

Coordinating Teams in Uncertain Environments: A Hybrid BDI-POMDP Approach

Ranjit Nair and Milind Tambe

Computer Science Department
University of Southern California
Los Angeles, CA 90089
{nair,tambe}@usc.edu

Abstract. Distributed partially observable Markov decision problems (POMDPs) have emerged as a popular decision-theoretic approach for planning for multiagent teams, where it is imperative for the agents to be able to reason about the rewards (and costs) for their actions in the presence of uncertainty. However, finding the optimal distributed POMDP policy is computationally intractable (NEXP-Complete). This paper is focussed on a principled way to combine the two dominant paradigms for building multiagent team plans, namely the “belief-desire-intention” (BDI) approach and distributed POMDPs. In this hybrid BDI-POMDP approach, BDI team plans are exploited to improve distributed POMDP tractability and distributed POMDP-based analysis improves BDI team plan performance. Concretely, we focus on role allocation, a fundamental problem in BDI teams – which agents to allocate to the different roles in the team. The hybrid BDI-POMDP approach provides three key contributions. First, unlike prior work in multiagent role allocation, we describe a role allocation technique that takes into account future uncertainties in the domain. The second contribution is a novel decomposition technique, which exploits the structure in the BDI team plans to significantly prune the search space of combinatorially many role allocations. Our third key contribution is a significantly faster policy evaluation algorithm suited for our BDI-POMDP hybrid approach. Finally, we also present experimental results from two domains: mission rehearsal simulation and RoboCupRescue disaster rescue simulation. In the RoboCupRescue domain, we show that the role allocation technique presented in this paper is capable of performing at human expert levels by comparing with the allocations chosen by humans in the actual RoboCupRescue simulation environment.

1 Introduction

Teamwork, whether among software agents, or robots (and people) is a critical capability in a large number of multiagent domains ranging from mission rehearsal simulations to RoboCup soccer and disaster rescue to personal assistant teams. Already a large number of multiagent teams have been developed for a range of domains [31, 44, 36, 19, 13, 9, 38, 7]. These existing practical approaches can be characterized as situated within the general “belief-desire-intention” (BDI) approach, a paradigm for designing multiagent systems, made increasingly popular due to programming frameworks [38, 9, 39] that facilitate the design of large-scale teams. Within this approach, inspired explicitly or

implicitly by BDI logics, agents explicitly represent and reason with their team goals and plans [41].

This paper focuses on the quantitative evaluation of multiagent teamwork, to provide feedback to aid human developers and possibly to agents participating in a team, on how the team performance in complex domains can be improved. Such quantitative evaluation is especially vital in domains like disaster rescue [21] and mission rehearsal simulations [38], where the performance of the team is linked to important metrics such as loss of human life and property. Both these and other such complex domains exhibit uncertainty, which arises from partial observability and non-determinism in the outcomes of actions. However, tools for such quantitative evaluations of teamwork in the presence of uncertainty are currently absent. Thus, given these uncertainties, we may be required to experimentally recreate a large number of possible scenarios (in a real domain or in simulations) in order to correctly evaluate team performance.

Fortunately, the emergence of distributed **P**artially **O**bservable **M**arkov **D**ecision **P**roblems (POMDPs) provides models [3, 4, 30, 43] that are well-suited for quantitative analysis of agent teams in uncertain domains. These models are powerful enough to express the uncertainty in these dynamic domains and in principle, can be used to generate and evaluate complete policies for the multiagent team. However, as shown by Bernstein et al. [3], the problem of deriving the optimal policy is generally computationally intractable (the corresponding decision problem is NEXP-complete).

This paper deals with this issue of intractability in distributed POMDPs by combining in a principled way the two dominant paradigms for building multiagent teams, namely distributed POMDPs and the “belief-desire-intention” (BDI) approach. While BDI frameworks facilitate human design of large scale teams, their key shortcoming is their inability to quantitatively reason about team performance, especially in the presence of uncertainty. This hybrid BDI-POMDP approach combines the native strengths of the BDI and POMDP approaches, i.e., the ability in BDI frameworks to encode large-scale team plans and the POMDP ability to quantitatively evaluate such plans. This approach focuses on the analysis of BDI team plans, to provide feedback to human developers on how the team plans can be improved. In particular, it focuses on the critical challenge of role allocation in building teams [40, 18], i.e. which agents to allocate to the various roles in the team. For instance, in mission rehearsal simulations [38], we need to select the numbers and types of helicopter agents to allocate to different roles in the team. Similarly, in disaster rescue [21], role allocation refers to allocating fire engines and ambulances to fires and it can greatly impact team performance. In such domains, the role allocation chosen directly impacts the team performance, which is linked to metrics like loss of human life and property; and thus, it is critical to find the best role allocation.

In order to analyze role allocations quantitatively, we derive RMTDP (Role-based Multiagent Team Decision Problem), a distributed POMDP framework for quantitatively analyzing role allocations. Using this framework, we show that, in general, the problem of finding the optimal role allocation policy is computationally intractable (the corresponding decision problem is still NEXP-complete). This shows that improving the tractability of analysis techniques for role allocation is a critically important issue.

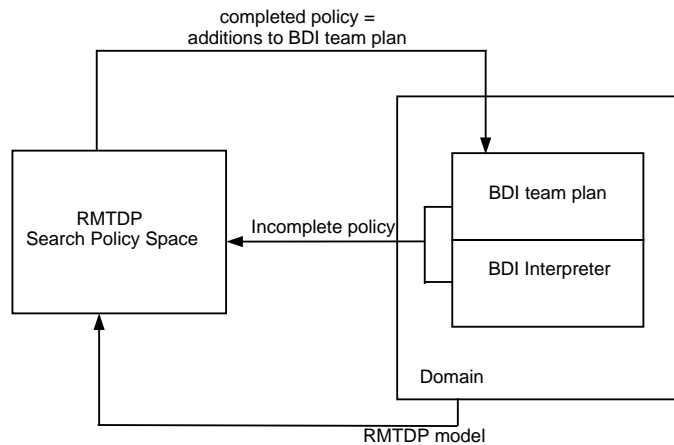


Fig. 1. Integration of BDI and POMDP.

The hybrid BDI-POMDP approach is based on three key interactions that improve the tractability of RMTDP and the optimality of BDI agent teams. The first interaction is shown in Figure 1. In particular, suppose we wish to analyze a BDI agent team (each agent consisting of a BDI team plan and a domain independent interpreter that helps coordinate such plans) acting in a domain. Then as shown in Figure 1, we model the domain via an RMTDP, and rely on the BDI team plan and interpreter for providing an incomplete policy for this RMTDP. The RMTDP model evaluates different completions of this incomplete policy and provides an optimally completed policy as feedback to the BDI system. Thus, the RMTDP fills in the gaps in an incompletely specified BDI team plan optimally. Here the gaps we concentrate on are the role allocations, but the method can be applied to other key coordination decisions. By restricting the optimization to only role allocation decisions and fixing the policy at all other points, we are able to come up with a restricted policy space. We then use RMTDPs to effectively search this restricted space in order to find the optimal role allocation.

While the restricted policy search is one key positive interaction in our hybrid approach, the second interaction consists of a more efficient policy representation used for converting a BDI team plan and interpreter into a corresponding policy (see Figure 1) and a new algorithm for policy evaluation. In general, each agent's policy in a distributed POMDP is indexed by its observation history [3, 30]. However, in a BDI system, each agent performs its action selection based on its set of privately held beliefs which is obtained from the agent's observations after applying a belief revision function. In order to evaluate the team's performance, it is sufficient in RMTDP to index the agents' policies by their *belief state* (represented here by their privately held beliefs) instead of their observation histories. This shift in representation results in considerable savings in the amount of space and time needed to evaluate a policy.

The third key interaction in our hybrid approach further exploits BDI team plan structure for increasing the efficiency of our RMTDP-based analysis. Even though RMTDP policy space is restricted to filling in gaps in incomplete policies, many policies may result given the large number of possible role allocations. Thus enumerating and evaluating each possible policy for a given domain is difficult. Instead, we provide a branch-and-bound algorithm that exploits task decomposition among sub-teams of a team to significantly prune the search space and provide a correctness proof and worst-case analysis of this algorithm.

In order to empirically validate our approach, we have applied RMTDP for allocation in BDI teams in two concrete domains: mission rehearsal simulations [38] and RoboCupRescue [21]. We first present the (significant) speed-up gained by our three interactions mentioned above. Next, in both domains, we compared the role allocations found by our approach with state-of-the-art techniques that allocate roles without uncertainty reasoning. This comparison shows the importance of reasoning about uncertainty when determining the role allocation for complex multiagent domains. In the RoboCupRescue domain, we also compared the allocations found with allocations chosen by humans in the actual RoboCupRescue simulation environment. The results showed that the role allocation technique presented in this paper is capable of performing at human expert levels in the RoboCupRescue domain.

This paper is organized as follows: In Section 2, background and motivation are presented. In Section 3, we introduce the RMTDP model and present key complexity results. Section 4 explains how a BDI team plan can be evaluated using RMTDP. Section 5 describes the analysis methodology for finding the optimal role allocation, and also presents an empirical evaluation of this methodology. Section 6 describes the JESP approach for finding locally optimal distributed POMDP policies. In Section 7, we present related work, in Section 8, we list our conclusions and in Section 9 we describe future work.

2 Background

This section first describes the two domains that we consider in this paper: an abstract mission rehearsal domain [38] and the RoboCupRescue domain [21]. Each domain requires us to allocate roles to agents in a team. Next, team-oriented programming (TOP), a framework for describing team plans is described in the context of these two domains. While we focus on TOP, as discussed further in Section 7.1, our techniques would be applicable in other frameworks for tasking teams [36, 9].

2.1 Domains

The first domain that we consider is based on mission rehearsal simulations [38]. For expository purposes, this has been intentionally simplified. The scenario is as follows: A helicopter team is executing a mission of transporting valuable cargo from point X to point Y through enemy terrain (see Figure 2). There are three paths from X to Y of different lengths and different risk due to enemy fire. One or more scouting sub-teams must be sent out (one for each path from X to Y), and the larger the size of a scouting

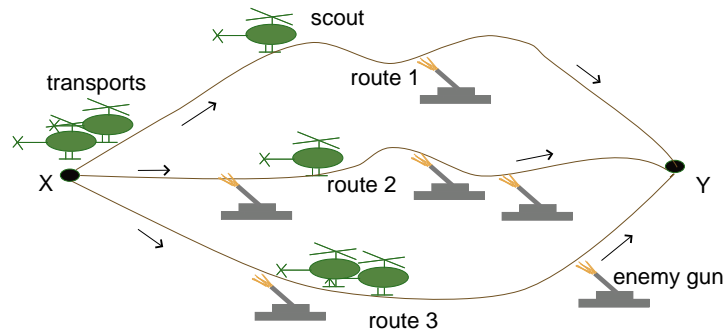


Fig. 2. Mission rehearsal domain.

sub-team the safer it is. When scouts clear up any one path from X to Y, the transports can then move more safely along that path. However, the scouts may fail along a path, and may need to be replaced by a transport at the cost of not transporting cargo. Owing to partial observability, the transports may not receive an observation that a scout has failed or that a route has been cleared. We wish to transport the most amount of cargo in the quickest possible manner within the mission deadline.

The key role allocation decision here is given a fixed number of helicopters, how should they be allocated to scouting and transport roles? Allocating more scouts means that the scouting task is more likely to succeed, but there will be fewer helicopters left that can be used to transport the cargo and consequently less reward. However, allocating too few scouts could result in the mission failing altogether. Also, in allocating the scouts, which routes should the scouts be sent on? The shortest route would be preferable but it is more risky. Sending all the scouts on the same route decreases the likelihood of failure of an individual scout; however, it might be more beneficial to send them on different routes, e.g. some scouts on a risky but short route and others on a safe but longer route.

Thus there are many role allocations to consider. Evaluating each one is difficult because role allocation must look-ahead to consider future implications of uncertainty, e.g. scout helicopters can fail during scouting and may need to be replaced by a transport. Furthermore, failure or success of a scout may not be visible to the transport helicopters and hence a transport may not replace a scout or transports may never fly to the destination.

The second example scenario (see Figure 3), set up in the RoboCupRescue disaster simulation environment [21], consists of five fire engines at three different fire stations (two each at stations 1 & 3 and the last one at station 2) and five ambulances stationed at the ambulance center. Two fires (in the top left and bottom right corners of the map) start that need to be extinguished by the fire engines. After a fire is extinguished, ambulance agents need to save the surviving civilians. The number of civilians at each location is not known ahead of time, although the total number of civilians is known. As time passes, there is a high likelihood that the health of civilians will deteriorate and fires



Fig. 3. RoboCupRescue Scenario: C1 and C2 denote the two fire locations, F1, F2 and F3 denote fire stations 1, 2 and 3 respectively and A denotes the ambulance center.

will increase in intensity. Yet the agents need to rescue as many civilians as possible with minimal damage to the buildings. The first part of the goal in this scenario is therefore to first determine which fire engines to assign to each fire. Once the fire engines have gathered information about the number of civilians at each fire, this is transmitted to the ambulances. The next part of the goal is then to allocate the ambulances to a particular fire to rescue the civilians trapped there. However, ambulances cannot rescue civilians until fires are fully extinguished. Here, partial observability (each agent can only view objects within its visual range), and uncertainty related to fire intensity, as well as location of civilians and their health add significantly to the difficulty.

2.2 Team-Oriented Programming

The aim of the team-oriented programming (TOP) [31, 38, 39] framework is to provide human developers (or automated symbolic planners) with a useful abstraction for tasking teams. For domains such as those described in Section 2.1, it consists of three key aspects of a team: (i) a team organization hierarchy consisting of roles; (ii) a team (reactive) plan hierarchy; and (iii) an assignment of roles to sub-plans in the plan hierarchy. The developer need not specify low-level coordination details. Instead, the TOP interpreter (the underlying coordination infrastructure) automatically enables agents to decide when and with whom to communicate and how to reallocate roles upon failure. The TOP abstraction enables humans to rapidly provide team plans for large-scale teams, but unfortunately, only a qualitative assessment of team performance is feasible. Thus, a key TOP weakness is the inability to quantitatively evaluate and optimize

team performance. For example, in allocating roles to agents only a qualitative matching of capabilities may be feasible. As discussed later, our hybrid BDI-POMDP model addresses this weakness by providing techniques for quantitative evaluation.

As a concrete example, consider the TOP for the mission rehearsal domain. We first specify the team organization hierarchy (see Figure 4(a)). *Task Force* is the highest level team in this organization and consists of two roles *Scouting* and *Transport*, where the *Scouting* sub-team has roles for each of the three scouting sub-sub-teams. Next we specify a hierarchy of reactive team plans (Figure 4(b)). Reactive team plans explicitly express joint activities of the relevant team and consist of: (i) pre-conditions under which the plan is to be proposed; (ii) termination conditions under which the plan is to be ended; and (iii) team-level actions to be executed as part of the plan (an example plan will be discussed shortly). In Figure 4(b), the highest level plan **Execute Mission** has three sub-plans: **DoScouting** to make one path from X to Y safe for the transports, **DoTransport** to move the transports along a scouted path, and **RemainingScouts** for the scouts which have not reached the destination yet to get there.

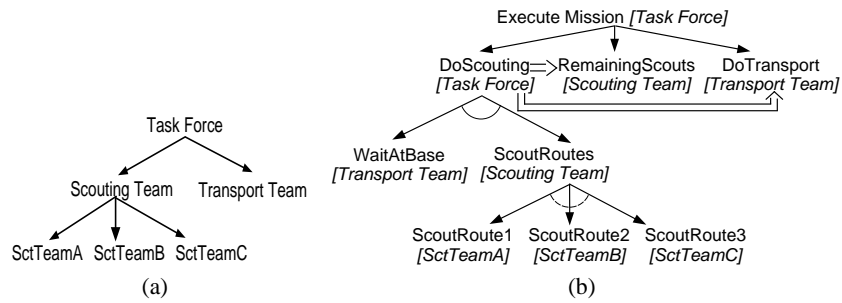


Fig. 4. TOP for mission rehearsal domain. a) Organization hierarchy; b) Plan hierarchy.

Figure 4(b) also shows coordination relationships: An AND relationship is indicated with a solid arc, while an OR relationship is indicated with a dashed arc. Thus, **WaitAtBase** and **ScoutRoutes** must both be done while at least one of **ScoutRoute1**, **ScoutRoute2** or **ScoutRoute3** need be performed. There is also a temporal dependence relationship among the sub-plans, which implies that sub-teams assigned to perform **DoTransport** or **RemainingScouts** cannot do so until the **DoScouting** plan has completed. However, **DoTransport** and **RemainingScouts** execute in parallel. Finally, we assign roles to plans – Figure 4(b) shows the assignment in brackets adjacent to the plans. For instance, *Task Force* team is assigned to jointly perform **Execute Mission** while *SctTeamA* is assigned to **ScoutRoute1**.

The team plan corresponding to **Execute Mission** is shown in Figure 5. As can be seen, each team plan consists of a context, pre-conditions, post-conditions, body and constraints. The context describes the conditions that must be fulfilled in the parent plan while the pre-conditions are the particular conditions that will cause this sub-plan to begin execution. Thus, for **Execute Mission**, the pre-condition is that the team mu-

```

ExecuteMission:
Context: {}
Pre-conditions: (MB <TaskForce> location(TaskForce) = START)
Achieved: (MB <TaskForce> (Achieved(DoScouting) ∧
    Achieved(DoTransport))) ∧ (time > T ∨ (MB <TaskForce>
    Achieved(RemainingScouts) ∨ (∄ helo ∈ ScoutingTeam,
    alive(helo) ∧ location(helo) ≠ END)))
Unachievable: (MB <TaskForce> Unachievable(DoScouting)) ∨
    (MB <TaskForce> Unachievable(DoTransport) ∧
    (Achieved(RemainingScouts) ∨ (∄ helo ∈ ScoutingTeam,
    alive(helo) ∧ location(helo) ≠ END)))
Irrelevant: {}
Body:
    DoScouting
    DoTransport
    RemainingScouts
Constraints: DoScouting → DoTransport, DoScouting →
    RemainingScouts

```

Fig. 5. Example team plan. MB refers to mutual belief.

tually believes (MB) that they are the “start” location. The post-conditions are divided into Achieved, Unachievable and Irrelevant conditions under which this sub-plan will be terminated. The body consists of sub-plans that exist within this team plan. Lastly, constraints describe any temporal constraints that exist between sub-plans in the body.

During execution, each agent has a copy of the TOP. The agent also maintains a set of private beliefs, which are a set of propositions that the agent believes to be true (see Figure 6). When an agent receives new beliefs, i.e. observations (including communication), the belief update function is used to update its set of privately held beliefs. For instance, upon seeing the last scout crashed, a transport may update its privately held beliefs to include the belief “CriticalFailure(DoScouting)”. In practical BDI systems, such belief update computation is of low complexity (e.g. constant or linear time). Once beliefs are updated, an agent selects which plan to execute by matching its beliefs with the pre-conditions in the plans. The basic execution cycle is similar to standard reactive planning systems such as PRS [12].

During team plan execution, observations, in the form of communications, often arise because of the coordination actions executed by the TOP interpreter. For instance, TOP interpreters have exploited BDI theories of teamwork, such as Levesque et al.’s theory of joint intentions [22], which require that when an agent comes to privately believe a fact that terminates the current team plan (i.e. matches the achievement or unachievability conditions of a team plan), then it communicates this fact to the rest of the team. By performing such coordination actions automatically, the TOP interpreter enables coherence at the initiation and termination of team plans within a TOP. Some further details and examples of TOPs can be seen in [31, 38, 39]

Figure 7 shows the TOP for the RoboCupRescue scenario. As can be seen, the plan hierarchy for this scenario consists of a pair of **ExtinguishFire** and **RescueCivil-**

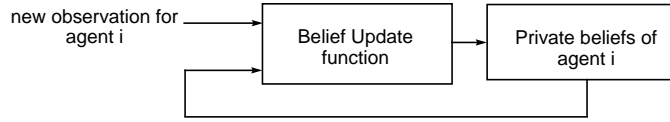


Fig. 6. Mapping of observations to beliefs.

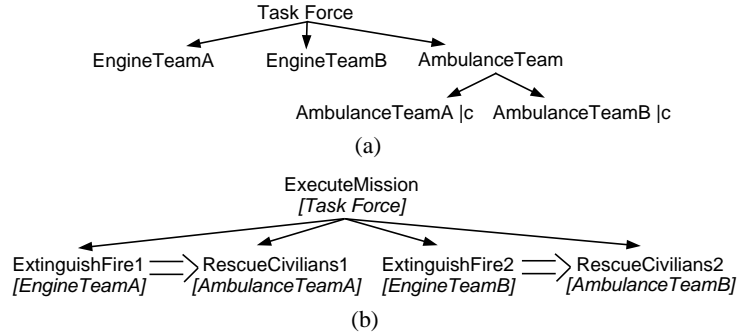


Fig. 7. TOP for RoboCupRescue scenario. a) Organization hierarchy; b) Plan hierarchy.

Plans are done in parallel, each of which further decomposes into individual plans. (These individual plans get the fire engines and ambulances to move through the streets using specific search algorithms. However, these individual plans are not relevant for our discussions in this paper; interested readers should refer to the description of our RoboCupRescue team entered into the RoboCup competitions of 2001 [25].) The organizational hierarchy consists of *Task Force* comprising of two *Engine* sub-teams, one for each fire and an *Ambulance Team*, where the engine teams are assigned to extinguishing the fires while the ambulance team is assigned to rescuing civilians. In this particular TOP, the assignment of ambulances to *AmbulanceTeamA* and *AmbulanceTeamB* is conditioned on the communication “c”. For instance, “*AmbulanceTeamA|c*” is the allocation of ambulances to *AmbulanceTeamA* on receiving communication “c” from the fire engines that describes the number of civilians present at each fire. The problem is which engines to assign to each *Engine Team* and for each possible value of “c”, which ambulances to assign to each *Ambulance Team*. Note that engines have differing capabilities owing to differing distances from fires while all the ambulances have identical capabilities.

3 Role-based Multiagent Team Decision Problem

In order to do quantitative analysis of key coordination decisions in multiagent teams, we extend Multiagent Team Decision Problem (MTDP) [30] for the analysis of the coordination actions of interest. For example, the COM-MTDP [30] is an extension of MTDP for the analysis of communication. In this paper, we illustrate a general method-

ology for analysis of other aspects of coordination and present the RMTDP model for quantitative analysis of role allocation and reallocation as a concrete example. In contrast to BDI systems introduced in the previous section, RMTDP enables explicit quantitative optimization of team performance. Note that, while we use MTDP, other possible distributed POMDP models could potentially also serve as a basis [3, 43].

3.1 Multiagent Team Decision Problem

Given a team of n agents, an MTDP [30] is defined as a tuple: $\langle S, A, P, \Omega, O, R \rangle$. It consists of a finite set of states $S = \Xi_1 \times \dots \times \Xi_m$ where each Ξ_j , $1 \leq j \leq m$, is a feature of the world state. Each agent i can perform an action from its set of actions A_i , where $\times_{1 \leq i \leq n} A_i = A$. $P(s, \langle a_1, \dots, a_n \rangle, s')$ gives the probability of transitioning from state s to state s' given that the agents perform the actions $\langle a_1, \dots, a_n \rangle$ jointly. Each agent i receives an observation $\omega_i \in \Omega_i$ ($\times_{1 \leq i \leq n} \Omega_i = \Omega$) based on the function $O(s, \langle a_1, \dots, a_n \rangle, \omega_1, \dots, \omega_n)$, which gives the probability that the agents receive the observations, $\omega_1, \dots, \omega_n$ given that the world state is s and they perform $\langle a_1, \dots, a_n \rangle$ jointly. The agents receive a single joint reward $R(s, \langle a_1, \dots, a_n \rangle)$ based on the state s and their joint action $\langle a_1, \dots, a_n \rangle$. This joint reward is shared equally by all members and there is no other private reward that individual agents receive for their actions. Thus, the agents are motivated to behave as a team, taking the actions that jointly yield the maximum expected reward.

Each agent i in an MTDP chooses its actions based on its local *policy*, π_i , which is a mapping of its observation history to actions. Thus, at time t , agent i will perform action $\pi_i(\omega_i^0, \dots, \omega_i^t)$. This contrasts with a single-agent POMDP, where we can index an agent's policy by its *belief state* – a probability distribution over the world state [20], which is shown to be a *sufficient statistic* in order to compute the optimal policy [35]. Unfortunately, we cannot directly use single-agent POMDP techniques [20] for maintaining or updating belief states [20] in a MTDP – unlike in a single agent POMDP, in MTDP, an agent's observation depends not only on its own actions, but also on unknown actions of other agents. Thus, as with other distributed POMDP models [3, 43], in MTDP, local policies π_i are indexed by observation histories. $\pi = \langle \pi_1, \dots, \pi_n \rangle$ refers to the joint policy of the team of agents.

3.2 Extension for explicit coordination

Beginning with MTDP, the next step in our methodology is to make an explicit separation between domain-level actions and the coordination actions of interest. Earlier work introduced the COM-MTDP model [30], where the coordination action was fixed to be the communication action, and got separated out. However, other coordination actions could also be separated from domain-level actions in order to investigate their impact. Thus, to investigate role allocation and reallocations, actions for allocating agents to roles and to reallocate such roles are separated out. To that end, we define RMTDP (Role-based Multiagent Team Decision Problem) as a tuple $\langle S, A, P, \Omega, O, R, \mathcal{RL} \rangle$ with a new component, \mathcal{RL} . In particular, $\mathcal{RL} = \{r_1, \dots, r_s\}$ is a set of all roles that the agents can undertake. Each instance of role r_j may be assigned some agent i to fulfill it. The actions of each agent are now distinguishable into two types:

Role-Taking actions: $\mathcal{Y}_i = \{v_{ir_j}\}$ contains the role-taking actions for agent i . $v_{ir_j} \in \mathcal{Y}_i$ means that agent i takes on the role $r_j \in \mathcal{R}\mathcal{L}$.

Role-Execution Actions: $\Phi_i = \bigcup_{r_j \in \mathcal{R}\mathcal{L}} \Phi_{ir_j}$ contains the execution actions for agent i where Φ_{ir_j} is the set of agent i 's actions for executing role $r_j \in \mathcal{R}\mathcal{L}$

In addition we define the set of states as $S = \Xi_1 \times \dots \times \Xi_m \times \Xi_{roles}$, where the feature Ξ_{roles} (a vector) gives the current role that each agent has taken on. The reason for introducing this new feature is to assist us in the mapping from a BDI team plan to an RMTDP. Thus each time an agent performs a new role-taking action successfully, the value of the feature Ξ_{roles} will be updated to reflect this change. The key here is that we not only model an agent's initial role-taking action but also subsequent role reallocation. Modeling both allocation and reallocation is important for an accurate analysis of BDI teams. Note that an agent can observe the part of this feature pertaining to its own current role but it may not observe the parts pertaining to other agents' roles.

The introduction of roles allows us to represent the specialized behaviors associated with each role, e.g. a transport vs. a scout role. While fulfilling a particular role, r_j , agent i can perform only role-execution actions, $\phi \in \Phi_{ir_j}$, which may be different from the role-execution actions Φ_{ir_l} for role r_l . Thus, the feature Ξ_{roles} is used to filter actions such that only those role-execution actions that correspond to the agent's current role are permitted. In the worst case, this filtering does not affect the computational complexity but in practice, it can significantly improve performance when trying to find the optimal policy for the team, since the number of domain actions that each agent can choose from is restricted by the role that the agent has taken on. Also, these different roles can produce varied effects on the world state (modeled via transition probabilities, P) and the team's reward. Thus, the policies must ensure that agents for each role have the capabilities that benefit the team the most.

Just as in MTDP, each agent chooses which action to perform by indexing its local policy π_i by its observation history. In the same epoch some agents could be doing role-taking actions while others are doing role-execution actions. Thus, each agent's local policy π_i can be divided into local role-taking and role-execution policies such that for all observation histories, $\omega_i^0, \dots, \omega_i^t$, either $\pi_{i\mathcal{Y}}(\omega_i^0, \dots, \omega_i^t) = \mathbf{null}$ or $\pi_{i\Phi}(\omega_i^0, \dots, \omega_i^t) = \mathbf{null}$. $\pi_{\mathcal{Y}} = \langle \pi_{1\mathcal{Y}}, \dots, \pi_{n\mathcal{Y}} \rangle$ refers to the joint role-taking policy of the team of agents while $\pi_{\Phi} = \langle \pi_{1\Phi}, \dots, \pi_{n\Phi} \rangle$ refers to the joint role-execution policy. In this paper, we do not explicitly model communicative actions as a special action. Thus communication is treated like any other role-execution action and the communication received from other agents are treated as observations.

Solving the RMTDP for the optimal role-taking policy, assuming that the role-execution policy is fixed, is highly intractable [28]. In general the globally optimal role-taking policy will be of doubly exponential complexity, and so we may be left no choice but to run a brute-force policy search, i.e. to enumerate all the role-taking policies and then evaluate them, which together determine the run-time of finding the globally optimal policy. The number of policies is $\left(|\mathcal{Y}| \frac{|\Omega|^T - 1}{|\Omega| - 1} \right)^n$, i.e. doubly exponential in the number of observation histories and the number of agents. Thus, while RMTDP enables quantitative evaluation of team's policies, computing optimal policies is intractable; furthermore, given its low level of abstraction, in contrast to TOP, it is

difficult for a human to understand the optimal policy. This contrast between RMTDP and TOP is at the root of our hybrid model described in the following section.

4 Hybrid BDI-POMDP approach

Having explained TOP and RMTDP, we can now present a more detailed view of our hybrid methodology to quantitatively evaluate a TOP with the help of Figure 1. We first provide a more detailed interpretation of Figure 1. BDI team plans are essentially TOP plans, while the BDI interpreter is the TOP coordination layer. As shown in Figure 1, an RMTDP model is constructed corresponding to the domain and the TOP and its interpreter are converted into a corresponding (incomplete) RMTDP policy. We can then analyze the TOP using analysis techniques that rely on evaluating the RMTDP policy using the RMTDP model of the domain.

Thus, our hybrid approach combines the strengths of the TOPs (enabling humans to specify TOPs to coordinate large-scale teams) with the strengths of RMTDP (enabling quantitative evaluation of different role allocations). On the one hand, this synergistic interaction enable RMTDPs to improve the performance of TOP-based BDI teams. On the other hand, we have identified at least six specific ways in which TOPs make it easier to build RMTDPs and to efficiently search RMTDP policies: two of which are discussed in this section, and four in the next section. In particular, the six ways are:

1. TOPs are exploited in constructing RMTDP models of the domain (Section 4.1);
2. TOPs are exploited to present incomplete policies to RMTDPs, thus restricting the RMTDP policy search (Section 5.1);
3. TOP belief representation is exploited in enabling faster RMTDP policy evaluation (Section 4.2);
4. TOP organization hierarchy is exploited in hierarchically grouping RMTDP policies (Section 5.1);
5. TOP plan hierarchy is exploited in decomposing RMTDPs (Section 5.2);
6. TOP plan hierarchies are also exploited in cutting down the observation or belief histories in RMTDPs (Section 5.2).

The end result of this efficient policy search is a completed RMTDP policy that improves TOP performance. While we exploit the TOP framework, other frameworks for tasking teams, e.g. Decker and Lesser [9] and Stone and Veloso [36] could benefit from a similar synergistic interaction.

4.1 Guidelines for constructing an RMTDP

As shown in Figure 1, our analysis approach uses as input an RMTDP model of the domain, as well as an incomplete RMTDP policy. Fortunately, not only does the TOP serve as a direct mapping to the RMTDP policy, but it can also be utilized in actually constructing the RMTDP model of the domain. In particular, the TOP can be used to determine which domain features are important to model. In addition, the structure in the TOP can be exploited in decomposing the construction of the RMTDP.

The elements of the RMTDP tuple, $\langle S, A, P, \Omega, O, R, \mathcal{RL} \rangle$, can be defined using a procedure that relies on both the TOP as well as the underlying domain. While this procedure is not automated, our key contribution is recognizing the exploitation of TOP structures in constructing the RMTDP model. First, in order to determine the set of states, S , it is critical to model the variables tested in the pre-conditions, termination conditions and context of all the components (i.e. sub-plans) in the TOP. Note that a state only needs to model the features tested in the TOP; if a TOP pre-condition expresses a complex test on the feature, that test is not modeled in the state, but instead gets used in defining the incomplete policy input to RMTDP. Next we define the set of roles, \mathcal{RL} , as the leaf-level roles in the organization hierarchy of the TOP. Furthermore, as specified in Section 3.2, we define a state feature \vec{x}_{roles} as a vector containing the current role for each agent. Having defined \mathcal{RL} and \vec{x}_{roles} , we now define the actions, A as follows. For each role $r_j \in \mathcal{RL}$, we define a corresponding role-taking action, v_{ir_j} which will succeed or fail depending on the agent i that performs the action and the state s that the action was performed in. The role-execution actions, Φ_{ir_j} for agent i in role r_j , are those allowed for that role according to the TOP.

Thus, we have defined S , A and \mathcal{RL} based on the TOP. To illustrate these steps, consider the plans in Figure 4(b). The pre-conditions of the leaf-level plan **ScoutRoute1**, for instance, tests start location of the helicopters to be at start location X, while the termination conditions test that scouts are at end location Y. Thus, the locations of the helicopters are modeled as features in the set of states in the RMTDP. Using the organization hierarchy, we define the set of roles \mathcal{RL} with a role corresponding to each of the four different kinds of leaf-level roles, i.e. $\mathcal{RL} = \{memberSctTeamA, memberSctTeamB, memberSctTeamC, memberTransportTeam\}$. Using these roles, we can define the role-taking and role-execution actions as follows:

- A role-taking action is defined corresponding to each of the four roles in \mathcal{RL} , i.e. becoming a member of one of the three scouting teams or of the transport team. The domain specifies that only a transport can change to a scout and thus the role-taking action, *joinTransportTeam*, will fail for agent i , if the current role of agent i is a scout.
- Role-execution actions are obtained from the TOP plans corresponding to the agent’s role. In the mission rehearsal scenario, an agent, fulfilling a scout role (members of SctTeamA, SctTeamB or SctTeamC), always goes forward, making the current position safe, until it reaches the destination and so the only execution action we will consider is “move-making-safe”. An agent in a transport role (members of Transport Team) waits at X until it obtains observation of a signal that one scouting sub-team has reached Y and hence the role-execution actions are “wait” and “move-forward”.

We must now define Ω , P , O , R . We obtain the set of observations Ω_i for each agent i directly from the domain. For instance, the transport helos may observe the status of scout helos (normal or destroyed), as well as a signal that a path is safe. Finally, determining the functions, P , O , R requires some combination of human domain expertise and empirical data on the domain behavior. However, as shown later in Section 6, even an approximate model of transitional and observational uncertainty is sufficient to deliver significant benefits. Defining the reward and transition function may sometimes

require additional state variables to be modeled, if they were only implicitly modeled in the TOP. In the mission rehearsal domain, the time at which the scouting and transport mission were completed determined the amount of reward. Thus, time was only implicitly modeled in the TOP and needed to be explicitly modeled in the RMTDP.

Since we are interested in analyzing a particular TOP with respect to uncertainty, the procedure for constructing an RMTDP model can be simplified by exploiting the hierarchical decomposition of the TOP in order to decompose the construction of the RMTDP model. The high-level components of a TOP often represent plans executed by different sub-teams, which may only loosely interact with each other. Within a component, the sub-team members may exhibit a tight interaction, but our focus is on the “loose coupling” across components, where only the end results of one component feed into another, or the components independently contribute to the team goal. Thus, our procedure for constructing an RMTDP exploits this loose coupling between components of the plan hierarchy in order to build an RMTDP model represented as a combination of smaller RMTDPs (factors). Note that if such decomposition is infeasible, our approach still applies except that the benefits of the hierarchical decomposition will be unavailable.

We classify sibling components as being either parallel or sequentially executed (contains a temporal constraint). Components executed in parallel could be either *independent* or *dependent*. For *independent* components, we can define RMTDPs for each of these components such that the sub-team executing one component cannot affect the transitions, observations and reward obtained by the sub-teams executing the other components. The procedure for determining the elements of the RMTDP tuple for component k , $\langle S_k, A_k, P_k, \Omega_k, O_k, R_k, \mathcal{RL}_k \rangle$, is identical to the procedure described earlier for constructing the overall RMTDP. However, each such component has a smaller set of relevant variables and roles and hence specifying the elements of its corresponding RMTDP is easier.

We can now combine the RMTDPs of the independent components to obtain the RMTDP corresponding to the higher-level component. For a higher level component l , whose child components are independent, the set of states, $S_l = \times_{\forall \Xi_x \in F_{S_l}} \Xi_x$ such that $F_{S_l} = \bigcup_{\forall k \text{ s.t. } Child(k,l)=\text{true}} F_{S_k}$ where F_{S_l} and F_{S_k} are the sets of features for the set of states S_l and set of states S_k . A state $s_l \in S_l$ is said to correspond to the state $s_k \in S_k$ if $\forall \Xi_x \in F_{S_k}, s_l[\Xi_x] = s_k[\Xi_x]$, i.e. the state s_l has the same value as state s_k for all features of state s_k . The transition function is defined as follows, $P_l(s'_l, a_l, s_l) = \prod_{\forall k \text{ s.t. } Child(k,l)=\text{true}} P_k(s'_k, a_k, s_k)$, where s_l and s'_l of component l corresponds to states s_k and s'_k of component k and a_k is the joint action performed by the sub-team assigned to component k corresponding to the joint action a_l performed by the sub-team assigned to component l . The observation function is defined similarly as $O_l(s_l, a_l, \omega_l) = \prod_{\forall k \text{ s.t. } Child(k,l)=\text{true}} O_k(s_k, a_k, \omega_k)$. The reward function is defined as $R_l(s_l, a_l) = \sum_{\forall k \text{ s.t. } Child(k,l)=\text{true}} R_k(s_k, a_k)$.

In the case of sequentially executed components (those connected by a temporal constraint), the components are loosely coupled since the end states of the preceding component specify the start states of the succeeding component. Thus, since only one component is active at a time, the transition function is defined as follows, $P_l(s'_l, a_l, s_l) = P_k(s'_k, a_k, s_k)$, where component k is the only active child compo-

ment, s_k and s'_k represent the states of component k corresponding to states s_l and s'_l of component l and a_k is the joint action performed by the sub-team assigned to component k corresponding to the joint action a_l performed by the sub-team corresponding to component l . Similarly, we can define $Q_l(s_l, a_l, \omega_l) = O_k(s_k, a_k, \omega_k)$ and $R_l(s_l, a_l) = R_k(s_k, a_k)$, where k is the only active child component.

Consider the following example from the mission rehearsal domain where components exhibit both sequential dependence and parallel independence. Concretely, **DoScouting** is executed first followed by **DoTransport** and **RemainingScouts**, which are parallel and independent and hence, either **DoScouting** is active or **DoTransport** and **RemainingScouts** are active at any point in the execution. Hence, the transition, observation and reward functions of their parent **Execute Mission** is given by the corresponding functions of either **DoScouting** or by the combination of the corresponding functions of **DoTransport** and **RemainingScouts**.

We use a top-down approach in order to determine how to construct a factored RMTDP from the plan hierarchy, where we replace a particular sub-plan by its constituent sub-plans if they are either independent or sequentially executed. If not, then the RMTDP is defined using that particular sub-plan. This process is applied recursively starting at the root component of the plan hierarchy. As a concrete example, consider again our mission rehearsal simulation domain and the hierarchy illustrated in Figure 4(b). Given the temporal constraints between **DoScouting** and **DoTransport**, and **DoScouting** and **RemainingScouts**, we exploited sequential decomposition, while **DoTransport** and **RemainingScouts** were parallel and independent components. Hence, we can replace **ExecuteMission** by **DoScouting**, **DoTransport** and **RemainingScouts**. We then apply the same process to **DoScouting**. The constituent components of **DoScouting** are neither independent nor sequentially executed and thus **DoScouting** cannot be replaced by its constituent components. Thus, RMTDP for the mission rehearsal domain is comprised of smaller RMTDPs for **DoScouting**, **DoTransport** and **RemainingScouts**.

Thus, using the TOP to identify relevant variables and building a factored RMTDP utilizing the structure of TOP to decompose the construction procedure, reduce the load on the domain expert for model construction. Furthermore, as shown in Section 5.2, this factored model greatly improves the performance of the search for the best role allocation.

4.2 Evaluating RMTDP policies by exploiting TOP beliefs

We now present a technique for exploiting TOPs in speeding up evaluation of RMTDP policies. Before we explain our improvement, we first describe the original algorithm for determining the expected reward of a joint policy, where the local policies of each agent are indexed by its entire observation histories [30, 26]. Here, we obtain an RMTDP policy from a TOP as follows. We obtain $\pi_i(\omega_i^t)$, i.e. the action performed by agent i for each observation history ω_i^t , as the action a performed by the agent i following the TOP when it has a set of privately held beliefs corresponding to the observation history, ω_i^t . We compute the expected reward for the RMTDP policy by projecting the team's execution over all possible branches on different world states and different observations. At each time step, we can compute the expected value of a joint policy, $\pi = \langle \pi_1, \dots$,

π_n \succ , for a team starting in a given state, s^t , with a given set of past observations, $\omega_1^t, \dots, \omega_n^t$, as follows:

$$\begin{aligned}
V_\pi^t(s^t, \langle \omega_1^t, \dots, \omega_n^t \rangle) &= R(s^t, \langle \pi_1(\omega_1^t), \dots, \pi_n(\omega_n^t) \rangle) + \\
&\sum_{s^{t+1} \in S} P(s^t, \langle \pi_1(\omega_1^t), \dots, \pi_n(\omega_n^t) \rangle, s^{t+1}) \cdot \\
&\sum_{\omega_{t+1} \in \Omega} O(s^{t+1}, \langle \pi_1(\omega_1^t), \dots, \pi_n(\omega_n^t) \rangle, \langle \omega_1^{t+1}, \dots, \omega_n^{t+1} \rangle) \cdot \\
&V_\pi^{t+1}(s^{t+1}, \langle \omega_1^{t+1}, \dots, \omega_n^{t+1} \rangle)
\end{aligned} \tag{1}$$

The expected reward of a joint policy π is given by $V_\pi^0(s^0, \langle \mathbf{null}, \dots, \mathbf{null} \rangle)$ where s^0 is the start state. At each time step t , the computation of V_π^t performs a summation over all possible world states and agent observations and so has a time complexity of $O(|S| \cdot |\Omega|)$. This computation is repeated for all states and all observation histories of length t , i.e. $O(|S| \cdot |\Omega|^t)$ times. Therefore, given a time horizon T , the overall complexity of this algorithm is $O(|S|^2 \cdot |\Omega|^{T+1})$.

As discussed in Section 2.2, in a team-oriented program, each agent's action selection is based on just its currently held private beliefs (note that mutual beliefs are modeled as privately held beliefs about all agents as per footnote 2). A similar technique can be exploited when mapping TOP to an RMTDP policy. Indeed, the evaluation of a RMTDP policy that corresponds to a TOP can be speeded up if each agent's local policy is indexed by its private beliefs, ψ_i^t . We refer to ψ_i^t , as the TOP-congruent belief state of agent i in the RMTDP. Note that this belief state is not a probability distribution over the world states as in a single agent POMDP, but rather the privately held beliefs (from the BDI program) of agent i at time t .

Belief-based RMTDP policy evaluation leads to speedup because multiple observation histories map to the same belief state, ψ_i^t . This speedup is a key illustration of exploitation of synergistic interactions of TOP and RMTDP. In this instance, belief representation techniques used in TOP are reflected in RMTDP, and the resulting faster policy evaluation can help us optimize TOP performance. A detailed example of belief state is presented later after a brief explanation of how such belief-based RMTDP policies can be evaluated.

Just as with evaluation using observation histories, we compute the expected reward of a belief-based policy by projecting the team's execution over all possible branches on different world states and different observations. At each time step, we can compute the expected value of a joint policy, $\pi = \langle \pi_1, \dots, \pi_n \rangle$, for a team starting in a given state, s^t , with a given team belief state, $\langle \psi_1^t, \dots, \psi_n^t \rangle$ as follows:

$$\begin{aligned}
V_{\pi}^t(s^t, \langle \psi_1^t \dots \psi_n^t \rangle) &= R(s^t, \langle \pi_1(\psi_1^t), \dots, \pi_n(\psi_n^t) \rangle) + \\
&\sum_{s^{t+1} \in S} P(s^t, \langle \pi_1(\psi_1^t), \dots, \pi_n(\psi_n^t) \rangle, s^{t+1}) \cdot \\
&\sum_{\omega_{i+1} \in \Omega} O(s^{t+1}, \langle \pi_1(\psi_1^t), \dots, \pi_n(\psi_n^t) \rangle, \langle \omega_1^{t+1}, \dots, \omega_n^{t+1} \rangle) \cdot \\
&V_{\pi}^{t+1}(s^{t+1}, \langle \psi_1^{t+1}, \dots, \psi_n^{t+1} \rangle)
\end{aligned} \tag{2}$$

where $\psi_i^{t+1} = \mathbf{BeliefUpdateFunction}(\psi_i^t, \omega_i^{t+1})$

The complexity of computing this function (expression 2) is $O(|S| \cdot |\Omega|) \cdot O(\mathbf{BeliefUpdateFunction})$. At each time step the computation of the value function is done for every state and for all possible reachable belief states. Let $|\Psi_i| = \max_{1 \leq t \leq T}(|\psi_i^t|)$ represent the maximum number of possible belief states that agent i can be in at any point in time, where $|\psi_i^t|$ is the number of belief states that agent i can be in at t . Therefore the complexity of this algorithm is given by $O(|S|^2 \cdot |\Omega| \cdot (|\Psi_1| \cdot \dots \cdot |\Psi_n|) \cdot T) \cdot O(\mathbf{BeliefUpdateFunction})$. Note that, in this algorithm T is not in the exponent unlike in the algorithm in expression 1. Thus, this evaluation method will give large time savings if: (i) the quantity $(|\Psi_1| \cdot \dots \cdot |\Psi_n|) \cdot T$ is much less than $|\Omega|^T$ and (ii) the belief update cost is low. In practical BDI systems, multiple observation histories map often onto the same belief state, and thus usually, $(|\Psi_1| \cdot \dots \cdot |\Psi_n|) \cdot T$ is much less than $|\Omega|^T$. Furthermore, since the belief update function mirrors practical BDI systems, its complexity is also a low polynomial or a constant. Indeed, our experimental results show that significant speedups result from switching to our TOP-congruent belief states ψ_i^t . However, in the absolute worst case, the belief update function may simply append the new observation to the history of past observations (i.e., TOP-congruent beliefs will be equivalent to keeping entire observation histories) and thus belief-based evaluation will have the same complexity as the observation history-based evaluation.

We now turn to an example of belief-based policy evaluation from the mission rehearsal domain. At each time step, the transport helicopters may receive an observation about whether a scout has failed based on some observation function. If we use the observation-history representation of the policy, then each transport agent would maintain a complete history of the observations that it could receive at each time step. For example, in a setting with two scout helicopters, one on route 1 and the other on route 2, a particular transport helicopter may have several different observation histories of length two. At every time step, the transports may receive an observation about each scout being alive or having failed. Thus, at time $t = 2$, a transport helicopter might have one of the following observation histories of length two, $\langle \{sct1OnRoute1Alive, sct2OnRoute2Alive\}^1, \{sct1OnRoute1Failed, sct2OnRoute2Failed\}^2 \rangle, \langle \{sct1OnRoute1Alive, sct2OnRoute2Failed\}^1, \{sct1OnRoute1Failed, sct2OnRoute2Failed\}^2 \rangle, \langle \{sct1OnRoute1Failed, sct2OnRoute2Alive\}^1, \{sct2OnRoute2Failed\}^2 \rangle$, etc. However, the action selection of the transport helicopters depends on only whether a critical failure (i.e. the last remaining scout has crashed) has taken place to change its role. Whether a failure is

critical can be determined by passing each observation through a belief-update function. The exact order in which the observations are received or the precise times at which the failure or non-failure observations are received are not relevant to determining if a critical failure has taken place and consequently whether a transport should change its role to a scout. Thus, many observation histories map onto the same belief states. For example, the above three observation histories all map to the same belief *CriticalFailure(DoScouting)* i.e. a critical failure has taken place. This results in significant speedups using belief-based evaluation, as Equation 2 needs to be executed over a smaller number of belief states, linear in T in our domains, as opposed to the observation history-based evaluation, where Equation 1 is executed over an exponential number of observation histories ($|\Omega|^T$). The actual speedup obtained in the mission rehearsal domain is demonstrated empirically in Section 6.

5 Optimizing role allocation

While Section 4 focused on mapping a domain of interest onto RMTDP and algorithms for policy evaluation, this section focuses on efficient techniques for RMTDP policy search, in service of improving BDI/TOP team plans. The TOP in essence provides an incomplete, fixed policy, and the policy search optimizes decisions left open in the incomplete policy; the policy thus completed optimizes the original TOP (see Figure 1). By enabling the RMTDP to focus its search on incomplete policies, and by providing ready-made decompositions, TOPs assist RMTDPs in quickly searching through the policy space, as illustrated in this section. We focus, in particular, on the problem of role allocation [18, 24, 40, 11], a critical problem in teams. While the TOP provides an incomplete policy, keeping open the role allocation decision for each agent, the RMTDP policy search provides the optimal role-taking action at each of the role allocation decision points. In contrast to previous role allocation approaches, our approach determines the best role allocation, taking into consideration the uncertainty in the domain and future costs. Although demonstrated for solving the role allocation problem, the methodology is general enough to apply to other coordination decisions.

5.1 Hierarchical grouping of RMTDP policies

As mentioned earlier, to address role allocation, the TOP provides a policy that is complete, except for the role allocation decisions. RMTDP policy search then optimally fills in the role allocation decisions. To understand the RMTDP policy search, it is useful to gain an understanding of the role allocation search space. First, note that role allocation focuses on deciding how many and what types of agents to allocate to different roles in the organization hierarchy. This role allocation decision may be made at time $t = 0$ or it may be made at a later time conditioned on available observations. Figure 8 shows a partially expanded role allocation space defined by the TOP organization hierarchy in Figure 4(a) for six helicopters. Each node of the role allocation space completely specifies the allocation of agents to roles at the corresponding level of the organization hierarchy (ignore for now, the number to the right of each node). For instance, the root node of the role allocation space specifies that six helicopters are assigned to the *Task*

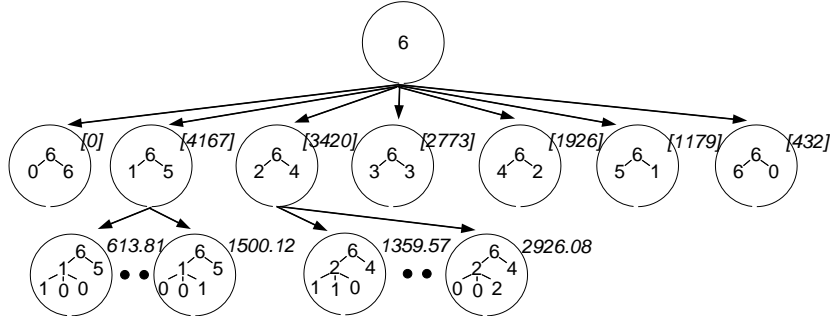


Fig. 8. Partially expanded role allocation space for mission rehearsal domain(six helos).

Force (level one) of the organization hierarchy while the leftmost leaf node (at level three) in Figure 8 specifies that one helicopter is assigned to *SctTeamA*, zero to *SctTeamB*, zero to *SctTeamC* and five helicopters to *Transport Team*. Thus, as we can see, each leaf node in the role allocation space is a complete, valid role allocation of agents to roles in the organization hierarchy.

In order to determine if one leaf node (role allocation) is superior to another we evaluate each using the RMTDP by constructing an RMTDP policy for each. In this particular example, the role allocation specified by the leaf node corresponds to the role-taking actions that each agent will execute at time $t = 0$. For example, in the case of the leftmost leaf in Figure 8, at time $t = 0$, one agent (recall from Section 2.2 that this is a homogeneous team and hence which specific agent does not matter) will become a member of *SctTeamA* while all other agents will become members of *Transport Team*. Thus, for one agent i , the role-taking policy will include $\pi_{i\mathcal{R}}(\mathbf{null}) = \text{joinSctTeamA}$ and for all other agents, $j, j \neq i$, it will include $\pi_{j\mathcal{R}}(\mathbf{null}) = \text{joinTransportTeam}$. In this case, we assume that the rest of the role-taking policy, i.e. how roles will be reallocated if a scout fails, is obtained from the role reallocation algorithm in the BDI/TOP interpreter, such as the *STEAM* algorithm [38]. Thus for example, if the role reallocation is indeed performed by the *STEAM* algorithm, then *STEAM*'s reallocation policy is included into the incomplete policy that the RMTDP is initially provided. Thus, the best role allocation is computed keeping in mind *STEAM*'s reallocation policy. In *STEAM*, given a failure of an agent playing $Role_F$, an agent playing $Role_R$ will replace it if:

$$\begin{aligned} &Criticality(Role_F) - Criticality(Role_R) > 0 \\ &Criticality(x) = 1 \text{ if } x \text{ is critical; } = 0 \text{ otherwise} \end{aligned}$$

Thus, if based on the agents' observations, a critical failure has taken place, then the replacing agent's decision to replace or not will be computed using the above expression and then included in the incomplete policy input to the RMTDP. Since such an incomplete policy is completed by the role allocation at each leaf node using the technique above, we have been able to construct a policy for the RMTDP that corresponds to the role allocation.

We are thus able to exploit the TOP organization hierarchy to create a hierarchical grouping of RMTDP policies. In particular, while the leaf node represents a complete

RMTDP policy (with the role allocation as specified by the leaf node), a parent node represents a group of policies. Evaluating a policy specified by a leaf node is equivalent to evaluating a specific role allocation while taking future uncertainties into account. We could do a brute force search through all role allocations, evaluating each in order to determine the best role allocation. However, the number of possible role allocations is exponential in the leaf roles in the organization hierarchy. Thus, we must prune the search space.

5.2 Pruning the role allocation space

We prune the space of valid role allocations using upper bounds (MaxEstimates) for the parents of the leaves of the role allocation space as admissible heuristics. Each leaf in the role allocation space represents a completely specified policy and the MaxEstimate is an upper bound of maximum value of all the policies under the same parent node evaluated using the RMTDP. Once we obtain MaxEstimates for all the parent nodes (shown in brackets to the right of each parent node in Figure 8), we use branch-and-bound style pruning. The key novelty of this branch-and-bound algorithm, which we discuss below, is how we exploit the structure of the TOP in order to compute upper bounds for parent nodes. The other details of this algorithm can be found in [28].

We will now discuss how the upper bounds of parents, called MaxEstimates, can be calculated for each parent. The MaxEstimate of a parent is defined as a strict upper bound of the maximum of the expected reward of all the leaf nodes under it. It is necessary that the MaxEstimate be an upper bound or else we might end up pruning potentially useful role allocations. In order to calculate the MaxEstimate of each parent we could evaluate each of the leaf nodes below it using the RMTDP, but this would nullify the benefit of any subsequent pruning. We, therefore, turn to the TOP plan hierarchy (see Figure 4(b)) to break up this evaluation of the parent node into components, which can be evaluated separately thus decomposing the problem. In other words, our approach exploits the structure of the BDI program to construct small-scale RMTDPs unlike other decomposition techniques which just assume decomposition or ultimately rely on domain experts to identify interactions in the agents' reward and transition functions [8, 15].

For each parent in the role allocation space, we use these small-scale RMTDPs to evaluate the values for each TOP component. Fortunately, as discussed in Section 4.1, we exploited small-scale RMTDPs corresponding to TOP components in constructing larger scale RMTDPs. We put these small-scale RMTDPs to use again, evaluating policies within each component to obtain upper bounds. Note that just like in evaluation of leaf-level policies, the evaluation of components for the parent node can be done using either the observation histories (see Equation 1) or belief states (see Equation 2). We will describe this section using the observation history-based evaluation method for computing the values of the components of each parent, which can be summed up to obtain its MaxEstimate (an upper bound on its children's values). Thus, whereas a parent in the role allocation space represents a group of policies, the TOP components (sub-plans) allow a component-wise evaluation of such a group to obtain an upper bound on the expected reward of any policy within this group.

Algorithm 1 exploits the smaller-scale RMTDP components, discussed in Section 4.1, to obtain upper bounds of parents. First, in order to evaluate the MaxEstimate for each parent node in the role allocation space, we identify the start states for each component from which to evaluate the RMTDPs. We explain this step using a parent node from Figure 8 – *Scouting Team* = two helos, *Transport Team* = four helos (see Figure 9). For the very first component which does not have any preceding components, the start states corresponds to the start states of the policy that the TOP was mapped onto. For each of the next components – where the next component is one linked by a sequential dependence – the start states are the end states of the preceding component. However, as explained later in this section, we can significantly reduce this list of start states from which each component can be evaluated.

Algorithm 1 MAXEXP method for calculating upper bounds for parents in the role allocation space.

```

1: for all parent in search space do
2:   MAXEXP[parent]  $\leftarrow$  0
3:   for all component  $i$  corresponding to factors in the RMTDP from Section 4.1 do
4:     if component  $i$  has a preceding component  $j$  then
5:       Obtain start states,  $states[i] \leftarrow endStates[j]$ 
6:        $states[i] \leftarrow \text{removeIrrelevantFeatures}(states[i])$  {discard features not
           present in  $S_i$ }
7:       Obtain corresponding observation histories at start  $OHistories[i] \leftarrow$ 
            $endOHistories[j]$ 
8:        $OHistories[i] \leftarrow \text{removeIrrelevantObservations}(OHistories[i])$ 
9:     else
10:      Obtain start states,  $states[i]$ 
11:      Observation histories at start  $OHistories[i] \leftarrow \text{null}$ 
12:       $maxEval[i] \leftarrow 0$ 
13:      for all leaf-level policies  $\pi$  under parent do
14:         $maxEval[i] \leftarrow \max(maxEval[i], \max_{s_i \in states[i], oh_i \in OHistories[i]}($ 
            $Evaluate(RMTDP_i, s_i, oh_i, \pi)))$ 
15:   MAXEXP[parent]  $\leftarrow^+ maxEval[i]$ 

```

Similarly, the starting observation histories for a component are the observation histories on completing the preceding component (no observation history for the very first component). BDI plans do not normally refer to entire observation histories but rely only on key beliefs which are typically referred to in the pre-conditions of the component. Each starting observation history can be shortened to include only these relevant observations, thus obtaining a reduced list of starting observation sequences. Divergence of private observations is not problematic, e.g. will not cause agents to trigger different team plans. This is because as indicated earlier in Section 2.2, TOP interpreters guarantee coherence in key aspects of observation histories. For instance, as discussed earlier, TOP interpreter ensures coherence in key beliefs when initiating and terminating team plans in a TOP; thus avoiding such divergence of observation histories.

In order to compute the maximum value for a particular component, we evaluate all possible leaf-level policies within that component over all possible start states and observation histories and obtain the maximum (Algorithm 1:steps 13-14). During this evaluation, we store all the end states and ending observation histories so that they can be used in the evaluation of subsequent components. As shown in Figure 9, for the evaluation of **DoScouting** component for the parent node where there are two helicopters assigned to *Scouting Team* and four helos to *Transport Team*, the leaf-level policies correspond to all possible ways these helicopters could be assigned to the teams *SctTeamA*, *SctTeamB*, *SctTeamC* and *Transport Team*, e.g. one helo to *SctTeamB*, one helo to *SctTeamC* and four helos to *Transport Team*, or two helos to *SctTeamA* and four helos to *Transport Team*, etc. The role allocation tells the agents what role to take in the first step. The remainder of the role-taking policy is specified by the role replacement policy in the TOP infrastructure and role-execution policy is specified by the **DoScouting** component of the TOP.

To obtain the MaxEstimate for a parent node of the role allocation space, we simply sum up the maximum values obtained for each component (Algorithm 1:steps 15), e.g. the maximum values of each component (see right of each component in Figure 9) were summed to obtain the MaxEstimate ($84 + 3330 + 36 = 3420$). As seen in Figure 8, third node from the left indeed has an upper bound of 3420.

The calculation of the MaxEstimate for a parent nodes should be much faster than evaluating the leaf nodes below it in most cases for two reasons. Firstly, parent nodes are evaluated component-wise. Thus, if multiple leaf-level policies within one component result in the same end state, we can remove duplicates to get the start states of the next component. Since each component only contains the state features relevant to it, the number of duplicates is greatly increased. Such duplication of the evaluation effort cannot be avoided for leaf nodes, where each policy is evaluated independently from start to finish. For instance, in the **DoScouting** component, the role allocation, *SctTeamA=1*, *SctTeamB=1*, *SctTeamC=0*, *TransportTeam=4* and the role allocation *SctTeamA=1*, *SctTeamB=0*, *SctTeamC=1*, *TransportTeam=4* will have end states in common after eliminating irrelevant features when the scout in *SctTeamB* for the former allocation and the scout in *SctTeamC* for the latter allocation fail. This is because through feature elimination (Algorithm 1:steps 6), the only state features retained for the **DoTransport** component are the scouted route and number of transports (some transports may have replaced failed scouts) as shown in Figure 9.

The second reason computation of MaxEstimates for parents is much faster is that the number of starting observation sequences will be much less than the number of ending observation histories of the preceding components. This is because not all the observations in the observation histories of a component are relevant to its succeeding components (Algorithm 1:steps 8). Thus, the function **removeIrrelevantObservations** reduces the number of starting observation histories from the observation histories of the preceding component.

We refer to this methodology of obtaining the MaxEstimates of each parent as MAXEXP. A variation of this, the maximum expected reward with no failures (NO-FAIL), is obtained in a similar fashion except that we assume that the probability of any agent failing is 0. We are able to make such an assumption in evaluating the parent

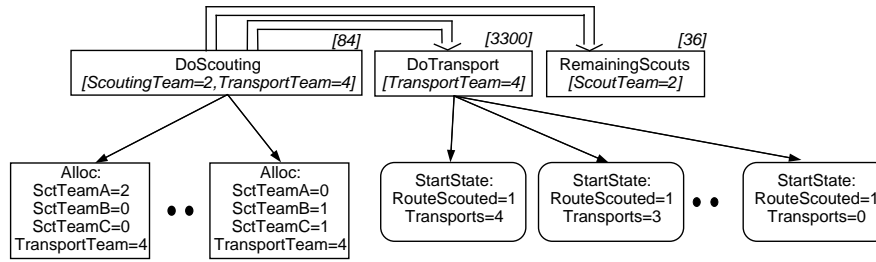


Fig. 9. Component-wise decomposition of a parent by exploiting TOP.

node, since we focus on obtaining upper bounds of parents, and not on obtaining their exact value. This will result in less branching and hence evaluation of each component will proceed much quicker. The NOFAIL heuristic only works if the evaluation of any policy without failures occurring is higher than the evaluation of the same policy with failures possible. This should normally be the case in most domains. The evaluation of the NOFAIL heuristics for the role allocation space for six helicopters is shown in square brackets in Figure 8.

6 Experimental Results

This section presents four sets of results in the context of the two domains introduced in Section 2.1, viz. mission rehearsal and RoboCupRescue [21]. First, we investigated empirically the speedups that result from using the TOP-congruent belief states ψ_i (belief-based evaluation) over observation history-based evaluation and from using the algorithm from Section 5 over a brute-force search. Here we focus on determining the best assignment of agents to roles; but assume a fixed TOP and TOP infrastructure. Second, we conducted experiments to investigate the benefits of considering uncertainty in determining role allocations. For this, we compared the allocations found by the RMTDP role allocation algorithm with (i) allocations which do not consider any kind of uncertainty, and (ii) allocations which do not consider observational uncertainty but consider action uncertainty. Third, we conducted experiments in both domains to determine the sensitivity of the results to changes in the model. Fourth, we compare the performance of allocations found by the RMTDP role allocation algorithm with allocations of human subjects in the more complex of our domains – RoboCupRescue simulations.

6.1 Results in Mission Rehearsal Domain

For the mission rehearsal domain, the TOP is the one discussed in Section 2.2. As can be seen in Figure 4(a), the organization hierarchy requires determining the number of agents to be allocated to the three scouting sub-teams and the remaining helos must be allocated to the transport sub-team. Different numbers of initial helicopters were attempted, varying from three to ten. The probability of failure of a scout at each time step

on routes 1, 2 and 3 are 0.1, 0.15 and 0.2, respectively. The probability of a transport observing an alive scout on routes 1, 2 and 3 are 0.95, 0.94 and 0.93, respectively. False positives are not possible, i.e. a transport will not observe a scout as being alive if it has failed. The probability of a transport observing a scout failure on routes 1, 2 and 3 are 0.98, 0.97 and 0.96, respectively. Here too, false positives are not possible and hence a transport will not observe a failure unless it has actually taken place.

Figure 10 shows the results of comparing the different methods for searching the role allocation space. We show four methods. Each method adds new speedup techniques to the previous:

1. NOPRUNE-OBS: A brute force evaluation of every role allocation to determine the best. Here, each agent maintains its complete observation history and the evaluation algorithm in Equation 1 is used. For ten agents, the RMTDP is projected to have in the order of 10,000 reachable states and in the order of 100,000 observation histories per role allocation evaluated (thus the largest experiment in this category was limited to seven agents).
2. NOPRUNE-BEL: A brute force evaluation of every role allocation. The only difference between this method and NOPRUNE-OBS is the use of the belief-based evaluation algorithm (see Equation 2).
3. MAXEXP: The branch-and-bound search algorithm described in Section 5.2 that uses upper bounds of the evaluation of the parent nodes to find the best allocation. Evaluation of the parent and leaf nodes uses the belief-based evaluation.
4. NOFAIL: The modification to branch-and-bound heuristic mentioned in Section 5.2. In essence it is same as MAXEXP, except that the upper bounds are computed making the assumption that agents do not fail. This heuristic is correct in those domains where the total expected reward with failures is always less than if no failures were present and will give significant speedups if agent failures is one of the primary sources of stochasticity. In this method, too, the evaluation of the parent and leaf nodes uses the belief-based evaluation. (Note that only upper bounds are computed using the no-failure assumption – no changes are assumed in the actual domains.)

In Figure 10(a), the Y-axis is the number of nodes in the role allocation space evaluated (includes leaf nodes as well as parent nodes), while in Figure 10(b) the Y-axis represents the runtime in seconds on a *logarithmic* scale. In both figures, we vary the number of agents on the X-axis. Experimental results in previous work using distributed POMDPs are often restricted to just two agents; by exploiting hybrid models, we are able to vary the number of agents from three to ten as shown in Figure 10(a). As clearly seen in Figure 10(a), because of pruning, significant reductions are obtained by MAXEXP and NOFAIL over NOPRUNE-BEL in terms of the numbers of nodes evaluated. This reduction grows quadratically to about 10-fold at ten agents.¹ NOPRUNE-OBS is identical to NOPRUNE-BEL in terms of number of nodes evaluated, since in both methods all the leaf-level policies are evaluated, only the method of evaluation differs.

¹ The number of nodes for NOPRUNE up to eight agents were obtained from experiments, the rest can be calculated using the formula $[m]^n/n! = (m+n-1) \cdot \dots \cdot m/n!$, where m represents the number of heterogeneous role types and n is the number of homogeneous agents. $[m]^n = (m+n-1) \cdot \dots \cdot m$ is referred to as a rising factorial.

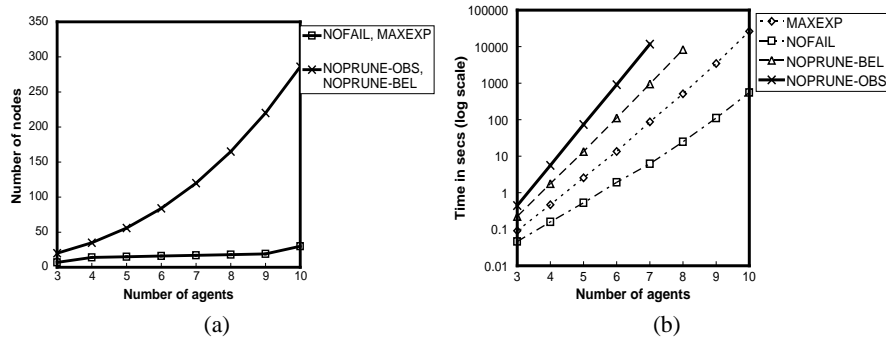


Fig. 10. Performance of role allocation space search in mission rehearsal domain, a) Number of nodes evaluated, b) Run-time in seconds on a log scale.

It is important to note that although NOFAIL and MAXEXP result in the same number of nodes being evaluated for this domains, this is not necessarily true always. In general, NOFAIL will evaluate at least as many nodes as MAXEXP since its estimate is at least as high as the MAXEXP estimate. However, the upper bounds are computed quicker for NOFAIL.

Figure 10(b) shows that the NOPRUNE-BEL method provides a significant speedup over NOPRUNE-OBS in actual run-time. For instance, there was a 12-fold speedup using NOPRUNE-BEL instead of NOPRUNE-OBS for the seven agent case (NOPRUNE-OBS could not be executed within a day for problem settings with greater than seven agents). This empirically demonstrates the computational savings possible using belief-based evaluation instead of observation history-based evaluation (see Section 4). For this reason, we use only belief-based evaluation for the MAXEXP and NOFAIL approaches and also for all the remaining experiments in this paper. MAXEXP heuristic results in a 16-fold speedup over NOPRUNE-BEL in the eight agent case.

The NOFAIL heuristic which is very quick to compute the upper bounds far outperforms the MAXEXP heuristic (47-fold speedup over MAXEXP for ten agents). Speedups of MAXEXP and NOFAIL continually increase with increasing number of agents. The speedup of the NOFAIL method over MAXEXP is so marked because, in this domain, ignoring failures results in much less branching.

Next, we conducted experiments illustrating the importance of RMTDP's reasoning about action and observation uncertainties on role allocations. For this, we compared the allocations found by the RMTDP role allocation algorithm with allocations found using two different methods (see Figure 11):

1. Role allocation via constraint optimization (COP [24, 23]) allocation approach: In the COP approach², leaf-level sub-teams from the organization hierarchy are treated as variables and the number of helicopters as the domain of each such vari-

² Modi et al.'s work [24] focused on decentralized COP, but in this investigation our emphasis is on the resulting role allocation generated by the COP, and not on the decentralization per se.

able (thus, the domain may be 1, 2, 3,..helicopters). The reward for allocating agents to sub-teams is expressed in terms of constraints:

- Allocating a helicopter to scout a route was assigned a reward corresponding to the route’s distance but ignoring the possibility of failure (i.e. ignoring transition probability). Allocating more helicopters to this subteam obtained proportionally higher reward.
 - Allocating a helicopter a transport role was assigned a large reward for transporting cargo to the destination. Allocating more helicopters to this subteam obtained proportionally higher reward.
 - Not allocating at least one scout role was assigned a reward of negative infinity
 - Exceeding the total number of agents was assigned a reward of negative infinity
2. RMTDP with complete observability: In this approach, we consider the transition probability, but ignore partial observability; achieved by assuming complete observability in the RMTDP. An MDP with complete observability is equivalent to a Markov Decision Problem (MDP) [30] where the actions are joint actions. We, thus, refer to this allocation method as the MDP method.

Figure 11(a) shows a comparison of the RMTDP-based allocation with the MDP allocation and the COP allocation for increasing number of helicopters (X-axis). We compare using the expected number of transports that get to the destination (Y-axis) as the metric for comparison since this was the primary objective of this domain. As can be seen, considering both forms of uncertainty (RMTDP) performs better than just considering transition uncertainty (MDP) which in turn performs better than not considering uncertainty (COP). Figure 11(b) shows the actual allocations found by the three methods with four helicopters and with six helicopters. In the case of four helicopters (first three bars), RMTDP and MDP are identical, two helicopters scouting route 2 and two helicopters taking on transport role. The COP allocation however consists of one scout on route 3 and three transports. This allocation proves to be too myopic and results in fewer transports getting to the destination safely. In the case of six helicopters, COP chooses just one scout helicopter on route 3, the shortest route. The MDP approach results in two scouts both on route 1, which was longest route albeit the safest. The RMTDP approach, which also considers observational uncertainty choose to allocate the two scouts to route 1 and route 2, in order to take care of the cases where failures of scouts go undetected by the transports.

6.2 Results in RoboCupRescue Domain

Speedups in RoboCupRescue Domain In our next set of experiments, we highlight the computational savings obtained in the RoboCupRescue domain. The scenario for this experiment consisted of two fires at different locations in the city. Each of these fires has a different initially unknown number of civilians in it, however the total number of civilians and the distribution from which the locations of the civilians is chosen is known ahead of time. For this experiment, we fix the number of civilians at five and set the distribution used to choose the civilians’ locations to be uniform. The number of fire engines is set at five, located in three different fire stations as described in Section 2.1 and vary the number of ambulances, all co-located at an ambulance center, from two

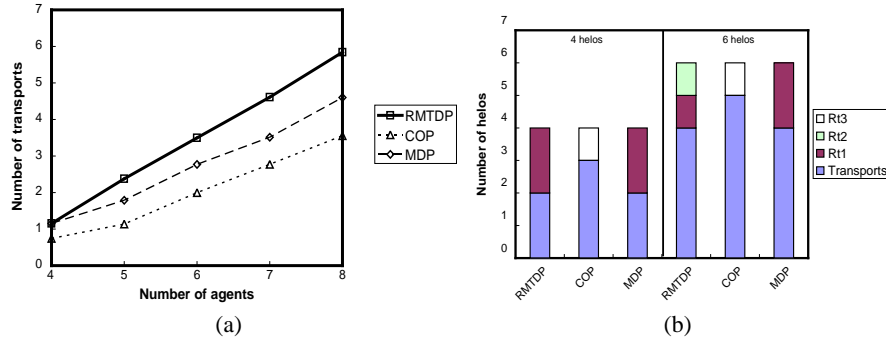


Fig. 11. a) Comparison of performance of different allocation methods, b) Allocations found using different allocation methods.

to seven. The reason we chose to change only the number of ambulances is because small number of fire engines are unable to extinguish fires, changing the problem completely. The goal is to determine which fire engines to allocate to which fire and once information about civilians is transmitted, how many ambulances to send to each fire location.

Figure 12 highlights the savings in terms of the number of nodes evaluated and the actual runtime as we increase the number of agents. We show results only from NOPRUNE-BEL and MAXEXP. NOPRUNE-OBS could not be run because of slowness. Here the NOFAIL heuristic is identical to MAXEXP since agents cannot fail in this scenario. The RMTDP in this case had about 30,000 reachable states.

In both Figures 12(a) and 12(b), we increase the number of ambulances along the X-axis. In Figure 12(a), we show the number of nodes evaluated (parent nodes + leaf nodes)³ on a logarithmic scale. As can be seen, the MAXEXP method results in about a 89-fold decrease in the number of nodes evaluated when compared to NOPRUNE-BEL for seven ambulances, and this decrease becomes more pronounced as the number of ambulances is increased. Figure 12(b) shows the time in seconds on a logarithmic scale on the Y-axis and compares the run-times of the MAXEXP and NOPRUNE-BEL methods for finding the best role allocation. The NOPRUNE-BEL method could not find the best allocation within a day when the number of ambulances was increased beyond four. For four ambulances (and five fire engines), MAXEXP resulted in about a 29-fold speedup over NOPRUNE-BEL.

³ The number of nodes evaluated using NOPRUNE-BEL can be computed as $(f_1 + 1) \cdot (f_2 + 1) \cdot (f_3 + 1) \cdot (a + 1)^{c+1}$, where f_1 , f_2 and f_3 are the number of fire engines at station 1, 2 and 3, respectively, a is the number of ambulances and c is the number of civilians. Each node provides a complete conditional role allocation, assuming different numbers of civilians at each fire station.

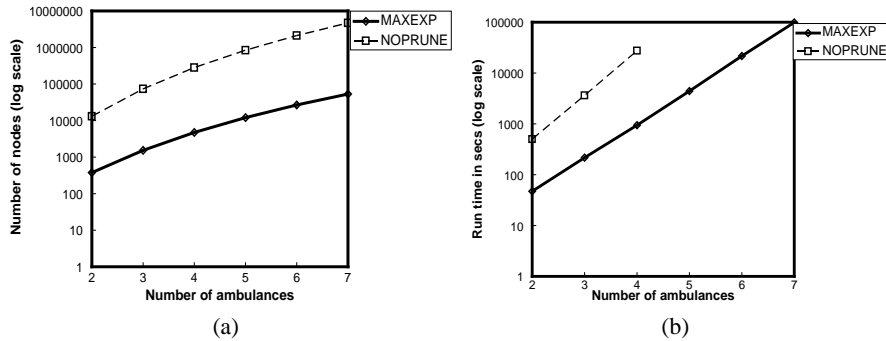


Fig. 12. Performance of role allocation space search in RoboCupRescue, a) Number of nodes evaluated on a log scale, and b) Run-time in seconds on a log scale.

Allocation in RoboCupRescue Our next set of experiments shows the practical utility of our role allocation analysis in complex domains. We are able to show significant performance improvements in the actual RoboCupRescue domain using the role allocations generated by our analysis. First, we construct an RMTDP for the rescue scenario, described in Section 2.1, by taking guidance from the TOP and the underlying domain (as described in Section 4.1). We then use the MAXEXP heuristic to determine the best role allocation. We compared the RMTDP allocation with the allocations chosen by human subjects. Our goal in comparing RMTDP allocations with human subjects was mainly to show that RMTDP is capable at performing at or near human expert levels for this domain. In addition, in order to determine that reasoning about uncertainty actually impacts the allocations, we compared the RMTDP allocations with allocations determined by two additional allocation methods:

1. RescueISI: Allocations used by the our RoboCupRescue agents that were entered in the RoboCupRescue competitions of 2001 [25] (RescueISI), where they finished in third place. These agents used local reasoning for their decision making, ignoring transitional as well and observational uncertainty.
2. RMTDP with complete observability: As discussed earlier, complete observability in RMTDP leads to an MDP, and we refer to this method as the MDP method.

Note that these comparisons were performed using the RoboCupRescue simulator with multiple runs to deal with stochasticity⁴. The scenario is as described in Section 6.2. We fix the number of fire engines, ambulances and civilians at five each. For this experiment, we consider two settings, where the location of civilians is drawn from:

- Uniform distribution – 25% of the cases have four civilians at fire 1 and one civilian at fire 2, 25% with three civilians at fire 1 and two at fire 2, 25% with two civilians

⁴ For the mission rehearsal domain, we could not run on the actual mission rehearsal simulator since that simulator is not public domain and no longer accessible, and hence the difference in how we tested role allocations in the mission rehearsal and the RoboCupRescue domains.

at fire 1 and three at fire 2 and the remaining 25% with one civilian at fire 1 and four civilians at fire 2. The speedup results of Section 6.2 were obtained using this distribution.

- Skewed distribution – 80% of the cases have four civilians at fire 1 and one civilian at fire 2 and the remaining 20% with one civilian at fire 1 and four civilians at fire 2.

Note that we do not consider the case where all civilians are located at the same fire as the optimal ambulance allocation is simply to assign all ambulances to the fire where the civilians are located. A skewed distribution was chosen to highlight the cases where it becomes difficult for humans to reason about what allocation to choose.

The three human subjects used in this experiment were researchers at USC. All three were familiar with RoboCupRescue. They were given time to study the setup and were not given any time limit to provide their allocations. Each subject was told that the allocations were going to be judged first on the basis of the number of civilian lives lost and next on the damage sustained due to fire. These are exactly the criteria used in RoboCupRescue [21].

We then compared “RMTDP” allocation with those of the human subjects in the RoboCupRescue simulator and with RescueISI and MDP. In Figure 13, we compared the performance of the allocations on the basis of the number of civilians who died and the average damage to the two buildings (lower values are better for both criteria). These two criteria are the main two criteria used in RoboCupRescue [21]. The values shown in Figure 13 were obtained by averaging forty simulator runs for the uniform distribution and twenty runs for the skewed distribution for each allocation. The average values were plotted to account for the stochasticity in the domain. Error bars are provided to show the standard error for each allocation method.

As can be seen in Figure 13(a), the RMTDP allocation did better than the other five allocations in terms of a lower number of civilians dead (although human3 was quite close). For example, averaging forty runs, the RMTDP allocation resulted in 1.95 civilian deaths while human2’s allocation resulted in 2.55 civilian deaths. In terms of the average building damage, the six allocations were almost indistinguishable, with the humans actually performing marginally better. Using the skewed distribution, the difference between the allocations was much more perceptible (see Figure 13(b)). In particular, we notice how the RMTDP allocation does much better than the humans in terms of the number of civilians dead. Here, human3 did particularly badly because of a bad allocation for fire engines. This resulted in more damage to the buildings and consequently to the number of civilians dead.

Comparing RMTDP with RescueISI and the MDP approach showed that reasoning about transitional uncertainty (MDP) does better than a static reactive allocation method (RescueISI) but not as well as reasoning about both transitional and observational uncertainty. In the uniform distribution case, we found that RMTDP does better than both MDP and RescueISI, with the MDP method performing better than RescueISI. In the skewed distribution case, the improvement in allocations using RMTDP is greater. Averaging twenty simulation runs, RMTDP allocations resulted in 1.54 civilian deaths while MDP resulted in 1.98 and RescueISI in 3.52. The allocation method used by

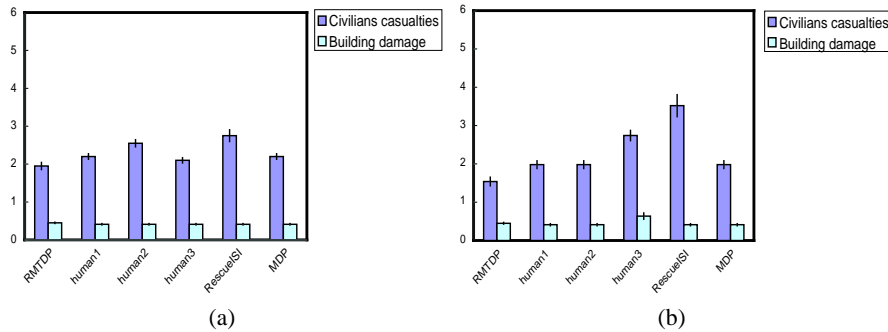


Fig. 13. Comparison of performance in RoboCupRescue, a) uniform, and b) skewed.

RescueISI often resulted in one of the fires being allocated too few fire engines. The allocations determined by the MDP approach turned out to be the same as human1.

A two-tailed t-test was performed in order to test the statistical significance of the means for the allocations in Figure 13. The means of number of civilians dead for the RMTDP allocation and the human allocations were found to be statistically different (confidence > 96%) for both the uniform as well as the skewed distributions. The difference in the fire damage was not statistically significant in the uniform case, however, the difference between the RMTDP allocation and human3 for fire damage was statistically significant (> 96%) in the skewed case.

These experiments show that the allocations found by the RMTDP role allocation algorithm performs significantly better than allocations chosen by human subjects and RescueISI and MDP in most cases (and does not do significantly worse in any case). In particular when the distribution of civilians is not uniform, it is more difficult for humans to come up with an allocation and the difference between human allocations and the RMTDP allocation becomes more significant. From this we can conclude that the RMTDP allocation performs at near-human expertise.

7 Related Work

There are three related areas of research that we wish to highlight. First, there has been a considerable amount of work done in the field of multiagent teamwork (Section 7.1). The second related area of research is the use of decision theoretic models, in particular distributed POMDPs (Section 7.2). Finally, in Section 7.3, the related work in role allocation and reallocation in multiagent teams is described.

7.1 BDI-based Teamwork

Several formal teamwork theories such as *Joint Intentions* [6], *SharedPlans* [14] were proposed that tried to capture the essence of multiagent teamwork in the logic of Beliefs-Desires-Intentions. Next, practical models of teamwork such as COLLAGEN [32],

GRATE* [19], STEAM [37] built on these teamwork theories [6, 14] and attempted to capture the aspects of teamwork that were reusable across domains. In addition, to complement the practical teamwork models, the team-oriented programming approach [31, 39] was introduced to allow large number of agents to be programmed as teams. This approach was then expanded on and applied to a variety of domains [31, 44, 7]. Other approaches for building practical multiagent systems [36, 9], while not explicitly based on team-oriented programming, could be considered in the same family.

The research reported in this paper complements this research on teamwork by introducing hybrid BDI-POMDP models that exploit the synergy between BDI and POMDP approaches. In particular, TOP and teamwork models have traditionally not addressed uncertainty and cost. Our hybrid model provides this capability, and we have illustrated the benefits of this reasoning via detailed experiments.

While this paper uses team-oriented programming [38, 7, 39] as an example BDI approach, it is relevant to other similar techniques of modeling and tasking collectives of agents, such as Decker and Lesser's [9] TAEMS approach. In particular, the TAEMS language provides an abstraction for tasking collaborative groups of agents similar to TOP, while the GPGP infrastructure used in executing TAEMS-based tasks is analogous to the "TOP interpreter" infrastructure shown in Figure 1. While Lesser et al. have explored the use of distributed MDPs in analyses of GPGP coordination [42], they have not exploited the use of TAEMS structures in decomposition or abstraction for searching optimal policies in distributed MDPs, as suggested in this paper. Thus, this paper complements Lesser et al.'s work in illustrating a significant avenue for further efficiency improvements in such analyses.

7.2 Distributed POMDP models

Three different approaches have been used to solve distributed POMDPs. One approach that is typically taken is to make simplifying assumptions about the domain. For instance, in Guestrin et al. [15], it is assumed that each agent can completely observe the world state. In addition, it is assumed that the reward function (and transition function) for the team can be expressed as the sum (product) of the reward (transition) functions of the agents in the team. Becker et al. [2] assume that the domain is factored such that each agent has a completely observable local state and also that the domain is transition-independent (one agent cannot affect another agent's local state).

The second approach taken is to simplify the nature of the policies considered for each of the agents. For example, Chadès et al. [5] restrict the agent policies to be memoryless (reactive) policies, thereby simplifying the problem to solving multiple MDPs. Peshkin et al. [29] take a different approach by using gradient descent search to find local optimum finite-controllers with bounded memory. Nair et al. [26, 27] present an algorithm for finding a locally optimal policy from a space of unrestricted finite-horizon policies. The third approach, taken by Hansen et al. [16], involves trying to determine the globally optimal solution without making any simplifying assumptions about the domain. In this approach, they attempt to prune the space of possible complete policies by eliminating dominated policies. Although a brave frontal assault on the problem, this method is expected to face significant difficulties in scaling up due to the fundamental complexity of obtaining a globally optimal solution.

The key difference with our work is that our research is focused on hybrid systems where we leverage the advantages of BDI team plans, which are used in practical systems, and distributed POMDPs that quantitatively reason about uncertainty and cost. In particular, we use TOPs to specify large-scale team plans in complex domains and use RMTDPs for finding the best role allocation for these teams.

7.3 Role allocation and reallocation

There are several different approaches to the problem of role allocation and reallocation. For example, Tidhar et al. [40] and Tambe et al. [38] performed role allocation based on matching of capabilities, while Hunsberger and Grosz [18] proposed the use of combinatorial auctions to decide on how roles should be assigned. Modi et al. [24] showed how role allocation can be modeled as a distributed constraint optimization problem and applied it to the problem of tracking multiple moving targets using distributed sensors. Shehory and Kraus [34] suggested the use of coalition formation algorithms for deciding quickly which agent took on which role. Fatima and Wooldridge [11] use auctions to decide on task allocation. It is important to note that these competing techniques are not free of the problem of how to model the problem, even though they do not have to model transition probabilities. Other approaches to reforming a team are reconfiguration methods due to Dunin-Keplicz and Verbrugge [10], self-adapting organizations by Horling and Lesser [17] and dynamic re-organizing groups [1]. Scerri et al. [33] present a role (re)allocation algorithm that allows autonomy of role reallocation to shift between a human supervisor and the agents.

The key difference with all this prior work is our use of stochastic models (RMTDPs) to evaluate allocations: this enables us to compute the benefits of role allocation, taking into account uncertainty and costs of reallocation upon failure. For example, in the mission rehearsal domain, if uncertainties were not considered, just one scout would have been allocated, leading to costly future reallocations or even in mission failure. Instead, with lookahead, depending on the probability of failure, multiple scouts were sent out on one or more routes, resulting in fewer future reallocations and higher expected reward.

8 Conclusion

While the BDI [38, 9, 39] approach to agent teamwork has provided successful applications, tools and techniques that provide quantitative analyses of team coordination and other team behaviors under uncertainty are lacking. The emerging field of distributed POMDPs [3, 4, 30, 43, 29, 16] provides a rich framework for modeling uncertainties and utilities in complex multiagent domains. However, as shown by Bernstein et al. [3], the problem of deriving the optimal policy is computationally intractable.

In order to deal with this issue of intractability in distributed POMDPs, this paper presented a principled way to combine the two dominant paradigms for building multiagent teams, namely the BDI approach and distributed POMDPs. In this hybrid BDI-POMDP approach, BDI team plans are exploited to improve distributed POMDP

tractability and distributed POMDP-based analysis improves BDI team plan performance. Thus, this approach leverages the benefits of both the BDI and POMDP approaches to analyze and improve key coordination decisions within BDI-based team plans using POMDP-based methods. In order to demonstrate these analysis methods, we concentrated on role allocation – a fundamental aspect of agent teamwork. First, we introduced RMTDP, a distributed POMDP based framework, for analysis of role allocation. Second, this paper presented an RMTDP-based methodology for optimizing key coordination decisions within a BDI team plan for a given domain. Concretely, the paper described a methodology for finding the best role allocation for a fixed team plan. Given the combinatorially many role allocations, we introduced methods to exploit task decompositions among sub-teams to significantly prune the search space of role allocations.

We constructed RMTDPs for two domains – RoboCupRescue and mission rehearsal simulation – and determined the best role allocation in these domains. Furthermore, we illustrated significant speedups in RMTDP policy search due to the techniques introduced in this paper. Detailed experiments revealed the advantages of our approach over state-of-the-art role allocation approaches that do not reason with uncertainty. In the RoboCupRescue domain, we showed that the role allocation technique presented in this paper is capable of performing at human expert levels by comparing with the allocations chosen by humans in the actual RoboCupRescue simulation environment.

References

1. S. Barber and C. Martin. Dynamic reorganization of decision-making groups. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-01)*, pages 513–520, 2001.
2. R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Transition-independent decentralized Markov decision processes. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-03)*, pages 41–48, 2003.
3. D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of MDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 32–37, 2000.
4. C. Boutilier. Planning, learning & coordination in multiagent decision processes. In *Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge (TARK-96)*, pages 195–210, 1996.
5. I. Chadès, B. Scherrer, and F. Charpillet. A heuristic approach for solving decentralized-pomdp: Assessment on the pursuit problem. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC-02)*, pages 57–62, 2002.
6. P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.
7. J. L. T. da Silva and Y. Demazeau. Vowels co-ordination model. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002)*, pages 1129–1136, 2002.
8. T. Dean and S. H. Lin. Decomposition techniques for planning in stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1121–1129, 1995.
9. K. Decker and V. Lesser. Quantitative modeling of complex computational task environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 217–224, 1993.

10. B. Dunin-Keplicz and R. Verbrugge. A reconfiguration algorithm for distributed problem solving. *Engineering Simulation*, 18:227–246, 2001.
11. S. S. Fatima and M. Wooldridge. Adaptive task and resource allocation in multi-agent systems. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-01)*, pages 537–544, May 2001.
12. M. P. Georgeff and A. L. Lansky. Procedural knowledge. *Proceedings of the IEEE special issue on knowledge representation*, 74:1383–1398, 1986.
13. B. Grosz, L. Hunsberger, and S. Kraus. Planning and acting together. *AI Magazine*, 20(4):23–34, 1999.
14. B. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.
15. C. Guestrin, S. Venkataraman, and D. Koller. Context specific multiagent coordination and planning with factored MDPs. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 253–259, 2002.
16. E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, pages 709–715, 2004.
17. B. Horling, B. Benyo, and V. Lesser. Using self-diagnosis to adapt organizational structures. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-01)*, pages 529–536, 2001.
18. L. Hunsberger and B. Grosz. A combinatorial auction for collaborative planning. In *Proceedings of the Fourth International Conference on Multiagent Systems (ICMAS-2000)*, pages 151–158, 2000.
19. N. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2):195–240, 1995.
20. L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(2):99–134, 1998.
21. H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjoh, and S. Shimada. RoboCup-Rescue: Search and rescue for large scale disasters as a domain for multiagent research. In *Proceedings of IEEE Conference on Systems, Men, and Cybernetics (SMC-99)*, pages 739–743, 1999.
22. H. J. Levesque, P. R. Cohen, and J. Nunes. On acting together. In *Proceedings of the National Conference on Artificial Intelligence*, pages 94–99. Menlo Park, Calif.: AAAI press, 1990.
23. R. T. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation.
24. P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proceedings of the Second International Joint Conference on Agents and Multiagent Systems (AAMAS-03)*, pages 161–168, 2003.
25. R. Nair, T. Ito, M. Tambe, and S. Marsella. Task allocation in the rescue simulation domain. In *RoboCup 2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Computer Science*, pages 751–754. Springer-Verlag, Heidelberg, Germany, 2002.
26. R. Nair, D. Pynadath, M. Yokoo, M. Tambe, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 705–711, 2003.
27. R. Nair, M. Roth, M. Yokoo, and M. Tambe. Communication for improving policy computation in distributed pomdps. In *Proceedings of the Third International Joint Conference on Agents and Multiagent Systems (AAMAS-04)*, pages 1098–1105, 2004.
28. R. Nair, M. Tambe, and S. Marsella. Role allocation and reallocation in multiagent teams: Towards a practical analysis. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS-03)*, pages 552–559, 2003.

29. L. Peshkin, N. Meuleau, K.-E. Kim, and L. Kaelbling. Learning to cooperate via policy search. In *Proceedings of the Sixteenth Conference in Uncertainty in Artificial Intelligence (UAI-00)*, pages 489–496, 2000.
30. D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.
31. D. V. Pynadath and M. Tambe. Automated teamwork among heterogeneous software agents and humans. *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 7:71–100, 2003.
32. C. Rich and C. Sidner. COLLAGEN: When agents collaborate with people. In *Proceedings of the First International Conference on Autonomous Agents (Agents-97)*, pages 284–291, 1997.
33. P. Scerri, L. Johnson, D. Pynadath, P. Rosenbloom, M. Si, N. Schurr, and M. Tambe. A prototype infrastructure for distributed robot, agent, person teams. In *Proceedings of the Second International Joint Conference on Agents and Multiagent Systems (AAMAS-03)*, pages 433–440, 2003.
34. O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.
35. E. J. Sondik. The optimal control of partially observable Markov processes. *Ph.D. Thesis, Stanford*, 1971.
36. P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, 1999.
37. M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.
38. M. Tambe, D. Pynadath, and N. Chauvat. Building dynamic agent organizations in cyberspace. *IEEE Internet Computing*, 4(2):65–73, 2000.
39. G. Tidhar. Team-oriented programming: Social structures. Technical Report 47, Australian Artificial Intelligence Institute, 1993.
40. G. Tidhar, A. Rao, and E. Sonenberg. Guided team selection. In *Proceedings of the Second International Conference on Multi-agent Systems (ICMAS-96)*, pages 369–376, 1996.
41. M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.
42. P. Xuan and V. Lesser. Multi-agent policies: from centralized ones to decentralized ones. In *Proceedings of the First International Joint Conference on Agents and Multiagent Systems (AAMAS-02)*, pages 1098–1105, 2002.
43. P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multiagent cooperation. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-01)*, pages 616–623, 2001.
44. J. Yen, J. Yin, T. R. Ioerger, M. S. Miller, D. Xu, and R. A. Volz. Cast: Collaborative agents for simulating teamwork. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 1135–1144, 2001.