SI-Dec-POMDPs)

In the case of sparsely interacting agents in teams, the problem can further be factored into the Sparsely Interacting Dec-POMDP (SI-Dec-POMDP). This form is a generalization of the DPCL [65] and EDI-CR [37] subclasses of Dec-POMDP. SI-Dec-POMDP explicitly models sparse interaction between agents through the factorization of the $\mathcal{R}$, $\mathcal{P}$ and $\mathcal{O}$ functions into independent and interacting subfunctions.

There are two key differences between the SI-Dec-POMDP model and the canonical Dec-POMDP model. First, in the SI-Dec-POMDP model, the state space is assumed to be factorable into global states and local states. Global states represent tasks and environmental properties, while local states are specific to particular agents. Thus, $\mathcal{S} := S_g \times S_1 \times \ldots \times S_N$ where $S_n$ is a set of local states of agent $n$ for $1 \leq n \leq N$ and $S_g$ is a set of tasks and global properties representing the state of the environment.

Second, SI-Dec-POMDP represents interactions between agents sparsely by factoring the $\mathcal{P}$, $\mathcal{R}$, and $\mathcal{O}$ functions into independent functions over the majority of state-action space and joint functions defined at specific points in the joint state-action space where agents’ behavior depends on more than just their local state and the global state. This corresponds to limited interactions between tuples of robots that differ from the behavior of the factored, independent functions.

In an SI-Dec-POMDP, a set of functions $\mathcal{P}_{ind}$, $\mathcal{R}_{ind}$, and $\mathcal{O}_{ind}$ is defined, where each function is decomposable into independent functions per agent, i.e. $\mathcal{P}_{ind} = \times_i \mathcal{P}_{i,ind}$. These independent functions define the effects of agent actions in regions of space and time where they do not
interact, such as when two agents are moving around in distant regions of a world. In these regions, agents can reason about the effects of their actions without considering their teammates. When a region of space and time is encountered where the actions of one agent do affect another agent, an interaction is defined. An interaction $I$ over a set of $m$ agents is formally defined as the tuple of

$$\langle s^I_g, \{s^I_i\}_1^m, \{a^I_i\}_1^m, P^I, R^I, O^I \rangle$$

where $s^I_g$ is the global state when the interaction occurs, $s^I_i$ is the local state of agent $i$, $a^I_i$ is the action taken by agent $i$ and $P^I, R^I, O^I$ are $m$-dimensional joint transition, reward, and observation functions. We assume that the set of interactions is given and available to each agent. We assume that these interactions are defined such that each agent can only be involved in one at any time.

From this, the joint $P, R,$ and $O$ functions of the problem are constructed as piecewise functions split over the interactions. For example, if there is a single two-agent interaction $I$, the $P$ function can be written as:

$$P = \begin{cases} 
P^I \times P^I_{\text{ind}} \times \ldots \times P^I_{\text{ind}} & \text{if } \langle s_g, s_1, s_2, a_1, a_2 \rangle = \langle s^I_g, s^I_1, s^I_2, a^I_1, a^I_2 \rangle; \\
P^I_{\text{ind}} \times P^I \times P^I_{\text{ind}} \times \ldots \times P^I_{\text{ind}} & \text{if } \langle s_g, s_1, s_2, a_1, a_3 \rangle = \langle s^I_g, s^I_1, s^I_2, a^I_1, a^I_3 \rangle; \\
\ldots & \text{otherwise.}
\end{cases}$$

### 3.3 Distributed Iterative Model-Shaping (DIMS)

In order to be able to solve SI-Dec-POMDPs for large teams, it is necessary to cater to several properties of the problems of interest. Firstly, and most significantly, it is necessary to decompose the problem into independent chunks that scale at worst linearly with the number of agents. Next, agents may not be able to exchange their local models easily, so any approach must be able to allow coordination between agents in a way that is model-agnostic. Furthermore, although there is sparse interaction among agents, in these joint regions, there must be some way to synchronize actions across agents. Finally, it will often be the case that particular problems will lack some model complexities or retain additional structure that can be exploited to plan more efficiently. It is desirable for a technical solution to be able to make use of these problem specifics without having to reformulate the problem around them.

Thus, a modular solver framework is constructed based on common decentralization strategies in the body of multiagent planning and learning literature. This framework is architected to allow the substitution of domain-specific algorithm components for improved efficiency on particular problems, while still retaining the ability to generally solve the remaining attributes of the problem. This allows the analysis and reuse of component planning strategies across a range of different problem domains that vary widely in complexity, while retaining behavioral characteristics of the algorithm.

Durfee [19] reviews the literature on distributed (decoupled) planning for multiagent problems. In summarizing the key aspects of planning under iterative plan-formation approaches, the following general steps are defined:
Table 1: Comparison of DIMS and decompositions of distributed problem solving in literature

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Role Allocation</td>
<td>Goal assignment</td>
<td>Credit assignment¹</td>
</tr>
<tr>
<td>Independent Planning</td>
<td>Local planning</td>
<td>Learning</td>
</tr>
<tr>
<td>Interaction Detection</td>
<td>Plan Exchange</td>
<td>Modeling of other agents</td>
</tr>
<tr>
<td>Model Shaping</td>
<td>Adding constraints</td>
<td>Dynamics of learning</td>
</tr>
</tbody>
</table>

1. Goal assignment: Assigning specific goals to agents.
2. Local planning: Agents formulating their local plans.
3. Plan Exchange: Local plans being exchanged and combined between agents.
4. Adding constraints: Imposing messaging and/or time commitments to resolve negative plan interactions.

Panait and Luke propose a similar generalization of the problem of “concurrent learning” [44], which describes a multiagent learning problem in which multiple distributed learners must simultaneously try to find local optima. In organizing the literature, the concurrent learning problem is divided into three sub-problems:

1. Credit assignment: Assigning joint reward to individual agents¹
2. Dynamics of learning: Shaping agents’ coevolution of policies.
3. Modeling of other agents: Creating models for each agent of what other agents are and will be doing.

The consistency of these decompositions and the wide range of approaches that fall under them suggest a particular organization that is effective for the problem of distributed problem solving among teams of agents that must plan with only partial shared knowledge. Thus, the Distributed Iterative Model Shaping framework is introduced. DIMS presents a particular approach to solving SI-Dec-POMDPs that exploits the near-decomposability described in Section 3.2 to scalably find approximate solutions.

In DIMS, the SI-Dec-POMDP is distributed into subproblems for each agent that contain estimations of other agents’ actions to determine what will happen in joint interactions, while using local models to describe the world outside of these interactions. Once local plans are determined, agents exchange some coordination information to allow other agents to estimate what the rest of the team is doing, then replan to get a better result.

More concretely, the process can be divided into the following four stages: Role allocation, Independent Planning, Interaction Detection, Model Shaping. By iterating over these steps, it is possible to form joint plans that account for agent interactions while still scaling only linearly with number of agents and distributing computational load. Table 1 overviews the relationship between these steps and the decompositions described in the literature.

¹While credit assignment typically deals with the division of rewards received through joint actions among agents, role-allocation can be loosely considered a form of a-priori assignment of rewards to individual agents for the completion of specific tasks.
3.3.1 Role Allocation

In the first step, roles (i.e., a set of tasks that an agent must accomplish) are allocated to each agent. The purpose of this step is to reduce the size of each agent’s search space when possible by partitioning relevant environmental state into independent problem models for each agent. In many cases, a simple role allocation step can focus each agent’s search significantly. For example, if there are several robots assigned to rescue victims in an environment, assigning victims to robots using a simple task-allocation algorithm allows them to reduce their state to only include assigned victims, which can greatly reduce their planning time. In cases where there are no explicit roles to allocate, this step can be used to simply eliminate state that is not pertinent to a particular agent. From the original Dec-POMDP problem, we create $N$ augmented independent POMDP models of the form:

$$\langle S_n^g \times \mathbb{X}_n, \Omega_n, \mathcal{P}_n', \mathcal{R}_n', \mathcal{O}_n', b_n \rangle$$

In these models, we define the following additional parameters:

- $S_n^g$ is the relevant part of global state space to agent $n$, as determined by role allocation.
- $\mathcal{P}_n'$ is an augmented independent transition function.
- $\mathcal{R}_n'$ is an augmented independent reward function.
- $\mathcal{O}_n'$ is an augmented independent observation function.
- $b_n$ is the marginal initial belief distribution for agent $n$.

The augmented functions $\mathcal{P}_n'$, $\mathcal{R}_n'$, and $\mathcal{O}_n'$ take two additional parameters for each interaction $I$: the probability of occurrence, $p_I$, and the expected marginal change in joint reward to the team, $R_{ext}^I$, if the interaction does occur. These terms are then integrated into the independent model using model-shaping:

$$\begin{align*}
\mathcal{P}_n' &= \begin{cases} 
\mathcal{P}_n^{ind} & \text{if } s_1 \neq s_1^I; \\
 p_1^I \cdot \mathcal{P}_n^{ind} + (1 - p_1^I) \cdot \mathcal{P}_n' & \text{if } s_1 = s_1^I;
\end{cases} \\
\mathcal{R}_n' &= \begin{cases} 
\mathcal{R}_n^{ind} & \text{if } s_1 \neq s_1^I; \\
 p_1^I \cdot (\mathcal{R}_n^{ind} + \mathcal{R}_{ext}^I) + (1 - p_1^I) \cdot \mathcal{R}_n^{ind} & \text{if } s_1 = s_1^I;
\end{cases} \\
\mathcal{O}_n' &= \begin{cases} 
\mathcal{O}_n^{ind} & \text{if } s_1 \neq s_1^I; \\
 p_1^I \cdot \mathcal{O}_n^{ind} + (1 - p_1^I) \cdot \mathcal{O}_n' & \text{if } s_1 = s_1^I;
\end{cases}
\end{align*}$$

Intuitively, this shaping corrects the independent functions by adding an expectation over the states and actions of the remainder of the team. The estimation of $p_I$ and $R_{ext}^I$ takes place in the interaction exchange step of DIMS.
3.3.2 Independent Planning

Once the problem is decomposed into nearly-independent planning problems, it is possible to distribute the planning computation across agents. Each agent is given its factored model to solve, using any efficient POMDP, MDP, or graph-search solver, depending on the level of uncertainty in the model. This is particularly useful as it allows the incorporation of existing state-of-the-art solvers that match the complexity of the underlying problem. Thus, while SI-Dec-POMDPs are expressive enough to represent partially-observable, uncertain actions, in cases where actions are certain or state is observable, a more efficient local planner can be used. For example, when dealing with problems that are fully-observable, an MDP solver can be applied to the factored models instead of a POMDP solver. DIMS assumes that such a solver can take an independent model and return an optimal or near-optimal policy. Thus, we can compute a set of local policies \( \Pi = \{ \pi_1, \pi_2, \ldots, \pi_n \} \) that are optimal for the augmented independent models.

3.3.3 Interaction Exchange

Once independent models are solved, agents must discover potential interactions. Agents do this by self-evaluating reward and probability of being involved in each of the interactions. This information is then exchanged between agents as compact messages to correct each others’ model functions. As is shown in later sections, this turns out to be an especially important step. Even in situations where the results of joint interaction are relatively well known (i.e. collisions), the actual detection of such interactions occurring in a large team is non-trivial. In addition, agents’ evaluations of self-reward and probability of involvement must be computed carefully. Because agents are simultaneously replanning, simply reporting current estimates of these values can lead to undesirably dynamic behavior such as oscillation and overly myopic search. It is therefore necessary to take additional steps to ensure that the process of exchanging of this information is dynamically stable in itself.

3.3.4 Model Shaping

Once messages are exchanged, the augmented parts of the independent models are rewritten using the updated information about interaction values and probabilities. Social model shaping is used to model the effects of joint interactions as it provides a widely used mechanism for propagating these changes in a way that will result in local planners preferring jointly optimal strategies. This completes a single iteration of the DIMS planner. At this point, the team can choose either to reallocate roles or retry independent planning to improve their policies.

The DIMS planning framework offers near-linear computational complexity with team size and allows representation of most types of sparse inter-agent interactions. However, its iterative nature makes it highly sensitive to the nature of the model shaping and precise order and exchange of messages.
3.4 Problem Domains

3.4.1 Urban Search and Rescue

In this work, the problem of urban search and rescue is modeled by agents moving and executing actions on a graph that represents the topology of a hypothetical disaster site. In this problem, a team of heterogeneous robots needs 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.

An example of a disaster site can be seen in Figure 2. This world is represented as a 4-connected lattice of fixed size, where each vertex in the lattice represents a location with some properties. The locations can be unsafe, which means that agents will incur a penalty upon traversal, or they may be narrow, in which case only a single agent can be at the location at a given time. In addition, they may contain debris, which can block agents that are not equipped to deal with that, or they may contain one of more victims, which are worth a high reward if rescued. A summary of these attributes and their graphical representations can be seen in Figure 1.

![Figure 1: Descriptions of the symbols used in the simple rescue domain.](image)

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. However, the simplification of modeling collisions and unsafe cells as negative rewards means that when these rewards are sufficiently large enough to impact policies, it is sometimes possible for policies that avoid risk to achieve higher values than policies that successfully rescue many victims, leading to unintuitive rankings of solutions.
3.4.2 Convoy Planning

In this work, convoy planning is modeled as a discrete decision making problem over a topological graph of locations. Resources start at particular locations, and are needed at other locations. Delivery of resources yields positive reward, but each location in the graph has partially-observable state that may limit movement and incur negative penalties.

Three classes of agent are available, transport agents, escort agents, and observer agents. Transport agents are able to move goods from one location to another, but are susceptible to delays and penalties as they move across the graph. Escort agents make locations safer for transport agents by moving through them, but cannot attain positive reward on their own. Observer agents can make the state of locations completely observable to transport or escort agents if they are at the same location, but once again cannot attain positive reward on their own.

4 Preliminary Work

In previous work, we study two subproblems of the general SI-Dec-POMDP formulation, multi-agent path planning and Distributed POMDPs with Coordination Locales (DPCLs). The reasons for starting with these subproblems are severalfold. First, as they model only restricted pieces of the full problem, they are simpler to understand, characterize, and compute. Second, the application of the DIMS framework to these problems results in algorithms that are closely related to existing approaches in the literature, so performance and behavior is comparable to some extent with the related work. In addition, by applying the DIMS framework to subproblems, the generality and efficacy of the framework can be tested in cases when additional problem structure arises.

In the case of multiagent path planning on a discrete graph, this simple problem allows us to test the specifics of exchanging messages and detecting conflicts without the computational cost of solving a full POMDP problem. Dec-POMDPs with Coordination Locales (DPCLs) are...
a more specific subclass of Dec-POMDP than an SI-Dec-POMDP. In DPCLs, interactions are restricted to affecting reward and transition functions. This provides a rich test environment for exploring the intricacies of dealing with agent interactions under uncertainty. The performance of DIMS-like solver algorithms is assessed for both path-planning and DPCL problems. Though the problems differ in scale, complexity, and uncertainty, there are similarities in the behavior and limitations of the applied algorithms.

4.1 Distributed Path Planning

We begin by looking at the problem of multiagent path planning on a graph. In this problem, a team of agents is given a graph of traversable states and some set of start and goal states, and must find collision-free paths through time and space between the two sets of states, taking into account varying traversal costs and graph connectivity. This can be thought of as a special case of the rescue domain, where there is no action or observational uncertainty and role-allocation has been done a-priori, in an environment where every location is a narrow corridor and there is an infinite collision penalty (as shown in Figure 3).

The complexity of solving a full SI-Dec-POMDP can make it hard to assess the dynamics of message exchange, interaction detection, and model shaping at large scales. The simplicity of the multiagent path planning problem allows fast evaluation of analogous problems, in potentially larger teams, to study the dynamics of these steps.

Formally, a discrete multiagent path planning problem can be described as the tuple

\[ (A_g, M, C, S, G, h) \]

where we define the following:

- \( A_g = \{1, \ldots, n\} \) is the set of agents.
- \( M \) is the graph that must be traversed, where \( V_M \) is the set of vertices in \( M \) and \( E_M \) is the set of unordered edges in \( M \).
- \( C := E_M \rightarrow \mathbb{R} \) is a function that maps edges to traversal costs.
- \( S = \{s_1, \ldots, s_n\} \) is the set of starting vertices, where \( s_i \) is the starting vertex for agent \( i \).
- \( G = \{g_1, \ldots, g_n\} \) is the set of goal vertices, where \( g_i \) is the starting vertex for agent \( i \).
- \( h \) is the time horizon of the problem.

The objective in the problem is to find a set of collision-free solution paths \( \Pi = \{\pi_1, \ldots, \pi_n\} \) that minimizes cost. Paths are defined as a sequence of neighboring vertices in \( M \), that is, \( \pi = \{v_0, \ldots, v_h\} \) such that \( v_i \in V_M \) and for every consecutive pair \( v_i, v_{i+1} \), \( (v_i, v_{i+1}) \in E_M \). The cost of \( \Pi \) is \( \sum_{i \in A_g} \sum_{t=0}^{h} C(\pi_i^t) \). In order to be a solution path for some agent \( i \), it must be the case that \( \pi_i^0 = s_i \) and \( \pi_i^h = g_i \). In order to be collision-free, \( \Pi \) should not contain any two agents that traverse the same edge at the same time or remain at the same vertex at the same time. Thus for any \( t \) in \( 0 \leq t < h \), it must be the case that \( \forall i \neq j, \pi_i^t \neq \pi_j^t \), and either \( \pi_i^{t+1} \neq \pi_j^t \) or \( \pi_i^t \neq \pi_j^{t+1} \).

Multiagent path planning problems can be reduced to SI-Dec-POMDPs. First, since agents’ state is locally observation-independent, create an observation \( o_i \) for each vertex \( v_i \in V_M \) and assign the observation function \( O \) to be an appropriately sized identity matrix. Next, enumerate
outgoing edges from each vertex in the graph arbitrarily, and assign an action set where each action $a_j$ corresponds to the $j$-th outgoing edge. The transition function $P$ can be constructed as a binary matrix valued at 1 for $s_i, a_j, s_k$ when $a_j$ corresponds to the outgoing edge $(v_i, v_k)$ and 0 otherwise. Finally, interactions can be defined as the collisions which may occur between robots. The set of interactions can be enumerated as the sets $I_{col}$ and $I_{pass}$ where

$$I_{col} := \{ \langle s_i, s_j, a_i, a_j \rangle : \exists s_k \text{ s.t. } P(s_i, a_i, s_k) \cdot P(s_j, a_j, s_k) \geq 0 \}$$

$$I_{pass} := \{ \langle s_i, s_j, a_i, a_j \rangle : P(s_i, a_i, s_j) \cdot P(s_j, a_j, s_i) \geq 0 \}.$$ 

Intuitively these cases correspond to the two logical conditions mentioned above regarding collisions, that a pair of robots cannot be at the same vertex at the same time, and two robots cannot exchange vertices, which given a non-multiedge graph, would mean that the same edge was used by two different robots at the same time.

Given a multiagent path planning problem, one approach which has been shown effective for reasonably large teams is prioritized planning \[61\]. In this approach, given a team of robots and an obstacle map, a priority is assigned to each robot in the team according to some prioritization function. Then, a central process plans for the robots sequentially, in the order of their priorities. As each robot is assigned a path, that path is added as a dynamic obstacle in the map for subsequent robots to plan around. Assuming that independent planning for each robot is successful, a set of collision-free paths can be generated in exactly $n$ planning cycles. While this approach is successful at generating solutions, this means paths must be planned in order, resulting in a linear increase in planning time with the number of robots. Prioritized planning is also necessarily centralized, creating a potential computational bottleneck in teams where computational resources are distributed.

However, in many domains, the strict ordering of sequential planning is likely to be unnecessarily expensive. In most cases, not all robots need to avoid all other robots. Online prioritized approaches such as \[15\] and \[16\] take advantage of this property by determining sets of robots that need to be planned sequentially by detecting interactions via local observations.

### 4.1.1 Distributed Prioritized Planning (DPP)

In \[66\], we introduce Distributed Prioritized Planning (DPP), a DIMS-based approach that has each robot plan simultaneously, then look for collisions between paths that require lower priority robots to replan. The intuition behind this is that if robot paths are not dependent on all other paths (i.e. interactions between agents are sparse), the number of replanning iterations necessary should be much less than the number of agents, reducing overall planning time because no single, central process needs to plan for each robot in sequence.

The DPP algorithm can be described using the DIMS framework as follows:

**Role Allocation.** In the typical description of the path planning problem, agents are given specific start and goal locations. Thus, role allocation simply becomes the task of assigning agents to their given goals.

**Independent Planning.** In DPP, each agent runs an A* planner in configuration-time space to determine an optimal path on an time-augmented graph. This path is saved to be exchanged with other agents, and to be used as an initial guess for future planning steps. The amount
of time spent planning is also recorded in the first iteration for use as a prioritization over
the robots. Previous work [61] showed that a path-length heuristic performed well in many
environments, with the intuition that robots with shorter paths have time to move around
other robots. The best-first nature of the underlying A* planner provides a similar intuition
for using planning time. If the planner is taking a long time to search, it is because the most
direct paths for the robot cannot be taken.

**Interaction Detection.** For the path planning problem, there is only one type of interaction:
collision. This makes detection and message exchange much simpler, as agents share
the same penalty and preference for avoiding collision interactions. Thus, the interaction
exchange process proceeds as follows: if agents generate a new plan at any iteration, they
send this path to all members of the team. When an agent receives a path from a neighbor,
it first checks the priority of the sending agent. If the agent is lower priority, the path is
discarded. If the agent is higher priority, the receiving agent retains the path for model
shaping.

**Model Shaping.** Once again, the simple nature of the path planning problem makes this step
relatively simple. For each path that has been retained from the interaction detection step,
each agent modifies its map of CT-space to add an arbitrarily large negative reward for
colliding with another agent. Then, it returns to the independent planning step to plan a
new path around these new obstacles.

Distributing the planning in this way can be proven to converge to the same result as the cen-
tralized planner, if agents are forced to replan at every step (as shown in [66]). However, given
static prioritization, and assuming lossless communication, it is possible to add a simple opti-
mization to dramatically reduce the number of messages communicated. Each agent $i$ maintains
a dependency list $D_i$ and a path cache $\Pi_{\text{cache}}$. The list is initialized to contain all of the robots in
the team, and the cache is initially empty. In each iteration, when an agent $i$ receives a message
$m_j = \langle j, p_j, \pi_j \rangle$, it compares the priority to its own, $p_i$. If $p_j > p_i$, the agent will remove the
sender $j$ from its dependency list and add $\pi_j$ to $\Pi_{\text{cache}}$. If $p_j < p_i$, it is discarded, but $j$ remains
on the dependency list $D_i$. At the end of each iteration, if an agent has replanned, it sends its new
plan to all robots on its dependency list. The result of adding this optimization is Algorithm 1.
We call this variant Reduced DPP.

Table 2: Probabilistic cellular automaton used to generate path planning maps.

| Upper Cell | □ | □ | □ | □ | ■ | ■ | ■ | ■ |
| Left Cell  | □ | □ | □ | ■ | ■ | ■ | □ | □ |
| Corner Cell| □ | □ | □ | □ | □ | □ | □ | □ |
| $P_{\text{Obstacle}}$ | 0.1 | 0.0 | 0.2 | 0.3 | 0.2 | 0.3 | 0.4 | 0.6 |

**4.1.2 Experimental results using DPP**

Experiments were carried out in a 2D grid simulation on binary obstacle maps. Maps were
generated by iterating through cells in row-first order using a probabilistic cellular automaton,
where occupancy was determined as a function of the upper, left, and upper-left corner cells
Algorithm 1 REDUCED DISTRIBUTED PLANNER(i)

1: \( D_i = \{1, \ldots, n\} \), \( \Pi_{cache} = \emptyset \)
2: \( \pi_i \leftarrow \text{PLAN}(O, \emptyset) \)
3: \( p_i \leftarrow \text{COMPUTE_PRIORITY}(i, \pi_i) \)
4: \( m_i \leftarrow \langle i, p_i, \pi_i \rangle \)
5: \( \text{SEND_TO_TARGETS}(m_i, D_i) \)
6: \( \text{repeat} \)
7: \( M_{recv} \leftarrow \text{RECEIVE_MESSAGES}() \)
8: \( \text{for all} \langle j, p_j, \pi_j \rangle \in (M_{recv}) \text{ do} \)
9: \( \text{if} p_j > p_i \text{ then} \)
10: \( \Pi_{cache} \leftarrow \Pi_{cache} \cup \{\pi_j\} \)
11: \( D_i \leftarrow D_i \setminus \{j\} \)
12: \( \text{isCollided} \leftarrow \text{COLLISION_CHECK}(\pi_i, \Pi_{cache}) \)
13: \( \text{if isCollided then} \)
14: \( \pi_i \leftarrow \text{PLAN}(M, \Pi_{cache}) \)
15: \( m_i \leftarrow \langle i, p_i, \pi_i \rangle \)
16: \( \text{SEND_TO_TARGETS}(m_i, D_i) \)
17: \( \text{everyoneDone} \leftarrow \text{WAIT_FOR_Others}(!\text{isCollided}) \)
18: \( \text{until everyoneDone} \)

on the map. This yielded randomized maps containing partially-connected obstacles. The exact function is described in Table 2. Figure 4 shows two examples of these maps. Start locations were uniformly randomly sampled from the free cells in the map, and goal locations were uniformly randomly sampled from a square region 60 cells wide, centered at the start location. The time horizon of the problem was set to \( h = 80 \).

Two problem sets were generated to test the performance of the planners, a team-size dataset and a map-density dataset. 15 problem instances were generated for each parameter set and tested using a centralized prioritized planner to verify that a solution existed and to establish a baseline for performance. In the first dataset, the number of robots in the team was varied between 40 and 240. In order to keep the problem difficulty similar across different team sizes, the maps were sized such that the density (the ratio of robots to map cells) was held constant at 0.125. In the second dataset, the number of robots was fixed at 240, while the density of the map varied between 0.125 and 0.03125.

The results of the experiments can be seen in Figure 5 and 6. To ensure that efficient paths were being generated, the cumulative path costs of each solution were compared to those of a set of independently planned shortest paths for each robot from start to goal. In both of the problem sets, the cumulative path costs of the solutions (Figures 5(a) and 6(a)) found by the planners were less than 6% higher than the costs of the shortest paths, suggesting that both the centralized and distributed prioritized solutions were very efficient.

In these results, the distributed planner would be expected to outperform centralized planning in planning time in larger teams, as it is dividing computational load over the team of robots. However, as team size was varied, the results show no clear trends in the planning times, and, surprisingly, the distributed planners take longer than the centralized planner. Figure 5(b) shows
Figure 3: Multiagent path planning describes a rescue problem where there is no action or observational uncertainty, role-allocation has been done a-priori, and every cell is a narrow corridor with an infinite collision penalty.

the times as normalized by the time taken by a centralized prioritized planner. Part of this result is related to the characteristics of the underlying path planner. The A* planner takes widely varying amounts of time on these maps in the best and worst cases. For example, in a problem instance of 240 robots, A* was found to take 0.0047s for one robot and 4.4822s for another. While this means that the latter robot will be prioritized higher initially, this three orders of magnitude difference in planning times means that often, the distributed approaches are delayed by a few iterations in which they replan a path that takes A* a long time, something a centralized prioritized planner is guaranteed to only plan once.

If the density of the agents on the map is varied (Figure 6(b)) planning time shows a more clear trend. As density increases, relative planning time for the distributed planners also increases. At the lowest densities, the distributed planners resolve paths in less than half the time taken by the centralized planner. This shows that when the environment is sparse, the parallelization was more helpful.

However, the potential for the distributed planners to reduce planning time, specifically reduced DPP, is particularly evident when considering the number of iterations taken for convergence across problem sets, as seen in Figures 5(c) and 6(c). The distributed planner converges across all cases with a relatively small number of iterations, considering the team sizes. In the 240-robot problem instances, (the right side of Figure 5(c) and all of Figure 6(c)), it finds a so-
Figure 4: Two typical maps used in DPP experiments. Circles denote robot start positions while stars denote goals. In the smaller map, dotted lines connect associated start and goal positions.
Figure 5: Average results of centralized and distributed prioritized planners on the team-size dataset. Each point represents 15 runs.

As the presented algorithms spend most of their time repairing paths to avoid new obstacles, incremental planners [32] or roadmaps [28] might tremendously improve running-time by reducing this replanning time.

In summary, a DIMS-based distributed planner is constructed based on a prioritized planner, and tested on a number of multiagent path planning problems. Because the independent planning step uses a non-incremental planner, and there is a large variation in planning times among agents, the distributed planner still takes comparable time to the centralized planner even though it can make use of the available computation distributed over the team. However, it is shown to produce solutions of similar quality to the centralized algorithm it is based upon while requiring far fewer sequential planning steps.
Figure 6: Average results of centralized and distributed prioritized planners on the map-density datasets. Each point represents 15 runs.

4.2 Distributed POMDP Solving

Next, we consider the full uncertainty of a Dec-POMDP problem using the Distributed POMDPs with Coordination Locales (DPCL) formulation introduced in [65], a more specific subclass of Dec-POMDPs than the SI-Dec-POMDP. In particular, we consider the rescue domain previously described in Section 3.4.1.

In this problem, rescue robots are tasked with reaching victims in a grid world, impeded by debris which can be cleared by cleaner robots and narrow corridors which allow passage of only a single robot at a time. Within the model used here, the actions that robots take also have some probability of succeeding or failing, and robots can only observe either the success of their actions or the safety of neighboring cells, both with some probability of accurate observation. Thus, the model has a rich combination of interaction between robots, action uncertainty, and observational uncertainty.
4.2.1 DPCL

A Distributed POMDP with Coordination Locales (DPCL) represents a subclass of the DEC-POMDP model in which interactions in transitions and rewards between agents can be factored explicitly at particular locations in space called Coordination Locales. It is represented using the same tuple of \( \langle A_g, S, A, \Omega, \mathcal{P}, R, \mathcal{O}, b_0, h \rangle \), where \( A_g, S, A, \Omega \) are the agents, states, actions and observations and \( \mathcal{P}, R, \mathcal{O} \) are the joint transition, reward and observation functions respectively.

Like SI-Dec-POMDP, DPCL assumes the state space \( S \) is factorable into sets of global states and local states. The global state primarily represents the set of tasks in the environment, while the local state corresponds to agent-specific state information. Thus formally:

\[
S := S_g \times S_1 \times \ldots \times S_N
\]

where \( S_n \) is a set of local states of agent \( n \) for \( 1 \leq n \leq N \) and \( S_g \) is a set of task states that keep track of the execution of tasks.

DPCL also assumes that the interactions among agents are limited and explicit. Thus, independent transition \( \mathcal{P}_i \), observation \( \mathcal{O}_i \), and reward \( \mathcal{R}_i \) functions for each agent can be defined separately (i.e. assuming no other agents are present in the environment). In addition, DPCL supports two specific types of interactions between agents: Same Time Coordination Locales (STCLs) and Future Time Coordination Locales (FTCLs).

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. For example, consider the case of two agents \( i \) and \( j \) attempting to enter a narrow corridor simultaneously. They would collide and one of them would be forced to transition back to its starting state. This cannot be represented by an independent transition function of just \( S_g \) and \( S_i \) or just \( S_g \) and \( S_j \). In a particular region of state-action space, \( \mathcal{P}_{ij} \) must be represented by a joint function over \( S_g, S_i, \) and \( S_j \).

FTCLs represent situations where actions of one agent impact actions of others in the future. Informally, these interactions describe situations in which agents modify the global state \( s_g \) as they execute their tasks, which can have a future impact on other agents’ transitions and rewards since both \( \mathcal{P}_i \) and \( \mathcal{R}_i \) depend on \( s_g \). For example, if a cleaner agent removes debris, changing the state of a debris task in \( S_g \), it will change the transition function for a rescue agent later moving through that location, by allowing it to move unimpeded.

To represent these interactions, DPCL maintains a set of these CLs. A CL is formally 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 \( \Gamma \) 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. Informally, a CL is “active” for an agent when it has a significant probability of entering the states and actions described by the CL. The set of coordination locales is assumed to be available to each agent a-priori, having been pre-computed from the joint transition and reward functions when the DPCL is constructed.

Once again, DPCLs can be trivially reduced to SI-Dec-POMDPs by simply reformatting the set of STCLs and FTCLs into a set of the more general SI-Dec-POMDP interaction tuples. The independent agent models remain the same, and the remainder of the problem remains the same.
4.2.2 TREMOR

In this section, we review the previous work with the TREMOR (Team REshaping of MOdels for Rapid execution) algorithm [65], an efficient approximate solver for DPCLs of up to 10 agents. TREMOR was a multistage planner that shared many similarities with the DIMS framework, but contained several monolithic processes forcing it to be centralized. It consisted of the same four stages as DIMS: role allocation, independent planning, interaction detection, and model shaping.

Role Allocation. Role allocation was used to assign agents to particular tasks specified by the DPCL. These tasks were assignments of the victims, for rescue agents, and debris, for the cleaner agents. Allocations were found using a centralized branch-and-bound search over a simpler MDP model of the problem. A tree of all possible allocations of agents to tasks was constructed. For each allocation, a centralized MDP solver was run to approximate the optimal joint MDP policy. This was used as an upper bound for the potential value obtainable for that allocation. The unpruned allocations were evaluated in full POMDP form over the using Algorithm 2 until an overall optimum was found.

Independent Planning. For each allocation of interest from the role allocation stage, independent EVA [64] POMDP solvers were first run to find optimal policies for each of the agents using the factored POMDP models provided by the DPCL (Alg. 2 line 1). After the interaction detection and model shaping steps, these solvers are re-run to find new optimal policies for the factored models (Alg. 2 line 10), until convergence or for some number of iterations (Alg. 2 line 4).

Interaction Detection. The resulting set of policies was evaluated empirically using a centralized simulation, which allowed the detection of relevant interactions (those which occur with some probability \( > \epsilon \)). Through numerous runs, marginal distributions of the STCL and FTCL interactions were computed (Alg. 2 line 5).

Model shaping. Given these distributions of interactions, social model shaping was used as a mechanism to integrate the interactions into the factored POMDP models. In states where a particular action led to an interaction with another agent, the model is updated with using social model shaping to incentivize or penalize the action based on the expectation of the

\begin{algorithm}
\begin{algorithmic}[1]
\State \( \pi^* \leftarrow \text{SOLVEINDIVIDUALPOMDPs}(\{\mathcal{P}_i\}_{i \leq N}) \)
\State \( \pi \leftarrow \phi \)
\State \( \text{iter} \leftarrow 0 \)
\While {\( \pi \neq \pi^* \) \& \& \( \text{iter} < \text{MAX\_ITERATIONS} \)}
\State ActiveCLs \( \leftarrow \text{COMPUTEACTIVECLs}(\{\mathcal{P}_i\}_{i \leq N}, \text{AllCLs}) \)
\ForAll {cl \in \text{ActiveCLs}}
\State \{val_a\}_{a \in \text{cl.agents}} \leftarrow \text{EVALUATECL}(cl)
\State \{\mathcal{P}_a\} \leftarrow \text{SHAPEMODELS}(cl, \{\{val_a\}, \{\mathcal{P}_a\}\}_{a \in \text{cl.agents}}) \)
\State \( \pi^* \leftarrow \pi d \)
\State \( \pi \leftarrow \text{SOLVEINDIVIDUALPOMDP}(\{\mathcal{P}_i\}_{i \leq N}) \)
\State \( \text{iter} \leftarrow \text{iter} + 1 \)
\EndFor
\EndWhile
\end{algorithmic}
\end{algorithm}
joint outcome (Alg. 2, lines 6-8). 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 joint policy value over the team with and without the Coordination Locale. This updated model is used in the next iteration of independent planning.

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. In [65], TREMOR was shown to be effective in a rescue domain similar to that described in Section 3.4.1 in which rescue and cleaner robots navigated grid worlds, and positive reward was achieved by moving rescue robots to the known locations of victims on the map (the primary goal of the team). As in the rescue domain described in this work, grid cells could be safe or unsafe, could be narrow, and/or could contain debris. However, in [65], the model also included a health state for each agent, a value which would decrease upon dangerous actions, and would result in robots being unable to succeed at actions at all. In the domain models used in this work, dangerous actions directly lead to negative rewards.

4.2.3 D-TREMOR

D-TREMOR, introduced in [67], avoids the scalability problems inherent in TREMOR and other distributed POMDP approaches by distributing the planning effort between agents and employing intelligent heuristics in CL communication and model shaping. The primary changes between TREMOR and D-TREMOR lie in the decentralization of the role allocation and interaction detection stages. In this section, we begin by describing the basic distributed planning algorithm of D-TREMOR and then describe the various heuristics employed to improve its performance. By fully decentralizing the planner, we are able to scale D-TREMOR to teams of a hundred agents or more.

<table>
<thead>
<tr>
<th>Role Allocation</th>
<th>Policy Solution</th>
<th>Interaction Detection</th>
<th>Model Shaping</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREMOR Branch &amp; Bound MDP</td>
<td>Independent EVA [64] solvers</td>
<td>Joint policy evaluation</td>
<td>Social model shaping of independent models</td>
</tr>
<tr>
<td>D-TREMOR Decentralized role allocation</td>
<td>Sampling &amp; message passing</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Algorithm [3] provides the pseudocode executed at each agent in performing these steps.

**Role Allocation** The joint evaluation of an MDP to find upper bounds on policy expectation scales exponentially in complexity as the number of agents is increased, quickly becoming
Algorithm 3 D-TREMOR(Agent i)

1: \( \pi_i \leftarrow \text{ObtainInitialPolicy}(\mathcal{M}_i, \text{allCLs}) \)
2: \( \text{iter} \leftarrow 0 \)
3: while \( \text{iter} < \text{MAX\_ITERATIONS} \) do
4: \( \alpha\text{CLs} \leftarrow \text{ComputeActiveCLs}(\mathcal{M}_i, \text{allCLs}, \pi_i) \)
5: for all \( cl \in \alpha\text{CLs} \) do
6: \( \text{val}_{i,cl} \leftarrow \text{EvaluateCL}(cl, \mathcal{M}_i, \pi_i) \)
7: \( \text{CommunicateCL}(i, cl, pr_{i,cl}, \text{val}_{i,cl}) \)
8: \( \text{recCLs} \leftarrow \text{ReceiveCLs}() \)
9: \( \mathcal{M}_i \leftarrow \text{ShapeModel}(\text{recCLs}, \mathcal{M}_i) \)
10: \( \pi_i \leftarrow \text{SolveIndividualPOMDP}(\mathcal{M}_i) \)
11: \( \text{iter} \leftarrow \text{iter} + 1 \)

prohibitively expensive. As seen in Table[3] role allocation in D-TREMOR is done instead through the used of a decentralized mechanism. Existing distributed role allocation algorithms \([22, 51, 52]\) can be used to compute the allocation of tasks to agents. Once again, for the rescue domain, these tasks are assignments of victims to rescue agents and debris to cleaner agents, with known locations.

**Independent Planning** As in TREMOR, independent planning is carried out by an array of EVA POMDP solvers [64] (lines 1 and 10). The remainder of this section focuses on the remaining two steps, which are architected in very different ways to ensure complete distributability of the algorithm.

**Interaction Exchange** Another major difference between D-TREMOR and TREMOR lies in how interactions are evaluated. Rather than depend on an expensive joint evaluation step, D-TREMOR uses a sampling-based approximation scheme to determine which CLs are active at any time.

Each agent computes the set of CLs which could be active given its own policy, i.e., \( \alpha\text{CLs} = \{cl | cl = \langle t, s_g, \{s_i\}^m_1, \{a_i\}^m_1, \Gamma \rangle, Pr_{\pi_i}((s_g, s_i), a_i) > \epsilon \} \). 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, this means that given a particular pair of messages, it is possible to compute the exact 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 is \( V_{cl_i} + V_{cl_j} \), and the probability of the event is \( Pr_{cl_i} \cdot Pr_{cl_j} \). From this, the expected joint utility can be computed to be \( Pr_{cl_i} \cdot Pr_{cl_j} \cdot (V_{cl_i} + V_{cl_j}) \). To avoid introducing unwanted message dynamics, the current version of D-TREMOR simply enforces complete communication between agents. While this is \( O(n^2) \) with the size of the team, the relatively small size of
the message tuples allows many of them to be exchanged with reasonably low bandwidth.

**Model Shaping** Upon receiving CL messages from other agents, each agent shapes the transition and reward function of its individual model. Each agent \( i \) that receives a CL message from \( j \) computes the probability of occurrence of \( cl \), \( Pr_{cl} \), and the value of the CL, \( V_{cl} \). The probability of occurrence of a coordination locale with respect to both agents is then computed, i.e., \( \hat{c}_{cl} = Pr_{cl_i} \cdot Pr_{cl_j} \).

In TREMOR, the new transition probability \( P'_{e} \) at decision epoch \( e \) for STCLs is computed by using a shaping heuristic. According to this heuristic, we take the weighted average of \( P_{i,cl}^e \) and \( P_{1,cl}^e \). \( P'_{i,cl} \) is the transition probability without any interactions, i.e., \( P_i \). In D-TREMOR, we provide a new improved heuristic as described in Section 4.2.8. While the expressions below are for STCLs, the expressions for FTCLs are similar as explained in [65].

\[
P_{i,cl}^e((s_g, s_i), a_i, (s'_g, s'_i)) \leftarrow \sum_{s' \in S: s' = (s'_g, s'_i)} P((s_g, s_i, s_j), (a_i, a_j), (s'_g, s'_i, s'_j)) \tag{4}
\]

\[
P_i^e \leftarrow \hat{c}_{cl} \cdot P_{i,cl}^e + (1 - \hat{c}_{cl}) \cdot P_{i,\neg cl}^e \tag{5}
\]

\[
R_{i,cl}^e((s_g, s_i), a_i, (s'_g, s'_i)) \leftarrow \sum_{s' \in S: s' = (s'_g, s'_i)} R((s_g, s_i, s_j), (a_i, a_j), (s'_g, s'_i, s'_j)) \tag{6}
\]

\[
R_i^e \leftarrow \hat{c}_{cl} \cdot R_{i,cl}^e + (1 - \hat{c}_{cl}) \cdot R_{i,\neg cl}^e \tag{7}
\]

In the next section, we describe the key characteristics that allow the D-TREMOR algorithm to improve its performance over existing algorithms considerably. As shown in 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.

### 4.2.4 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 \( cl : \left((e, (s_g, s_i), (a_i, a_j))\right)\), agent \( i \) computes the probability for \((e, (s_g, s_i), a_i)\) given its policy \( \pi_i \) and agent \( j \) computes the probability for \((e, (s_g, s_i), a_i)\) given its policy \( \pi_j \). 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.

4.2.5 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 [70]. It is governed by a parameter $\delta$, 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 $\delta$, 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 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, $\{r_i\}_{i=1}^n$, such that $r_1 > r_2 > r_3 \ldots > r_n$. 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 rescue domain of Section 3.4.1, 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.

4.2.6 Computing $Pr_{cl_i}$ and $V_{cl_i}$ efficiently

While the parallelism in 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. To improve the efficiency of these computations, we provide an approach inspired from a sampling approach developed for solving large Markov Decision Processes [29]. 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 4 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 $V_{cl_i}$.

Algorithm 4 COMPUTEPRCL($i, cl, \hat{\pi}_i, b^0$)

1: $iter \leftarrow 0$
2: $val = 0$
3: while $iter < NUM - SAMPLES$ do
4: \hspace{0.5cm} $\pi_i \leftarrow \hat{\pi}_i; s \leftarrow GETSIMSTATE(b^0); \tau \leftarrow 0$
5: \hspace{1cm} while $\tau < cl.t$ do
6: \hspace{1.5cm} $act \leftarrow \pi_i.a$
7: \hspace{1.5cm} $s' \leftarrow GETSIMFUTURESTATE(s, act)$
8: \hspace{1.5cm} $\omega \leftarrow GETSIMOBS(s', act)$
9: \hspace{1.5cm} $\pi_i \leftarrow \pi_i(\omega); s \leftarrow s'$
10: \hspace{1cm} if $s = cl.s_i$ and $act = cl.a_i$ then
11: \hspace{1.5cm} $val \leftarrow val + 1$
12: \hspace{1cm} return $val$
13: return $NUM - SAMPLES$

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 $\epsilon$ is close to zero) given the time horizon. Given an action $a$ and observation $\omega$, the maximization problem for belief probability of state $s_t$ (state $s$ at decision epoch $t$) is given by:

$$
\max_{b_{t-1} \in B_{t-1}} \frac{\sum_{s_{t-1}} O_t(s_t, a, \omega) \Sigma_{s_{t-1}} P_{t-1}(s_{t-1}, a, s_t) b_{t-1}(s_{t-1})}{\sum_{s_{t-1}} O_t(s_t, a, \omega) \Sigma_{s_{t-1}} P_{t-1}(s_{t-1}, a, s_t) b_{t-1}(s_{t-1})}
$$
This is solved in polynomial time using the lagrangian techniques presented in [63].

4.2.7 Capturing dependencies between CLs

In TREMOR, each CL is treated independently of others, i.e. assuming that model shaping corresponding to a CL does not affect any other CL. In weakly coupled domains, i.e., ones with fewer 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 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 and V_cl, 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 and V_cl, while not sacrificing on the efficiency. Such accurate estimates of Pr_cl and V_cl essentially reduce the difference between the shaped models and the joint model and hence provide improved solutions.

4.2.8 Shaping Heuristics

In the context of the expressions in Equation 5 and Equation 7, 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:

\[ \mathcal{P}^{new}_i \leftarrow \sum_{cl \in CL_i,s,a} \hat{c}_{cl} \cdot \mathcal{P}^{e}_{i,cl} + (1 - \sum_{cl \in CL_i,s,a} \hat{c}_{cl}) \cdot \mathcal{P}^{e}_{i,-cl} \]  

\[ \mathcal{R}^{new}_i \leftarrow \sum_{cl \in CL_i,s,a} \hat{c}_{cl} \cdot \mathcal{R}^{e}_{i,cl,s} + (1 - \sum_{cl \in CL_i,s,a} \hat{c}_{cl}) \cdot \mathcal{R}^{e}_{i,-cl,s} \]  

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

4.2.9 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, consider the example in Figure 7. 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 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 $R(s_g, (s_i, s_j), (a_i, a_j)) > R_i(s_g, s_i, a_i) + R_j(s_g, s_j, a_j)$, then $Pr_{i,cl} = 1$. 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.

4.3 Initial results

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 8(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.

In the density dataset, a square $9 \times 9$ 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 8(b) 8(c) and 8(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. 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 4.2.9. 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 simply
do not move from their original locations, and in the random policy, each agent independently selects their next 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, assigning to each rescue robot the single closest unassigned victim, and to each cleaner robot the single closest unassigned debris. In subsequent iterations, all activated CLs are communicated to every team member. 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 Figures 9 and 10. In the former, data are normalized to the policies of independent planning by subtracting its performance from that of the other algorithms. Figure 9(a) compares the average joint value of the various solution policies. If the maximum-value iteration of D-TREMOR (Max D-TREMOR in Figure 9(a)) is evaluated, it outperforms or matches the value of the other techniques in every case. This establishes the ability of the algorithm to find good joint solutions in complex environments. However, simply choosing the last (20th) iteration of D-TREMOR yields much poorer performance, suggesting that in its current form, D-TREMOR cannot necessarily detect when it has reached a good solution. In a single run (Figure 10(c)), 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 9(b)), 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 9(c)), however, D-TREMOR, random, and do-nothing all achieve similar low values, while optimistic and independent collide more often.

While cleaner robots clear many debris under the optimistic policies (Figure 9(e)), their number of rescue robots colliding with debris is higher than that of the independent policies (Figure 9(d)) 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
Figure 9: Performance measures for algorithms on the scaling dataset.
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 possible to directly compare the running times across the scaling dataset, as agents need not compete for CPU resources. Figure 10(b) 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 10(a)).

Results of the density dataset are seen in Figure 11. As expected, increasing the density of narrow corridors decreases the performance of all policies except the do-nothing policy (Figure 11(a)). The abundance of narrow corridors causes optimistic and independent policies to suffer a very high number of collisions (Figure 11(b)), dropping their overall value despite the fact that they manage to secure some victims (Figure 11(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 11(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.

### 4.3.1 Summary of results

These runs confirmed several desirable properties of D-TREMOR. First, it is able to remain tractable in problem sizes of up to 100 agents. Second, the fact that it can generate policies
that are better than independent suggested that it is capable of exploring useful areas of the joint policy solution space. Finally, it is possible to confirm empirically the near-linear time scaling of D-TREMOR with number of agents (Figure 10(b)). As the complexity of these models is additionally dependent on the number of interactions that need to be considered, as long as interactions are relatively sparse, the computation time taken to complete each iteration of D-TREMOR should increase only linearly with the number of agents.

While D-TREMOR is shown to be effective at finding joint solutions in team planning problems with complex interactions, uncertainty, and at least a hundred agents, there are several key deficiencies in its current incarnation. First, it does not currently have the ability to handle simultaneous interactions with multiple agents accurately, instead assuming conditional independence between interactions. Second, it is limited in generality, being unable to handle interactions in agent observations. Finally, and most importantly, it is unclear what the performance and convergence properties of the algorithm are over all potential types of interactions. It is evident that the algorithm can be effective on collisions and debris interactions, but it is critical to know how this behavior extrapolates to other types of interactions. It is therefore necessary to consider a more principled line of investigation into the dynamics of interaction exchange and model prediction when using the DIMS framework.

5 Proposed Work

While preliminary work suggests that DIMS-based planning is a promising approach to solving SI-Dec-POMDP problems, there are a number of remaining theoretical and practical issues that remain unresolved. The aim of this thesis is to explore these issues to ascertain the validity and scope of the DIMS approach.

In preliminary work, two interactions, negative pairwise reward/transition interactions (collisions) and positive pairwise transition interactions (debris-clearing), have been explored, and results suggest that the two interactions need to be dealt with using distinct heuristic mechanisms. However, in general, the potential classes of interactions that can and cannot be solved using these heuristics is not clear. In order to understand how and when DIMS planning will be effective, it is necessary to understand what exactly makes solving each of these types of interactions difficult, and if there are any other interesting classes of interaction that could be expected to arise in the domains of interest.

Once these interesting classes of interaction are well defined, heuristic strategies addressing each of the classes can be designed and evaluated to determine the efficacy of DIMS planning in each case. This is necessary to establish the generality of DIMS planning as a a solution strategy in multiagent sequential decision making problems.

In order to address these two issues, proposed work will be divided into the following two components: the development of an interaction taxonomy and the development of targeted heuristics to solve relevant interactions.
5.1 Interaction Taxonomy

While preliminary work with D-TREMOR and DPP demonstrate that it is possible for DIMS planning to solve specific SI-Dec-POMDPs, it is desirable to determine what salient features in SI-Dec-POMDPs make them more or less solvable. Starting from the two distinct classes of interaction identified in the preliminary work (those corresponding to collisions and those corresponding to debris), this work will formalize the characteristics of the classes of interactions in our domains and develop a richer taxonomy of potential interactions. This taxonomy will identify the key features that cause DIMS planning to behave differently when handling particular interactions, and separate out the space of potential interactions into classes of similar behavior.

The preliminary work suggests several key features that can act as starting points for this characterization. The choice of which $P$, $R$, and $O$ functions are modified appears to affect the complexity of interaction resolution: modifications to local reward only seem to have simpler effects than other combinations. In addition, the existence of a preferred configuration of the interaction (e.g., for a rescue robot, it is always better for debris to be cleared rather than present) can be exploited through optimistic heuristics. However, determining if such configurations exists can be surprisingly complex. For example, while intuitively, it seems as though this should be the case for collision interactions (e.g. "for a pair of robots, it is better to not have a collision than have a collision"), the resulting change in $P$ when a collision occurs allows situations in which a collision has a positive outcome: robots can be pushed into cells to achieve positive rewards or reduce uncertainty, increasing their future expected reward.

Once a taxonomy begin to emerge, interaction classes can be prioritized by their relevance to the domains and problems of interest. Certain classes may be pathological in nature, but many of these classes may also be unlikely to arise in the planning problems studied here. Since DIMS itself is an approximate planning technique, this will allow subsequent work to be focused on developing heuristics to solve the classes that do commonly occur, while making a clear distinction as to which types of problems are not well suited to the approach.

5.2 Heuristic Development

Following the creation of a taxonomy of interaction, heuristics can be targeted to address each of the interaction classes of interest. In preliminary work, we studied a prioritization heuristic in order to solve collision interaction, and a probabilistic model-shaping heuristic for handling the debris interaction. However, these are two specific instances of broader classes of interactions between agents. Once relevant interaction classes are identified, these and other heuristics that independently target each class of interaction can be developed to ensure reasonable performance on problems with arbitrary sets of the addressed interactions without requiring problem-specific heuristics.

As described in Section 2.3, several approaches to solving MSDM problems in the literature have used game-theoretic formulations to model the process of coordinating agents. While directly applying these methods is generally too complex given the scales of our problems, studying these existing strategies for equilibria-finding or model-learning in smaller coordination games may provide further inspiration into potential solution heuristics.
5.3 Evaluation

The performance of the DIMS planner will be tested on the two previously described domains of search and rescue and humanitarian convoy planning. Each of these two domains will be modeled at two levels of fidelity. First, a low-fidelity, discrete event simulation will be created to test the behaviors and optimality of the DIMS planner against comparable approaches. Then, high-fidelity, real time physical simulations will be performed to demonstrate the practicality of the approach, as well as study the real-world issues arising from the SI-Dec-POMDP model approximation. Two existing simulation environments, the Unified System for Automation and Robot Simulation (USARSim, Figure 12(a)) and Virtual Battlespace 2 (VBS2, Figure 12(b)) will provide the basis and validation for this modeling. In addition, three SI-Dec-POMDP models of varying levels of abstraction will be constructed: a fully-observable, certain model, a fully-observable uncertain model, and a partially-observable, uncertain model. A summary of the domains and models that will be evaluated can be seen in Table 4. The table describes the applicability of prior algorithms to the domain models, while dashes indicate where the proposed algorithmic extensions are necessary.

Figure 12: Two high-fidelity simulation environments will be used to create and verify rescue and convoy domain models.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Search and Rescue</th>
<th>Humanitarian Convoy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Discrete Event</td>
<td>USARSim</td>
</tr>
<tr>
<td>Graph</td>
<td>DPP</td>
<td>-</td>
</tr>
<tr>
<td>MDP</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>POMDP</td>
<td>D-TREMOR</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4: Proposed domains and models
5.4 Timeline

The proposed research will be carried out over a period of 18 months, as summarized in Table 5. Proposed work will begin with the development of an interaction taxonomy relevant to the DIMS framework and the identification of relevant and interesting classes of interaction, which will finally enable the development of a DIMS planner capable of handling these interactions. Empirical results will be collected in the rescue and convoy domains using models of several levels of abstraction. A more detailed breakdown of tasks can be found in the Gantt chart in Figure 13.

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct-Feb 2011</td>
<td>Construct taxonomy of interactions</td>
</tr>
<tr>
<td>Feb-May 2011</td>
<td>Develop heuristics to solve common classes of interaction</td>
</tr>
<tr>
<td>Mar-Jul 2011</td>
<td>Implementation of DIMS solver with interaction heuristics</td>
</tr>
<tr>
<td>Jul-Oct 2011</td>
<td>Rescue experiments</td>
</tr>
<tr>
<td>Oct-Jan 2012</td>
<td>Convoy experiments</td>
</tr>
<tr>
<td>Feb-May 2012</td>
<td>Thesis preparation</td>
</tr>
<tr>
<td>May 2012</td>
<td>Defend thesis</td>
</tr>
</tbody>
</table>
Figure 13: Detailed timeline of proposed research
5.5 Risk Mitigation

Preliminary and prior work suggest that the proposed research is feasible. Major risks that may arise during the proposed research, and steps that have been taken to mitigate those risks, are presented in Table 6.

Table 6: Risks and mitigation strategies

<table>
<thead>
<tr>
<th>Risk</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unable to create formal taxonomy</td>
<td>Moderate</td>
<td>Mild</td>
<td>Develop of informal taxonomy. Create empirical interaction “test sets” for evaluation</td>
</tr>
<tr>
<td>Failure to solve interaction class</td>
<td>Mild</td>
<td>Moderate</td>
<td>Generate empirical results exploring why interaction cannot be solved. Restrict final performance evaluation to models excluding specific interaction class</td>
</tr>
</tbody>
</table>

6 Conclusion

This thesis proposal presents a generalized framework (DIMS) for solving planning problems in large teams of agents with sparse interactions and a taxonomy of potential sparse interactions between agents as relevant to the framework. DIMS applies to problems with and without transitional uncertainty and partial observability. It builds upon recent work using sparse interaction to accelerate searches for approximate planning solutions, but adds the novel contributions of complete distributability, fast heuristic interaction detection, and local message exchange to achieve high scalability. The approach will be validated in two realistic domains involving hundreds of agents, demonstrating its efficacy and efficiency when applied to real-world problems.

6.1 Expected Contributions

The expected contributions of the proposed research are:

- A distributed POMDP planner capable of handling hundreds of agents
- A generalized framework for solving large-scale planning problems of varying uncertainty
- Empirical results of distributed planning in teams of at least 100 agents.

6.2 Significance and Future

This research presents a significant step forward in the scalability of team planning algorithms. The proposed framework will allow coordinated planning of hundreds of agents in highly uncertain environments. This improves team planning methods by at least an order of magnitude, making it possible to study and apply team coordination strategies to team sizes one to two orders of magnitude larger than previously addressed in the literature. In addition, the near-linear
computational complexity of the algorithm has the potential to handle even larger team planning problems, where agents potentially number into the thousands.

7 Bibliography


[48] Maayan Roth, Reid Simmons, and Manuela Veloso. Reasoning about joint beliefs for execution-time communication decisions. Proceedings of the Fourth International Joint


