Role Allocation and Reallocation in Multiagent Teams: Towards A Practical Analysis

Ranjit Nair
Computer Science Dept
Univ. of Southern California
Los Angeles, CA 90089
nair@usc.edu

Milind Tambe
Computer Science Dept
Univ. of Southern California
Los Angeles, CA 90089
tambe@usc.edu

Stacy Marsella
Information Sciences Institute
Univ. of Southern California
Marina del Rey, CA 90292
marsella@isi.edu

ABSTRACT

Despite the success of the BDI approach to agent teamwork, initial role allocation (i.e. deciding which agents to allocate to key roles in the team) and role reallocation upon failure remain open challenges. What remain missing are analysis techniques to aid human developers in quantitatively comparing different initial role allocations and competing role reallocation algorithms. To remedy this problem, this paper makes three key contributions. First, the paper introduces RMTDP (Role-based Multiagent Team Decision Problem), an extension to MTDP [9], for quantitative evaluations of role allocation and reallocation approaches. Second, the paper illustrates an RMTDP-based methodology for not only comparing two competing algorithms for role reallocation, but also for identifying the types of domains where each algorithm is suboptimal, how much each algorithm can be improved and at what computational cost (complexity). Such algorithmic improvements are identified via a new automated procedure that generates a family of *locally optimal* policies for comparative evaluations. Third, since there are combinatorially many initial role allocations, evaluating each in RMTDP to identify the best is extremely difficult. Therefore, we introduce methods to exploit task decompositions among subteams to significantly prune the search space of initial role allocations. We present experimental results from two distinct domains.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Measurement, Performance, Algorithms

Keywords

Role allocation, Reallocation, Analysis of Teams

1. INTRODUCTION

The belief-desire-intention (BDI) approach to agent teamwork has led to many practical multiagent applications [12, 14, 13]. Initial role allocation, i.e. which agents to allocate to the various roles in the team, and role reallocation upon failures or new tasks, are two continuing challenges for building teams [15, 7]. For instance, in mission rehearsal simulations [12], we need to select the numbers and types of helicopter agents to allocate to different roles in the team, and to decide how role substitution should occur upon failures. Similarly, in disaster rescue [8], initial role allocations (e.g. which brigades for each fire) can greatly impact team performance.

Critically needed now are analysis techniques to aid human developers in quantitatively comparing and evaluating different initial role allocations and competing role reallocation algorithms. Such evaluations are currently difficult because there are significant uncertainties and costs associated with agents' execution of roles and roles may need to be reallocated upon execution failures. In particular, given domain uncertainty and costs, experimentally evaluating all possible role allocation and reallocation combinations can be expensive or infeasible. Fortunately, the recent emergence of distributed partially observable Markov decision problems (POMDPs) provides quantitative multiagent analysis techniques [1, 2, 9, 16] that allow such evaluations. The key idea is to encode different multiagent coordination protocols as policies in distributed POMDPs, and compare them against specific baseline policies to investigate any potential for improvements in the protocols.

Unfortunately, while the previous distributed-POMDP analysis techniques are powerful, from our perspective they suffer from two key limitations. First, analysis in previous work focused on communication [9, 16], rather than role allocation or any other coordination decisions. To address this limitation, we demonstrate an overall methodology to analyze different aspects of teamwork coordination. In particular, we derive RMTDP (Role-based Multiagent Team Decision Problem), a distributed POMDP framework for analyzing role allocation and reallocation.

A second limitation is that even after defining a model such as RMTDP, techniques tailored for analysis of role allocation and reallocation are unavailable. Partly, the problem is the difficulty in deriving baseline policies for comparison: as we prove using RMTDP, the derivation of a globally optimal role-allocation policy is NEXP-complete. Partly, the logical separation of initial role allocation from reallocation in BDI teams needs to be reflected in the analysis. Indeed, in BDI teams, initial role allocation is considered part of the domain-specific organization structure; this interacts with but is separate from role reallocation, which is often part of the domain-independent coordination infrastructure to coordinate the organization [13]. For example, such separation is clearly seen in the team-oriented program (TOP) approach to building BDI teams [12, 14, 13]. Thus, in many instances, analyses must focus on selecting just the right role allocation for a given domain, since the coordination

AAMAS'03, July 14–18, 2003, Melbourne, Australia.

infrastructure is deployed and fixed. In other instances, since external constraints fix the organization structure, analyses should aim to improve an existing, domain-independent role reallocation algorithm within the coordination infrastructure — this improvement must be effective across different domains.

To address this second limitation, we treat role allocation and reallocation as two separate, interacting design tasks, and tailor analysis techniques for each. Thus, we first analyze role reallocation assuming a fixed initial role allocation. We apply RMTDP to quantitatively compare two published role reallocation algorithms over differing domain conditions — comparisons that were previously very difficult to obtain. We also illustrate a quantitative comparison of these strategies with "locally optimal" baseline policies to identify the types of domains where individual strategies may be suboptimal, how much a strategy can be improved and the computational cost of such improvements. While previous work proposed one restricted kind of "locally optimal" policy as a baseline [9], this paper presents: (i) an *automated* procedure to generate an entire family of locally optimal policies and (ii) a methodology for role reallocation analysis using such policies.

Next we use RMTDPs to improve initial role allocations, given fixed role reallocation strategies. While we could enumerate each possible role allocation for a given domain and evaluate each as a separate RMTDP policy, combinatorially many role allocations would be evaluated. Instead, we exploit task decomposition among subteams of a team to significantly prune the search space.

This work is motivated by the needs of role allocation and reallocation in two concrete domains: RoboCupRescue [8] and mission rehearsal simulation [12]. We have constructed large-scale teams using BDI approaches for both domains [12]; in particular, we have used a team-oriented programming approach (TOP) [12, 14, 13], described in Section 2. We assume these BDI systems as a given and apply RMTDP only to analyze them — presenting experimental results from both domains. While we use TOP as an example BDI approach, our methodology applies to other related teamwork approaches that need role (re)allocation as well [11].

2. DOMAINS AND MOTIVATION

For expository purposes, we have intentionally simplified the mission rehearsal domain and describe it first (the more complex RoboCupRescue domain is described later): A helicopter team is executing a mission of transporting valuable cargo from X to Y through enemy terrain (see Figure 1). There are three paths from X to Y of different lengths and different risk due to enemy fire. One or more scouting subteams must be sent out, and the larger the size of a scouting subteam 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 TOPs for domains such as these consist 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 execute plans. Thus, 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. In the TOP for this example, we first specify the team organization hierarchy (see Figure 2(a)). *Task Force* is the highest level team in this organization and consists of two roles *Scouting*

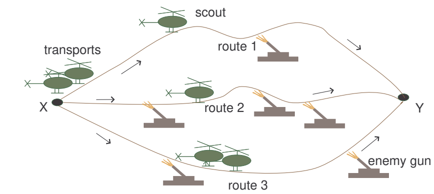
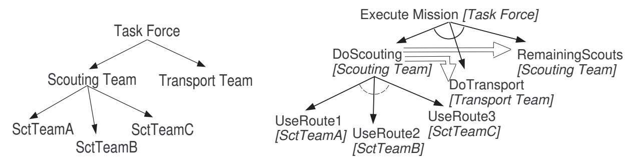


Figure 1: Helicopter Domain



(a) (b)

Figure 2: TOP a: Organization hierarchy; b: Plan hierarchy

and *Transport*, where the *Scouting* subteam has roles for each of the three scouting sub-subteam. Next we specify a hierarchy of reactive team plans (Figure 2(b)). Reactive team plans explicitly express joint activities of the relevant team and consist of: (i) initiation 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. In Figure 2(b), the highest level plan **Execute Mission** has three subplans: **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 haven't reached the destination yet to get there.

Figure 2(b) also shows coordination relationships: An AND relationship is indicated with a solid arc, while an OR relationship is indicated with a dotted arc. Thus, **DoScouting**, **DoTransport** and **RemainingScouts** must all three be done while at least one of **UseRoute1**, **UseRoute2** or **UseRoute3** need be performed. There is also a temporal dependence relationship among the subplans, which implies that subteams 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 2(b) shows the assignment in brackets adjacent to the plans. For instance, *Task Force* team is assigned to jointly perform **Execute Mission**.

This example scenario helps explain the key challenges faced in role allocation and reallocation. First, a human developer must allocate available agents to the organization hierarchy (Figure 2(a)). However, there are combinatorially many allocations to choose from [7, 12]. For instance, starting with even 6 homogeneous helicopters results in 84 different ways of deciding how many agents to assign to each scouting and transport subteam. This problem is exacerbated by the fact that the best allocations varies significantly based on domain variations, e.g. Figure 3 shows three different assignments of agents to the team organization hierarchy, each found in our analysis to be the best for a given setting of probability of helicopter failures (details in Section 6). For example, interchanging the probability of failures for routes 2 and route 3 resulted in the number of transports in the best allocation changing from 3 (see Figure 3(b)) to 4 (see Figure 3(c)). If the probability of failure on route 3 was reduced further, the number of transports increased to 5 (see Figure 3(a)). Furthermore, what reallocation algorithm to use to reallocate transports to scouting role, is a critical challenge for the TOP interpreter (coordination infrastructure). For instance, should the algorithm require all scouts to fail before role reallocation, or would a more pre-emptive reallocation approach work

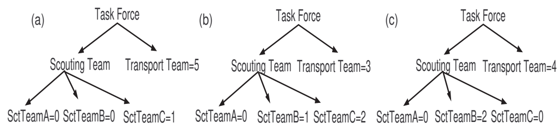


Figure 3: Best role allocations for different settings

better for this given range of domains? Our analysis takes a step towards answering the above questions.

The second example scenario, set up in the RoboCupRescue disaster simulation environment [8], consists of 7 fire brigades at three different fire stations (2 each at stations 1 & 2 and the rest at station 3) and 5 ambulances stationed at the ambulance center. Two fires start that need to be extinguished by the fire brigades. After a fire is extinguished, ambulance agents need to save the surviving civilians. As time passes, the health of civilians deteriorates and fires increase in intensity and so the goal is to rescue as many civilians as possible with minimal damage to the buildings. Here, partial observability (each agent can only view objects within its visual range), and large action uncertainty add significantly to the difficulty.

The plan hierarchy for this scenario, consists of two **Execute-Rescue** plans executed in parallel, one for each of the fires. Each such plan consists of a **ExtinguishFire** plan and a **RescueCivilians** plan, which further decompose into individual plans. The organizational hierarchy consists of *Task Force* comprising of two *Rescue* subteams, one for each fire. Each such subteam is comprised of a *Brigade Team* and an *Ambulance Team*, where the brigade team is assigned to extinguishing the fire while the ambulance team is assigned to rescuing civilians. The problem is which brigades and ambulances to assign to each *Brigade Team* and *Ambulance Team*. Note that brigades have differing capabilities due to differing distances from fires.

3. MULTIAGENT TEAM DECISION PROBLEM

For quantitative analysis of role allocation and reallocation, we extend the Multiagent Team Decision Problem (MTDP) [9]. While our extension focuses on role (re)allocation, it also illustrates a general methodology for analysis of other aspects of team coordination. Note that, while we use MTDP, other possible distributed POMDP models could potentially also serve as a basis [1, 16].

Given a team of agents α, an MTDP [9] is defined as a tuple: $\langle S, A, P, \Omega, O, R\rangle$. It consists of a finite set of states $S = \Xi_1 \times \cdots \times \Xi_n$. Each agent i can perform an action from its set of actions A_i. $P(s, < a_1, \ldots, a_{|\alpha|} >, s')$ gives the probability of transitioning from state s to state s' given that the agents perform the actions $< a_1, \ldots, a_{|\alpha|} >$ jointly. Each agent i receives an observation $\omega_i \in \Omega_i$ based on the function $O(s, < a_1, \ldots, a_{|\alpha|} >, \omega_1, \ldots, \omega_{|\alpha|})$, which gives the probability that the agents receive the observations, $\omega_1, \ldots, \omega_{|\alpha|}$ given that the world state is s and they perform $< a_1, \ldots, a_{|\alpha|} >$ jointly. The agents receive a single joint reward $R(s, a_1, \ldots, a_{|\alpha|})$.

The state of the world, s need not be observable to the agent. Thus, each agent i 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 $\pi_i(\omega_i^0, \ldots, \omega_i^t)$. $\pi = < \pi_1, \ldots, \pi_{|\alpha|} >$ refers to the joint policy of the team of agents.

3.1 Extension for explicit coordination: $\mathcal{RL}$

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 [9] where the coordination action was fixed to be the communication action. 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, $\mathcal{RL} = \{r_1, \ldots, r_s\}$ 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. Agents' actions are now distinguishable into two types:

Role-Taking actions: $\Upsilon = \prod_{i\in\alpha} \Upsilon_i$ is a set of combined role-taking actions, where $\Upsilon_i = \{\upsilon_{ir_j}\}$ contains the role-taking actions for agent i. $\upsilon_{ir_j} \in \Upsilon_i$ means that agent i takes on the role $r_j \in \mathcal{RL}$.

Role-Execution Actions: $\Phi = \prod_{i\in\alpha} \Phi_i$ is a set of combined execution actions, where $\Phi_i = \bigcup_{\forall r_j \in \mathcal{RL}} \Phi_{ir_j}$ contains the execution actions for agent i. Φ_{ir_j} is the set of agent i's actions for executing role $r_j \in \mathcal{RL}$

Thus, in RMTDP, successive epochs alternate between role-taking (Υ) and role-execution actions(Φ). If the time index is divisible by 2, agents are in the role-taking epoch, executing role-taking actions, and otherwise they are in the role-execution epoch. Although this sequencing of role-taking and role-execution epochs restricts different agents from running role-taking and role-execution actions in the same epoch, it is conceptually simple and synchronization is automatically enforced. More importantly, the distinction between role-taking and -execution actions is critical to enable a separation in their costs, so as to more easily analyze the costs of role-taking actions. To this end, in RMTDP, reward is role-taking reward, $R_\Upsilon(s, a_1, \ldots, a_{|\alpha|})$, for even time indices and role-execution reward, $R_\Phi(s, a_1, \ldots, a_{|\alpha|})$, otherwise. We view the role-taking reward as the cost (negative reward) for taking up different roles in different teams. For instance, in our example domain, when transports change roles to be scouts, there is cost for dumping its cargo and loading scout equipment. However, such change of roles may potentially provide significant future rewards.

Within this model, we can represent the specialized behaviors associated with each role, e.g. a transport vs. a scout role. While filling a particular role, r_j, agent i can only perform role-execution actions, $\phi \in \Phi_{ir_j}$, which may be different from the role-execution actions Φ_{ir_l} for role r_l. These different roles can produce varied effects on the world state (modeled via transition probabilities, P) and the team's utility. Thus, the policies must ensure that agents for each role have the capabilities that benefit the team the most.

3.2 Complexity results with RMTDP

While previous sections qualitatively emphasized the difficulty of role (re)allocation, RMTDP helps in understanding the complexity more precisely. In particular, we can define a role-taking policy, $\pi_{i\Upsilon}$ for each agent's role-taking action, a role-execution policy, $\pi_{i\Phi}$ for each agent's role-execution action. The goal in RMTDP is then to come up with joint policies π_Υ and π_Φ that will maximize the total reward over a finite horizon T. Such an optimal role taking policy not only provides for role allocation, but it also takes into account optimal future role reallocations. The following theorem illustrates the complexity of finding such optimal joint policies.

THEOREM 1. *The decision problem of determining if there exist policies, π_Υ and π_Φ, for an RMTDP, that yield a reward at least K over some finite horizon T is NEXP-complete.*

PROOF. Proof follows from the reduction of MTDP [9] to/from RMTDP. To reduce MTDP to RMTDP, we set RMTDP's role taking actions, Υ' to null. To reduce RMTDP to MTDP, we generate a new MTDP whose state space contains an additional feature to indicate if the current state corresponds to a role-taking or -executing stage of the RMTDP. The transition function, P', augments the original function P: $P'(\langle \xi_{1b}, \ldots, \xi_{nb}, \text{taking}\rangle, \upsilon_1, \ldots, \upsilon_{|\alpha|}, \langle \xi_{1e}, \ldots, \xi_{ne}, \text{executing}\rangle) = P(\langle \xi_{1b}, \ldots, \xi_{nb}\rangle, \upsilon_1, \ldots, \upsilon_{|\alpha|}, \langle \xi_{1e}, \ldots, \xi_{ne}\rangle)$ where $\upsilon_1, \ldots, \upsilon_{|\alpha|}$ is a role-taking action in the RMTDP(similarly from executing to taking). Finding the required policy in MTDP is NEXP-complete [9]. □

While the previous theorem focused on the complexity of combined role-taking and role execution actions, we can focus on the complexity of just determining the role taking actions, given fixed role-execution actions.

THEOREM 2. *The decision problem of determining if there exists a role-taking policy, π_Υ, for an RMTDP, that yields a reward at least K together with a fixed role-execution policy π_Φ, over some finite horizon T is NEXP-complete.*

PROOF. We reduce an MTDP to an RMTDP with a different role-taking action for every action in the MTDP. The role-execution policy is to perform the action corresponding to the current role. □

Note that Theorem 2 refers to a completely general *globally optimal* role-taking policy, where any number of agents can change roles at any point in time. Given the above result, 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 determines the run-time of finding the globally optimal policy. The number of policies is $\left(|\Upsilon|^{\frac{|\Omega|^{T}-1}{|\Omega|-1}}\right)^{|\alpha|}$, i.e. doubly exponential in the finite horizon and the number of agents. This clearly illustrates the point made in Section 1, that the search for a globally optimal policy is intractable.

Note that, in the worst case, cost of evaluating a single policy can be given by $O\left((|S| \cdot |\Omega|)^{T}\right)$ [9]. We will in general assume a fixed procedure for policy evaluation and primarily focus on the number of policies being evaluated.

3.3 Constructing an RMTDP

Constructing an RMTDP for evaluating a TOP is a key step in our approach. To that end, we must define each of the elements of the RMTDP tuple, specifically, $\langle S, A, P, \Omega, O, R, \mathcal{RL}\rangle$, by a process that relies on both the TOP plans as well as the underlying domain. While this step has not been automated, we briefly describe mapping techniques based on the work on our two domains.

First, we need to define the set of states S. To this end, it is critical to model the variables tested in the preconditions and termination conditions of the TOP plans. For complex domains, it is useful to consider abstract descriptions of the state modeling only the significant variables. Agents' role-taking and -execution actions in RMTDP are defined as follows. For each role in the TOP organization hierarchy, we define a role-taking action in each state s. The role-execution actions are those allowed for that role in the TOP plan hierarchy given the variable values in state s.

To illustrate these steps, consider the plans in Figure 2(b). The preconditions of plans such as **UseRoute1** and others test the 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 all the helicopters are critical variables modeled in our set of states S. For role-taking, each helicopter can perform one of four actions, i.e. being a member of one of the three scouting teams or of the transport team. Role-execution actions are the TOP actions for the plan that the agent's role is assigned in the TOP. In our case, the role execution policy for the scout role is to always go forward until it reaches Y, while for the transport role the policy is to wait at X until it obtains observation of a signal that one scouting subteam has reached Y.

Further, the types of observations for each agent must be defined. We define the set of observations to be the variables tested in the preconditions and termination conditions of the TOP plans and individual agent plans. 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, we must define the transition, observation and reward functions. Determining these functions requires some combination of human domain expertise and empirical data on the domain behavior. However, as shown later in Section 6, even an approximate dynamic and observational model, is sufficient to deliver significant benefits. Defining the reward and transition function may sometimes require additional state variables to be modeled. In our helicopter domain, the time at which each the scouting and transport mission was completed determined the amount of reward and hence time was included as a state variable.

4. ANALYSIS OF ROLE REALLOCATION

This section and the next one will now illustrate how RMTDP could be applied for iterative improvements in role allocation (in the TOP) and role reallocation algorithms (in the TOP interpreter). In this section, we focus on selecting the right reallocation strategy for a range of domains of interest — an important issue for developers of BDI coordination infrastructures. We show the application of RMTDP in quantitatively contrasting alternative reallocation strategies used in BDI systems and present an automated approach to suggest improvements to those strategies.

Our approach starts from the perspective of how BDI coordination actions, reallocation in particular, actually work. For illustration, we consider the reallocation strategy in the STEAM "interpreter", given that STEAM has been applied in several real domains [12]. Inspired by the SharedPlans theory [5], this reallocation strategy focuses on role replacement, where a failed agent must be replaced by another. Such events, as the failure of an agent, often trigger a BDI system's "local" decision on whether to reallocate roles. Thus, in STEAM, given a trigger of a failure of an agent F, an agent R will decide to replace a failed agent F only if the following inequality holds:

$$Criticality\,(Role_F) - Criticality\,(Role_R) > 0 \quad (1)$$
$$Criticality\,(x) = 1 \;\; if\; x\; is\; critical; = 0 \;\; otherwise$$

Replacement occurs if F's role is considered critical and R's role is not critical. Thus STEAM's classification of reallocation triggers is role-failure-critical or role-failure-not-critical.

Is this STEAM approach guaranteed to be better than other related approaches in other BDI systems? For instance, the role exchange strategy of the FC Portugal RoboCup soccer team [10] allows two agents to exchange roles, thus performing a pairwise reallocation of roles. Their approach employs a utility calculation based on the tradeoffs between the assumed rewards/costs of each agent taking on the other's role versus not doing this exchange. Exchanges are triggered when the two agents agree that the payoff is positive. Assuming the two agents are A and B:

$$Utility\,(Role_B, A) + Utility\,(Role_A, B)$$
$$-Utility\,(Role_A, A) - Utility\,(Role_B, B) > 0 \quad (2)$$

Such an approach can be straightforwardly adapted to role reallocation upon failures. Agent A would replace a failed agent B if the

```
epochs ← {0,...,T}; policySpace ← null
for each agent decision epoch t <= T
  for each joint obs. history ω^0...ω^{t-1}
    for each policy π in policySpace
      π' ← π; π'' ← π
      for each joint observation ω^t
        for each trigger in trigger list
          if trigger.triggered(ω^0...ω^t)=true
            i ← trigger.respondingAgent
            π'_i[ω_i^0...ω_i^t] ← respond
            π''_i[ω_i^0...ω_i^t] ← dontRespond
            policySpace ← policySpace.add(π')
            policySpace ← policySpace.add(π'')
          else
            π'_i[ω_i^0...ω_i^t] ← π_original[ω_i^0...ω_i^t]
            policySpace ← policySpace.add(π')
      policySpace ← policySpace.remove(π)
return best(policySpace)
```

Figure 4: Locally optimal policy generation.

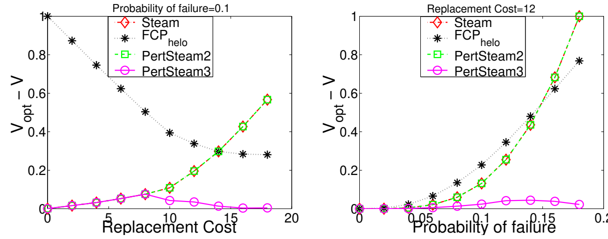
following inequality holds (where utility of a failed agent is 0):

$$Utility\,(Role_B, A) - Utility\,(Role_A, A) > 0 \qquad (3)$$

RMTDP analysis can be used to quantitatively contrast such alternative role reallocation strategies and determine under what condition one is preferable. Furthermore, we can compare these strategies with a "locally optimal" policy to determine the types of domains where improvements are possible in each strategy and by how much. A locally optimal policy is one that considers reallocations at exactly the same junctures (reallocation triggers) as the strategy it is being compared with (as opposed to a globally optimal policy that may consider reallocations at other times). The derivation of alternative local optimal policy provides a means to check if, for a trigger like role failure, whether the BDI system's behavior could be improved or not. In defining such trigger events for RMTDP analysis, the classification can be more fine-grained than the BDI system being analyzed, e.g. first-role-failure vs. second-role-failure vs. third-failure, etc. The finer grain allows the analysis to uncover cases where the BDI's decision-making is doing well versus not so well.

To further facilitate the analysis, we have automated the process of deriving these alternative locally optimal policies. Earlier methodology, as specified in [9] dictated that we first derive an algorithm for a "locally optimal" policy by hand. However, there are two problems in this derivation: (i) Deriving such a complex expression by hand is cumbersome and hinders analysis; (ii) The focus there remained on a single trigger, and no guidance is provided on multiple triggers. It is possible to automatically generate various locally optimal policies by replacing or perturbing the response of a particular reallocation trigger. For example, our approach can perturb STEAM's response to the first and second failure that occurs by replacing it with an optimal RMTDP policy that is derived under the assumption that the response to all other triggers remains the same as STEAM's. Furthermore, the response may also vary with time, even for a single trigger. Indeed, Figure 4 presents such an algorithm that automatically generates a locally optimal replacement policy. In this algorithm, we require a list of trigger events and a specific agent that would respond to each trigger. For each decision epoch and trigger, the responding agent chooses between performing "respond" (i.e. replace) or "dontRespond". We can obtain a family of locally optimal policies by varying the list of triggers.

Deriving a local optimal policy achieves considerable computational savings over the global optimal. Recall derivation of the



(a) (b)

Figure 5: Sub-optimality of replacement policies, a: Varying replacement cost and b: Varying probability of failure

"globally optimal" policy must evaluate $\left(|\Upsilon|^{\frac{|\Omega|^{T}-1}{|\Omega|-1}}\right)^{|\alpha|}$ policies. The number of policies that need to be considered for a locally optimal policy depends on the how many triggers it considers. For example, 2^T policies have to be evaluated for a locally optimal policy that varies its response depending on what time the trigger (only one trigger) occurred, while for deriving a locally optimal policy that varies its response depending on both time of the trigger and which trigger (1st, 2nd, etc.), it has to evaluate $2^{|triggers|*T}$ policies, as shown in Figure 4.

To demonstrate the empirical utility of automated RMTDP reallocation analysis, consider the helicopter domain from Section 2. These agents must decide whether to do a role replacement when a failure occurs. We compare the performance of various policies, across a space of distinct domains obtained by varying the replacement cost and probability of failure. For this experiment, we start with 6 helicopters and various starting role allocations. Here, we will discuss the results for an allocation of 3 scouts (all assigned to path 2) and 3 transports. The results for other allocations are similar. When a scout is replaced by a transport, a *Role replacement cost* is incurred. (We assume that there is an ordering that determines which transport will perform a role replacement.) To ensure a focus on role replacement, we assume that for all approaches, the policies for role-execution and communication are the same, as given in Section 3.3. The reward is higher if more transports reach the destination and if they reach early rather than late.

We compared the performance of 4 policies. In the *STEAM* policy, the transports use the inequality1 to determine whether to replace a failed scout. In STEAM, failure of the last remaining scout would be seen as *critical* and all other roles as non-critical. In the policy we call FCP_{helo} (based on [10]), inequality 3 is used to determine whether to replace a failed scout. The FCP_{helo} policy would replace a failed scout by a transport if the utility for the successful completion of the scouting mission and transport mission exceeds the replacement cost and the loss of reward because there is one less transport (under the assumption of no subsequent failures). PertSTEAM2 and PertSTEAM3 are locally optimal perturbations of the STEAM policy, generated automatically by running the algorithm in Figure 4, considering the trigger to be the 2nd and 3rd failure respectively. The above 4 policies were then compared to a benchmark policy, generated from Figure 4 with list of all failures as the list of triggers. As discussed earlier, such a policy is more expensive to compute than PertSTEAM2 and PertSTEAM3, given that it varies its response depending on which trigger it is (1st, 2nd, etc). Note that this is still cheaper than the globally optimal policy.

In Figure 5 we compare the sub-optimality of various replacement polices. In Figure 5(a), we varied the replacement cost, keeping the probability of failure fixed at 0.1. In Figure 5(b) we varied the probability of failure keeping the replacement cost fixed at 12. In both figures we plot the sub-optimality with respect to the benchmark policy, $V_{opt} - V$, on the Y-axis (lower values are better). The two graphs identify that:

- In domains with low replacement cost or low probability of failure, STEAM is close to benchmark, suggesting that a more expensive reallocation strategy is not required.
- If we must choose between STEAM and FCP$_{helo}$, then for domains with low probability of failure and low cost, use STEAM else we use FCP$_{helo}$.
- STEAM's response to second failure is optimal since STEAM and PertSTEAM2 exhibit identical performance in both graphs.
- However, in both graphs, PertSTEAM3 does better than the other 3 policies, clearly identifying room for improvement. In particular, on third failures, STEAM will always replace and thus, in both high replacement cost and failure rate domains, must be over-eager to replace, even when helicopters will not be able to meet the mission deadline.
- STEAM would be improved by factoring in replacement cost and failure rate.

In summary, RMTDP analysis has identified where the two BDI strategies do well, where they do more poorly as well as which decisions and factors need to be taken into account in order to improve them. It does this analysis both in terms of contrasting the strategies in relation to each other and in relation to local optimums. Finally, it provides a worst case estimate of the cost of making the improvements, via the cost of deriving the local optimum.

5. ANALYSIS OF ROLE ALLOCATION

While the previous section focused on analysis of role reallocation in the TOP interpreter, this section focuses on role allocation done in the TOP itself. As mentioned earlier, role allocation focuses on deciding how many and what types of agents to allocate to different roles in the organization hierarchy. Figure 6 shows a partially expanded role allocation space defined by the TOP organization hierarchy in Figure 2 for 6 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 6 helicopters are assigned to the *Task Force* (level 0) of the organization hierarchy while the leftmost leaf node (at level 2) in Figure 6 specifies that 1 helicopter is assigned to *SctTeamA*, 0 to *SctTeamB*, 0 to *SctTeamC* and 5 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 compare by constructing an RMTDP policy for each leaf. The role allocation specified by the leaf node corresponds to the role-taking actions that each agent will execute at time=0. For example, in the case of the left most leaf in Figure 6, at time 0, one agent (recall from Section 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*. The rest of the role-taking policy will be the role replacement policy determined in Section 4, i.e. for the range of domains of interest it is the *STEAM* policy. Each agent's role-execution policies are determined by the plan associated to their role. Thus, we have been able to construct a policy for the RMTDP that corresponds to the role allocation.

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

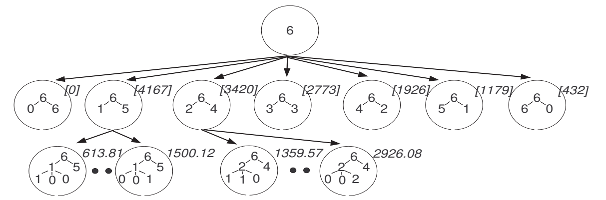


Figure 6: Partially expanded role allocation space (6 helos)

leaf roles in the organization hierarchy. Thus, we must prune the search space.

5.1 Pruning the role allocation space

We prune the space of valid role allocations using heuristic overestimates (max estimates) for the parents of the leaves of the role allocation space (Section 5.2). In particular, once we obtain max estimates for all the parent nodes (shown in brackets to the right of each parent node in Figure 6), we use branch-and-bound style pruning. First, we sort the parent nodes by their estimates and then start evaluating children of the parent with the highest max estimate. In the case of the role allocation space in Figure 6, we would start with evaluating the leaves of the parent node that has 1 helicopter in *Scouting Team* and 5 in *Transport Team*. The value of evaluating each leaf node is shown to the right of the leaf node. Once we have obtained the value of the best leaf node, in this case 1500.12, we compare this with the max estimates of the other parents of the role allocation space. As we can see from Figure 6 this would result in pruning of 3 parent nodes (left most parent and right two parents). Next, we would then proceed to evaluate all the leaf nodes under the parent with 2 helos in *Scouting Team* and 4 in *Transport Team*. This would result in pruning of all the remaining unexpanded parent nodes and we will return the leaf with the highest value, which in this case is the node corresponding to 2 helos allocated to *SctTeamA* and 4 to *Transport Team*. Although demonstrated for a 3-level hierarchy, extending to deeper hierarchies is straightforward.

5.2 Calculating over-estimates for parents

We will now discuss how the max estimates can be calculated for each parent. The max estimate of a parent must necessarily be an overestimate of the maximum expected reward of all the leaf nodes under it or else we might end up pruning potentially useful role allocations. In order to calculate the max estimate 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 plan hierarchy (see Figure 2(b)) to see how this evaluation of the parent node can be broken up into components, which can be evaluated separately thus decomposing the problem. Our approach exploits the structure of the BDI program to construct small-scale RMTDPs, unlike other decomposition techniques [3, 6]. For each parent in the role allocation space, we use these RMTDPs to evaluate the values for each component. The values of the components of each parent can be added to obtain its max estimate (an upper bound on its children's values).

As shown in Figure 8, the first step in our approach involves deciding how to decompose the plan hierarchy into components and then create smaller RMTDPs, one for each component. We explain this methodology using the plan hierarchy in Figure 2(b). Using the temporal constraints in the plan hierarchy, we choose a level of the plan hierarchy at which to do a decomposition. For example, since temporal constraints exist between **DoScouting**, **DoTransport** and **RemainingScouts**, we choose these as the components for our max estimation. The process of constructing an RMTDPs for a component is similar to the method described in Section 3.3. In particular, we determine the set of state variables relevant to the component that it corresponds to, e.g. those variables that are present in the preconditions and termination conditions of this component are clearly relevant. Fortunately, all state variables that are irrelevant to the component can be eliminated. For example, in the **DoTransport** component, only the variables that refer to number of members of *Transport Team*, the locations of each of these transports and the route that they should follow (the scouted route) are relevant. Information about the other scouts is not important to this component and can be eliminated from the state.

After constructing RMTDPs for each component, we evaluate the max estimate for each parent node in the role allocation space (See Figure 8). First, we identify the start states for each component from which to evaluate the RMTDPs. We explain this step using a parent node from Figure 6 — *Scouting Team* = 2 helos, *Transport Team* = 4 helos (see Figure 7). For the very first component, the start states corresponds to all the role allocations under the parent node. As shown in Figure 7, the start states of the **DoScouting** component for this parent, correspond to all possible role allocations of 2 helos to *Scouting Team* and 4 helos to *Transport Team*, e.g. 1 helo to *SctTeamB*, 1 helo to *SctTeamC* and 4 helos to *Transport Team*. The role allocation corresponding to a start state tells the agents what role to take in that start state. The remainder of the role-taking policy is specified by the role replacement policy. 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. Note, if the next component is not sequentially dependent on the prior one, then its start states are determined from its own children.

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 observations which are typically referred to in the preconditions 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.

In order to obtain the max estimate for a parent node of the role allocation space, we simply sum up the maximum of the evaluation for each component over all its start states and starting observations. E.g. the maximum values of each component (see right of each component in Figure 7) were summed to obtain the max estimate $(84 + 3330 + 36 = 3420)$. The calculation of the max estimate for a parent nodes should be much faster than evaluating the leaf nodes below it in most cases for three reasons. Firstly, parent nodes are evaluated component-wise. Thus, if multiple start states result in the same end state, we can remove duplicates to get the start states of the next component. This prevents a lot of duplication of the evaluation effort, something that cannot be avoided for leaf nodes, where each state is evaluated independently from start to finish. Secondly, since each component only contains the state variables relevant to it, we can further reduce the set of start states drastically. As seen in Figure 7, the start states of the **DoTransport** component only considers the scouted route and number of transports (some transports may have replaced failed scouts), thus greatly reducing the set of end states of **DoScouting** component. Finally, the number of starting observation sequences will be much less than the number of ending observation histories of the preceding components.

We refer to this methodology of obtaining the max estimates of each parent as MAXEXP. A variation of this, the maximum expected reward with no failures (NOFAIL), is obtained in a similar fashion except that we assume that the probability of any agent failing is 0. 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 6 helicopters is shown in square brackets in Figure 6.

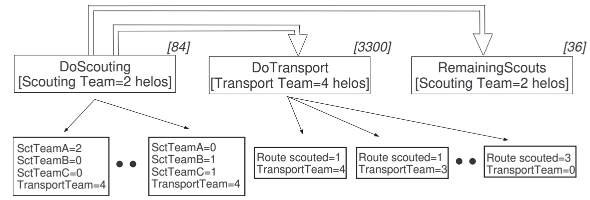


Figure 7: Component-wise decomposition of a parent node

Constructing RMTDPs
1. Divide the plan hierarchy into components
2. For each component construct an RMTDP by determining $S, \Omega, P, O,$ and R

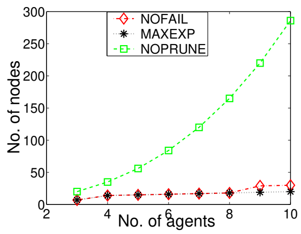
MaxEstimate for a parent node
1. For each component RMTDP
2. Obtain start states, $states$ and corresponding observation histories at start $OHistories$
3. MaxEstimate =+ $\max_{s \in states, oh \in OHistories}$(evaluate($s, oh$))

Figure 8: Calculating over-estimates for parents.

6. EXPERIMENTAL RESULTS

For the two domains introduced in Section 2, helo and RoboCupRescue [8], we focus on determining the best initial assignment of agents to roles; but assume a fixed TOP and role replacement strategy (we use the STEAM policy from Section 4). Thus, the initial role assignment considers future role replacements, as discussed in Section 1. For the helicopter domain, the TOP is the one discussed in Section 2. As can be seen in Figure 2(b), the organization hierarchy requires determining the number of agents to be allocated to the three scouting subteams and the remaining helos must be allocated to the transport subteam. Different numbers of initial helicopters were attempted, varying from 3 to 10.

Figure 9 shows the results of comparing the different methods for searching the role allocation space; in particular, we show results from MAXEXP, NOFAIL and NOPRUNE (brute force). In Figure 9(a), the Y-axis is the number of nodes in the role allocation space evaluated, while in Figure 9(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. Figure 9(a) clearly shows significant reductions over NOPRUNE in the numbers of nodes evaluated due to the pruning by MAXEXP and NOFAIL. This reduction grows quadratically to more than 20-fold at 10 agents. Note that the NOFAIL heuristic results in less pruning than MAXEXP for 9 and 10 agents because, although it is cheaper to evaluate, its estimate is higher than the MAXEXP estimate. Figure 9(b) shows how the MAXEXP heuristic results in a 14-fold speedup over the NOPRUNE in the 10 agent case. The NOFAIL heuristic, which is very quick to compute the max estimates, far out performs the MAXEXP heuristic (180-fold speedup over MAXEXP for 10 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



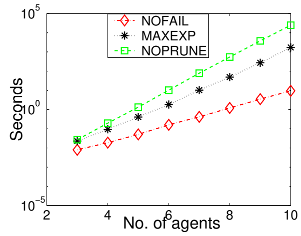


Figure 9: Performance of role allocation space search, a: Number of nodes evaluated, and b: Run-time in seconds

Table 1: Performance of role allocations in RoboCupRescue

	Civilians saved	Bldg. damage(%)	RoboCupRescue
Best Alloc.	6	0.10	1.17
Alt. Alloc. 1	4	1.42	3.29
Alt. Alloc. 2	2	8.18	5.42

much less branching. There were different best role allocations for different settings of the probability of failure, as shown in Figure 3, for 6 helos, indicating the difficulty of finding the best allocation.

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, by taking guidance from the TOP and the underlying domain (as described in Section 3.3). We then use the NOFAIL heuristic to determine the best role allocation — NOPRUNE could not be run because of its slowness. The best allocation recommended assigning three ambulances and three fire brigades(2 from station1 and 1 from station2) for the first fire and the remaining agents to the second fire. We tested this best allocation in the actual RoboCup Rescue simulation. For comparison, we considered two alternate allocations. Alternate 1 was the next best allocation with a significant value difference (RMTDP ranked four allocations to be extremely close in value, so we consider the fifth). It allocated 3 ambulances and 2 fire brigades (1 from station1 and 1 from station3) for first fire, remaining agents to second fire. While, alternate 2 was an allocation that RMTDP predicted would perform poorly — 2 ambulances and 2 fire brigades (both from station1) for first fire, remaining agents to second fire. In Table 1, we show how the best role allocation compared to the alternate allocations. The best allocation resulted in 6 survivors out of 7 trapped civilians, while the alternate allocations results in 4 and 2 survivors, respectively. Also, the best allocation resulted in less percentage damage to the buildings due to fire and received a better (lower) RoboCupRescue score, which considers civilian lives lost, damage to buildings and injuries (most weight for lives lost).

7. CONCLUSION AND RELATED WORK

While the BDI approach to agent teamwork remains the most successful within multiagents, initial role allocation and reallocation remain open challenges. Critically needed now are analysis techniques to help human developers in quantitatively comparing different initial role allocations and competing role reallocation algorithms. To this end, this paper provided three key contributions. First, it introduced RMTDP, a distributed POMDP based framework, for analysis of role (re)allocation, generalizing prior work on analysis of communication. RMTDP analysis enables a virtuous cycle of improvements to role allocation and role reallocation to ensue. Second, for role reallocation, the paper illustrated a methodology not only in comparing two competing algorithms, but also identified the types of domains where each is suboptimal, how much each algorithm can be improved and at what computational cost (complexity). Such algorithmic improvements are identified via a new automated procedure that generates a family of locally optimal policies for comparative evaluations. Third, given the combinatorially many initial role allocations, we introduced methods to exploit task decompositions among subteams to significantly prune the search space of initial role allocations. We presented results from two distinct domains to illustrate our methodology.

While the work used team-oriented programming [12, 13, 14] as an example BDI approach, it is relevant to other similar techniques of modeling and tasking collectives of agents, such as Lesser *et al*'s [4] TAEMS approach. In other related work, role allocation based on matching of capabilities [15] and combinatorial auctions [7] has been proposed earlier. The key difference with 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 costs of reallocation upon failure. Our key contributions focused on improving the efficiency of this stochastic analysis. Finally, in terms of MDP research, MTDP is itself identical to the DEC-POMDP [1] model; and could be seen to generalize Boutilier's MMDP [2] model. RMTDP extends MTDP to analyze role allocation and reallocation in BDI teams, which has required the development of novel, practical techniques for analysis.

Acknowledgement: Thanks to J. Blythe, A. Cassandra, H. Jung, S. Kapetanakis, S. Koenig, M. Littman, D. Pynadath and P. Scerri for valuable discussions. This research was supported by NSF grant #0208580.

8. REFERENCES

[1] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of MDPs. In *UAI*, 2000.

[2] C. Boutilier. Planning, learning & coordination in multiagent decision processes. In *TARK*, 1996.

[3] T. Dean and S. H. Lin. Decomposition techniques for planning in stochastic domains. In *IJCAI*, 1995.

[4] K. Decker and V. Lesser. Quantitative modeling of complex computational task environments. In *AAAI*, 1993.

[5] B. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.

[6] C. Guestrin, S. Venkataraman, and D. Koller. Context specific multiagent coordination and planning with factored MDPs. In *AAAI*, 2002.

[7] L. Hunsberger and B. Grosz. A combinatorial auction for collaborative planning. In *ICMAS*, 2000.

[8] H. Kitano, S. Tadokoro, and I. Noda. Robocup-rescue: Search and rescue for large scale disasters as a domain for multiagent research. In *IEEE Conference SMC*, 1999.

[9] D. Pynadath and M. Tambe. Multiagent teamwork: Analyzing the optimality complexity of key theories and models. In *AAMAS*, 2002.

[10] L. P. Reis, N. Lau, and E. C. Oliveira. Situation based strategic positioning for coordinating a team of homogeneous agents. LNCS 2103:175.

[11] P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intel.*, 110(2):241–273, 1999.

[12] M. Tambe, D. Pynadath, and N. Chauvat. Building dynamic agent organizations in cyberspace. *IEEE Internet Computing*, 4(2), 2000.

[13] J. Tavares and Y. Demazeau. Vowels co-ordination model. In *AAMAS*, 2002.

[14] G. Tidhar. Team-oriented programming: Social structures. Technical Report 47, Australian A.I. Institute, 1993.

[15] G. Tidhar, A. Rao, and E. Sonenberg. Guided team selection. In *ICMAS*, 1996.

[16] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multiagent cooperation. In *Agents*, 2001.