Learning Abstracted Models and Hierarchies of Markov Decision Processes

Submission 46 (Authors removed for blind review)

Abstract

In real-world environments, intelligent agents often face challenges with complex goals in large state-action spaces. Solving these tasks is computationally intractable for standard planning and learning methods due to the need to search over extremely long sequences of primitive actions to reach a goal. Hierarchical methods enforce structure over generated policies to simplify these problems, typically by identifying repeated patterns of behavior and decomposing the overall goal into a graph of subtasks. Solutions to smaller tasks are found independently and then combined to complete the original task. Abstract Markov decision processes (AMDPs) support this process by defining an individual Markov decision process for every subtask, each with its own separate model, abstracted relative to the base domain. While this framework is effective at solving large planning problems, it has previously required a domain expert to hand-define the complete hierarchical structure and the transition and reward dynamics for each task. To automate the process of creating these hierarchies, we propose R-AMDP, which integrates task hierarchy construction algorithms with model-based reinforcement learning to learn entire abstract hierarchies solely from an agent's experiential data. The R-AMDP agent shows computational and performance improvements when compared to the baseline, R-MAXQ.

Introduction

Decision-making agents operating in large, stochastic, and feature-rich environments require planning over long action sequences to complete their goals. The state-action spaces of these domains typically grow combinatorially as the number of objects in the state increases. Traditional planning methods scale poorly in such domains, because they must search for a solution trajectory consisting solely of primitive actions. Real-world environments quickly become infeasible, even for agents with considerable resources.

To address this combinatorial explosion, hierarchical methods in reinforcement learning and planning are developed to scale appropriately, making such problems tractable. This aim is achieved through various means, such as by identifying repeated patterns of behavior, as with options (Sutton, Precup, and Singh 1999), or decomposing a complex goal into simpler, reusable subtasks, as with MAXQ (Dietterich 2000). Hierarchies allow an agent to reason over only relevant actions at the appropriate level of abstraction for each task. This approach resembles the way in which humans naturally tackle complex tasks by breaking them down into sequences of simpler, more manageable steps. Hierarchical methods also facilitate knowledge transfer, where experience gained from solving one subtask is retained and reused for future, related tasks. When agents need to complete the same or similar tasks multiple times to accomplish their overall "root" goal, reuse of task solutions is key to not repeatedly solving the same problem. One significant consideration with hierarchical methods, however, is the source of structure. Frequently, human experts hand-design the components of a hierarchy, which include the hierarchical structure, subtask policies, value functions, and models.

A recently developed hierarchical method, planning over abstract Markov decision processes (AMDPs), decomposes a decision-making domain into a task hierarchy: a directed acyclic graph of subtasks (Gopalan et al. 2017). AMDPs go beyond the existing MAXQ formalism by defining each subtask as a separate MDP, complete with a local model (reward function and transition probabilities) and its own state-action space abstracted from the base domain. They provide a considerable increase over the classic options and MAXQ-style approaches in terms of both performance and efficiency. Despite AMDPs' vastly improved planning time, to date, they require a significant amount of expert knowledge in the generation of both the state abstraction and the models used in each AMDP subtask, as well as the specification of the task hierarchy itself.

Our goal is to enable agents to learn components that were previously expert-defined, hypothesizing the hierarchical structure of subtasks and then learning the models for each task, purely from experience. We present a novel formalism, R-AMDP, that builds a task hierarchy and employs separate model-based reinforcement learning processes at every subtask to approximate the local transition and reward functions of AMDPs. We demonstrate empirically that R-AMDP offers considerable gains in performance over existing methods, learning both hierarchies and abstract task models, without the need of expert knowledge to define any subtask relations, state abstractions, transitions, or rewards.

Background

We consider decision-making in uncertain domains, and discuss the model-based and hierarchical methods that provide the backbone of our contribution.

MDPs. Stochastic planning and reinforcement learning are typically described by Markov decision processes (MDPs). Formally, an MDP is a four-tuple $\{\mathcal{S}, \mathcal{A}, \mathrm{T}, \mathrm{R}\}$, with states $\mathcal{S}$, actions $\mathcal{A}$, transition probabilities T, and a reward function R. An object-oriented MDP (OO-MDP) (Diuk, Cohen, and Littman 2008) is a factored-state representation that extends the MDP formalism by representing states as collections of objects, each with a set of instantiated values for predefined attributes. The state of an OO-MDP domain is defined as the union of the attribute values for all objects in the domain. OO-MDPs allow designers to convey transition and reward information through simple conditional structures that express intuitive relationships between the attributes.

R-MAX. Model-based agents are a class of reinforcement learners that approximate models (T and R of an MDP) from experiential data obtained as they explore a domain. These estimated models grant the agent an understanding of how its environment works, allowing it to plan a policy that achieves its intended goal. As the agent gathers more data during exploration, the agent's model converges to the true model, and the resulting policies converge towards optimality. R-MAX (Brafman and Tennenholtz 2002) is one such approach that employs the strategy of optimism-under-uncertainty.

MAXQ. One classic hierarchical method is MAXQ (Dietterich 2000). MAXQ begins by decomposing an MDP into a set of smaller MDPs, one for each subtask, with transitions and rewards derived from the base MDP. These MDPs are organized into a task hierarchy: a directed acyclic graph from a root goal to child (subtask) tasks, down to leaf tasks that use primitive actions. Each subtask is more condensed than the root goal, allowing the agent to make decisions across reduced search spaces, then combine subtask solutions together in a manner that achieves the root goal. By defining such repeated tasks as subtasks of a hierarchy, an agent produces subtask policies that are shared among all of its parent tasks.

An agent leverages the MAXQ hierarchy by learning the value function for each node expressed in terms of its children's value functions. MAXQ uses the value functions to calculate an action value expressed as the value of the child task plus a completion function. MAXQ's completion functions represent the expected discounted reward of the current task i after completing subtask (or action) a while in state s, $C(i, s, a)$. Then, the action value is computed recursively—$Q(i, s, a) = V(a, s) + C(i, s, a)$—in terms of the value of the state for the child a (which in turn is computed using Q at that lower level) plus the completion function of the respective task. MAXQ achieves a recursively optimal solution, meaning that the policy is optimal at each level of the hierarchy under the assumption that its children are optimal. However, plans in MAXQ can be suboptimal from a global perspective, depending the structure of the MAXQ hierarchy. By trading optimality for near-optimal solutions, recursively optimal methods offer significantly faster planning times.

R-MAXQ. R-MAXQ (Jong and Stone 2008) is a unique learning agent that builds partial models of each task in a given task graph. R-MAXQ applies R-MAX to learn the rewards and transitions of primitive actions while the model dynamics of higher-level tasks are found by applying Bellman equations that relate the parent's model to the children's model and recursively to the base actions. After the knowledge of the ground MDP is propagated up the task hierarchy, the agent is able to plan at an abstract level using a Greedy-Q policy. After a bounded amount of exploration, the agent's hierarchical model will converge to a nearly optimal solution.

AMDPs. An abstract Markov decision process (AMDP) hierarchy (Gopalan et al. 2017) is a framework derived from the MAXQ method of decomposing a complex planning problem into a series of actionable subtasks. Given some base MDP to solve, an AMDP hierarchy is represented as a graph similar to MAXQ. The root corresponds to the root goal of the MDP; the leaves represent primitive actions to be executed in the ground MDP; and all other nodes are subtasks that function as the actions of the parent nodes. Compared to MAXQ or R-MAXQ tasks, a key difference is that each AMDP task node is a complete MDP, possessing its own locally (not recursively) defined transition and reward functions. Thus, an AMDP is an MDP where each state is an abstracted representation of the base MDP. Formally, an AMDP is a six-tuple $\{\tilde{\mathcal{S}}, \tilde{\mathcal{A}}, \tilde{\mathrm{T}}, \tilde{\mathrm{R}}, \tilde{\mathcal{E}}, F\}$, consisting of the OO-MDP components with the addition of a state projection function of $F : \mathcal{S} \rightarrow \tilde{\mathcal{S}}$ for mapping states from the base MDP into the abstract state space of the AMDP. $\tilde{\mathrm{T}}, \tilde{\mathrm{R}}, \tilde{\mathcal{E}}$ are unique to the given AMDP, and $\tilde{\mathcal{A}}$ consists of the AMDP's child subtasks, which can be either primitive actions or other AMDPs.

AMDP hierarchies produce optimal policies at each AMDP and, like MAXQ hierarchies, are recursively optimal given correct local state abstraction, reward, and transition functions. AMDP hierarchies enable top-down planning in stochastic environments, such that an agent plans only for subtasks that help achieve its main goal without computing plans for irrelevant subtasks. An AMDP is Markovian with respect to its own state-action space and transitions. The use of state abstractions, however, makes the abstract higher-level problems not necessarily Markovian relative to the base domain (Bai, Srivastava, and Russell 2016). The execution of AMDP plans, thus, functions similarly to options in semi-Markov decision processes (Sutton, Precup, and Singh 1999). One consideration that arises from this property is the handling of failure in subtasks. There is no inherent guarantee that non-goal termination conditions of child AMDPs are reflected in the state space of a parent AMDP. Thus, it is necessary when creating AMDPs that the designer ensures both the projection function and terminal sets are sufficiently expressive to capture such failure cases.

A limitation of AMDPs is that the structure and components (the hierarchy, task nodes, mapping function, task reward functions, and transition probabilities) must be specified by an expert. Moreover, expert-designed hierarchies, either of AMDPs or with MAXQ, yield resulting policies that, while recursively optimal with respect to themselves, can be arbitrarily worse than other possible hierarchies with different subtasks that solve the same root goal.

To address these issues and the broader goal of hierarchical decision-making, we introduce R-AMDP. R-AMDP learns a hierarchies of subtasks with their state abstraction and internal models from experiential data. This approach retains the performance boost afforded by AMDPs while completely alleviating the need for expert-encoded knowledge.

R-AMDP Approach

R-AMDP provides a comprehensive hierarchical planning and learning formalism that learns a MAXQ-like task hierarchy first, statically from solution trajectories, and then builds approximate models for each AMDP subtask in the hierarchy online while solving its specific, global task.

Learning the Hierarchy Structure

To apply MAXQ in domains without an expert-designed hierarchy, HI-MAT (Mehta et al. 2008), and later HierGen (Mehta 2011), learns the structure of a task hierarchy by leveraging the causal relationship between actions and state variables (the attribute values of objects in state space). The input to these methods is one or more solution trajectories, in the form of vectorized state-action-state-reward transition tuples, from an initial state to a terminal, or goal state. HI-MAT and HierGen then annotate each trajectory with the causal effects of actions. A causal edge between two actions, represented by $a \xrightarrow{v} b$, is created when a state variable, v, is relevant to two actions, a and b. In this context, v is relevant to a if a changes v, or v is checked by a in order to change a different variable or calculate the reward of executing a. These causal edges induce a MAXQ hierarchy by tracing the edges backwards from the goal to find useful subtasks. HI-MAT is more restrictive, relying on only a single solution trajectory, while HierGen makes several improvements (learning action models, allowing for multiple input trajectories), resulting in a more general hierarchy that solves a broader range of target tasks than HI-MAT. Trajectories can be created simply by training a standard agent (e.g., Q-learning) on several MDPs drawn from a distribution for the same root task in a particular domain.

HierGen does causal annotation by training a decision tree for each individual action to learn a dynamic Bayesian network model of what state variables change upon executing the action in the environment. With this set of decision trees serving as learned action models, HierGen then analyzes how actions causally affect changes in the state transitions of the sampled trajectories. Causal annotation of a trajectory proceeds by adding links that connect two actions, a and b, over a state variable v, if and only if (1) a changes v, (2) b checks v, and (3) v is not changed by any intervening action. Here, a checks v if v is found in an internal node of the decision tree of the variable that changed when a was executed. Using this information, the algorithm searches for and produces tasks by following the causal edges from the goals in all trajectories, and accumulating the variables relevant to those sequences of edges such that generated tasks generalize across all input trajectories. This produces the subtasks structure learned completely from agent experience, which is adapted into the AMDP hierarchy.

Creating the R-AMDP Hierarchy

AMDP hierarchies are similar to MAXQ hierarchies in terms of the structure of tasks. R-AMDP leverages this similarity by using a MAXQ hierarchy generator to create a template for the AMDP hierarchy. In particular, an AMDP is assembled from the MAXQ task components by creating a state projection function that abstracts away the state variables that are causally irrelevant for that task. The action space for the AMDP consists of its child nodes in the task hierarchy, while the reward and transition model are undefined until learned by the R-AMDP agent through model-based reinforcement learning. A set of terminal states is also defined implicitly for each task from the ending conditions of subtask found by HierGen. By directly defining each component of the MDP tuple, an AMDP is a complete MDP, which can be solved independent of the other MDPs in the hierarchy. The state abstraction generated by HierGen, consisting solely of the relevant variables, guards against the semi-Markovian issue of failure cases in AMDPs. In particular, any actions or variables that could lead to failure cases will be placed at the highest task needed to recover from them. For instance, the learned hierarchy we show in Figure 1 has a primitive action as one of its subtasks because this action, and the variables related to it, could lead to an irrecoverable failure state for that domain. However it is still possible that HierGen may produce abstractions that lead to other issues related to the semi-Markovian nature of AMDPs; we plan to develop and analyze more sophisticated methods in future work. Finally, the generated AMDP hierarchy is capable of planning and learning with R-AMDP to handle the undefined transition and reward functions.

Learning Models with R-AMDP

Learning the reward and transition functions for each AMDP in a hierarchy follows the same process as learning on any ordinary MDP.We select R-MAX as the method of learning the transition and reward models to permit a more meaningful comparison of R-AMDP results to the functionally similar hierarchical model learner, R-MAXQ, which also employs R-MAX.

With R-MAX, when the agent enters an environment for the first time, each action is treated as returning the maximum possible reward, and each state-action-state combination is considered equally likely[1] (normalized to the number of states, a uniform probability distribution over successor

[1]Such highly connected planning problems are intractable in practice. We use a heuristic to select under-sampled actions first.

Algorithm 1 Planning with an R-AMDP hierarchy while learning a model from experience

function R-AMDP-PLAN(H, i)
  $s_i \leftarrow F_i(s)$ ▷ project environment state s
  while $s_i \notin \mathcal{E}_i$ **do** ▷ execute until termination
    $\pi \leftarrow$ PLAN(s_i, i)
    $a \leftarrow \pi(s_i)$
    $j \leftarrow$ LINK(H, i, a) ▷ a links to child task j
    $s'_0 \leftarrow$ R-AMDP-PLAN(H, j)
    $s'_i \leftarrow F_i(s'_0)$
    if TASK-COMPLETE(j,s'_i) **then**
      UPDATE R-MAX MODEL(i, s_i, a, s'_i, r)
    $s_i \leftarrow s'_i$
return s'_0

states):

$$\mathrm{R}(s,a) = r_{\max} \; \forall \; s \in \mathcal{S}, a \in \mathcal{A},$$
$$\mathrm{T}(s'|s,a) = |\mathcal{S}|^{-1} \; \forall \; s, s' \in \mathcal{S}, a \in \mathcal{A}.$$

After each action taken by the agent, the corresponding transition is recorded in the model according to the following update rules: total reward $\mathrm{r} : \mathcal{S} \times \mathcal{A}$ is initialized to 0 and updated by $\mathrm{r}(s,a) \leftarrow \mathrm{r}(s,a) + r$, the state-action-state observations $\mathrm{t} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ are initialized to 0, and incremented each time a given state-action-state transition is observe. The counter of state-action observations $\mathrm{n} : \mathcal{S} \times \mathcal{A}$ is also initialized to 0 and incremented after each observation. Once $\mathrm{n}(s,a)$ reaches a threshold of observations m, the recorded rewards and transitions are used to estimate the reward and transition probability for that state-action pair:

$$\mathrm{R}(s,a) \leftarrow \tfrac{\mathrm{r}(s,a)}{\mathrm{n}(s,a)} \quad \text{and} \quad \mathrm{T}(s'|s,a) \leftarrow \tfrac{\mathrm{t}(s,a,s')}{\mathrm{n}(s,a)}.$$

Once the m threshold is reached, the model is approximated more accurately with each new observation.

As an R-MAX agent plans using its approximate model, actions that have not reached the sample threshold are favored because these transitions are treated as if they return the maximum reward. This method effectively balances the need for exploration with exploiting the known information, where unknown transitions are thought of as promising and preferred until the threshold is reached and the agent switches behavior, exploiting its collected knowledge of the domain. R-MAX is a well studied algorithm, proven to be probably approximately correct in Markov decision processes (PAC-MDP), with guarantees of convergence and bounded space and sample complexity (Strehl, Li, and Littman 2009). Although model learning is effective in reasonably sized domains, in larger domains, the strategy becomes intractable due to the time complexity and space requirements of maintaining tabular representations of the MDP's reward and transition functions. Our results will demonstrate how hierarchical methods and model learning are mutually beneficial in significantly reducing the information that must be calculated and stored at any given point in decision making.

When approximating a MDP model in a hierarchical structure, the model update procedure must be altered slightly to account for the unique qualities of hierarchical modeling. In a flat MDP, the transitions, s, a, s', r, are sampled from the real environment, so they are known to reflect the MDP's true dynamics. In an abstract MDP, the actions are either primitive actions or non-primitive child tasks. When a child task completes, it either completes its goal or terminates in a failure state. These failure states are states in the child's state space in which the subtask is completely unreachable using the child's action set. If a task's child completes its goal in state s' with reward r, sampled from the task's pseudo-reward function defined by R-MAX, then the transition will be included as part of the model of the task. The other possible outcome is that s' is a failure state of the child. If the child was working correctly, it never returns the failed state to the parent, so the R-MAX model ascribes a zero probability to the task transitioning from s to s' when executing a. Therefore, this transition cannot be recorded in the parent's model, because doing so skews the probability distribution of executing subtask a in state s. These inaccuracies in the model produce incorrect plans at the parent node and, when combined, introduce greater error into the final plan.

The optimistic estimation of R-MAX ensures a degree of initial exploration of the state space for each AMDP in the hierarchy. By maintaining approximated models at every abstract task, rather than only at the base MDP, R-AMDP encourages more high-level experimentation and recombination of intermediary subtasks. As the AMDP models begin to converge, the policies created for each task likewise converge to near-optimal solutions. After a bounded exploration phase, our inspection found that R-AMDP produces hierarchical policies that are the same as or better than those found by expert-designed AMDP hierarchies. The R-AMDP hierarchy also maintains the guarantees of planning-time efficient behavior that AMDP hierarchies provide. The learning of models in R-AMDP, thus, takes full advantage of AMDPs' power to solve hard problems, transferring knowledge between similar tasks, while avoiding the need for domain experts to define abstract models.

Methodology

Taxi Domain. The fickle Taxi domain (Dietterich 2000) is a discrete environment with a taxi agent, passengers, depot locations, and impassable walls positioned on a constrained map. The objective is for the taxi to deliver a passenger from a source location to a goal location, each of which could be any of the depots. The taxi can move in any cardinal direction that is not blocked by a wall, pick up a passenger when they share the same position on the map, or drop off a held passenger when the taxi is at a depot. Stochasticity in this domain is generated from both the taxi's slipperiness, where the taxi has a 0.20 probability of moving perpendicular to its intended direction on any navigational action, and the passenger's fickleness, where the passenger has a 0.05 probability of changing its desired destination on any transition when in the taxi.

The taxi domain benefits greatly from hierarchical planning because the root objective is recursively decomposed into disjoint subtasks. (However, the introduction of the

fickle passenger effectively penalizes hierarchical methods because the passenger's goal changes are abstracted away from navigational subtasks. In the worst case, the agent may complete the current navigation task, then return to the parent task only to discover that the initially selected navigation is no longer an optimal decision.) The top hierarchy in Figure 1 shows the expert-defined AMDP hierarchy for Fickle Taxi. This hierarchy is the same as the originally published taxi hierarchy, with one major difference: the inclusion of bringon and dropoff AMDPs.

We evaluate the learning of task hierarchies from experience on the Taxi domain. The bottom hierarchy in Figure 1 shows the learned AMDP hierarchy for Fickle Taxi. Compared to the expert-defined hierarchy, the most notable difference is that learned tasks are able to identify state abstractions that parameterize a task by object attributes. Specifically, the learning process for the Taxi hierarchy found that the x-y coordinates of an object were causal parameters to the actions related to navigation. Thus, the generated hierarchy reuses the same navigation AMDP model both for moving to a passenger and for moving to a goal. This sharing leads to much greater sample efficiency while learning the model, directly transferring behavioral knowledge among object classes. Another benefit from using HierGen when to construct the hierarchy from data is that more concise state abstractions are generated for each subtask, since only the state variables that are causally relevant are preserved.

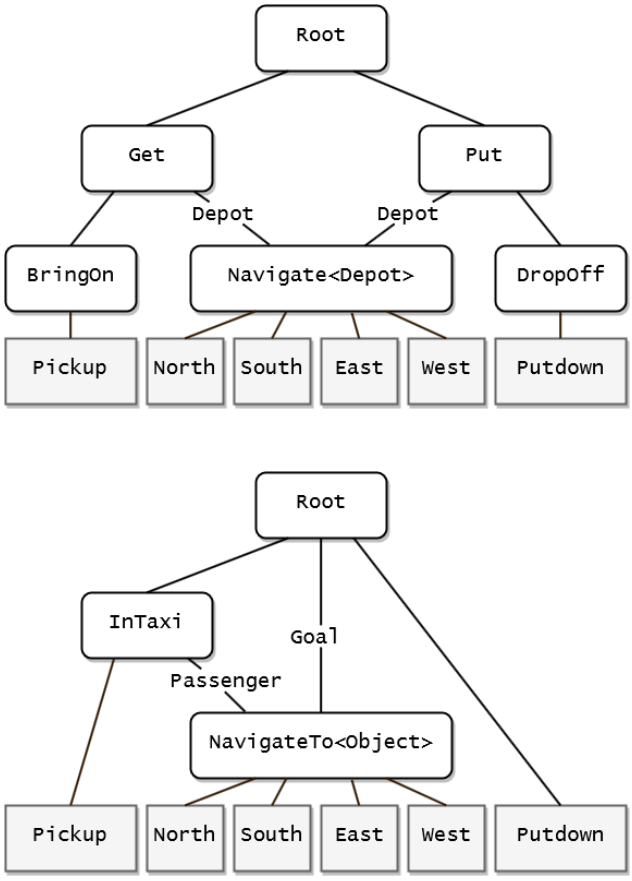

Figure 1: The expert-defined (top) and learned (bottom) AMDP hierarchies for the Taxi domain. Internal nodes are each defined as an AMDP, each with its own model and state abstraction; leaf nodes are primitive actions. Edges denote the subtasks of the parent node that comprise its action set. A labeled edge specifies a parameterized task's object type.

Cleanup Domain. In this domain, an agent must navigate a grid world composed of rooms, doors connecting them, and objects that are maneuvered by being pushed and pulled. Each state has one agent, a set of impenetrable block objects, some number of rectangular rooms that enclose open, traversable space, and some number of doors connecting the rooms. The agent's goal is to put every object into a specific destination room, simulating a robot that needs to tidy a house by putting things away where they belong, similar to the game of Sokoban (MacGlashan et al. 2015). The agent moves in the cardinal directions, pushing blocks forward by moving into them. The agent also has an explicit action for pulling blocks, which the agent uses when immediately facing a block to switch spaces with it (such as to move a block out of a corner, away from a wall).

We use the same hierarchy of AMDPs as described in previous work (Gopalan et al. 2017). The ROOT node of the task graph consists of an abstract MDP where the agent and block objects simply have an attribute for the current region, room or door, that they occupy. ROOT goal states are those in which each block is in its correct destination region. The ROOT possesses two parameterized subtasks: PICKAROOMFORTHEAGENT and PICKAROOMFORTHEBLOCK. For example, in a Cleanup domain with two rooms and one block, there are four "pick" actions, two each for picking a room for the agent and picking a room for a block—that is, one per room per object. Each pick action corresponds to an AMDP that uses the same region-based abstraction as the root AMDP, and has its goal met when the agent or block is in the region specified by the action. The pick AMDPs have child tasks for managing the movement of the agent or block among rooms and doors, with four possible parameterized actions: MOVEAGENTTODOOR, MOVEAGENTTOROOM, MOVEBLOCKTODOOR, and MOVEBLOCKTOROOM. The "move" AMDPs do not use state abstraction, so they are similar to options (Sutton, Precup, and Singh 1999). In solving a move AMDP, a policy is computed for navigating to the destination room or door in the base domain. These are the lowest-level AMDP tasks, with access to the five primitive actions. The task hierarchy for Cleanup differs from Taxi because, rather than cleanly dividing an agent's intentions, as GET and PUT do for Taxi, the subtasks overlap in their effects. Moreover, the Cleanup hierarchy is overcomplete in the sense that there are several ways to reach the same global goal using completely different combinations of subtasks (unlike Taxi, where GET and PUT must always be performed in precisely that order). Thus, we include the Cleanup domain to demonstrate how doing model-based learning over hierarchies (R-MAXQ and R-AMDP) works

Hyperparameter	Taxi	Cleanup
Independent trials per experiment	20	10
Episodes per trial	60	50
Max steps per episode	2000	300
Max Δ threshold (value iteration)	0.01	0.0001
Max planner policy rollouts	1000	1000
R-MAX sample threshold	5	2

Table 1: Values used in experimental evaluation.

well in large domains, even when those hierarchies are wide, offering multiple valid solutions.

Experimental Setup

For both the Fickle Taxi and Cleanup domains, we analyze the performance of an R-AMDP model-learning agent, using R-MAXQ as a baseline for comparison. Each experiment consists of a set of independent trials. Value iteration serves as the planner inside the PLAN function of Algorithm 1. Table 1 shows the details of the experimental setup for both domains.

For Cleanup, the initial configuration consists of two rooms, each 2-by-3 in size, connected in the middle horizontally by a door, with a block in the lower right of the left room and the agent in the upper left of the right room. The goal is to take the block in the left room to the room on the right (the agent must navigate through the door to the left, then push/pull the block back through the door into the right room).

Additionally, we investigate the effectiveness of R-AMDP to learn models with hierarchies learned from data by applying HierGen's structure learning to the Taxi domain, learning action models and causally annotating trajectories to construct a task graph. We generate sample solution trajectories using Q-learning execution traces run on a variety of small, randomly generated Taxi domains. As in prior work (Mehta 2011), our agent learns the action models via decision trees (Weka's J48 algorithm) and then creates a task graph from the causally annotated trajectories. Finally, this HierGen-produced task graph is converted to a hierarchy of AMDPs (with implicit state abstraction, based on the causally relevant features) and compared with another R-AMDP agent using the classic (expert-specified) Taxi task hierarchy.

Results and Discussion

Model Learning

We compare the performance of R-AMDP to R-MAXQ using a hand-crafted hierarchy on a fickle taxi domain with stochastic movement actions (Figure 2). The domain we consider is a smaller version of Taxi: a 5-by-1 grid world containing four depots (red, blue, green, and yellow), the taxi (agent), and a passenger starting at the blue depot who desires to go to the red depot. While both agents successfully explore and learn the environment at each node of the task hierarchy, R-AMDP converges to the optimal policy much sooner in the trial, with greater consistency across all

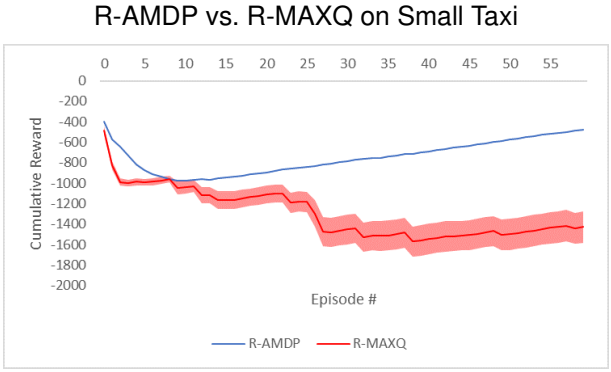

Figure 2: Cumulative reward for R-AMDP and R-MAXQ on the Taxi domain, each using the expert-made hierarchy. R-MAXQ must explore possible subtasks, repeatedly executing long sequences of primitive actions that do not help its overall policy. R-AMDP offer better high-level exploration, effectively pruning unhelpful subtrees of its hierarchy.

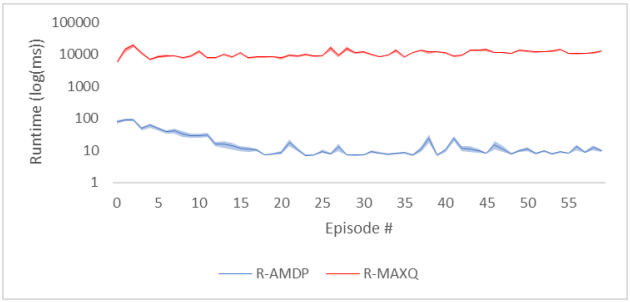

Figure 3: Runtime comparison of R-AMDP and R-MAXQ. R-AMDP agents are significantly faster in execution, completing trials in seconds that takes R-MAXQ an hour or more.

episodes. R-MAXQ sometimes fails to converge to a policy that consistently find a solution, due to the stochastic nature of a fickle passenger. Further, R-MAXQ experiences episodes where the agent's lack of a complete model results in random exploration of unknown states, causing decreases in reward in later episodes. R-AMDP converges to a full model of each task, leading to consistent results after convergence to a near-optimal policy.

Another notable benefit to R-AMDP in comparison with R-MAXQ is the difference in the algorithms' runtime. Figure 3 shows the time (in milliseconds, on a logarithmic scale) that each episode took to run the same set of trials.[2] Overall, R-MAXQ takes several orders of magnitude more computation time than R-AMDP to learn a model and policy for the same domain. R-AMDP improves the runtime as the policy converges towards near-optimality, while R-MAXQ

[2]Runtime analysis is performed on an Intel Core i7-4790K CPU @ 4.00 GHz with 20GB of RAM.

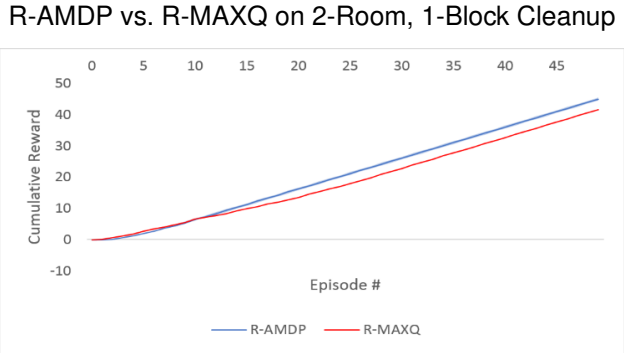

Figure 4: Cumulative reward for R-AMDP and R-MAXQ on the Cleanup domain. The confidence regions around the graphed lines are visualized but small, indicating a statistically significant difference.

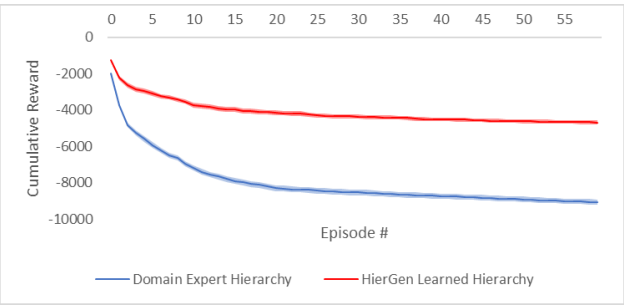

Figure 5: R-AMDP with expert-defined and a HierGen-learned Hierarchy on standard Taxi Domain.

continues to exhibit poor performance.

Results for the Cleanup domain are presented in Figure 4. Both algorithms converge to an optimal hierarchical policy, but the cumulative reward is a statistically significant difference, with R-AMDP outperforming R-MAXQ. Additionally, R-AMDP is able to scale to larger Cleanup domains, in terms of computational runtime, whereas R-MAXQ does not. While we found a similar cumulative reward curve when R-AMDP agents were run on 3-room-2-block domains and more complex domains, R-MAXQ was unable to complete trials of these types of domains in a reasonable amount of time. This is due to the exponential growth of sampling relative to the number of terminal states and height of the hierarchy.

Expert-Defined vs. Learned Hierarchies

The hierarchy generation technique was tested by comparing two R-AMDP agents in the taxi domain, one on the expert-designed task hierarchy and the other on the hierarchy created by HierGen. Both agents were tested on the 5-by-5 taxi state containing four depots and one passenger with stochastic movement and fickle goal location probabilities. The learned hierarchy outperforms the expert-designed hierarchy. While both agents converge to similar optimal policies (Figure 5), the expert-designed hierarchy requires more exploration because its hierarchy contains coarser task parameterizations.

Advantages of R-AMDP

The advantages of R-AMDP are made evident by the results on the Taxi domain. R-AMDP requires fewer Bellman updates for each task node in the hierarchy, since R-MAXQ must compute a model over all possible future states in a planning envelope after each action. Therefore, R-AMDP is much more scalable to large domains than R-MAXQ. In exploring larger domains with more depots, we found that R-AMDP could rapidly solve domains that R-MAXQ was unable to complete in a practical time frame. As the state-action space from R-MAXQ expands with non-zero transition values, more computation is required to recursively compute a model used in planning at each task for all states in the enlarged planning envelope. Further, the update of the model in R-MAXQ occurs by recursively computing models for the children of each composite action, causing the reachability computation to be exponential to the height of the hierarchy. Since each task node in an AMDP hierarchy is an independent learning problem, the execution of R-MAX happens at each task node, and increases linearly with the size of the hierarchy.

In general, the higher branching factor at the abstract levels of the hierarchy used in Cleanup causes the R-AMDP agent to explore more initially, relative to R-MAXQ, before it finds a good solution. However, by episode 10, on average, R-AMDP is able to more consistently exploit its abstract models. The self-contained nature of the abstract "pick" models means that the R-AMDP agent determines which "move" subtasks are not relevant to the task at hand sooner. For example, suppose the agent is performing PICKROOMFORBLOCK(ROOM2, BLOCK1) (which solves the root goal), and consider it to be in the state after moving from the original room to the room with the block. Once the R-AMDP agent determines that moving back to the door or the original room is not helpful to solving the goal at that abstract level, it will cease the needless explorations down those branches of subtasks, essentially (and correctly) pruning them. R-MAXQ, on the other hand, computes its abstract models recursively up from the subtasks, continually re-entering the original room to explore as many possible configurations of base states it finds. A central benefit of AMDPs relative to MAXQ is the ability to expand only those subtasks needed in the rollout of a hierarchical policy. The key insight demonstrated here is that the R-AMDP formalism not only preserves this property, but propagates this efficient handling of the exploration/exploitation up to higher abstraction levels, allowing it to scale to larger domains. The strength of R-AMDP to deal with subtask branching allows it to solve the 3-room, 2-block Cleanup domains (and larger ones), whereas R-MAXQ agents do not reach a conclusion in reasonable runtime. R-AMDP agents achieve greater focus in their construction of abstract models.

The experiment with the learned hierarchy shows the benefit of a learned AMDP structure over an expert-designed

one. While our expert hierarchy gives all possible parameterizations of the child tasks, the learned structure only includes subtasks that have been observed to be beneficial to the current task's goal. That is, while both hierarchies contain tasks with their unknown rewards and transitions approximated via exploration, the tasks learned via HierGen are parameterizations that offer greater reuse by parent tasks. This property means some execution paths down the hierarchy are shorter and more efficient in how tasks are used in specific contexts. For example, the generated hierarchy contains a NAVIGATE task node parameterized by coordinates, rather than a specific depot. The hierarchical policies for solving the PUT task are done more directly, first by navigating to the coordinates of the target depot, then recursing back to ROOT by dropping off the passenger and terminating. Because it had a reduced number of tasks available at any one time, this agent has to learn fewer R-MAX models compared to the expert hierarchy, resulting in less unnecessary exploration at all nodes in the AMDP hierarchy and more cumulative reward collected in each trial. Essentially, it is a condensed hierarchy of minimal task nodes, organized to minimally represent the target task.

Future Work

R-AMDP extends the AMDP planning framework by removing the need to hand-design the models of AMDPs, even in hierarchies derived from task graphs learned from data. The state spaces are defined by MAXQ feature relevance as subsets of the original state space. AMDPs are more powerful than a MAXQ hierarchy because AMDPs in a hierarchy are independent of the other AMDPs in the structure. Therefore, the state spaces can be defined by a more advanced state abstraction technique than MAXQ irrelevance. The hierarchy learning process exploits this property by creating state spaces with unique attributes that achieve better performance.

HierGen creates a single hierarchical structure from multiple training trajectories. If there is an inaccuracy in the final structure, the R-AMDP agent has no way to improve the hierarchy. One solution to avoid this rigidity is to pick the best candidate structure from a collection of hierarchies proposed by HierGen. By scoring each hierarchy using a set of metrics related to performance and computational resources, all hierarchies could be ordered and the best structure become the AMDP. Additionally, when constructing a hierarchy, HierGen does not differentiate between causative attributes and correlated attributes. One could view this issue as analogous to the development of superstitions in humans: if one always performs better when wearing a "lucky" shirt, one might attribute the performance to the wearing of the shirt and not to the factors that directly cause the increase in performance.

We selected R-MAX for this research to more fairly compare performance with the existing baseline, R-MAXQ; we intend to explore alternate algorithms.

Conclusion

R-AMDP demonstrates the viability of a model-based approach on the AMDP framework for hierarchical planning, where totally abstract models are learned from experience gained and propagated across the abstract level itself. In particular, R-AMDP allows an agent to learn the models that best suit each task in a hierarchy of decision problems learned from data, without requiring any human engineering of reward structure or expert-defined transition dynamics, all while maintaining the benefits of planning with AMDP hierarchies. We have deployed an existing structure-learning method, HierGen, to autonomously produce a task graph that, when converted to an AMDP hierarchy, yields a more compact representation with higher performance. In combination with R-AMDP, this work introduces a complete top-down, stochastic, hierarchical planning framework in which the models are learned bottom-up from data.

References

Bai, A.; Srivastava, S.; and Russell, S. 2016. Markovian state and action abstractions for mdps via hierarchical mcts. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 3029–3037. AAAI Press.

Brafman, R. I., and Tennenholtz, M. 2002. R-MAX: A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3(Oct):213–231.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Diuk, C.; Cohen, A.; and Littman, M. L. 2008. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, 240–247.

Gopalan, N.; desJardins, M.; Littman, M. L.; MacGlashan, J.; Squire, S.; Tellex, S.; Winder, J.; and Wong, L. L. 2017. Planning with abstract Markov decision processes. In *International Conference on Automated Planning and Scheduling*.

Jong, N. K., and Stone, P. 2008. Hierarchical model-based reinforcement learning: R-MAX+ MAXQ. 432–439.

MacGlashan, J.; Babeş-Vroman, M.; desJardins, M.; Littman, M. L.; Muresan, S.; Squire, S.; Tellex, S.; Arumugam, D.; and Yang, L. 2015. Grounding English commands to reward functions. In *Robotics: Science and Systems*.

Mehta, N.; Ray, S.; Tadepalli, P.; and Dietterich, T. 2008. Automatic discovery and transfer of MAXQ hierarchies. In *Proceedings of the 25th International Conference on Machine Learning*, 648–655.

Mehta, N. 2011. *Hierarchical structure discovery and transfer in sequential decision problems*. Ph.D. Dissertation, Oregon State University.

Strehl, A. L.; Li, L.; and Littman, M. L. 2009. Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research* 10(Nov):2413–2444.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1-2):181–211.