Efficient Multi-scale POMDPs for Robotic Object Search and Delivery

Efficient Multi-scale POMDPs for Robotic Object Search and Delivery

Luc Holzherr, Julian Förster, Michel Breyer, Juan Nieto, Roland Siegwart and Jen Jen Chung

***Abstract*— We present a novel hierarchical POMDP framework to solve an object search and delivery task where the agent is given a prior belief about the possible item locations. Solving POMDPs is computationally demanding and, as such, applications have typically been limited to small environments. The proposed hierarchical POMDP framework performs reasoning on multiple spatial scales in order to reduce computation time. The problem is first solved in the top layer of the hierarchy with a coarsely discretized state space. Its solution is refined in the lower layers with increasing resolution. Three different methods for propagating information down the spatial hierarchy are discussed and validated in simulation. We show that a two-layer multi-scale POMDP decreases computation time by an order of magnitude allowing for real-time applications while maintaining high solution quality. For large problems that require three layers to reach the desired resolution, computation time speedups by two orders of magnitude are achieved.**

I. INTRODUCTION

Partially observable Markov decision processes (POMDPs) are a powerful tool for planning under uncertainty. They describe a mathematical framework for sequential decision making problems with uncertain states (represented by a belief) in stochastic environments [1]. Solving POMDPs exactly is computationally intractable for many real-world problems [2]. Reasoning over a task such as finding and delivering objects in a multi-room environment requires reasoning over all possible object locations as well as the optimal order in which to execute the task if multiple deliveries are involved [3]–[5].

Instead, anytime algorithms which approximate the optimal solution through sampling are used. These can be divided in two major groups, Monte Carlo methods and point-based methods. Monte Carlo methods are online solvers [5]–[8]. They perform a Monte Carlo lookahead search on a belief tree to find the next best action to execute. They can also be adapted for problems with a continuous state space [8]. Point-based methods accelerate the solving process by sampling a small set of points from the belief space and approximating the value function through a set of α-vectors [9]–[12]. These are offline solvers, i.e. the output is a value function which can be used to compute a policy.

Another strand of work aims at modifying the problem structure in order to decrease the computational burden. Sutton et al. extended the MDP framework to include options [13], sometimes referred to as macro-actions, which can be applied to POMDPs if the option is a closed-loop policy acting on the *belief* space as shown in the work of Omidshafiei et al. [14]. An option is an action that takes multiple timesteps to execute and solves a subproblem of the task like picking up an item. An option can be added as separate actions to the action space of the (PO)MDP. This speeds up the solving process since it enables larger state transitions. However, the state space considered remains identical to the original (PO)MDP problem, which limits the computational benefits.

Similar methods related to abstract MDPs [15] and hierarchical MDPs [16]–[19] perform some form of problem partitioning to achieve computational efficiency. However, this often comes at the cost of the final solution quality. These methods also sometimes require that the original (PO)MDP can be split into decoupled or non-interacting subproblems, which is an invalid assumption in many real-world problems.

In particular, for hierarchical (PO)MDP formulations, the higher layer transition, reward and observation models need to be computed by either first solving the hierarchy from the bottom up [16], [18] or by using heuristics [19]. The former approach requires solving many (PO)MDP problems, which sacrifices part of the computational gains, while for the latter approach it can be difficult to find good heuristics.

In this paper, we propose three multi-scale POMDP methods for the task of object search and delivery that address these problems by creating the higher layer models through fixed policies on the lower layers. The multi-scale methods differ in how information is propagated through the layers: either through the higher layer's computed action, both the computed action and value function, or simply propagating the value function to the next layer. We use SARSOP [11], a state-of-the-art point-based method, as our underlying POMDP solver. Our results on single and multi-object search and delivery tasks show a substantial speedup in computation time compared to directly solving the full-resolution (flat) POMDP, which for larger problems often fails due to solver time-out. At the same time, our multi-scale methods maintain comparable solution quality on modest problem sizes while actually demonstrating superior performance (faster delivery times) compared to the solutions found by the flat POMDP when tested on larger problems (larger environments, more items to deliver).

II. PROBLEM FORMULATION

The agent's task is to find one or multiple items in an office environment with multiple rooms and deliver each item to a specified goal location in as little time as possible. A map of the static, closed environment is given and the agent has

This work was supported in part by ABB Corporate Research, the Luxembourg National Research Fund (FNR) 12571953 and the ETH Foundation with an unrestricted gift from Huawei Technologies.

The authors are with the Autonomous Systems Lab, ETH Zürich, Zürich 8092, Switzerland. {luch; fjulian; mbreyer; jnieto; rsiegwart; chungj}@ethz.ch

a prior estimate about the item locations in the form of a probability distribution. An object's goal location is specified with coordinates. It is further assumed that items are uniquely identifiable (e.g. via an instance segmentation pipeline). The agent modeled for this task is a simple ground robot with a forward-facing camera and a manipulator arm. The robot can only carry one item at a time.

We formulate the object search and navigation problem as a POMDP. Formally, a POMDP is defined by the tuple $P = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, b_0, \gamma \rangle$, where $\mathcal{O}$ is a discrete set of observations, Z is the observation model, b_0 is the initial belief of the state and $\gamma \in [0,1]$ is the discount factor. At each time step the agent is in a state $s \in \mathcal{S}$, chooses an action $a \in \mathcal{A}$, receives a reward $R(s,a)$, transitions according to $T(s,a,s')$ to state s' and then receives an observation $o \in \mathcal{O}$ according to $Z(o,s',a)$.

Given an initial belief distribution of the item locations, b_0, the goal is to solve for the agent policy π (or corresponding value function $V(b(s))$) that will maximize the discounted sum of future rewards as defined by R. For our object search and delivery task, the reward model is defined by,

$$R = R_{\text{time}} + R_{\text{subtask}} + R_{\text{task}}, \tag{1}$$

where the first term is a time penalty (negative), the second term provides an intermediate reward for picking up a target item, and the final term rewards the delivery of an item to its target location. As discussed in Section I, solving a POMDP scales poorly with problem size and dimensionality, even when using approximation methods such as SARSOP [11]. Hierarchical solutions are key to achieving computational tractability, yet it is still unclear how information should be propagated between the different hierarchical planning layers. Thus, this is the core investigation of our work.

III. OBJECT SEARCH AND DELIVERY DOMAIN

We adapt the work from [20], [21] for a search and delivery task involving K items. The environment is divided into N separate non-overlapping regions, referred to as *nodes*. Fig. 1b shows an example of a small office with three rooms that has been partitioned into $N = 9$ nodes n_0 to n_8. Nodes can have any shape and do not need to be rectangles.

The state $s = [x_a, x_1, \cdots, x_K]$ is a vector of size $K+1$ with one variable, x_a, describing the agent's position (current node) and K variables for the items, where

$$\begin{aligned} &x_a \in \{n_0, \cdots, n_{N-1}\}, \\ &x_k \in \{n_0, \cdots, n_{N-1}, \texttt{agent}, \texttt{goal}\}, \quad 1 \le k \le K, \\ &|\mathcal{S}| = N(N+2)^K. \end{aligned} \tag{2}$$

Since the object search and delivery task has a defined and observable goal, then a state is considered a terminal state, iff all item variables take the value `goal`.

The action space consists of four types of actions: (i) $\texttt{nav}(n_i, n_j)$ navigates the agent between adjacent nodes; (ii) `look_around` moves the camera field of view (e.g. by rotating) to observe the node the agent is in; (iii) `pickup`k instructs the agent to pick up item k if it is observed within the current node; and (iv) `release` allows the agent to deliver the item it is carrying within the current node.

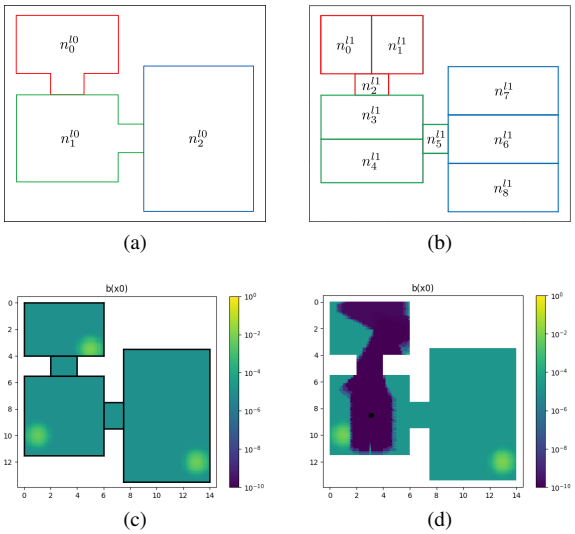


Fig. 1: A three-room office environment decomposed into (a) three or (b) nine nodes. (c) Initial belief of an item's location. (d) Updated belief after executing the `look_around` and `nav` actions.

The observation $o = [o_1, \cdots, o_K]$ is a vector of size K, where each variable corresponds to an item. The observation o_k takes the value `no` if item k is not observed in this time step, n_i if the item was observed in node i, and `agent` if the agent is currently carrying item k. If the item variable x_k takes the value `goal`, the observation always returns `no`.

$$\begin{aligned} &o_k \in \{\texttt{no}, n_0, \cdots, n_{N-1}, \texttt{agent}\}, \quad 1 \le k \le K, \\ &|\mathcal{O}| = (N+2)^K \end{aligned} \tag{3}$$

In this work we adopt a deterministic transition function and assume conditional independence between the state variables given the current state-action. Specifically, $\texttt{nav}(n_i, n_j)$ only affects the agent state x_a (indeed, it is the only action that affects x_a). It results in x_a transitioning to n_j with probability 1 if the agent started the action in n_i or vice versa. `look_around` allows the agent to search the current node for items but does not cause changes to the state. `pickup`k transitions item k to state `agent` if the agent is in the same node as the item and is not already carrying another item. `release` successfully delivers item k if the agent is in the same node as the goal location, otherwise the item is dropped and takes the state of the current node.

The observation model we use also assumes conditional independence between the observation variables (item locations, o_k) given the action and resulting state. Note that this may not always hold, e.g. some items like salt and pepper are often co-located; however, it is a valid assumption for our particular problem domain. The observation model takes into account the robot's object recognition rate and false positive detection rate as well as an approximation of the swept camera field of view (FOV) during the `nav` and `look_around` actions. For example, the probability of observing an item during a `nav` action in node i is approximated as the ratio between the expected observed

area of node i as swept out by the camera FOV and the total area of node i. If item k is picked up, its state $x'_k = \texttt{agent}$ becomes perfectly observed, while we assume that no other items are observed during this action. Similarly, for the `release` action observations.

As shown in (1), the reward function consists of three components. The time penalty R_{time} is approximated for each high-level action since execution times may differ depending on various low-level factors. The run-time of an action subroutine depends on the specific agent location, item positions and the current state of the belief grid. Modeling R_{time} requires approximating these execution times, either through a simulation using sampling to get robust averaged values or by using a heuristic. In this work we implemented the former simulation averaging approach. For R_{subtask}, the agent receives a positive reward r_{pickup} for successfully grabbing an item. When the agent releases the item in a later time step the same reward must be subtracted to prevent the agent from repeatedly picking up and releasing an item to accumulate the subtask reward. Similarly, there is a positive reward r_{delivery} for delivering an item to its goal location.

The initial belief is provided as a distribution that defines the probability for each partially observable state variable. We assume that the agent state x_a is fully observable and so only the item locations are represented in b. Following [5], we exploit the independence between object locations and factor the belief into K high-resolution discretized grids, where the value of each cell is the belief that the item occupies the cell. Figures 1c and 1d show the initial and updated belief grid after the agent has executed the `look_around` and `nav` actions.

Finally, discounting can be used to value future rewards less than current ones and is required for methods like SARSOP. Since the goal of this object search and delivery problem is to minimize the total time, the discounting factor is chosen close to one, $\gamma = 0.999$. Further implementation details can be found in our open-sourced code[1].

IV. MULTI-SCALE POMDP AGENTS

Here we present a novel hierarchical POMDP framework which greatly reduces computation times while maintaining comparable solution quality to solving the flat (full) POMDP. The core idea is to split the original POMDP into multiple smaller problems with different spatial resolutions. An example spatial decomposition is shown in Figs. 1a and 1b, where the top layer $l0$ constructs a POMDP with only three nodes, while the next layer $l1$ contains the original POMDP resolution with nine nodes. Each node of $l1$ can be mapped uniquely to one node of $l0$. This relationship is true in general between nodes in layers $l\,(\lambda+1)$ and $l\lambda$. Then, starting from $l0$, which has the coarsest resolution, we solve the resulting POMDP for its proposed next action and then refine this over the subsequent layers with increasing resolution. The lowest layer of the hierarchy corresponds to the original POMDP problem. In the following, we use $\star^{l\lambda}$ to indicate that the variable belongs to layer λ.

[1] https://github.com/ethz-asl/multi_scale_search

A. Multi-scale Transition, Reward and Observation Models

The higher layer POMDPs are formulated similarly to the original (lowest layer) POMDP as described in Section III. The main differences lie in the higher layer reward, transition and observation models, which are created from the lower layer ones in a recursive manner through the $(\lambda+1)$-layer policies that execute the λ-layer actions. In this work, we consider relatively simple actions at each layer and so we use handcrafted open-loop policies. We formulate the policy π_{OL} as an ordered list where element $\pi_{\text{OL}}[i]$ is the i-th action to execute. Instead of using open-loop policies, one could also use closed-loop policies on states $\pi(s)$. This corresponds to the options/macro-action model for MDPs [13], [15]. It is also possible to use automatically constructed closed-loop policies on beliefs [14].

Using the decomposition in Fig. 1 as an example, selecting $\texttt{nav}\left(n_0^{l0}, n_1^{l0}\right)$ when $x_a^{l0} = n_0^{l0}$ results in the execution of,

$$\pi_{\text{OL}}\left(\texttt{nav}\left(n_0^{l0}, n_1^{l0}\right)\right) = \begin{cases} \left[\texttt{nav}\left(n_0^{l1}, n_2^{l1}\right), \texttt{nav}\left(n_2^{l1}, n_3^{l1}\right)\right], & \text{if } x_a^{l1} = n_0^{l1}; \\ \left[\texttt{nav}\left(n_1^{l1}, n_2^{l1}\right), \texttt{nav}\left(n_2^{l1}, n_3^{l1}\right)\right], & \text{if } x_a^{l1} = n_1^{l1}; \\ \left[\texttt{nav}\left(n_2^{l1}, n_3^{l1}\right)\right], & \text{if } x_a^{l1} = n_2^{l1}. \end{cases} \tag{4}$$

Continuing with this example, the reward for executing π_{OL} in n_0^{l1} is the discounted sum of rewards for each action:

$$R^{\pi_{\text{OL}}}\left(x_a^{l1} = n_0^{l1}\right) = R\left(x_a^{l1} = n_0^{l1}, a^{l1} = \texttt{nav}\left(n_0^{l1}, n_2^{l1}\right)\right) + \gamma R\left(x_a^{l1} = n_2^{l1}, a^{l1} = \texttt{nav}\left(n_2^{l1}, n_3^{l1}\right)\right). \tag{5}$$

The layer 0 reward can now be calculated by averaging the rewards for executing π_{OL} for the different starting states:

$$R\left(x_a^{l0} = n_0^{l0}, a^{l0} = \texttt{nav}\left(n_0^{l0}, n_1^{l0}\right)\right) = \frac{1}{3}\left(R^{\pi_{\text{OL}}}\left(x_a^{l1} = n_0^{l1}\right) + R^{\pi_{\text{OL}}}\left(x_a^{l1} = n_1^{l1}\right) + R^{\pi_{\text{OL}}}\left(x_a^{l1} = n_2^{l1}\right)\right). \tag{6}$$

Note that multiple open-loop policies could be used and the respective rewards averaged to compute the layer 0 reward.

In a more general setup with a non-deterministic transition model the expected reward for following π_{OL} on layer λ starting in state $s^{l\lambda}$ can be computed recursively as:

$$R^{\pi_{\text{OL}}}\left(s^{l\lambda}\right) = R\left(s^{l\lambda}, \pi_{\text{OL}}[1]\right) + \gamma \sum_{s' \in \mathcal{S}^{l\lambda}} T\left(s^{l\lambda}, a^{l\lambda} = \pi_{\text{OL}}[1], s'\right) R^{\pi_{\text{OL}}[2:]}\left(s'\right), \tag{7}$$

where $\pi_{\text{OL}}[2:]$ is the same list starting at the second element.

The higher layer observation and transition models are constructed with the same approach. One action of layer λ corresponds to a policy on the next layer down $l\,(\lambda+1)$, which produces a sequence of observations. Thus, we first need to define a mapping from the $(\lambda+1)$-layer observations to the λ-layer observation. The observation of item k in layer λ takes the value n_i if any of the observations on layer $(\lambda+1)$ is within n_i. If all observations on the lower layer are `no` the higher layer observation also takes the value `no`:

$$o_k^{l\lambda} = \begin{cases} n_i^{l\lambda}, & \text{if any } o_k^{l(\lambda+1)} \in \mathcal{N}(n_i^{l\lambda}); \\ \texttt{no}, & \text{if all } o_k^{l(\lambda+1)} = \texttt{no}, \end{cases} \tag{8}$$

$\mathcal{N}(n_i^{l\lambda})$ is the set of nodes on layer $(\lambda+1)$ that lie in $n_i^{l\lambda}$.

Solving the top layer $l0$ yields the value function $V^{l0}(b)$ and the first action to execute a^{l0}. In subsequent layers, only a subset of the state, observation and action spaces is considered, based on the state-action of the layer above. Decision information only propagates down. The multi-scale methods we present do not compute the entire solution beforehand but only the next action to execute.

B. Reasoning over Multiple Spatial Scales

Given the higher layer model definitions it is now possible to solve each POMDP layer. In this subsection, we present three different methods for propagating the solutions of each spatial layer. The methods differ in computation speed and the reasoning capabilities of the lower layers. All methods consider restricted state, action and observation spaces on the lower layers to achieve a speedup in computation time.

1) Action Propagation (AP): The first method is the simplest and explores a leader-follower relationship where the lower layer attempts to directly execute the action chosen by the layer above. The lower layers consider as few variables and values as strictly needed to execute the higher layer action. The transition, reward and observation models remain unchanged for the lower layers. We illustrate the concept with an example of the restricted POMDP solved in layer 1 for a layer 0 `nav` action.

Example 1 Consider the three room environment from Fig. 1 where the agent is currently in n_0^{l0} and the task is to find and deliver K items. Solving layer 0 yields the action $a^{l0} = \texttt{nav}\left(n_0^{l0}, n_1^{l0}\right)$. The layer 1 POMDP only considers the layer 1 nodes that are within the current layer 0 node n_0^{l0} and the entry node to n_1^{l0} which is n_3^{l1}. To navigate the agent, the items do not need to be considered in this sub-problem, which makes layer 1 an MDP. The terminal state on layer 0 is when the agent is within the node n_1^{l0}. Further, the action space only consists of navigation actions:

$$\begin{aligned} x_a^{l1} &\in \left\{n_0^{l1}, n_1^{l1}, n_2^{l1}, n_3^{l1}\right\} \\ s_{term}^{l1} &= \left[x_a^{l1} = n_3^{l1}\right]^\intercal \\ a^{l1} &\in \left\{\texttt{nav}\left(n_0^{l1}, n_1^{l1}\right), \texttt{nav}\left(n_0^{l1}, n_2^{l1}\right), \texttt{nav}\left(n_1^{l1}, n_2^{l1}\right), \right. \\ &\qquad \left. \texttt{nav}\left(n_2^{l1}, n_3^{l1}\right)\right\}. \end{aligned} \tag{9}$$

□

AP yields a big speedup regarding computation time compared to solving the flat (single-layer) POMDP from Section III as the state, action and observation spaces of both layer 0 and layer 1 are smaller. The drawback is that the lower layers perform very little reasoning and just try to solve the action of the layer above in as little time as possible. Since the higher layers use a coarse resolution their proposed action could be suboptimal. Later in Section V we show that such problems start to occur for larger environments where more than two layers are used. One particular problem for large environments is inconsistent search behaviour. The AP agent stops the search in a room and goes to another room even though the belief that the item is in the room is still relatively high. The next two methods address those issues by giving the lower layers more flexibility and information about the entire problem.

2) Action-Value Propagation (AVP): The second method aims to improve the lower layer reasoning while still yielding a large computational speedup. The lower layers access the value function of the layer above to get global context about the problem. The value function can be incorporated through the terminal state reward as is shown in Example 2. Thus, the lower layers can, for example, choose to explore the nodes before executing the proposed action of the layer above. This changes the belief at the terminal state and therefore the terminal state reward. Similar to AP, the state, action and observation spaces of the lower layers are determined by the higher layer state and action. However, all item variables are considered on all layers, and in addition to the $(\lambda+1)$ node values, item variables can also take the value `not_here` for which the initial belief is given as one minus the belief that the item is in $n_i^{l\lambda}$.

Example 2 Consider the same problem as in Example 1. In contrast to AP, AVP accounts for all item variables on layer 1 and the action space now also contains the `look_around` action (note that x_a^{l1} remains the same as in Example 1 (9)):

$$\begin{aligned} x_k^{l1} &\in \left\{n_0^{l1}, n_1^{l1}, n_2^{l1}, n_3^{l1}, \texttt{not\_here}\right\}, \quad 1 \le k \le K, \\ s_{\text{term}}^{l1} &= \left[x_a^{l1} = n_3^{l1}, x_k^{l1}\right]^\intercal, \quad 1 \le k \le K, \\ o_k^{l1} &\in \left\{\texttt{no}, n_0^{l1}, n_1^{l1}, n_2^{l1}, n_3^{l1}, \texttt{agent}\right\}, 1 \le k \le K, \\ a^{l1} &\in \left\{\texttt{nav}\left(n_0^{l1}, n_1^{l1}\right), \texttt{nav}\left(n_0^{l1}, n_2^{l1}\right), \texttt{nav}\left(n_1^{l1}, n_2^{l1}\right), \right. \\ &\qquad \left. \texttt{nav}\left(n_2^{l1}, n_3^{l1}\right), \texttt{look\_around}\right\} \end{aligned} \tag{10}$$

This gives layer 1 the flexibility to search the first room before driving to the second room. The terminal state reward is an additional reward which is defined through the value function of layer 0. Note that there are 5^K terminal states for all item combinations, since each item state can take 5 different values. For example, in a two-item scenario, the terminal reward where the first item is in n_1^{l1} and the second item in n_3^{l1} is given by:

$$\begin{aligned} &R_{\text{term}}\left(s^{l1} = \left[n_2^{l1}, n_1^{l1}, n_3^{l1}\right]^\intercal, a^{l1} = \texttt{nav}\left(n_2^{l1}, n_3^{l1}\right)\right) = \\ &\quad V\left(x_a^{l0} = n_1^{l0}, b\left(x_1^{l0} = n_0^{l0}\right) = 1, b\left(x_2^{l0} = n_1^{l0}\right) = 1\right). \end{aligned} \tag{11}$$

The belief vector b^{l0} at the terminal state can be altered through the actions that the agent chooses. Through changing b^{l0} the expected terminal reward changes, thus giving layer 1 context about the entire problem. □

The improved lower layer reasoning leads to better solutions as is shown in Section V but has slightly longer computation times compared to AP since all items are considered in the lower layers and additional exploration actions are included in the action set. AVP resolves the inconsistency problems of AP as the lower layers allow for more exploration. However, the lower layers still have to follow the proposed action of the above layers. This can

become a problem for larger environments where the coarse discretization of layer 0 can lead to suboptimal decisions. *Value Propagation* resolves this limitation with additional terminal states in the lower layers.

3) Value Propagation (VP): The third method improves the lower layer reasoning by lifting the state restrictions coming from the proposed action of the layer above. The lower layer POMDPs are instead guided purely by the value function of the layer above, which is incorporated in the same way as in AVP. The concept is demonstrated in Example 3.

Example 3 Again, consider the same problem as in Example 1. Solving layer 0 yields the value function $V(b^{l0})$. The restricted state, action and observation space is determined solely by the agent variable of layer 0, i.e. x_a^{l0}. Thus, x_a^{l1} and o_k^{l1} remain the same as in Example 2 (10). Terminal states are reached when the agent leaves the current layer 0 node or if it successfully grabs an item:

$$\begin{aligned} x_k^{l1} &\in \left\{ n_0^{l1}, n_1^{l1}, n_2^{l1}, n_3^{l1}, \texttt{not\_here}, \texttt{agent} \right\}, 1 \le k \le K, \\ s_{\text{term}}^{l1} &\in \left\{ \left[x_a^{l1} = n_3^{l1}, x_k^{l1} \right]^\intercal, \left[x_a^{l1}, x_1^{l1} = \texttt{agent}, \cdots, x_K^{l1} \right]^\intercal, \right. \\ &\quad \left. \cdots, \left[x_a^{l1}, x_1^{l1}, \cdots, x_K^{l1} = \texttt{agent} \right]^\intercal \right\}, 1 \le k \le K, \quad (12) \\ a^{l1} &\in \left\{ \texttt{nav}\left(n_0^{l1}, n_1^{l1}\right), \texttt{nav}\left(n_0^{l1}, n_2^{l1}\right), \texttt{nav}\left(n_1^{l1}, n_2^{l1}\right), \right. \\ &\quad \left. \texttt{nav}\left(n_2^{l1}, n_3^{l1}\right), \texttt{look\_around}, \texttt{pickup}k \right\} \qquad \square \end{aligned}$$

Thus, layer 1 has complete freedom in its choice of action and does not need to follow the proposed action of layer 0. Layer 1 solves its reduced POMDP using the value function of layer 0 to inform its terminal rewards, similar to (11). Through the higher resolution of layer 1, the agent can correct for model approximation errors of layer 0, for example, finding shorter routes between nodes. However, as the example shows, the state, action and observation spaces are larger than for either AP or AVP, which increases overall computation time.

C. Multi-scale Search Properties

Each of the multi-scale methods splits one large POMDP into multiple smaller POMDPs. With the hierarchical layout from Fig. 1, where each node is split into $\nu = 3$ subnodes on the next lower layer, a logarithmic relationship between the number of nodes in the original POMDP and the number of layers in the hierarchy is given:

$$\lambda_{\max} = \log_\nu N. \quad (13)$$

This is a nice scaling property as the number of layers is proportional to the number of POMDP problems that must be solved. Moreover, the reduced POMDP at each layer only deals with ν nodes, which means that it also has an exponentially reduced state and observation space.

Of course, this speedup is achieved through discretization approximations and so in general, all multi-scale methods are susceptible to suboptimal solutions due to discretization inaccuracies. An example is depicted in Fig. 2 where the task involves two items in the three room environment. Both items are within the same layer 0 node n_0^{l0} and both goal-locations are in n_2^{l0}. Layer 0 is unable to differentiate between the two items regarding expected delivery time and randomly chooses either the `pickup1` or `pickup2` action first. Layer 1 is guided through the layer 0 action a^{l0} and value function $V(b^{l0})$. Neither reflects that the pink item's goal location is closer and should therefore be delivered first. Therefore, the multi-scale agents cannot correctly reason which item to deliver first. The flat (single-layer) POMDP delivers the pink item first as the initial positions and goal locations of the two items are in different nodes.

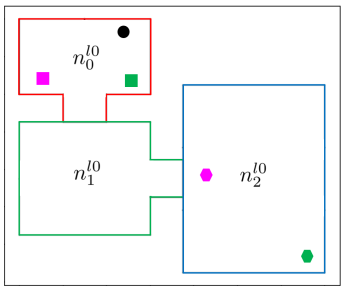


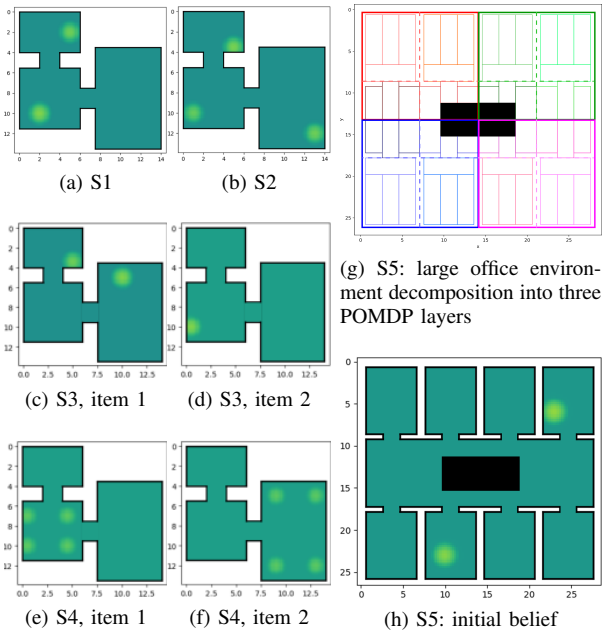
Fig. 2: A scenario where the multi-scale methods are unable to correctly reason over which item should be delivered first. The robot starts at the black circle; colored squares show the current item locations; hexagons show the desired delivery points.

V. RESULTS AND ANALYSIS

We evaluated our three multi-scale POMDP methods in the five test scenarios shown in Fig. 3. The scenarios vary in the number of items (S1, S2 and S5 each have one item to deliver, whereas S3 and S4 have two items), the number of peaks in the initial belief distribution over the item locations, as well as the size of the underlying map. S1–S4 are executed in the three-room office environment introduced earlier and are solved with two POMDP layers, while S5 is conducted in a large eight-room environment with 56 nodes in the bottom-layer POMDP and is solved using three layers (Fig. 3g shows the hierarchical decomposition).

The flat and multi-scale POMDP methods were tested in S1–S4 over 100 statistical runs each. At the start of each run, the true location of the item(s) was sampled from the initial belief distribution. Since the final delivery time is strongly influenced by the initial placement of the item(s), we report the difference in delivery time of each multi-scale POMDP solution as compared to the solution found by the flat POMDP. For all runs, we set the SARSOP solver timeout to 600s. If a timeout occurred, the policy found at the previous timestep was used (which could lead to very suboptimal actions) or a planning failure was recorded if there was no prior policy.

Results are shown in Fig. 4. On average, the multi-scale solutions result in comparable delivery times to the flat POMDP with suboptimality of the solutions demonstrated in several outlier cases. However, as the problem size increases (two items to deliver instead of one) the multi-scale solutions are just as often able to find solutions within the given solve time that lead to faster overall delivery times. When testing in S5, we were unable to achieve a solution from the flat POMDP within the allocated solve time. Thus, we only show results from 30 statistical trials comparing AVP and VP to the AP multi-scale algorithm. In this case, all three algorithms achieve similar mission performance.



(a) S1 (b) S2 (g) S5: large office environment decomposition into three POMDP layers (c) S3, item 1 (d) S3, item 2 (e) S4, item 1 (f) S4, item 2 (h) S5: initial belief

Fig. 3: Initial item belief distributions for our test scenarios S1-S5.

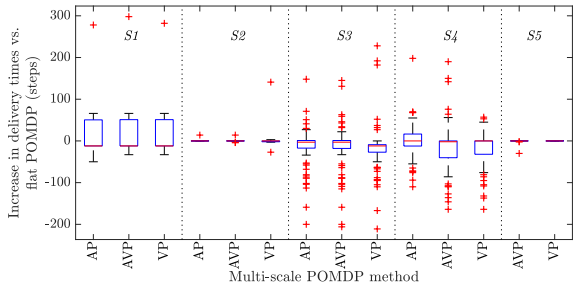


Fig. 4: Increase in mission times compared to solving with the flat POMDP. Note: S5 is compared to the AP multi-scale method since the flat POMDP failed to find a solution within the time budget.

We present the average time required to compute an action by each POMDP framework over all the tested scenarios in Fig. 5. The computational efficiency of the multi-scale algorithms are evident when compared to the flat POMDP. In particular, the flat POMDP scales poorly when the size of the state space increases, either through the introduction of more items to deliver or when the number of nodes increases. In contrast, the multi-scale methods average $\{0.18, 0.21, 0.13\}$s per action in S1 ($|\mathcal{S}| = 99$), which increases modestly to $\{0.65, 0.74, 1.53\}$s in S5 ($|\mathcal{S}| = 3248$).

Finally, in Fig. 6 we highlight a case where information propagation across the layers changes the delivery time performance between the AP, AVP and VP methods. Due to the coarse resolution at layer 0, AP and AVP are not able to correctly reason if traveling via n_1^{l0} or n_2^{l0} is faster for reaching the goal location (in n_3^{l0}). Since VP is not restricted to follow the higher layer actions, it correctly reasons that the path via n_1^{l0} is faster. Thus, demonstrating the benefits of allowing greater decision independence to the lower layers.

VI. CONCLUSIONS

In this work, we presented a multi-scale POMDP framework that allows efficiently solving an object search and delivery task. Three methods of propagating information

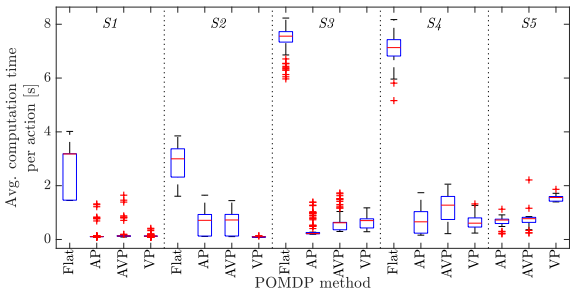
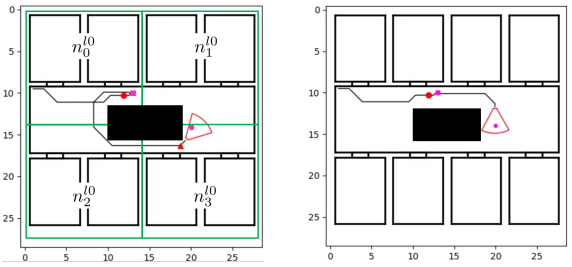


Fig. 5: Average computation time per action.



(a) AP, AVP solution (b) VP solution

Fig. 6: Solution differences as a result of decision-making power.

between layers of different resolutions were shown to produce a similar solution quality compared to solving a flat POMDP. All three methods achieve this at a significantly decreased computational cost, allowing them to tackle larger-scale problems that cannot be solved using the flat POMDP.

Interestingly, the three methods perform very similarly despite varying degrees of reasoning authority at lower levels. The VP method showed a slightly more consistent solution quality improvement especially for the multi-item scenarios. However, this comes at the expense of an increased computational cost compared to the AP and AVP methods.

A limitation of the multi-scale approach is that the node partitioning for each layer needs to be manually designed for each environment. Although not studied in detail here, the resolution and placement of the partitions (especially those at the interface of higher layer nodes) will also impact the solution quality. While the same needs to be done for the flat POMDP to make it tractable, we recognize that each extra layer of the multi-scale approach increases this effort. However, we believe that this can be justified by the subsequent performance improvements.

While we evaluated the proposed method in an object search and delivery task, we believe that it can be extended to most POMDPs where physical space is discretized for a state variable. To make our approach more readily applicable to unseen domains, we would like to alleviate the need to handcraft open-loop policies by automatically constructing closed-loop policies. Automating the partitioning of nodes could be tackled with a similar motivation. Furthermore, to improve the accuracy of the higher-layer models, an idea is to adapt them through a correction mechanism after the lower-layer POMDPs are solved. Finally, we would like to investigate whether our multi-scale approach is beneficial in a model-free reinforcement learning context, sharing Q-values over layers, similar to how the value function is shared in the presented AVP and VP approaches.

REFERENCES

[1] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable Markov processes over a finite horizon," *Operations Research*, vol. 21, no. 5, pp. 1071–1088, 1973.

[2] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[3] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1470–1477.

[4] S.-Y. Lo, S. Zhang, and P. Stone, "PETLON: planning efficiently for task-level-optimal navigation," in *Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems*, 2018, pp. 220–228.

[5] A. Wandzel, Y. Oh, M. Fishman, N. Kumar, L. L. Wong, and S. Tellex, "Multi-object search using object-oriented POMDPs," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 7194–7200.

[6] D. Silver and J. Veness, "Monte-carlo planning in large POMDPs," in *Advances in neural information processing systems*, 2010, pp. 2164–2172.

[7] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "DESPOT: Online POMDP planning with regularization," in *Advances in neural information processing systems*, 2013, pp. 1772–1780.

[8] H. Bai, D. Hsu, W. S. Lee, and V. A. Ngo, "Monte Carlo value iteration for continuous-state POMDPs," in *Algorithmic Foundations of Robotics IX*, 2010, pp. 175–191.

[9] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 3, 2003, pp. 1025–1032.

[10] M. T. Spaan and N. Vlassis, "Perseus: Randomized point-based value iteration for POMDPs," *Journal of Artificial Intelligence Research*, vol. 24, pp. 195–220, 2005.

[11] H. Kurniawati, D. Hsu, and W. S. Lee, "Sarsop: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Robotics: Science and Systems*, vol. 2008, 2008.

[12] N. P. Garg, D. Hsu, and W. S. Lee, "DESPOT-α: Online POMDP planning with large state and observation spaces." in *Robotics: Science and Systems*, 2019.

[13] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[14] S. Omidshafiei, A.-A. Agha-Mohammadi, C. Amato, and J. P. How, "Decentralized control of partially observable Markov decision processes using belief space macro-actions," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5962–5969.

[15] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier, "Hierarchical solution of markov decision processes using macro-actions," in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 1998, pp. 220–229.

[16] J. Pineau and S. Thrun, "An integrated approach to hierarchy and abstraction for POMDPs," Carnegie Mellon University, Tech. Rep., 2002.

[17] A. Foka and P. Trahanias, "Real-time hierarchical POMDPs for autonomous robot navigation," *Robotics and Autonomous Systems*, vol. 55, no. 7, pp. 561–571, 2007.

[18] N. K. Jong and P. Stone, "Hierarchical model-based reinforcement learning: R-max + MAXQ," in *Proceedings of the 25th International Conference on Machine Learning*, 2008, pp. 432–439.

[19] N. Gopalan, M. desJardins, M. L. Littman, J. MacGlashan, S. Squire, S. Tellex, J. Winder, and L. L. Wong, "Planning with abstract Markov decision processes," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2017.

[20] W. Van den Hof, "Robot search in unknown environments using POMDPs," Master's thesis, TU Delft, 2014.

[21] A. R. Mendes, M. T. J. Spaan, and P. U. Lima, "Planning under uncertainty for search and rescue," in *Robótica - 11th International Conference on Mobile Robots and Competitions*, 2011.