Decision Support in Organizations: A Case for OrgPOMDPs

Pradeep Varakantham
School of Information Systems
Singapore Management University
Singapore
pradeepv@smu.edu.sg

Nathan Schurr, Alan Carlin, Christopher Amato
Aptima Inc.
Boston, United States of America
{nschurr,acarlin,camato}@aptima.com

***Abstract*—In today's world, organizations are faced with increasingly large and complex problems that require decision-making under uncertainty. Current methods for optimizing such decisions fall short of handling the problem scale due to not exploiting the inherent structure of the organizations. We propose a new model called the *OrgPOMDP* (Organizational POMDP), which is based on the partially observable Markov decision process (POMDP). This new model combines two powerful representations for modeling large scale problems: hierarchical modeling and factored representations. In this paper we make three key contributions: (a) Introduce the OrgPOMDP model; (b) Present an algorithm to solve OrgPOMDP problems efficiently; and (c) Apply OrgPOMDPs to scenarios in an existing large organization, the Air and Space Operation Center (AOC). We conduct experiments and show that our OrgPOMDP approach results in greater scalability and greatly reduced runtime. In fact, as the size of the problem increases, we soon reach a point at which the OrgPOMDP approach continues to provide solutions while traditional POMDP methods cannot. We also provide an empirical evaluation to highlight the benefits of an organization implementing an OrgPOMDP policy.**

I. Introduction

Solving decision problems in uncertain domains with imperfect information is a difficult challenge. These problems include situations with uncertain action effects and only partial information about the current state of the environment. Partially observable Markov decision processes (POMDPs) provide a robust model for representing these problems. While many promising algorithms have been developed [1], [2], [3], [4], [5], [6], [7], [8], scalability to large real-world domains remains an open question.

Recently, work on hierarchical [9], [10], [11] and factored models [12], [13], [14], [15] has shown increased scalability by making use of inherent structure in a problem. These approaches allow the problem to be broken up into more manageable pieces which can be solved more easily by using either a hierarchy of more finely grained problems or factored problem variables which contain sets that are independent of one another. In this paper, we combine the benefits of both approaches by breaking up a large problem into a set of hierarchically related problems, each of which is made up of a factored model. This model is motivated by the need to find the best use of an organization's resources while taking into account the partially observable nature of a domain, leading us to call our model an Organizational POMDP, or OrgPOMDP. OrgPOMDP helps leverage the hierarchical nature of the organization and the structure in dependencies between different levels efficiently.

From the perspective of organizations such as Air and Space Operation Center (AOC), OrgPOMDP is an ideal model to represent (a) organizations' control hierarchy; (b) decision problems (primarily under uncertainty) faced at each level of the control hierarchy; and most importantly (c) the interactions between decision makers at different levels of the hierarchy. Due to such a rich representation of the decision problem, an OrgPOMDP policy ensures that an organization reacts to unexpected events in a coherent manner. We provide empirical evidence illustrating this aspect in the context of AOC. It is worth noting that a large number of hierarchical problems can be represented using the OrgPOMDP model.

Apart from presenting the OrgPOMDP model, we also introduce a novel algorithm to solve OrgPOMDPs. This algorithm provides methods to exploit the factored and hierarchical structure present in the OrgPOMDP, drastically reducing the solution complexity. Finally, we also show the performance of this solver on scenarios from the AOC domain. These results show that as the complexity of the problem increases, the benefits of the OrgPOMDP approach increase as well. In fact, as we increased the complexity of the Organizational Planning Scenario (Section II-B2), we quickly reached a point where the OrgPOMDP could provide solutions, whereas a traditional POMDP could not. As mentioned in the previous paragraph, we have conducted experiments to highlight the advantages for an organization in employing an OrgPOMDP policy.

II. Motivating Domains

The OrgPOMDP model is applicable to a wide range of problems. In order to make our discussions and experiments concrete, we present the general characteristics of domains that would benefit from the OrgPOMDP model.

A. Domain Characteristics

Informally, this paper focuses on representing and solving problems in which:(a)There is a hierarchical control structure. (b)Decisions are made at each level and they are made sequentially, i.e., initially a decision is made at the root (level

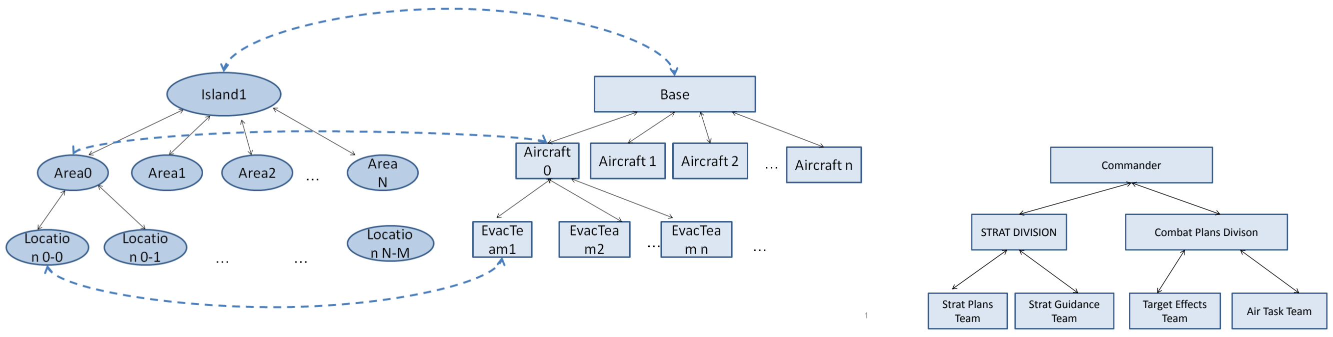

Figure 1. Rescue Mission and Planning Scenarios.

0) for which a set of decisions are made at level 1 (the next level) and for each decision at level 1 there are a set of decisions made at level 2 and so on. Decisions of a parent node can potentially affect all aspects (mentioned formally in Section III) of a child's decision problem. Child decisions affect state transitions of its parent.

B. Air and Space Operations Center (AOC)

Many military organizations possess the hierarchical structure described above. One organization that needs to quickly and seamlessly adapt is the United States Air Force's Air and Space Operations Center (AOC). The AOC is a command and control center with the capability to direct and supervise the activities of assigned and attached forces and to monitor the actions of both enemy and friendly forces. By design, the AOC consists of core personnel and augmenting elements. The AOC constantly evolves its staffing, structure and processes to changes in the environment and its missions. The policies for augmentations and commensurate personnel responsibility assignments are examples of organizational adaptation in the AOC.

In addition to the general organizational adaptation challenges described above, AOC is an example of an organization with hierarchical authority, distributed roles, and flexible staffing. Its hierarchical structure is embodied in the multiple command teams and cells. Each team has a set of core functions, and can be augmented with additional personnel if required. All elements of AOC have two main processes: standard proactive and planned tasks, and event-based time-critical tasks. Consequently, organization adaptation involves changes in the staffing for the teams and the assignment of tasks to teams. We consider two representative problems, in the form of scenarios for the AOC:

1) Rescue Scenario: In this scenario (depicted in Figure 1), AOC has been alerted that a hurricane is imminent, and that an island must be evacuated. There are two key aspects to the scenario, one is the hierarchical organization structure and the second is the hierarchical task structure. The task structure is representative of the several fragment areas of the island that need to be evacuated. The organization has selected resources to perform the evacuation, in the form of aircraft operating out of a nearby base. Each aircraft

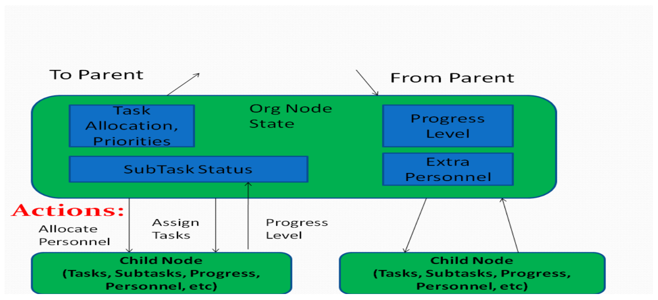
To Parent
From Parent
Task Allocation, Priorities
Org Node State
Progress Level
SubTask Status
Extra Personnel
Actions:
Allocate Personnel
Assign Tasks
Progress Level
Child Node (Tasks, Subtasks, Progress, Personnel, etc)
Child Node (Tasks, Subtasks, Progress, Personnel, etc)

Figure 2. OrgPOMDP representation.

can hold up two evacuation teams, and each evacuation team is capable of performing evacuation operations in a particular location within the area. Nodes in the control hierarchy can only be allocated to tasks on the same level of the hierarchy, as shown in the figure. When a node on the right is assigned to a task on the left, it is permitted to allocate its child nodes to subtasks. The dotted arrows in the figure show that (a) Base has been assigned to rescue Island 1, (b) Aircraft 1 has been assigned to rescue Area 1, and (c) EvacTeam1 has been assigned to rescue Location 0-0. We will refer to this scenario simply as *Rescue* for the experiments in Section V of this paper.

2) Organizational Planning Scenario: Second, we consider the much larger problem of planning for the whole organization. The hierarchical structure of Figure 1 represents the organization. Each division has a role in the planning process, and each team within the division has a more specific role. However, when multiple tasks arrive, it may be unclear which of the tasks should be handled by the teams, and how the organization should leverage its structure to handle them. We will refer to this scenario as *Planning* for the experiments in Section V of this paper.

3) Additional scenario details: Figure 2 shows the decisions that will be made by each organizational node. Parent nodes have three categories of actions which affect the child nodes by either: (1) redistributing extra resources (or personnel) among the child nodes, which will increase the probability of progress in the child nodes which receive the resources. (2) changes the allocation of subtasks to child nodes, which will again alter the probability of progress depending on the effectiveness of the child node at handling the allocated task. (3) doing nothing and waiting for progress.

Henceforth, we will refer to the extra resources in the *Rescue* domain and extra personnel in the *Planning* domain generically as Extra Resources (ER). In the *Rescue* scenario, ER represents additional team members or rescue equipment, and in the *Planning* scenario ER represents specialists (software expert, domain expert, etc) that assist in planning. Thus, increasing ER increases effectiveness at performing the tasks. The formal model in the next section will situate these two motivating domains into a broader class of problems.

III. Model

To represent the domains of interest in this paper, we introduce an extension to the well known partially observable Markov decision process (POMDP) model which we call the OrgPOMDP, $\mathcal{OP}$. The model is defined as the tuple

$$\mathcal{OP} = (\mathcal{P}, \{c\mathcal{OP}_1, c\mathcal{OP}_2, \cdots\}, \mathcal{SD}, \mathcal{MD}, H)$$

with the following attributes: $\mathcal{P}$ is the standard POMDP tuple, $c\mathcal{OP}_1, c\mathcal{OP}_2, \cdots$ are child OrgPOMDPs, $\mathcal{SD}$ are state dependencies and $\mathcal{MD}$ are model dependencies. As can be noted from the definition above, $\mathcal{OP}$ is recursively defined, thus an OrgPOMDP can be represented as a tree with each "node" in the tree representing an OrgPOMDP. Without loss of generality, we assume Q nodes in this tree and each node is referred to as $\mathcal{OP}_q$.

Factored POMDP model: In each OrgPOMDP node, the $\mathcal{P}$ is represented as the tuple $\langle S, A, \Omega, T, O, R, H\rangle$ with a factored state ($S = (S^i)_1^l$), action ($A = (A^i)_1^m$) and observation space ($\Omega = (\Omega^i)_1^n$). That is, each of the states, actions and observations can be broken up into a set of features. For instance, a state at a node in the planning scenario is broken into features such as progress, extra personnel, allocation of tasks at a node. T defines the probability of transitioning to state s' from state s given action a was taken, or $T(s'|s, a)$, O defines the probability of observing o given action a was taken and the resulting state is s', or $O(o|s', a)$, R defines the immediate reward for being in state s and taking action a, or $R(s, a)$ and H represents the time horizon for the decision making at the root node.

Child OrgPOMDPs: For each node $\mathcal{OP}_q$, its child nodes are represented as $c\mathcal{OP}_{q,1}, c\mathcal{OP}_{q,2}, \cdots$. These child nodes are further defined recursively. Time horizon for the child nodes can be different from the parent and the OrgPOMDP representation allows for such a variation.

Then, we have two aspects of the model related to the dependencies existing between nodes at different levels of the hierarchy:

State space dependencies: These are dependencies from child nodes to their parent nodes that arise due to the dependence between state space features. For instance in AOC type organizations, these dependencies arise because the performance of the organization as a whole (i.e. root $\mathcal{OP}$) depends on overall progress (feature in the state space of root node) achieved on various tasks, which in-turn is computed from the progress achieved by child nodes on sub-tasks (feature in the state space of child node). Thus, a state dependency, $sd \in \mathcal{SD}$ of $\mathcal{OP}_q$ is defined as the tuple $\langle k, c\mathcal{OP}_{q,1}, G_1, f()\rangle$, where $s^k = f(\{s_1^i\}_{i\in G_1})$, G_1 is a subset of the state space features in POMDP, $\mathcal{P}$ (of $c\mathcal{OP}_{q,1}$) and f can be any surjective function that is invertible. In the *Rescue* domain mentioned earlier, $f()$ is the "average" function over the progress achieved on its sub tasks. At this juncture, a state feature for the parent can only be obtained from state features of one child. In the future, we wish to extend this interaction to consider state features from multiple children.

Model dependencies: These are dependency links from a parent node to one of its child nodes. In this paper, we assume that the state and actions of a parent $\mathcal{OP}_q$ could affect all aspects of the child decision problem, except the observations. Therefore, a model dependency, $md_s^a \in \mathcal{MD}$ from $\mathcal{OP}_q$ to $c\mathcal{OP}_{q,1}$ is defined as $md_s^a : s \times a \rightarrow S \times A \times T \times R$, where $s \in \mathcal{OP}_q.\mathcal{P}.S$ and $a \in \mathcal{OP}_q.\mathcal{P}.A$. Thus, a model dependency, can have the following effects on the child node (depending on the state and action of parent node):

(a) Defining the state and action space: For example in the planning scenario, if at the parent node a decision (action) is made to avoid having extra personnel at a child node, then at that child node, the state and action space would not contain a feature associated with extra personnel.

(b) Defining the reward function: Organization (parent node) sets goals for individual groups (child nodes) to achieve in a planning scenario. Such goal setting by higher levels is modeled in terms of defining the reward function for the lower levels; and

(c)Defining the transition function: Continuing the planning scenario, depending on the number of extra personnel allocated to a child node, the dynamics (captured by the transition function) at the lower level vary.

Due to these dependencies between different levels of the hierarchy, the OrgPOMDP model is only partially specified. For instance, if there is a state dependency between two nodes in the hierarchy, the state transition can only be partially specified at the parent node (i.e. for features not dependent on child node) and similarly, if there is an action dependency, the aspects of the child node that are dependent on the parent node are partially specified. In the organizational planning scenario mentioned earlier, an example of a model dependency would be where the top level management decides to restructure the tasks each group is working on. Such a scenario would correspond to a model dependency, where most aspects of the child decision problem are affected.

A. Mapping to a Flat POMDP model

To illustrate the complexity of an OrgPOMDP decision problem, we provide a mapping from an OrgPOMDP model to an equivalent (flat) POMDP model where all the decisions are made using one single POMDP. In the flat model, different attributes of the POMDP tuple $\langle \mathbb{S}, \mathbb{A}, \Omega, \mathbb{T}, \mathbb{O}, \mathbb{R} \rangle$ are defined as follows:

(a) States, $\mathbb{S}$: The set of states is a cross product of a subset of state features from each node $\mathcal{OP}_q$ of the OrgPOMDP. Intuitively, this subset at each node includes all the features which cannot be aggregated from state features of a child node. Formally, this subset of features at each node in the OrgPOMDP, $\mathcal{OP}_q$ will include all those state features k, where $\nexists \langle k, cOP_{q,z}, G_1, f() \rangle \in \mathcal{OP}_q.SD$ s.t. $s_q^k = f(\{s_{q,z}^i\}_{i \in G_1}$, $s_q^k \in \mathcal{OP}_q.\mathcal{P}.S$ and $s_{q,z}^i \in c\mathcal{OP}_{q,z}.\mathcal{P}.S$. Therefore, $\forall st \in \mathbb{S}, st = \times_{q<Q} st_q$, where $st_q = \times_k s_q^k$.

(b) Actions, $\mathbb{A}$: The set of actions is a cross product of actions from each node in the OrgPOMDP,
i.e. $\mathbb{A} = \times_{q \in Q} \mathcal{OP}_q.\mathcal{P}.A$.

(c) Observations, Ω: The set of observations is also a cross product of observations from all the nodes in the OrgPOMDP,
i.e. $\Omega = \times_{q \in Q} \mathcal{OP}_q.\mathcal{P}.\Omega$

(d)Transition function, $\mathbb{T}$: Since the transition functions are primarily independent for the different nodes of the OrgPOMDP, the transition function for the flat POMDP is defined as the product of feature transition probabilities from individual OrgPOMDP nodes:

$$\forall st, st' \in \mathbb{S}, \mathbb{T}(st, a, st') = \prod_{q<Q, a \in \mathcal{OP}_q.\mathcal{P}.A} T(s_q, a, st'_q),$$

where $s_q \in \mathcal{OP}_q.\mathcal{P}.S, s_q = st_q \times \neg s_q$. Henceforth, we refer to $\neg s_q$ as being the state features obtained by aggregating features in st using state dependencies, $\mathcal{OP}_q.SD$.

(e) Observation function, $\mathbb{O}$: Similar to the transition function computation, observation function is obtained by computing a product of different OrgPOMDP nodes:

$$\forall st' \in \mathbb{S}, \omega \in \Omega, \mathbb{O}(st', a, \omega) = \prod_{q<Q, a \in \mathcal{OP}_q.\mathcal{P}.A, \omega^l \in \omega} O(s'_q, a, \omega^l),$$

where $s'_q \in \mathcal{OP}_q.\mathcal{P}.S, s'_q = st'_q \times \neg s'_q$.

(f) Reward function, $\mathbb{R}$: The reward function for the flat POMDP is computed by summing the rewards obtained at different nodes in the OrgPOMDP. Formally,

$$\forall st, st' \in \mathbb{S}, \mathbb{R}(st, a, st') = \sum_{q<Q, a \in \mathcal{OP}_q.\mathcal{P}.A} R(s_q, a, s'_q),$$

where $s_q, s'_q \in \mathcal{OP}_q.\mathcal{P}.S, s_q = st_q \times \neg s_q, s'_q = st'_q \times \neg s'_q$.

It should be noted that due to the sequential nature of policy execution between parent nodes and child nodes in an OrgPOMDP, the flat POMDP model will have high costs (in its reward function) for actions that are not executable at a decision.

B. *Differences with Current Methods*

A number of approaches have been developed for exploiting hierarchical structure in POMDPs [9], [10], [11]. These approaches have not exploited factored representations. Similarly, a number of approaches have exploited factored representations in POMDPs [12], [13], [14], [15] but have not accounted for hierarchical relationships. Our approach not only combines these representations in a novel manner, but also extends them in significant ways.

OrgPOMDPs are different from hierarchical POMDPs in several ways. Hierarchical approaches typically break up a problem into a set of smaller problems of finer granularity using methods such as HMMs [11], finite-state controllers [9] or action hierarchies [10]. Like these methods, OrgPOMDPs uses a sequential execution order between different levels, i.e. the root (zero) level first executes its action, based on which the first level nodes execute their policies and so on, but in our case, we break the problem up into whole POMDPs which are executed by each node in the hierarchy. This results in the state dependencies, in which the states of the parent depend on the state features of the child and model dependencies, which define the effect of parent OrgPOMDP's action on the child nodes. The effect of the action can either be *nothing* or a combination of the four effects described in *action dependency* above. These dependencies can be efficiently represented by using the factorized parameter space.

Specifically, the OrgPOMDP approach has three fundamental differences from current hierarchical methods: (a) Hierarchical POMDPs only capture the hierarchical task decomposition (state space), while with OrgPOMDPs we are able to capture both the control (action space) and task (state space) hierarchies, a key factor required in modeling dynamic organizations; (b) We assume general hierarchies: (i) existing at the level of state features and not entire states; and (ii) aggregation of state features with general surjective functions ($f()$); and (c) For solving hierarchical POMDPs, multiple message passing iterations through the hierarchy are proposed, which limit the scalability considerably. In our approach, we are able to solve the OrgPOMDP with only one pass through the hierarchy, which provides considerable computational savings.

OrgPOMDPs are also substantially different from current factored POMDPs. The factored POMDP model (introduced by Boutilier and Poole [12] with improved algorithms following [13], [14], [15]), separates states and observations into sets of features and uses a two-stage dynamic Bayesian network to define the independencies between these features for each action. In an OrgPOMDP, there exist state features for parent nodes that are not related to state features of any of its child nodes. Similarly, there exist state features for child nodes that are not dependent on states or actions of parent node. This allows the problem to be factored differently at each node in the hierarchy. Also, actions can be factored in the OrgPOMDP model to further simplify the representation and model multiagent scenarios (where a set of workers is completing a task, each possessing a different set of actions). It is worth noting that while the actions can be factored, it remains future work to also factor the rewards in such a way as to extend the efficient solution concepts of the cooperative multiagent MDP model of Guestrin et al. [16] to the partially observable case.

IV. Algorithm

In this section, we provide an algorithm for fully specifying and solving an OrgPOMDP problem. The key challenge in solving the OrgPOMDP is reasoning with circular dependencies that exist between the parent and child nodes in the hierarchy: (a) The model for the child nodes is constructed based on the actions selected at the parent node; and (b) Because certain features of the state space at the parent nodes are dependent on states at child nodes, the transition probabilities for parent nodes can only be computed by knowing child policies. In this paper, the key idea is to resolve the circular dependency by converting each node in the hierarchy into a fully specified POMDP and solving it. We achieve this in three steps:

(a) We start from the root of the hierarchy and move towards the leaf nodes, while initializing the POMDPs at all nodes with states, actions and observations.

(b) At the leaf nodes of the hierarchy, OrgPOMDP nodes are already full specified POMDPs. The parent nodes for the leaf nodes are not POMDPs and the models at the leaf levels can change based on the state and actions of the parent node (as explained in state and action dependencies). To account for this, we generate and solve all POMDPs corresponding to the set of states and actions of the parent. The policies thus generated are stored and used for computing state transitions for the parent POMDPs. Our first contribution in this paper is in exploiting structure in the domain to reduce the number of possible POMDPs that are generated and solved.

(c) We construct the parent model by using the policies computed at the child (corresponding to all possible state, action pairs). This stage involves simulating the execution of policy for the child and subsequently computing the transition and observation probability functions at the parent.

Algorithm 1 OPSOLVER ($\mathcal{OP}, OrgProb$)

```
1: $\mathcal{P} \leftarrow$ CONVERTTOPOMDP($\mathcal{OP}, OrgProb$)
2: $\pi \leftarrow$ SOLVEPOMDP ($\mathcal{P}$)
```

Algorithm 2 CONVERTTOPOMDP($\mathcal{OP}, OrgProb$)

```
1: $\mathcal{P} \leftarrow$ INITIALIZEPOMDP($\mathcal{OP}, OrgProb$)
2: if $\mathcal{OP}.children \neq \phi$ then
3:     for all $child \in \mathcal{OP}.children$ do
4:         $\mathcal{P}.T_{child} \leftarrow$ GETCHILDTRANSPROB($\mathcal{P}, child, OrgProb$)
5:     $\mathcal{P}.T \leftarrow$ MERGETRANSPROBS($\mathcal{P}.T, OrgProb$)
6:     $\mathcal{P}.R \leftarrow$ GETREWARDS($\mathcal{P}.R, OrgProb$)
7:     $\mathcal{P}.O \leftarrow$ GETOBSPROBS($OrgProb$);
8: return $\mathcal{P}$
```

We present the technique for solving an OrgPOMDP in Algorithm 1. This algorithm solves the decision problem at the top of the organizational hierarchy by converting an OrgPOMDP to a POMDP. We perform this conversion using the recursive function CONVERTTOPOMDP()in Algorithm 2. In this function, the transition, observation and reward functions are computed based on whether it is a leaf node or an intermediate node in the hierarchy (line 2). For the leaf nodes, the POMDP is readily specified given the state and action of the parent node and hence the model conversion is performed on line 1 itself. However, the interesting aspect of this algorithm is when the computation is performed for intermediate nodes (lines 3 - 7).

Algorithm 3 GETCHILDTRANSPROB($\mathcal{P}, \mathcal{OP}Child, OrgProb$)

```
1: for all $(s, a) \in (\mathcal{P}.S, \mathcal{P}.A)$ do
2:     if INSTORE($s, a$) then
3:         $\langle ch\mathcal{P}, \pi_{ch} \rangle \leftarrow$ RETRIEVE($s, a$)
4:     else
5:         $ch\mathcal{P} \leftarrow$ CONVERTTOPOMDP($\mathcal{OP}Child, OrgProb$)
6:         $\pi_{ch} \leftarrow$ SOLVEPOMDP($ch\mathcal{P}$)
7:         STORE ($s, a, ch\mathcal{P}, \pi_{ch}$)
8:     $chBel \leftarrow$ GETCHILDBELIEF($s, OrgProb$)
9:     $chBelArr \leftarrow$ PROPAGATEBELIEF($chBel, \pi$)
10:    $\{\mathcal{P}.T_{child}(s, a, s')\} \leftarrow$ GETPARENTTRANS($chBelArr$)
11: return $\mathcal{P}.T_{child}$
```

As we have illustrated in Algorithm 3, the computation of transition probability for an intermediate node relies on converting to and solving the POMDP for the child nodes, over all state action pairs (line 1). The policy obtained by converting to and solving the child POMDP (lines 5-6) is simulated (lines 8-9) to compute the state transition probability for the parent OrgPOMDP (line 10). This simulation of the policy accounts for the state dependencies that exist between parent and child OrgPOMDPs. To avoid resolving of the same POMDPs and improve the performance, we maintain a cache of the generated POMDPs and their solution policies (lines 2-3, line 7).

Computing the policy: While the policy for the root node is present at the start of execution, rest of the OrgPOMDP policy (i.e. policies for nodes at lower levels) is constructed as observations are received. The key idea is to cache the policies for the child node, while constructing transition probabilities at the parent nodes. Specifically, in computing T(s,a,s') at the parent node: (a) we construct the POMDP model for the child corresponding to the model dependency, md_s^a; and (b) we solve this new child model to obtain the policy to be executed at the child and use the beliefs at the end of executing this policy. Once these child node policies are cached with reference to the model dependency, the policy for the child corresponding to parent's action is computed by using the parent belief state and aggregating the child policies corresponding to that belief state and the action taken at the parent.

The following two key ideas employed with our algorithm improve its efficiency considerably:

A. Exploiting action dependency structure

Depending on the action dependencies between a parent node and child node, we would potentially need to run all possible POMDPs at the lower level. In this paper, we exploit the structure in the model dependencies to avoid solving $S \times A$ POMDPs at the lower level. The key idea here is that over all model dependencies, md^a_s, we would only solve POMDPs which have a distinct effect on the child node than with any other (s,a) pair. Due to a factored state and action spaces, model dependencies can be redefined more compactly with respect to the state and action features. Thus, an action dependency with the factoredness taken into account will be - $md : (\times_{i \in G^s} S^i) \times (\times_{i \in G^a} A^i) \rightarrow S_1 \times A_1 \times T_1 \times R_1$, where G^s, G^a represent a subset of the features of the state and action spaces of $\mathcal{OP}.\mathcal{P}$. This method of representing dependencies allows for grouping of (state,action) pairs for which only one child POMDP needs to be solved.

B. Single pass through the control hierarchy

In earlier research on solving hierarchical POMDPs [10], multiple passes from root to leaf and leaf to root were proposed to obtain the policy. This was primarily due to the circular dependencies that existed between child and parent nodes. However, in this research, we do a single pass from root to the leaf and back from leaf to the root of the control hierarchy. This is possible, because we generate all feasible options (i.e. fully specified POMDPs) efficiently (see Section IV-A) for action dependencies and this converts the circular dependencies into uni-directional dependencies (from child to parent). This mechanism of generating fully specified POMDPs at each node increases the scalability of the approach considerably.

V. Results

We evaluated the performance of our approach on both *Rescue* and *Planning* scenarios, as described in Section 2. We also compare the performance of the OrgPOMDP model to a Flat POMDP model, which is computed using the mapping method provided in Section III-A. Flat POMDP does not take advantage of organizational infrastructure to allocate subtasks to the various child nodes and merely considers all possible allocations of tasks to the nodes in organizational hierarchy. We solve a POMDP in both cases by invoking the point based solver [4].

Except where noted, experiments involve a 3-level control and task hierarchy, corresponding to different levels of granularity in the organization and tasks. Leaf nodes at the bottom of the hierarchy have a single action, to progress on their assigned task with their assigned number of personnel. Each task has a discrete number of progress levels, and a reward is assigned for completing each task. Transitions advance the progress based on the resources assigned to the leaf node (more resources = higher probability of progress).

At the root level, a single node is assigned a root task that is decomposable into subtasks. Reward is accumulated up the hierarchy through accumulation functions such as sum, min, max, etc. We use sum for this set of experiments. Observations report noisy observed progress levels on tasks.

	OrgPOMDP					
ER	TIME	MaxS	AvgS	MaxA	MaxO	POMDPs
0	2.188	36	18.0	5	9	6
1	2.422	36	18.0	5	9	10
2	4.953	54	21.8	6	9	14
3	9.922	72	26.0	7	9	18
4	25.687	108	32.7	9	9	22
5	116.094	198	46.4	14	9	26

	FlatPOMDP			
ER	TIME	$\vert S\vert$	$\vert A\vert$	$\vert O\vert$
0	-	486	8	81
1	-	1944	11	81
2	-	4860	17	81
3	-	9720	27	81
4	-	17010	42	81
5	-	27216	63	81

Table I
OrgPOMDP and Flat POMDP performance for *Rescue* scenario. '-' indicates solver failed to complete in 1 hour.

HORIZON	1	2	3	4	5
FLAT-HORIZON	3	14	39	84	155
TIME	5.6	13.8	52.1	140.8	466.6

Table II
Horizon versus time for *Rescue* scenario.

A. Rescue Scenario

We first report results on the *Rescue* scenario depicted in Figure 1. Extra resources are available from the highest level, and can be added to each aircraft and each team in turn. Table I compares performance of Flat POMDP to OrgPOMDP for varying numbers of extra personnel. For OrgPOMDPs, we report the key complexity indicators such as number of POMDP problems (POMDPs) generated by the OrgPOMDP representation, the size of the subproblem with the largest state (MaxS), action (MaxA), and observation space (MaxO) (subproblems faced at different levels are different, thus the table represents the size of the largest subproblem), and the average state space (AvgS) size in the organization. For Flat POMDPs, we report the number of states ($|S|$), actions ($|A|$), and observations ($|O|$) in it. The table shows that even for simpler versions of the problem with no extra personnel, the Flat POMDP representation of these problems was intractable for our POMDP solver, due to the large number of states and observations.

Next, we consider larger problems to analyze scalability properties of the OrgPOMDP model. Having established the relative inability of Flat POMDP to handle even the simpler problems, we do not report further on Flat POMDP time (which is no solution, unless stated otherwise) on more complicated problems. Rather, we use properties of the Flat POMDP in order to provide context to the scalability of OrgPOMDP with respect to problem features. We start with time horizon. Table II shows the solution time versus the horizon for the OrgPOMDP. Also shown is the time horizon for the equivalent Flat POMDP problem. For hierarchies of

Number of Areas	Time	MaxS	MaxA	MaxO
2	13.7	54	6	9
3	52.6	54	6	9
4	55.0	54	6	9
5	55.3	54	6	9

Table III
EFFECT OF ADDING MORE AREAS

Aircraft	Time	MaxS	MaxA	MaxO
2	55.3	54	6	9
3	228.2	162	10	9
4	1800	648	28	9

Table IV
EFFECT OF ADDING MORE AIRCRAFT.

size 3, in cases where the individual OrgPOMDP problems are of horizon H, the time horizon in the equivalent Flat POMDP problem will equal $H^3 + H^2 + H$ steps, as each child runs for H steps for each step of the parent. The table shows that solution time for OrgPOMDP grows roughly proportionally to the underlying Flat POMDP time horizon, not the OrgPOMDP time horizon.

We now compare the effects of adding to Task structure, as well as the effects of adding complexity to Organization structure. Table III shows the effect of adding tasks on the rescue problem while keeping other parameters constant. In the scenarios depicted, more areas (each with two sub locations) were added to the middle of the three levels of the hierarchy. As the table shows, the effect of adding areas to this particular scenario is negligible. This was because the most direct effect of adding areas is to add states to the root node, it can contemplate new task allocations. However, the larger POMDPs in the scenario are not at the root node, and thus the addition does not drive the timing.

However, adding components to the organizational structure does have an effect. Table IV shows the time taken versus number of aircraft (nodes at the middle level). When adding aircraft, the organizational POMDP grows exponentially. Each aircraft can be assigned to any of the areas, and furthermore each aircraft can be assigned any number of personnel. The Flat POMDP was $\langle 19440s, 35a, 81o \rangle$ for the 4 aircraft scenario, orders of magnitude larger than the size of a solvable Flat POMDP.

In summary, the effect of adding to either structure must be analyzed in terms of whether it increases the decision space of an organizational node and the position of that node in the hierarchy. If the node is a computational bottleneck, such as the root node in Table III, adding complexity will have considerable effect. Similarly adding to the decision space of a node also increases overall run-time. In either case, a great deal of scalability was demonstrated for this problem.

B. Planning Scenario

The goal of these experiments is to simulate operations within the AOC. In this domain, tasks consist of individual missions, and each mission task structure matches the organization structure, that is, there is no confusion in the AOC as to which organization node was appropriate for a task. However, staffing allocation decisions for each node must still be made. Thus, manipulation of Extra Resources is the key.

ER	Time	MaxS	MaxA	MaxO
0	36.1	36	5	9
1	75.5	36	5	9
2	158.7	54	6	9
3	320.0	72	7	9
4	739.7	108	9	9
5	2688.9	198	14	9

Table V
TIMING RESULTS FOR *Planning* SCENARIO.

The *Planning* task had 4 missions, 2 divisions each with 2 planning teams, as in Figure 1. In this scenario, as opposed to the *Rescue* scenario, each team was able to perform two tasks concurrently. The resulting Flat POMDP had $\langle 52488s, 9a, 6561o \rangle$ for the case with 0 extra personnel, and no solution was found within an hour, because again this is orders of magnitude larger than any Flat POMDP that is solvable by current approaches. For the OrgPOMDP, we plotted results for varying numbers of extra personnel in Table V. The effects of extra resources were similar to that in the *Rescue* scenario, in that adding extra resources added to the decision space which in turn made the probleme less tractable.

The policies generated from the OrgPOMDP are large, planning for all possible contingencies from each node of the organization. To give a sense of the dynamic organization structure produced by the policies, we include in Table VI examples of the decisions made by the organizational policy at execution time. Transitions and observations were chosen stochastically based on the transition and observation functions of the OrgPOMDP. It can be seen that the organization will adapt based on the availability of resources and the tasks assigned to the different workers.

Finally, we compare the performance of a dynamic organization represented using an OrgPOMDP and a static organization. Table VII compares the average quality of the OrgPOMDP policy, which dynamically changes the organizational structure over time, with the average quality of a static organization which does not adjust over time. Because the OrgPOMDP policy can reallocate tasking and resources, such as from completed tasks to incomplete tasks, or from tasks with low progress to more promising tasks in the last horizon steps, it is able to achieve higher scores. To generate these scores, a reward of 1 was allocated for each time step at which a low level task was complete, a reward of 3 was allocated for mid-level tasks, and a reward of 7 was allocated for the highest level tasks.

Node	Timestep	Action
Commander	1	TA(1,0)
Strat Division	2	TA(0,1)
Strat Plans Team	3-4	RA(0,4), RA(3,1)
Strat Guidance Team	3-4	Wait, Wait
Strat Division	5	TA(0,1)
Strat Plans Team	6-7	RA(0,4), Wait
Strat Guidance Team	6-7	Wait, Wait
CP Division	2	TA(0,1)
Target Effects Team	3-4	Wait, Wait
Air Task Team	3-4	Wait, Wait
CP Divison	5	TA(0,1)
Target Effects Team	6-7	Wait, Wait
Air Task Team	6-7	Wait, Wait
...	...	...
...	...	...

Table VI
EXECUTION OF ORGPOMDP POLICY FOR A HORIZON-2 PROBLEM FOR THE ORGANIZATION IN 1. "WAIT" DOES NOT CHANGE THE ORGANIZATION STRUCTURE, "TA" CHANGES TASK ALLOCATION OF CHILD NODES AND "RA" CHANGES RESOURCE ALLOCATION.

Dynamic	Static
355	188

Table VII
DYNAMIC ORGANIZATION (ORGPOMDP) VS. STATIC ORGANIZATION

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have introduced the OrgPOMDP model and presented an algorithm to solve it efficiently. The OrgPOMDP model efficiently represents problems having a hierarchical structure with general parent and child dependencies. Furthermore, this model can also take advantage of factored model parameters to even more efficiently represent the problem. The algorithm to solve OrgPOMDPs makes use of this problem structure to allow solutions to be found efficiently. The OrgPOMDP model is very general and many hierarchical problems in the of academic interest as well as real world scenarios can be represented, permitting more compact description and more scalable solutions.

We conducted experiments that demonstrate that our OrgPOMDP approach is both scalable and useful. We have applied OrgPOMDPs to two realistic scenarios used by the Air and Space Operations Center (AOC): Rescue Mission and Organizational Planning. Our results in two domains show that OrgPOMDPs dramatically reduces computation time. In fact, our results show that as we shifted to the more complex domain of planning for the entire organization, we quickly reached a point where the OrgPOMDP could provide optimal solutions, whereas a traditional POMDP could not. The OrgPOMDP's advantage is that it leverages the hierarchical nature of the organization and the structure in dependencies to compute policies for decision makers at various levels efficiently.

It is also worth noting that because the problem is transformed into a set of POMDPs, any POMDP solver can be used on these subproblems, allowing further improvements to performance as solvers improve. In this paper, we have employed PBVI [10], an approximate technique to solve POMDPs. However, any of the other leading approximate solvers can be employed [10], [6], [7] to reduce computation time while producing high quality approximate solutions for each POMDP. Lastly, OrgPOMDPs can leverage the factored POMDP algorithms such as factored PERSEUS [14] and factored HSVI [15] on the low level problems, further increasing scalability.

REFERENCES

[1] C. Amato, B. Bonet, and S. Zilberstein, "Finite-state controllers based on mealy machines for centralized and decentralized POMDPs," in *AAAI*, 2010.

[2] B. Bonet and H. Geffner, "Solving POMDPs: RTDP-Bel vs. point-based algorithms," in *IJCAI*, 2009.

[3] S. Ji, R. Parr, H. Li, X. Liao, and L. Carin, "Point-based policy iteration," in *AAAI*, 2007.

[4] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *IJCAI*, 2003.

[5] G. Shani, R. I. Brafman, and S. E. Shimony, "Forward search value iteration for POMDPs," in *IJCAI*, 2007.

[6] T. Smith and R. G. Simmons, "Point-based POMDP algorithms: Improved analysis and implementation," in *UAI*, 2005.

[7] M. T. J. Spaan and N.Vlassis, "Perseus: Randomized point-based value iteration for POMDPs," *JAIR*, vol. 24, pp. 195–220, 2005.

[8] P. Varakantham, R. Maheswaran, T. Gupta, and M.Tambe, "Towards efficient computation of quality bounded solutions in POMDPs." in *IJCAI*, 2007.

[9] E. A. Hansen and R. Zhou, "Synthesis of hierarchical finite-state controllers for POMDPs," in *ICAPS*, 2003.

[10] J. Pineau, N. Roy, and S. Thrun, "A hierarchical approach to POMDP planning and execution," in *ICML Workshop on Hierarchy and Memory in Reinforcement Learning*, 2001.

[11] G. Theocharous, K. Murphy, and L. P. Kaelbling, "Representing hierarchical POMDPs as DBNs for multi-scale robot localization," 2004.

[12] C. Boutilier and D. Poole, "Computing optimal policies for partially observable decision processes using compact representations," in *AAAI*, 1996.

[13] E. A. Hansen and Z. Feng, "Dynamic programming for POMDPs using a factored state representation," in *AIPS*, 2000.

[14] P. Poupart, "Exploiting structure to efficiently solve large scale partially observable Markov decision processes," Ph.D. dissertation, Department of Computer Science, University of Toronto, 2005.

[15] H. S. Sim, K.-E. Kim, J. H. Kim, D.-S. Chang, and M.-W. Koo, "Symbolic heuristic search value iteration for factored POMDPs," in *AAAI*, 2008.

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