# Planning with Abstract Markov Decision Processes

**Nakul Gopalan**[*] ngopalan@cs.brown.edu
**Marie desJardins**[†] mariedj@umbc.edu
**Michael L. Littman**[*] mlittman@cs.brown.edu
**James MacGlashan**[‡] james@cogitai.com
**Shawn Squire**[†] ssquire1@umbc.edu
**Stefanie Tellex**[*] stefie10@cs.brown.edu
**John Winder**[†] jwinder1@umbc.edu
**Lawson L.S. Wong**[*] lsw@brown.edu

[*] Brown University, Providence, RI 02912
[†] University of Maryland, Baltimore County, Baltimore, MD 21250
[‡] Cogitai, Inc.

**Abstract**

Robots acting in human-scale environments must plan under uncertainty in large state–action spaces and face constantly changing reward functions as requirements and goals change. Planning under uncertainty in large state–action spaces requires hierarchical abstraction for efficient computation. We introduce a new hierarchical planning framework called Abstract Markov Decision Processes (AMDPs) that can plan in a fraction of the time needed for complex decision making in ordinary MDPs. AMDPs provide abstract states, actions, and transition dynamics in multiple layers above a base-level "flat" MDP. AMDPs decompose problems into a series of subtasks with both local reward and local transition functions used to create policies for subtasks. The resulting hierarchical planning method is independently optimal at each level of abstraction, and is recursively optimal when the local reward and transition functions are correct. We present empirical results showing significantly improved planning speed, while maintaining solution quality, in the Taxi domain and in a mobile-manipulation robotics problem. Furthermore, our approach allows specification of a decision-making model for a mobile-manipulation problem on a Turtlebot, spanning from low-level control actions operating on continuous variables all the way up through high-level object manipulation tasks.

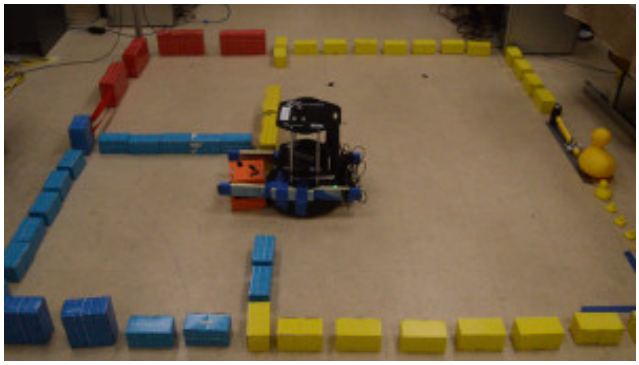


Figure 1: The continuous object manipulation domain, where the Turtlebot agent needs to fetch a block into a goal room. The task is solved online with an AMDP hierarchy that provides low-level actions to solve the abstractly defined task to "push the block into the blue room". The AMDP planner shows reactive control when moving the blocks to deal with stochasticity of the environment and the robot's controllers.

## 1 Introduction

When carrying out tasks in unstructured, multifaceted environments such as factory floors or kitchens, the resulting planning problems are extremely challenging due to the large state and action spaces (Bollini et al. 2012; Knepper et al. 2013). The state–action space in these domains grows combinatorially with the number of objects. Typical planning methods require the agent to explore the state–action space at its lowest level, resulting in a search for long sequences of actions in a combinatorially large state space. For example, cleaning a room requires arranging objects in their respective places. A naive approach for arranging object might have to search over all possible states by placing all objects in all possible locations, resulting in an intractable inference problem with increasing objects.

One promising approach is to decompose planning problems in such domains into a network of independent subgoals. This approach is appealing because the decision-making problem for each subgoal is typically much simpler than the original problem. There are two ways in which the decision problem can be simplified. First, instead of selecting between actions, the agent can select between subgoals that are recursively solved, decreasing the search depth. Second, the state representation of the world can be compressed to include only information that is relevant to the current decision problem. Importantly, planning algorithms for each subproblem can be custom-tailored, allowing each goal to be solved as efficiently as possible.

We propose *Abstract Markov Decision Process* (AMDP)

hierarchies as a method for reasoning about a network of subgoals. AMDPs offer a model-based hierarchical representation that encapsulates knowledge about abstract tasks at each level of the hierarchy, enabling much faster, more flexible top-down planning than previous methods (see Section 2). An AMDP is an MDP whose states are abstract representations of the states of an underlying *environment* (the ground MDP). The actions of the AMDP are either primitive actions from the environment MDP or subgoals to be solved. An AMDP hierarchy is an acyclic graph in which each node is a primitive action or an AMDP that solves a subgoal defined by its parent. The main advantage of such a hierarchy is that *only* subgoals that help achieve the main task need to be planned for; crucially, plans for irrelevant subgoals are never computed. Another desirable property of AMDPs is that agents can plan in stochastic environments, since each subgoal's decision problem is represented by an MDP. Moreover, each subgoal can be independently solved by any off-the-shelf MDP planner best suited for solving that subgoal. Finally, if each AMDP's transition dynamics accurately models the subgoal outcomes, then an optimal solution for each AMDP produces a recursively optimal solution for the whole problem; if the transition dynamics are not accurate, then the error associated with the overall solution can still be bounded (Section 3.1).

For example, consider the Taxi problem (Dietterich 2000) shown in Figure 3a and its AMDP hierarchy in Figure 2. The objective of the task is to deliver the passenger to their goal location out of four locations on the map. The subgoal of Get Passenger, which picks up the passenger from a source location, is represented by an MDP, with lower-level navigation subgoals, Nav(R), and a passenger-pickup subgoal, Pickup. The state space to solve the Get Passenger subgoal need not include certain aspects of the environment such as the Cartesian coordinates of the taxi and passenger. To solve this small MDP when picking up a passenger at the Red location, it is unnecessary to solve for the subpolicy to navigate to the Blue location. Our hierarchy enables a decision about which subgoal to solve without needing to solve the entire environment MDP. Moreover, since the tasks are abstractly defined (for example, "take passenger to blue location"), changing the task description from the "blue" to the "red" location is straightforward, and users do not have to directly manipulate the reward functions at each level of the hierarchy. This abstraction is useful in robotics, as human users can simply change the top-level task description and the required behavior will be achieved by the hierarchy.

In the next section, we discuss the relationship of our work to previous hierarchical planning methods. We then formally describe an AMDP hierarchy and planning over AMDPs (Section 3). Our results (Section 4) show that AMDPs use significantly less planning time than base-level planners and MAXQ to complete tasks in domains with large state spaces. In smaller domains, such as the Taxi problem, AMDPs do as well as a fast base planner like Bounded Real Time Dynamic Programming (BRTDP) (McMahan, Likhachev, and Gordon 2005); however, in large domains such as Cleanup World (MacGlashan et al. 2015)—which has multiple manipulable objects that require long plans—AMDPs can produce plans orders of magnitude faster than all other planners we have examined. We also implemented a continuous mobile manipulation domain on a Turtlebot, as shown in Figure 1, which is solved using an AMDP hierarchy that spans all the way from low-level control actions to very high-level abstract goals, with exceptionally efficient planning. Our approach enables the Turtlebot to choose actions at the lowest level of the hierarchy at 20Hz. Further, if the environment changes (for example, if a target object moves), the robot instantly replans and completes the task. AMDPs are the first hierarchical method to allow real-time planning on abstractly specified object manipulation tasks in stochastic environments with continuous variables.

## 2 Related Work

Subgoals and abstraction provide mechanisms for allowing an agent to efficiently search large state/action spaces. One of the earliest reinforcement learning (RL) methods using the ideas of subgoals and abstraction is the MAXQ model (Dietterich 2000). MAXQ decomposes a "flat" MDP into smaller subtasks, each accompanying a subgoal and potentially a state abstraction that is specific to the subtask's goal. A subtask is a sequence of actions that is used to complete a subgoal. MAXQ constrains the choice of subtasks that can be chosen in its hierarchy, depending on the context or parent task. MAXQ can also use relevant state abstractions for each context, which speeds up learning. MAXQ hierarchies were originally designed for use in model-free RL. However, they have been used in planning settings (Diuk, Littman, and Strehl 2006). An AMDP has the form of a non-primitive subtask of a MAXQ hierarchy, but also includes an "abstract" transition and reward function defined over the subtask's children. AMDPs are therefore model-based, in contrast to MAXQ. In AMDPs, these abstract transition and reward functions enable planning purely at the abstract level, without needing to further descend to lower-level subtasks or primitive actions until execution. In this work, we focus on the planning aspects of AMDP hierarchies, and assume these abstract transition and reward functions are given; future work will explore learning these models.

A major limitation of MAXQ is that value functions over the hierarchy are found by processing the state–action space at the lowest level and backing up values to the abstract subtask nodes. This *bottom-up* process requires full expansion of the state–action space, resulting in large amounts of computation. In contrast, Figure 2 shows the Taxi task hierarchy, with shaded cells indicating the nodes that get expanded or solved by an AMDP planner from the initial state in the environment. AMDPs model each subtask's transition and reward functions locally, resulting in faster planning, since backup across multiple levels of the hierarchy is unnecessary. This *top-down* planning approach decides what a good subgoal is before planning to achieve it. In this work, because we want to compare planning speed across methods, we have provided the transition and reward models for subtasks to the AMDP planner. However, we are currently developing methods to learn these models from scratch.

The concept of using a subtask model for computing a policy has also appeared in previous work, such as R-

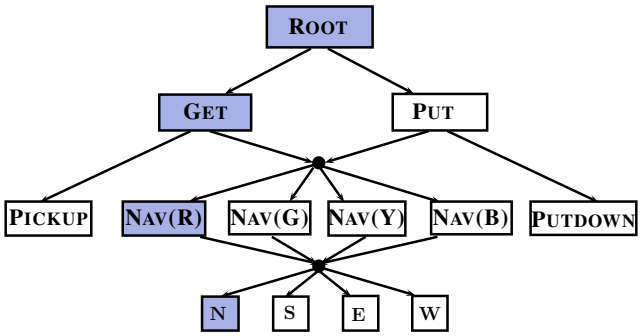



Figure 2: The Taxi AMDP hierarchy. Nodes indicate subgoals which are solved using an AMDP or a primitive action. The edges are actions belonging to the parent AMDP. Shaded nodes indicate which subgoals are expanded by AMDPs in a given state. In contrast, bottom-up approaches like MAXQ (Dietterich 2000) expand all nodes in the figure. These savings result in significant total planning computation gains: AMDP planning requires only 3% of the number of backups that MAXQ requires for the Taxi problem.

MAXQ (Jong and Stone 2008), a model-based learning algorithm for MAXQ. However, in R-MAXQ, the transition and reward functions violate our model-abstraction requirement: their local models are not self-contained. Instead, the effects of a subtask's child are computed recursively, requiring planning for each child and descendant for each state visited in the subtask. Learning transition functions in R-MAXQ requires recursion all the way to the base level. AMDPs, in contrast, open the door to new learning algorithms that operate separately at each level, potentially requiring much less data and computation to learn a model. Moreover, within most of these methods, there are no means to use specialized planners or heuristics specific to a subtask.

Temporally extended actions (McGovern, Sutton, and Fagg 1997) and options (Sutton, Precup, and Singh 1999) are other commonly used bottom-up planning approaches, which encapsulate reusable segments of plans into a more tractable form, and are another way to represent subgoals. Generally, options are used as pre-computed policies for planning; once available, options can speed up planning in large domains. However, these pre-computed policies may themselves be hard to find. When policies for options are planned for with the entire task, they are as slow as other bottom-up approaches. Moreover, although temporally extended actions can decrease plan length and complexity, their inclusion also increases the branching factor of search, and can lead to dramatic increases in planning time (Jong 2008). Recently, Konidaris (2016) discussed creating MDPs at multiple levels of the hierarchy with options, which were used as precomputed policies. This hierarchy plans for a task completely at one of the levels of the hierarchy, instead of planning only for the subgoals that are needed across the hierarchy. Construction of these sets of options at multiple levels requires searching through the entire hierarchy, bottom-up, which is slower than an AMDP-based approach.

Top-down approaches have been used previously for planning and have shown improvements in planning speeds when compared to bottom-up methods. Hierarchical Dynamic Programming (HDP) (Bakker, Zivkovic, and Krose 2005) is a top-down approach that was used for planning in navigation problems. However, the HDP work is specific to navigation domains, and relies on value iteration (Bellman 1956) for each subtask, whereas AMDPs are more general because they allow any suitable planner to be used for each subgoal. For example, for a navigation subtask, A* (Hart, Nilsson, and Raphael 1968) might be an appropriate method, but for a subtask of picking a particular object in a room full of different objects, BRTDP (McMahan, Likhachev, and Gordon 2005) might be a better choice. Furthermore, the state aggregations formed by HDP are based on the adjacency of physical locations, which do not transfer to mobile manipulation completely, since the abstraction is not based on completion of subgoals such as picking up passengers or objects. Nevertheless, the previous application of HDP to robotic navigation provides strong evidence that the AMDP approach generalizes to very large robotics-relevant problems; to support this claim, we demonstrate the use of AMDPs in a full-stack mobile-robot controller.

Angelic hierarchical planning (Marthi, Russell, and Wolfe 2008) also uses a top-down approach for planning. However, its use is limited to deterministic domains, and it does not offer the modularity of using different planning algorithms at each node. DetH* (Barry, Kaelbling, and Lozano-Pérez 2011) is another top-down hierarchical approach that learns a two-level hierarchy by breaking the state space into macro-states and planning deterministically over the macro-states. Domains like Taxi have stochasticity in the higher levels of abstraction, so DetH* cannot be directly used. Further, DetH* does not allow planning in MDPs with a continuous state space. AMDPs are a generalized top-down planner, which can solve in real time a variety of stochastic domains with combinatorial and/or continuous state variables, and can completely plan trajectories for a robot to perform mobile manipulation.

## 3 Abstract Markov Decision Processes

We consider the problem of an agent interacting with an environment that is modeled as an MDP (Bellman 1957). An MDP is defined by a five-tuple ($\mathcal{S}$,$\mathcal{A}$, $\mathcal{T}$, $\mathcal{R}$, $\mathcal{E}$), where $\mathcal{S}$ is the environment state space; $\mathcal{A}$ is the agent's action space; $\mathcal{T}(s, a, s')$ is a function defining the transition dynamics (i.e., the probability that a transition to state $s'$ will occur after taking action $a$ in state $s$); $\mathcal{R}(s, a, s')$ is the reward function, which returns the reward that the agent receives for transitioning to state $s'$ after taking action $a$ in state $s$; and $\mathcal{E} \subset \mathcal{S}$ is a set of terminal states that, once reached, prevent any future action. The goal of planning in an MDP is to find a *policy*—a mapping from states to actions—that maximizes the expected future discounted reward.

Solving an MDP can be quite challenging, so we introduce the concept of an *Abstract MDP* (AMDP) to simplify this process. An AMDP is an MDP in its own right, but captures higher-level transition dynamics that serve as an abstraction of the lower-level environment MDP with which the agent is interacting. Formally, we define an AMDP as

a six-tuple ($\tilde{\mathcal{S}}$, $\tilde{\mathcal{A}}$, $\tilde{\mathcal{T}}$, $\tilde{\mathcal{R}}$, $\tilde{\mathcal{E}}$, $F$). These are the usual MDP components, with the addition of $F : \mathcal{S} \rightarrow \tilde{\mathcal{S}}$, a *state projection* function that maps states from the environment MDP into the AMDP state space $\tilde{\mathcal{S}}$. Additionally, the actions ($\tilde{\mathcal{A}}$) of the AMDP are either primitive actions of the environment MDP, or are associated with subgoals to solve in the environment MDP. The transition function of the AMDP ($\tilde{\mathcal{T}}$) must capture the expected changes in the AMDP state space upon completion of these subgoals. With these action and state semantics, an AMDP, in effect, defines a decision problem over subgoals for the environment MDP.

Naturally, each subgoal for a task must be solved. However, even a single subgoal might be challenging to solve in the environment MDP. Therefore, we introduce the concept of an AMDP hierarchy $H = (V, E)$, which is a directed acyclic graph (DAG) with labeled edges. The vertices of the hierarchy $V$ consist of a set of AMDPs $\mathcal{M}$ and the set of the primitive actions $\mathcal{A}$ of the environment MDP. The edges in the hierarchy link multiple AMDPs together, with the edge label associating the action of an AMDP with either a primitive environment action or a subgoal that is formulated as an AMDP itself. Consequently, an AMDP hierarchy recursively breaks down a problem into a series of small subgoals.

For example, consider the Taxi problem, where the objective is to ferry a passenger to their goal location with AMDP hierarchy shown in Figure 2. The AMDP for the Get Passenger subgoal has access to the location-parameterized abstract action Nav(i) and the primitive action Pickup. The abstract states and projection function in this AMDP remove the Cartesian coordinates of the taxi and passenger, replacing both with a discrete value corresponding to one of the colored destinations. The termination states consist of any states in which the passenger is in the taxi. The reward function is simply defined to return 1 when the passenger is picked up. The navigate to Red subgoal, Nav(R), which is referenced by Get Passenger, is itself another AMDP. In this case, its actions consist only of the primitive environment actions, North, South, East, and West. Its states and state projection function exclude variables related to the passenger and its terminal states are defined to be states in which the taxi is at the Red location. The reward function returns 1 when the Red location is reached, 0 otherwise. The AMDP definition for the other Nav subgoals is similar.

Although there are many possible ways to define state abstractions, a useful heuristic is subgoal-based state abstraction. Subgoals specifically allow the condition of Result Distribution Irrelevance (Dietterich 2000), in which the state space collapses into a few states at the end of a temporally extended action, behaving as a "funneling action," allowing a concise representation of the abstract state space.

When a parent AMDP invokes a lower-level node, it expects that particular subgoal to be achievable; there is currently no mechanism for backtracking. This is potentially problematic if the lower-level AMDP's failure conditions are not expressible in the higher-level state abstraction, since the parent would then have no means to reason about the failure. We currently avoid this problem by ensuring that termination sets include all failure conditions, and designing higher-level state spaces that are capable of representing such failures. However, if the abstractions are themselves imperfect (e.g., learned), then we would likely need to address the problem of failure recovery.

**Algorithm 1** Online Hierarchical AMDP Planning

```
function SOLVE(H)
    GROUND(H, ROOT(H))
function GROUND(H, i)
    if i is primitive then                          ▷ recursive base case
        EXECUTE(i)
    else
        s_i ← F_i(s)                    ▷ project the environment state s
        π ← PLAN(s_i, i)
        while s_i ∉ E_i do              ▷ execute until local termination
            a ← π(s_i)
            j ← LINK(H, i, a)                       ▷ a links to node j
            GROUND(H, j)
            s_i ← F_i(s)
```

## 3.1 Planning in AMDPs

In this section, we describe how to plan with a hierarchy $H$ of AMDPs. The critical property of our planning approach is to make decisions online in a top-down fashion by exploiting the transition and reward function defined for each AMDP. In this top-down methodology, planning is performed by first computing a policy for the root AMDP for the current projected environment state, and then recursively computing the policy for the subgoals the root policy selects. Consequently, the agent never has to determine how to solve subgoals that are not useful subgoals to satisfy, resulting in significant performance gains compared to bottom-up solution methods. This top-down approach does require that the transition model and reward function for each AMDP are available. In this work, we assume they are given by a designer to demonstrate the power afforded by this top-down approach. However, in the future, we plan to investigate how to learn these abstract transition models and reward functions by using model-based reinforcement learning approaches. Our work is analogous to hierarchical task network (HTN) planners, which use designer-provided background knowledge to guide the planning process, but can also be learned from observation (Nejati, Langley, and Konik 2006)

Pseudocode for online hierarchical AMDP planning is shown in Algorithm 1. Planning begins by calling the recursive *ground* function from the root of $H$. If node $i$ passed to the ground function is a primitive action in the environment MDP, then it is executed in the environment. Otherwise, the node is an AMDP that requires solving. Before solving it, the current environment state $s$ is first projected into AMDP $i$'s state space with AMDP $i$'s projection function $F_i$. Next, any off-the-shelf MDP planning algorithm associated with AMDP $i$ is used to compute a policy. The policy is then followed until a terminal state of the AMDP is reached. Following actions selected by the policy for AMDP $i$ involves finding the node the actions links to in hierarchy $H$, and then calling the ground function on that node. Note that after

the ground function returns, at least one primitive action in the environment should have been executed. Therefore, after ground is called, the current state for the AMDP is updated by projecting the current state of the environment with $F_i$.

AMDP planning can be substantially faster than general MDP planning for two reasons. First, the hierarchical structure allows high-level AMDP actions to specify subgoals for the lower-level MDPs, dramatically decreasing search depth. Second, each subgoal can take advantage of strong heuristics or planning knowledge that is not easily incorporated into a planning algorithm solving the global problem. For example, in the Taxi problem, the navigation subgoals enable A* to be used with a strong heuristic, such as Euclidean distance from the goal, to help complete the subtasks. In contrast, bottom-up hierarchical algorithms, including MAXQ, do not afford the ability to use custom planning algorithms for each subgoal and require expanding subgoals that will ultimately prove unhelpful to solving the task.

Since hierarchical planners constrain which decisions can be made, optimal solutions to them do not necessarily maximize the expected future reward for the underlying environment MDP. Dietterich (2000) distinguishes between optimality for the environment and several notions of optimality that are constrained by the hierarchy. A *hierarchically optimal* policy is one that achieves the maximum reward at the base level, subject to the constraint that actions are consistent with the given hierarchical structure. In contrast, *recursive optimality* is a local concept in which the policy is optimal at each level of the hierarchy; that is, the policy at each node is optimal given the policies of its children. A MAXQ policy is recursively optimal (Dietterich 2000).

Actions in an AMDP are chosen to be optimal with respect to its level of abstraction, given any goal conditions from the level above. The notion of optimality within each level of abstraction is weaker than recursive optimality, since the abstract MDPs do not query the transition and reward functions of the flat MDP to learn a correct semi-MDP (SMDP) (Sutton, Precup, and Singh 1999) model of the task while planning. However, with appropriate design a planner optimal at each abstract level can be as effective as a recursively optimal planner. An AMDP hierarchy without state abstraction is equivalent to solving the base-level MDP with temporally extended actions in the form of subtasks. Without state abstraction, if the transition and reward functions for each AMDP respect the true multi-timestep transitions for completing those subgoals in the environment, and each AMDP is optimally solved, then the solutions of the hierarchical MDP planner will be recursively optimal by definition (Dietterich 2000). Further, Brunskill and Li (2014), in Lemma 5, prove that bounded error in an SMDP transition and reward function leads to bounded error in its value function and Q function. That is:

$$|Q_1^*(s,a) - Q_2^*(s,a)| \leq \max_{s,a} \epsilon_{s,a} \ , \tag{1}$$

where $Q_1^*(s,a)$ and $Q_2^*(s,a)$ are the optimal Q values of the state–action pair under two different SMDP models, and $\epsilon$ is a polynomial function of the difference in the transition and reward functions of the two SMDPs. Hence, without state abstraction, AMDPs with bounded error in their transition and reward functions would have bounded errors in their value function. Consequently, using imperfect AMDP models can still produce reasonable results.

### 3.2 Example AMDP Hierarchies

As examples of source (environment) MDPs and AMDP hierarchies, we present hierarchies for Cleanup World (MacGlashan et al. 2015) and the "fickle taxi" domain used by Dietterich (2000) for MAXQ. We use the object-oriented MDP (OO-MDP) formalism to represent these domains (Diuk, Cohen, and Littman 2008). These hierarchies are extensively defined in the supplementary material.[1]

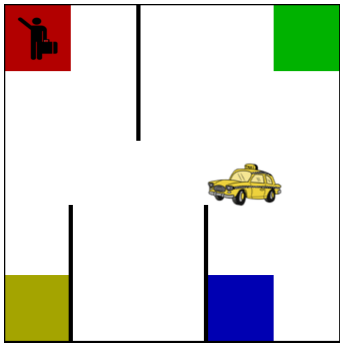
**Fickle Taxi** The Fickle Taxi problem was used by Dietterich (2000) to introduce the MAXQ planning hierarchy. We created AMDP analogs for the hierarchical structure used in the original MAXQ work. In the flat (level 0) MDP, there are six actions: north, south, east, west, pickup passenger, and drop passenger. States in the flat MDP consist of the physical grid-cell coordinates of passengers, taxi, and locations. Locations and passengers also have a color attribute, and the taxi has an attribute pointing to its current passenger, if any. Movement actions of the taxi are noisy with a probability of $0.2$, and the passenger changes their destination after getting in the taxi with a probability of $0.3$. The stochasticity of the fickle passenger penalizes hierarchical methods that use temporally extended navigation actions, which terminate only when the taxi reaches the goal location.

At level 1 of the hierarchy, the abstract actions are Pickup, which puts a passenger into the taxi if they are at the same location; Putdown, which drops a passenger off at the current location; and Nav($i$), which drives the taxi to location $i$. The transitions have been defined deterministic at this level; however, they can be defined stochastically according to the passenger's fickle behavior. The state for this AMDP abstracts away the physical coordinates of passengers, taxi, and goal locations, replacing them with level 1 passenger and taxi variables, each containing one of the four possible locations or an "on-road" designation that represents all states in which the taxi and the passengers are not at one of the four locations. For Nav(R), the reward function returns 1 when the taxi is at location Red, and 0 otherwise. Nav(R) terminates when the taxi is at the location Red.

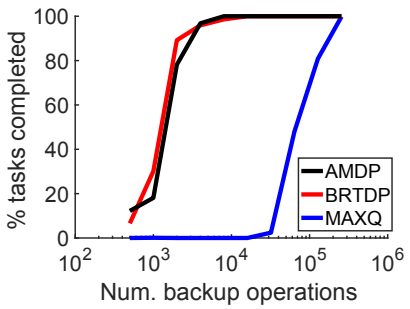
At level 2, the AMDP actions are Get, which puts the passenger into the taxi; and Put, which drops the passenger at their destination. The states abstract away the taxi and passenger locations. Hence, the subtask hierarchy of MAXQ is similar to AMDPs; however, the task decomposition is local in its reward functions, transition function, and value function or planning computations, resulting in substantial savings in planning time.

**Cleanup World** Our second evaluation domain is Cleanup World (MacGlashan et al. 2015), representing a mobile-manipulation robot, as shown in Figure 4a. Cleanup World is a good testbed for planning algorithms, because its state

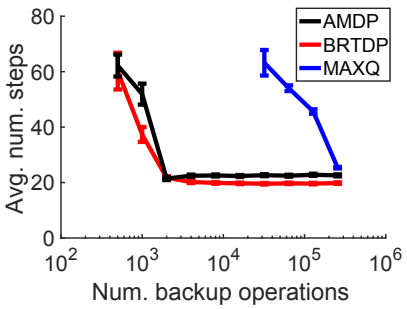
[1] http://h2r.cs.brown.edu/wp-content/uploads/2017/03/AMDP-ICAPS-SupplementaryMaterial.pdf



(a) Taxi problem




(b) Percentage of completed plans (of 1000 problems), given a backup budget.




(c) Average number of steps needed to deliver the passenger in completed plans.

Figure 3: Taxi domain using AMDPs, base-level planning (BRTDP), and MAXQ. AMDPs and the base-level planner have almost 100% completion rate after 4000 backup operations, whereas MAXQ needs orders of magnitude more backups to plan.

space grows combinatorially with the number of objects and rooms, similar to real-world robotics planning problems. The robot can have a variety of goals, such as moving the chair to the red room or moving all objects to the blue room. Abstract actions include moving to a door connected to the room in which the robot currently resides, moving from a door to a connected room, moving to an object currently in the same room (or doorway) as the robot, taking an object next to the robot to a door, and taking an object from a door to a connected room. The source MDP goal conditions for each of these AMDP actions can be defined based on the action arguments. Using the MoveToDoor($d$) AMDP as an example, its reward function returns 1 when the agent is at the specified doorway $d$, and 0 otherwise. MoveToDoor($d$) terminates at any state in which the robot is in the doorway.

The abstract actions define a corresponding state representation that abstracts away the geometric spatial information from the source OO-MDP. Such a representation retains the same objects as the source OO-MDP, but represents objects' positions and the room–door topology relationally instead of spatially. Specifically, the robot and household objects have an attribute that points to the doorway/room in which they reside; the robot has an attribute that points to adjacent household objects; and the room points to connected doorways (and vice versa). A second-level AMDP involves even higher-level actions, such as taking any given object to any given room. The corresponding high-level state space for this AMDP abstracts away the room-door topology.

## 4 Results

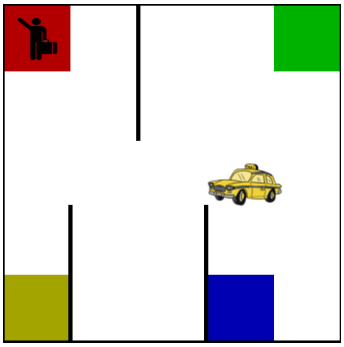
We compared AMDP planning performance against a flat planner and MAXQ in Fickle Taxi and Cleanup World. For each method, we generated plans using bounded RTDP (McMahan, Likhachev, and Gordon 2005), a state-of-the-art method with performance guarantees. We used MAXQ with state abstraction, that is, MAXQ-Q by Dietterich (2000). We compared approaches via two metrics: (1) Bellman updates/backups needed for effective planning and (2) steps taken to solve the task given a budget of Bellman updates. We plotted the average steps to completion only if the task was solved in more than 5% of 1000 trials. We recognize that MAXQ is learning a Q-function model-free; however, it is the closest comparison to AMDPs in terms of state and temporal abstraction. We also implemented options, but found, consistent with Jong (2008), that they performed less well than the base-level planner. We cannot use other top-down approaches, because HDP does not generalize to non-navigation based domains and DetH* does not allow multi-level hierarchies or stochastic transitions at higher levels of abstraction, which are needed for Taxi. We further used the AMDP hierarchy for Cleanup World to control a Turtlebot in a continuous space with planning over low-level control actions in the lowest level of the hierarchy.

### 4.1 Taxi Domain

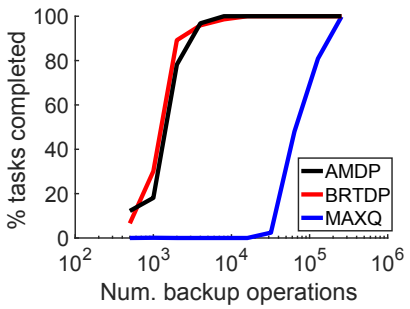
In Taxi, we defined a task to be completed if a planner found a solution that executed $\leq 100$ primitive actions. For MAXQ, we used the same state abstractions and tuned parameters given by Dietterich (2000). MAXQ required about 32,000 updates/backups to first becoming capable of completing the task, and about 256,000 backups to learn an optimal policy. In contrast, as shown in Figure 3b, AMDPs exhibited a 100% completion rate at 8000 backups, slightly faster than the base-level planner. The number of steps to complete the task with AMDPs (22.5) was higher than the base-level planner (19.8) (see Figure 3c), because the hierarchical action of navigate is penalized by the passenger's fickle behavior. The base-level planner outperforms AMDPs in this domain because the taxi task is relatively simple and plans are very short. However, this will no longer be the case when the domain's state space is combinatorially large.

### 4.2 Cleanup World

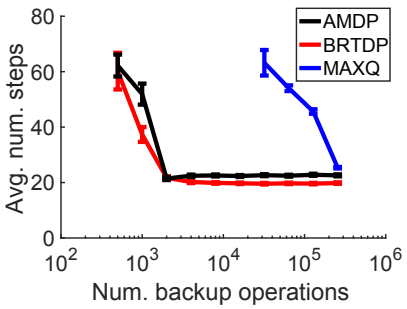
We constructed two different configurations of the Cleanup World domain. The first configuration (3rm) had 3 rooms with 3 objects and about 56 million states. The optimal plan was 25 steps. The second configuration (4rm, shown in Figure 4a) had 4 rooms with 3 objects and more than 900 million states. The optimal plan was 48 steps long. We defined the task to be completed if the agent is able to solve it within 5 times the number of steps to solve the task optimally.



(a) Taxi problem




(b) Percentage of completed plans (of 1000 problems), given a backup budget.




(c) Average number of steps needed to deliver the passenger in completed plans.

Figure 3: Taxi domain using AMDPs, base-level planning (BRTDP), and MAXQ. AMDPs and the base-level planner have almost 100% completion rate after 4000 backup operations, whereas MAXQ needs orders of magnitude more backups to plan.

space grows combinatorially with the number of objects and rooms, similar to real-world robotics planning problems. The robot can have a variety of goals, such as moving the chair to the red room or moving all objects to the blue room. Abstract actions include moving to a door connected to the room in which the robot currently resides, moving from a door to a connected room, moving to an object currently in the same room (or doorway) as the robot, taking an object next to the robot to a door, and taking an object from a door to a connected room. The source MDP goal conditions for each of these AMDP actions can be defined based on the action arguments. Using the MoveToDoor($d$) AMDP as an example, its reward function returns 1 when the agent is at the specified doorway $d$, and 0 otherwise. MoveToDoor($d$) terminates at any state in which the robot is in the doorway.

The abstract actions define a corresponding state representation that abstracts away the geometric spatial information from the source OO-MDP. Such a representation retains the same objects as the source OO-MDP, but represents objects' positions and the room–door topology relationally instead of spatially. Specifically, the robot and household objects have an attribute that points to the doorway/room in which they reside; the robot has an attribute that points to adjacent household objects; and the room points to connected doorways (and vice versa). A second-level AMDP involves even higher-level actions, such as taking any given object to any given room. The corresponding high-level state space for this AMDP abstracts away the room-door topology.

## 4 Results

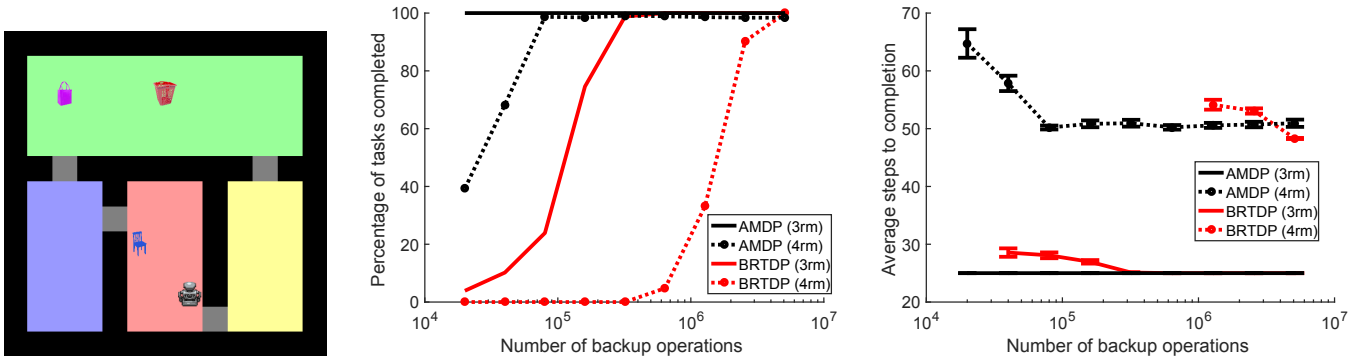
We compared AMDP planning performance against a flat planner and MAXQ in Fickle Taxi and Cleanup World. For each method, we generated plans using bounded RTDP (McMahan, Likhachev, and Gordon 2005), a state-of-the-art method with performance guarantees. We used MAXQ with state abstraction, that is, MAXQ-Q by Dietterich (2000). We compared approaches via two metrics: (1) Bellman updates/backups needed for effective planning and (2) steps taken to solve the task given a budget of Bellman updates. We plotted the average steps to completion only if the task was solved in more than 5% of 1000 trials. We recognize that MAXQ is learning a Q-function model-free; however, it is the closest comparison to AMDPs in terms of state and temporal abstraction. We also implemented options, but found, consistent with Jong (2008), that they performed less well than the base-level planner. We cannot use other top-down approaches, because HDP does not generalize to non-navigation based domains and DetH* does not allow multi-level hierarchies or stochastic transitions at higher levels of abstraction, which are needed for Taxi. We further used the AMDP hierarchy for Cleanup World to control a Turtlebot in a continuous space with planning over low-level control actions in the lowest level of the hierarchy.

### 4.1 Taxi Domain

In Taxi, we defined a task to be completed if a planner found a solution that executed $\leq 100$ primitive actions. For MAXQ, we used the same state abstractions and tuned parameters given by Dietterich (2000). MAXQ required about 32,000 updates/backups to first becoming capable of completing the task, and about 256,000 backups to learn an optimal policy. In contrast, as shown in Figure 3b, AMDPs exhibited a 100% completion rate at 8000 backups, slightly faster than the base-level planner. The number of steps to complete the task with AMDPs (22.5) was higher than the base-level planner (19.8) (see Figure 3c), because the hierarchical action of navigate is penalized by the passenger's fickle behavior. The base-level planner outperforms AMDPs in this domain because the taxi task is relatively simple and plans are very short. However, this will no longer be the case when the domain's state space is combinatorially large.

### 4.2 Cleanup World

We constructed two different configurations of the Cleanup World domain. The first configuration (3rm) had 3 rooms with 3 objects and about 56 million states. The optimal plan was 25 steps. The second configuration (4rm, shown in Figure 4a) had 4 rooms with 3 objects and more than 900 million states. The optimal plan was 48 steps long. We defined the task to be completed if the agent is able to solve it within 5 times the number of steps to solve the task optimally.




(a) Cleanup World (4rm) (b) Tasks completed within a planning budget. (c) Steps taken to move an object to a room.

Figure 4: (a) Cleanup World (4-room configuration shown) using AMDPs and base-level (BRTDP) planning. (b-c) AMDPs (black) have near-perfect completion rates, finding plans of similar length orders of magnitude faster than BRTDP (red).

In both configurations, AMDPs solved the task with an order of magnitude fewer backups than the base-level BRTDP, as shown in Figure 4. AMDPs solved 3rm and 4rm to almost $100\%$ completion rate with about $5000$ and $80{,}000$ backups, respectively, whereas BRTDP needed $320{,}000$ and $512{,}000$ backups, respectively, to achieve the same performance.

We also solved a basic Cleanup World problem with $3$ rooms and one object ($25{,}000$ states), in which both AMDPs and the base-level planners could find the optimal solution within $500$ backups, showing that the task was simple enough not to need complex hierarchies. However, we found that, even in this simplified configuration, MAXQ was unable to find a solution with $8{,}000{,}000$ backups.

AMDPs have a particular advantage in Cleanup World because long plans are needed, and solving tasks requires manipulating one out of many present objects. Such combinatorial state spaces are present in everyday manipulation tasks, and AMDPs are better equipped than other hierarchical methods or base-level planning to tackle them.

### 4.3 Continuous Cleanup World on Turtlebot

To illustrate the ability of AMDP hierarchies to transfer between domains and use different planners at different levels, we constructed a continuous version of a 3-room, 1-object Cleanup World domain, which we tested on a Turtlebot. The agent in Continuous Cleanup World is a modified TurtleBot with a pair of appendages that can guide blocks. The appendages make pushing the block easy; however, they make movement within the domain hard, as a point turn is not possible in most places, increasing the overall plan lengths.

At level $0$, the agent has a continuous position in Cartesian coordinates, in a $9\text{ft}\times 9\text{ft}$ world. The agent also has a continuous angle attribute for its orientation. The Turtlebot can move forward and turn clockwise/counter-clockwise. The continuous forward action moves the robot forward by about $0.1\text{ft}$. The turn actions change the orientation of the agent about $0.15$ radians. The object also has a continuous position attribute. The objective again is to move the object to a goal room, from any location in the world.

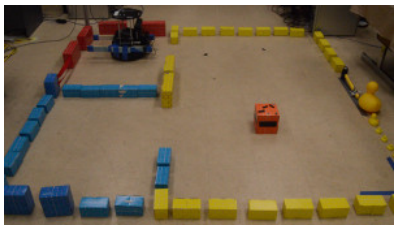
Our abstraction of the continuous source MDP connects to the (discrete) Cleanup World AMDP hierarchy from the previous subsection. At level $1$, we discretize the world into $9 \times 9$ cells and the agent's bearing into the $4$ cardinal directions. The subtasks at this level are moving forward by one grid cell, and turning clockwise/counter-clockwise by $\frac{\pi}{2}$ radians. This level-1 AMDP corresponds to the source MDP of (discrete) Cleanup World, so we immediately obtain higher-level abstractions for Continuous Cleanup World as well. Additionally, the top-down nature of AMDP planning enables the retention of all higher-level models and previously computed policies. In contrast, bottom-up approaches such as MAXQ would have needed to re-solve all subtasks.

In addition to the above level-1 abstraction, we also need to specify how to solve level-1 subtasks. Since the source MDP is continuous, low-level planners such as Rapidly Exploring Randomized Trees (LaValle and Kuffner 2001) could have been used to plan short distances of single grid movement. However, we use closed-loop controllers, because they allow faster planning and are less complicated to specify for these small movements. This domain application shows how different planners can be used at different levels of the AMDP hierarchy, depending on their suitability.

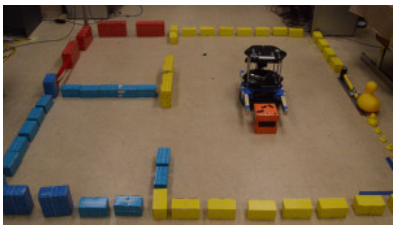
The Turtlebot is controlled using movement messages sent by our planner over ROS (Quigley et al. 2009) at $20\text{Hz}$. Our planners can publish commands at over $100\text{Hz}$, but $20\text{Hz}$ is the standard rate at which the Turtlebot publishes commands to its mobile base. The robot can be moved faster by publishing higher velocities per motion command, but we do not want the Turtlebot to move too quickly within the lab. The location and pose of the Turtlebot and the object are obtained using a motion capture system.

Put together, our AMDP hierarchy solves this continuous problem in real time. We can see in the video[2] that the robot plans in real time to complete the manipulation task as shown in Figure 5. The robot makes minor corrections to its trajectory and recovers from mistakes in real time. It also recovers when the object is removed from its arms, and replans to recover the object instantaneously to push the object to the goal location as shown in Figure 6. This shows that

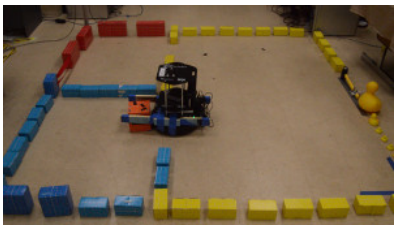
[2]https://youtu.be/Bp3VEO66WSg



(a) Starting position for the turtlebot.

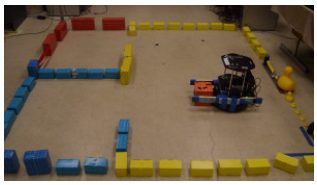


(b) Getting to the block.

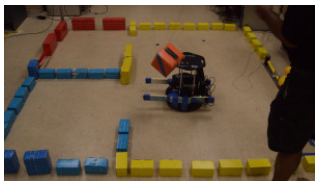


(c) Pushing the block into the goal room.

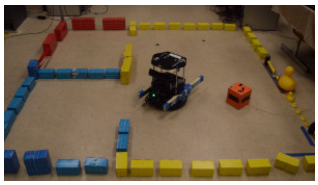
Figure 5: In this figure, we see that the robot starts from a random position and pushes the block into the goal room, with blue walls, using its arms. The robot makes minor corrections in trajectories when it overshoots.



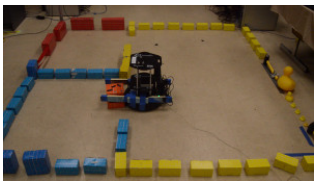
(a) Pushing the block.



(b) Losing the block.



(c) Adjusting trajectory.



(d) Getting to the goal.

Figure 6: This figure shows the importance of having MDPs to control the lowest level of the hierarchy. The Turtlebot starts to push the block into the goal location, but has the block removed from its arms, and replaced at another location. The Turtlebot corrects its trajectory immediately, getting to the block and pushing it into the goal.

fast reactive control is possible in top-down hierarchies such as AMDPs, even in domains with significant stochasticity. Furthermore, the hierarchy allows us to control the robot at different levels of abstraction, since we can execute any subgoal by only planning with its subtree in the task hierarchy.

## 5 Conclusions

We introduced a novel planning approach using Abstract Markov Decision Process (AMDP) hierarchies, which decompose large planning problems into AMDPs, representing subtasks that have local transition and reward functions. AMDPs compute plans for large problems orders of magnitude faster than other planners. Options and MAXQ may fail to plan in large domains because of the time spent on recursively decomposing the value functions from the base level. Hence, even though MAXQ and options provide strong theoretical guarantees of being recursively and hierarchically optimal, respectively, their planning time might be too long for actual agents to act in the world. AMDPs, on the other hand, offer a weaker notion of optimality at every abstract level, but allow faster planning in large domains. Previous top-down planning approaches such as HDP and DetH* are not generic enough to allow mobile manipulation in stochastic continuous domains. Moreover, the AMDP hierarchical structure allows AMDPs to be invariant to small changes in stochasticity at the flat level. This property has useful applications in robotics, when accurate models for the base-level transitions are not available, allowing the robot to recover reactively from missteps or environmental noise.

We demonstrated with the Taxi and Cleanup World domains that AMDPs trade orders-of-magnitude faster planning time for potentially suboptimal solutions. Additionally, in the Turtlebot Continuous Cleanup World, our AMDP hierarchy provides a model for a robot's entire capability stack. This unified model allows tasks, specified as high-level abstract goals, to be efficiently planned for and grounded into low-level control actions. This planning speedup is crucial for planning in large domains such as robotics, navigation, and search and rescue.

However, to fully realize the potential of AMDPs, we must go beyond provided abstract transition and reward functions, and fixed AMDP hierarchies and state abstractions. We believe that AMDP transition dynamics can be learned with model-learning methods without recursion to the base level, leading to a faster model-based approach to MAXQ reinforcement learning. Our current focus is on developing methods to learn the AMDP hierarchies and their state abstractions automatically from experiential data.

Further, the top-down hierarchy allows the choice of a specific subtask from the tree to be solved, instead of always solving the task from the root node. Subtasks such as "move forward one grid cell" can be solved by moving the root node to become the specific subtask, i.e., the new root for the AMDP hierarchy would be at "move forward." This flexibility would be useful when a robot is commanded by language, where directions of the form "go north three steps" and tasks like "go to the blue room with the block" can be solved in the same manner, within the same hierarchy. We are currently collecting natural language datasets to study this strategy. Ultimately, AMDPs will allow specifying robotics tasks at different levels of abstraction, such that we can command and guide robots using natural language.

## Acknowledgments

This material is based upon work supported by the National Science Foundation (grants IIS-1637614 and IIS-1637937), the US Army/DARPA (grant W911NF-15-1-0503), and the National Aeronautics and Space Administration (grant NNX16AR61G).

Lawson L.S. Wong was supported by a Croucher Foundation Fellowship.

## References

Bakker, B.; Zivkovic, Z.; and Krose, B. 2005. Hierarchical dynamic programming for robot path planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Barry, J. L.; Kaelbling, L. P.; and Lozano-Pérez, T. 2011. Deth: Approximate hierarchical solution of large markov decision processes. In *International Joint Conference on Artificial Intelligence*.

Bellman, R. 1956. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences* 42(10):767–769.

Bellman, R. 1957. A Markovian decision process. *Indiana University Mathematics Journal* 6:679–684.

Bollini, M.; Tellex, S.; Thompson, T.; Roy, N.; and Rus, D. 2012. Interpreting and executing recipes with a cooking robot. In *International Symposium on Experimental Robotics*.

Brunskill, E., and Li, L. 2014. PAC-inspired option discovery in lifelong reinforcement learning. In *International Conference on Machine Learning*, 316–324.

Dietterich, T. 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 *International Conference on Machine Learning*.

Diuk, C.; Littman, M. L.; and Strehl, A. 2006. A hierarchical approach to efficient reinforcement learning in deterministic domains. In *International Conference on Antonomous Agents and Multiagent Systems*.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Jong, N., and Stone, P. 2008. Hierarchical model-based reinforcement learning: R-max+ MAXQ. In *International Conference on Machine Learning*.

Jong, N. 2008. The utility of temporal abstraction in reinforcement learning. In *International Conference on Antonomous Agents and Multiagent Systems*.

Knepper, R.; Tellex, S.; Li, A.; Roy, N.; and Rus, D. 2013. Single assembly robot in search of human partner: Versatile grounded language generation. In *ACM/IEEE International Conference on Human-Robot Interaction Workshop on Collaborative Manipulation*.

Konidaris, G. 2016. Constructing abstraction hierarchies using a skill-symbol loop. In *International Joint Conference on Artificial Intelligence*.

LaValle, S. M., and Kuffner, J. J. 2001. Randomized kinodynamic planning. *The International Journal of Robotics Research* 20(5):378–400.

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*.

Marthi, B.; Russell, S. J.; and Wolfe, J. 2008. Angelic hierarchical planning: Optimal and online algorithms. In *International Conference on Automated Planning and Scheduling*.

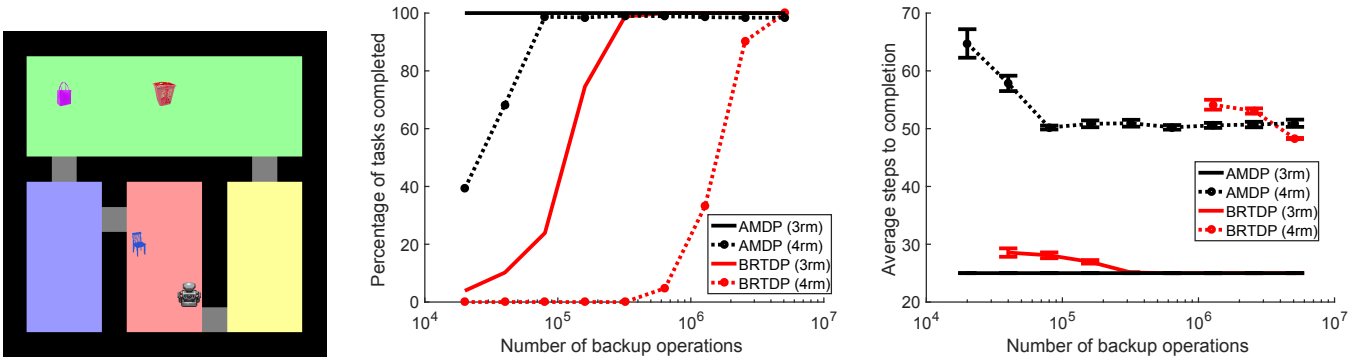
McGovern, A.; Sutton, R. S.; and Fagg, A. H. 1997. Roles of macro-actions in accelerating reinforcement learning. *Grace Hopper Celebration of Women in Computing* 1317.

McMahan, H.; Likhachev, M.; and Gordon, G. 2005. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *International Conference on Machine Learning*.

Nejati, N.; Langley, P.; and Konik, T. 2006. Learning hierarchical task networks by observation. In *Proceedings of the 23rd International Conference on Machine Learning*.

Quigley, M.; Faust, J.; Foote, T.; and Leibs, J. 2009. Ros: an open-source robot operating system. In *IEEE International Conference on Robotics and Automation Workshop on Open Source Software*.

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




(a) Cleanup World (4rm) (b) Tasks completed within a planning budget. (c) Steps taken to move an object to a room.

Figure 4: (a) Cleanup World (4-room configuration shown) using AMDPs and base-level (BRTDP) planning. (b-c) AMDPs (black) have near-perfect completion rates, finding plans of similar length orders of magnitude faster than BRTDP (red).

In both configurations, AMDPs solved the task with an order of magnitude fewer backups than the base-level BRTDP, as shown in Figure 4. AMDPs solved 3rm and 4rm to almost $100\%$ completion rate with about $5000$ and $80{,}000$ backups, respectively, whereas BRTDP needed $320{,}000$ and $512{,}000$ backups, respectively, to achieve the same performance.

We also solved a basic Cleanup World problem with $3$ rooms and one object ($25{,}000$ states), in which both AMDPs and the base-level planners could find the optimal solution within $500$ backups, showing that the task was simple enough not to need complex hierarchies. However, we found that, even in this simplified configuration, MAXQ was unable to find a solution with $8{,}000{,}000$ backups.

AMDPs have a particular advantage in Cleanup World because long plans are needed, and solving tasks requires manipulating one out of many present objects. Such combinatorial state spaces are present in everyday manipulation tasks, and AMDPs are better equipped than other hierarchical methods or base-level planning to tackle them.

### 4.3 Continuous Cleanup World on Turtlebot

To illustrate the ability of AMDP hierarchies to transfer between domains and use different planners at different levels, we constructed a continuous version of a 3-room, 1-object Cleanup World domain, which we tested on a Turtlebot. The agent in Continuous Cleanup World is a modified TurtleBot with a pair of appendages that can guide blocks. The appendages make pushing the block easy; however, they make movement within the domain hard, as a point turn is not possible in most places, increasing the overall plan lengths.

At level $0$, the agent has a continuous position in Cartesian coordinates, in a $9\text{ft}\times 9\text{ft}$ world. The agent also has a continuous angle attribute for its orientation. The Turtlebot can move forward and turn clockwise/counter-clockwise. The continuous forward action moves the robot forward by about $0.1\text{ft}$. The turn actions change the orientation of the agent about $0.15$ radians. The object also has a continuous position attribute. The objective again is to move the object to a goal room, from any location in the world.

Our abstraction of the continuous source MDP connects to the (discrete) Cleanup World AMDP hierarchy from the previous subsection. At level $1$, we discretize the world into $9 \times 9$ cells and the agent's bearing into the $4$ cardinal directions. The subtasks at this level are moving forward by one grid cell, and turning clockwise/counter-clockwise by $\frac{\pi}{2}$ radians. This level-1 AMDP corresponds to the source MDP of (discrete) Cleanup World, so we immediately obtain higher-level abstractions for Continuous Cleanup World as well. Additionally, the top-down nature of AMDP planning enables the retention of all higher-level models and previously computed policies. In contrast, bottom-up approaches such as MAXQ would have needed to re-solve all subtasks.

In addition to the above level-1 abstraction, we also need to specify how to solve level-1 subtasks. Since the source MDP is continuous, low-level planners such as Rapidly Exploring Randomized Trees (LaValle and Kuffner 2001) could have been used to plan short distances of single grid movement. However, we use closed-loop controllers, because they allow faster planning and are less complicated to specify for these small movements. This domain application shows how different planners can be used at different levels of the AMDP hierarchy, depending on their suitability.

The Turtlebot is controlled using movement messages sent by our planner over ROS (Quigley et al. 2009) at $20\text{Hz}$. Our planners can publish commands at over $100\text{Hz}$, but $20\text{Hz}$ is the standard rate at which the Turtlebot publishes commands to its mobile base. The robot can be moved faster by publishing higher velocities per motion command, but we do not want the Turtlebot to move too quickly within the lab. The location and pose of the Turtlebot and the object are obtained using a motion capture system.

Put together, our AMDP hierarchy solves this continuous problem in real time. We can see in the video[2] that the robot plans in real time to complete the manipulation task as shown in Figure 5. The robot makes minor corrections to its trajectory and recovers from mistakes in real time. It also recovers when the object is removed from its arms, and replans to recover the object instantaneously to push the object to the goal location as shown in Figure 6. This shows that

[2]https://youtu.be/Bp3VEO66WSg