Planning with Abstract Markov Decision Processes

Nakul Gopalan[1], **Marie desJardins**[2], **Michael L. Littman**[1], **James MacGlashan**[1], **Shawn Squire**[2], **Stefanie Tellex**[1], **John Winder**[2], **Lawson L.S. Wong**[1]

[1]Brown University, Providence, RI 02912 [2]University of Maryland, Baltimore County, Baltimore, MD 21250

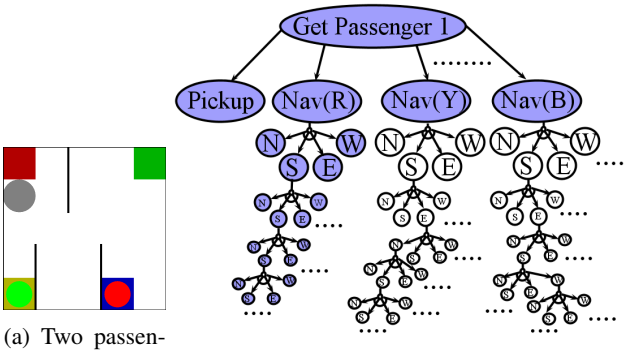
Abstract

Planning in large state-action spaces requires hierarchical abstraction for efficient computation. We introduce a new hierarchical planning framework called Abstract Markov Decision Processes (AMDPs) to find efficient solutions that, although possibly suboptimal, can solve complex planning problems in a fraction of the time needed for 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. Because the local reward and transition functions could be incorrect, the resulting solutions may be suboptimal; however, they also provide strong heuristic guidance, greatly decreasing planning time. The resulting hierarchical planning method, while not recursively optimal, 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 improved planning speed and equivalent solution quality, when compared to existing methods in the well studied Taxi domain and in a simulated mobile manipulation robotics problem.

1. Introduction

When carrying out tasks in unstructured, stochastic 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). Typical planning methods require the agent to explore the state–action space at the lowest level, requiring a combinatoric search through long sequences of actions.

A common method for dealing with the combinatorics of such long sequences is to provide the planner with temporally extended actions, encapsulating reusable segments of the plan into a more tractable form. These extended actions, with or without abstraction hierarchies, take the agents closer to important subgoals. Temporally extended



(a) Two passenger Taxi (b) Task hierarchy over the taxi problem

Figure 1. Abstract Markov Decision Process (AMDP) hierarchies avoid exploring the complete hierarchical tree, by using local transition and reward functions at each level of the hierarchy. The above figure is the task decomposition to explore a Get Passenger 1 node in the Taxi problem (Dieterrich, 2000), when the passenger is at the Red location. Approaches like MAXQ explore the complete subtree of each subtask, but AMDPs explore only the relevant branches, indicated with nodes colored blue, according to the local transition and reward functions.

actions can be provided to the planner in the form of options (Sutton et al., 1999) or macro-actions (McGovern et al., 1997). While extended actions can decrease plan length and complexity, their inclusion increases the branching factor of search leading to dramatic increases in planning time (Jong, 2008).

The MAXQ model (Dieterrich, 2000), in contrast, decomposes a flat MDP into smaller subtasks, each specifying a subgoal and potentially a state abstraction specific for the subtask's goal. MAXQ hierarchies were originally designed to be model-free and were used to accelerate model-free learning. However, they have also been used in a planning setting (Diuk et al., 2006). Planning in the MAXQ setting requires recursive exploration of all children in the hierarchy for value estimation of a node, which leads to long runtimes in large problems.

We introduce the concepts of an *abstract MDP* (AMDP) and an *AMDP hierarchy*. We define an AMDP to be a non-primitive subtask of a MAXQ hierarchy that includes an *abstract* semi-MDP transition function and reward function over the subtask's children and where the reward function reflects the goals of the subtask. By "abstract", we

mean that the model is computed, for reward and transition functions, without having to decompose the effects of its children, unlike a semi-MDP (SMDP) formulation (Sutton et al., 1999). We refer to a subtask with such an abstract model as an AMDP, because its states and actions consist of elements of a lower level MDP. An AMDP subtask includes a set of states abstracted from the underlying MDP's state space, a set of "actions" consisting of the subtask's children, a transition function mapping from these actions to probabilities of next state outcomes, and a reward function defining the goal of the subtask. An AMDP hierarchy, then, is a MAXQ task hierarchy whose subtasks are either AMDPs or primitive nodes. For example consider the Taxi problem shown in Figure 1a, a subtask to Get Passenger 1 has a set of states where the passenger is present in different locations, but the physical locations of the taxi or passengers are abstracted. Similarly the action of Nav(R) navigates the taxi to location Red in the abstract state in a single abstract MDP step, with deterministic transitions (in this case) defined for the subtask. The local reward function gives a goal reward of one when the passenger is picked. As we go higher up in an AMDP hierarchy the states abstract away more of the physical state and have actions that are temporally longer on ground level.

Because an AMDP has a transition and reward function, its policy can be determined using planning approaches for flat MDPs such as value iteration. The concept of using a subtask model for computing a policy has been used in previous work, such as R-MAXQ (Jong & Stone, 2008)—a model-based learning algorithm for MAXQ. However, in that work, the transition and reward function violated our model-abstraction requirement: the effects of a subtask's child was computed recursively, requiring planning for each child and descendant for each state visited in the subtask. The position we advocate in this work is that abstract models provide large savings in planning time by avoiding to plan for subtasks that are not selected. To illustrate this effect, consider Figure 1b, which shows a planning search tree for a Get Passenger 1 subtask in a multi-passenger version of the Taxi domain (Dieterrich, 2000). The blue nodes represent the state–action planning expansion performed when using an abstract model: the abstract transition model considers the effects of the pickup and navigate subtasks, which results in it selecting Nav(R). For that state, the AMDP planning only expands the Nav(R) subtask. In contrast, when recursive planning is carried out, as in MAXQ or R-MAXQ, *all* the nodes, both blue and white, need to be expanded to determine their effects. By avoiding deep expansion, AMDPs unlock the computational savings possible through hierarchical abstraction.

AMDPs also afford the opportunity to apply specialized planners for each subtask to further accelerate learning. For example, algorithms like bounded real time dynamic programming (BRTDP) (McMahan et al., 2005) with subtask-specific heuristics can be used to accelerate planning. It may also be possible to learn heuristics from previous experiences more easily over a variety of tasks, since heuristics can be bound to each relevant subtask and reused.

The work we present here is preliminary—our goal is to foster more discussion on the topic. Specifically, our results show that AMDPs use less planning time than base level planners and options to complete tasks in the Taxi and Cleanup World (MacGlashan et al., 2015) domains.

2. Related Work

An MDP (Bellman, 1957) is defined by a five-tuple ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{R}$, $\mathcal{E}$), where $\mathcal{S}$ is the agent's 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. Reasoning about large state and action spaces in the context of a standard MDP is hard, and hierarchical methods have been known to help speed up planning in these large domains (Sutton et al., 1999; Dieterrich, 2000).

Since hierarchical planners do not reason at the flat level, they do not maximize the expected future reward at the base level for a globally optimal plan. Dieterrich (2000) distinguishes between optimality at the base level, with 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* policy is defined as a hierarchically policy such that each subtask's corresponding policy is optimal for the SMDP defined by its set of states, actions, transition probabilities as defined multi-time step SMDP model (Sutton et al., 1999), and a rewards according to the original reward function.

3. Abstract Markov Decision Processes

AMDPs capture higher-level transition dynamics that serve as an abstraction of a lower level (A)MDP domain. Each AMDP directly abstracts either a single lower-level AMDP or the source MDP, inducing an AMDP hierarchy with a stack of (A)MDPs.[1] In effect, an AMDP defines a deci-

[1]This concept could be extended to build a DAG of MDPs.

sion problem over subgoals for its lower-level (A)MDP, using AMDP "actions". These actions also allow transitions between higher-level states of AMDPs. Planning in an AMDP can be orders of magnitude faster than planning in the lower-level MDP space because they do not require values to be backed up from all low-level MDP states. Instead, we only perform MDP-style planning at the *abstracted* level, where the state and action spaces are potentially much smaller. For this abstracted MDP planning, we require additionally, as input at each level, abstract transition and reward functions, and a state projection function to map ground states to abstract MDP states, all of which are engineered for this work. However, plans in AMDP space are specified in terms of abstract actions or subgoals that must be *grounded* in policies in the lower-level space to be carried out. We describe our approach to efficiently grounding abstract actions in Section 3.1.

We define an AMDP domain as a complete MDP domain in itself (state space, set of actions, transition dynamics, and terminal states) with the addition of a state-projection function F and a set of goal conditions G that are associated with each action. Given an MDP M, we define an AMDP domain $\tilde{M}$ based on M by adding a state projection function F and a set of goal conditions G that are associated with each action. The state-projection function maps states from the source MDP state space into the AMDP state space. That is, $\tilde{M}$ includes the state projection function $F : \mathcal{S} \rightarrow \tilde{\mathcal{S}}$, where $\mathcal{S}$ and $\tilde{\mathcal{S}}$ are state spaces of M and $\tilde{M}$. For example in the Taxi problem, an higher level abstract state might remove the Cartesian coordinates of the taxi, passenger and locations; and maintain relative attributes. For example the taxi can be at a location like Red or in transit between locations indicated by On Road. The goal conditions G in the AMDP specify the intended outcomes and termination conditions for the subtasks associated with each action. A goal condition for the AMDP action at level l in the hierarchy $\tilde{a}$ ($G_{\tilde{a}}$) has two components: (1) a reward function $G^{\mathcal{R}}_{\tilde{a}}$, which maps from states in the (A)MDP at level $l-1$ to a reward value,[2] and (2) $G^{\mathcal{E}}_{\tilde{a}} \subset \mathcal{S}$, a set of terminal states in the (A)MDP at level $l-1$. The reward function for the (A)MDP at $l-1$ is zero cost for taking each action until the terminal condition is reached for a reward of one. The transition function for AMDP action $\tilde{a}$ at level l is simply probability 1 of reaching an abstract state in $\tilde{S}$ where the goal condition for the AMDP action is satisfied in the (A)MDP at level $l-1$. The AMDP at the highest level uses similar termination conditions for its abstract states derived from the set of goal states in the flat MDP. Consider the level 1 action Nav(Red) action to navigate the taxi to the Red location. The goal condition at level 0 is for taxi to be present in the Red location at the base level. This goal condition defines the set of terminal next states with reward 1, otherwise the reward is 0 elsewhere. The transition probability at level 1 is 1.0 for the taxi to be at location Red with the rest of the state unchanged and 0 for other states.

When planning, 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 optimality within each level of abstraction is strictly weaker than recursively optimality, as we are not querying the reward function of the flat MDP while planning. Given an SMDP based multi time-step transition function and sum of rewards over the original reward function, AMDP policies become recursively optimal by definition (Dieterrich, 2000). However, computing SMDP based transition and reward functions lead to longer planning times. Recursive methods for learning transition functions have been applied in the context of R-MAXQ (Jong & Stone, 2008). However, we believe algorithms in the AMDP setting can learn these transition functions without the expense of recursion to the base level.
.

3.1. Planning in AMDPs

Algorithm 1 AMDP Solving and Execution

```
function SOLVE(s, G)
    s_0 ← s
    SOLVE-HELPER(G, 0)
end function
function SOLVE-HELPER(G, i)
    if i<max_layer then   ▷ project states to highest AMDP
        s_{i+1} ← PROJECT(s_i, i + 1)
        SOLVE-HELPER(G, i + 1)
    else                  ▷ ground actions from highest AMDP
        GROUND(G, i)
    end if
end function
function GROUND(G, i)
    π ← PLAN(s_i, G, i)
    while s_i ∉ G^E do    ▷ execute until local goal satisfied
        a ← π(s_i)
        if i > 0 then     ▷ give goal conditions to level below
            GROUND(a, i − 1)
        else              ▷ execute actions in environment
            s_i ← EXECUTE(a)
        end if
    end while
    s_{i+1} ← PROJECT(s_i, i + 1)
end function
```

Given a corresponding source MDP, an initial state in it, and an AMDP hierarchy, a computationally demanding "flat" planning problem can be simplified by using an online planning algorithm that first computes a high-level

[2] More generally, the reward function may also be defined as depending on the previous state, action, and next state.

AMDP policy and then reduces each selected action in the AMDP policy into a separate planning problem in the lower-level source MDP.

Planning and execution of AMDPs operates on multiple hierarchical levels of abstraction. First, the initial AMDP state is determined at the highest layer of abstraction by recursively applying each layer's state-projection function from the lowest-level MDP up. Second, a downward-recursive grounding procedure, consisting of planning and execution, is initiated from the top layer for the global goal. In the planning step, a policy for the goal condition rooted in the current abstract state is computed. In the execution step, the computed policy is *followed* until a terminal abstract state is reached. A policy is followed by selecting and executing actions until a terminal abstract state is reached. An AMDP action is executed by calling the grounding procedure on the layer below with the goal conditions of the AMDP action. At the completion of a grounding procedure, it projects its exiting state to the above layer. The complete pseudocode is shown in Algorithm 1.

AMDP planning can be substantially faster than previous hierarchical planners for two reasons. First, the AMDP planners do not need recursively compute the values, transitions and rewards at each node, unlike previous methods. Second, structured subgoals at individual levels often have extremely efficient algorithms. For example, in the Taxi problem, the navigation subgoals enable A* to be used with a goal based Euclidean distance heuristic for the navigation subtask. MAXQ based methods do not offer modularity on the choice of such algorithms at each level.

3.2. Example AMDP Hierarchies

To provide examples of a source MDPs and AMDP hierarchies, we present results for the cleanup domain (MacGlashan et al., 2015) and taxi domain (Dieterrich, 2000). We use the object-oriented MDP (OO-MDP) formalism to express all these problems (Diuk et al., 2008).

3.2.1. Two passenger Taxi

We now consider in detail AMDPs for the Taxi problem with two passengers with their own individual destinations instead of a single passenger. We consider the two passenger case because with two passengers differences between the notions of recursively optimal and hierarchical optimal are clear by the order in which the passengers are picked and dropped. We created AMDP analogs for the hierarchical structure used in the original MAXQ problem (Dieterrich, 2000). In the flat MDP, at level 0 of the hierarchy, there are six actions: north, south, east, west, pickup passenger, drop passenger. The state at the flat MDP consists of passengers, taxi and locations, with attributes of their physical locations in the grid. Locations and passengers also have color as an additional attribute while the taxi has an attribute pointing to a current passenger, if any. At level 1 of the hierarchy, the abstract actions are: Pickup, which moves 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 state for this AMDP abstracts away the physical location of passengers, taxi and the locations. The goal condition for the Nav(Red) AMDP action is a reward function that returns zero cost for actions until the taxi is at location Red, where the reward is 1; and a set of terminal states in which the taxi is at the location Red with the rest of the state unchanged. At level 2 of the AMDP stack, the actions are: Get Passenger(i) that puts passenger i into the taxi, and Put Passenger(i) that grounds to dropping passenger i to its destination. The states abstract away that the taxi and passengers have attributes pointing to their current 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. This local task decomposition gives AMDPs savings in planning times.

3.2.2. Cleanup World

Our second evaluation domain is Cleanup World (MacGlashan et al., 2015), representing a mobile-manipulator robot as shown in Figure 3a. Cleanup domain is a good test bed for planning algorithms as its state space explodes combinatorially with objects and rooms like real life robotics planning problems. The robot can have a variety of goals, such as moving the chair to the red room and moving all objects to the blue room. Abstract actions include the robot moving to a door connected to the room in which the robot currently resides, from a door to a connected room, to an object currently in the same room (or doorway) as the robot, taking an object to which the robot is adjacent 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. For example, the goal condition for the MoveToDoor AMDP action is a reward function that returns zero everywhere except when the agent is at the specified doorway, in which case it returns a reward of one; and the terminal states are all states in which the robot is in the doorway.

The abstract actions define a corresponding state representation that abstracts away the spatial information from the source OO-MDP. Such a representation retains the same objects as the source OO-MDP, but represent position attribute information and the room–door topology relationally instead of spatially. Specifically, the robot and household objects have an attribute that points to the door/room in which they reside; the robot has an attribute that points to household objects to which it is adjacent; and the room

points to connected doors (and vice versa). Projecting states from the source MDP into this AMDP is done by finding the room/door the robot or an object resides using comparisons of the position and region boundary attributes. Similarly, it is trivial to determine which doors are connected to which rooms by testing whether a room and door's rectangular regions intersect. A third-level AMDP with even higher-level actions, such as taking any given object to any given room, can also be introduced. The corresponding high-level state space for this AMDP abstracts away the room-door topology.

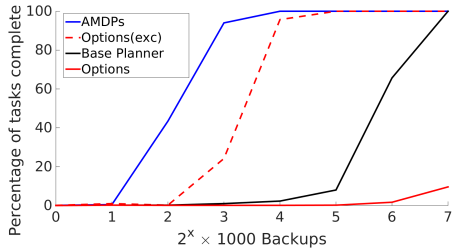
4. Results

We compared AMDP planning performance against a flat planner and options in Taxi and Cleanup World. For Cleanup World, we only have preliminary options results because of long run times. We used options in two ways: (1) *options* as used commonly, allowing base-level actions to be also chosen from each state; (2) *exclusively options*, where we use options without base-level actions for planning. We expected that using options exclusively would speeds up planning, since the planner has a lower action branching factor. For each method, we generated plans using BRTDP (McMahan et al., 2005), which has performance guarantees in large domains. We compared different abstraction methods over two metrics: (1) Bellman updates or backups needed for effective planning; (2) steps taken to complete the task, given a budget of Bellman updates.

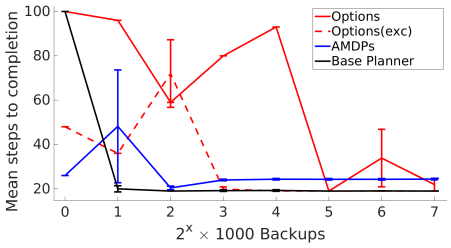
4.1. Taxi Domain

For the two-passenger taxi domain, we handcrafted an AMDP hierarchy, as mentioned in Section 3.2.1. The domain is deterministic in its transitions, but a stochastic transition function would not change the result trends. The base-level planner is BRTDP. The options provided are the same as the actions of the AMDP at level 1 (Pickup, Putdown, Nav(i)). The option-selection plan is derived by BRTDP, given initiation conditions and termination conditions similar to AMDPs. We ran each approach 1000 times to compute averages and completion rates, as shown in Figure 2. We counted a task as *completed* if it finished within 100 actions. Data points with low completion rates may have wide confidence intervals, but all methods found consistent plans with more backups.

AMDPs have a 100% completion rate at $16,000$ backups. The next fastest approach is options used exclusively, which is an order of magnitude slower. In two-passenger taxi, the order in which passengers are picked up and dropped off changes the number of actions required from 19 to 30, where 19 is optimal. Options find an optimal path of 19 steps, because they query the flat MDP reward function to calculate the expected values of the options from



(a) Percentage of completion given a backup budget.



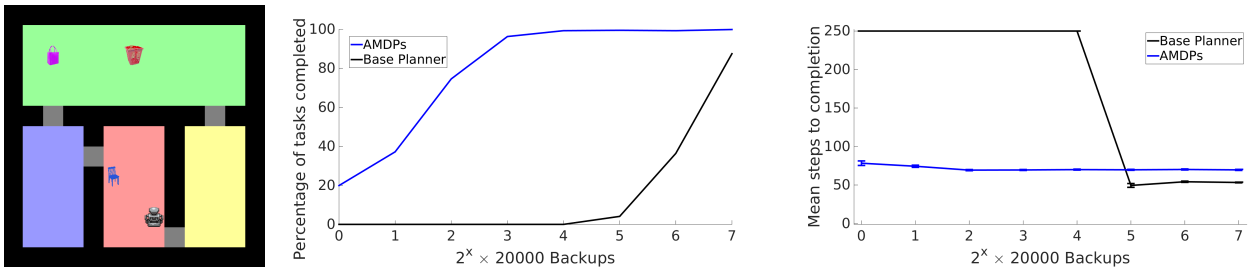
(b) Number of steps needed deliver both passengers.

Figure 2. Taxi domain using AMDPs, base level planning, and Options. AMDPs have almost a 100% completion rates from 8000 backups; options used exclusively need an order of magnitude more backups for similar completion rates.

states. AMDPs, being optimal only at every level of abstraction, cannot model a difference between dropping one passenger first vs. the other, and take about 24.5 actions to complete the task. The base-level planner and options with base actions take about $128,000$ and $350,000$ backups, respectively, to converge to 100% completions rates.

4.2. Cleanup World

We constructed a 4-room Cleanup World problem, and used AMDPs to solve them with the hierarchy mentioned in Section 3.2.2. The state space of the 4-room Cleanup World problem has more than 900 million states. A task is considered completed if the agent is able to solve it within 250 steps. The optimal number of steps for this problem is about 53 steps. AMDPs solve the task with a 100% completion rate with about $160,000$ backups. Even with 2.5 million backups, the base-level planner completes the task only about 87.5% of the time. Over 1000 runs, the average length of paths computed by AMDPs is about 70, and when the base level planners converge, they produce paths that are on average 53 steps long. Our preliminary results on options (based on AMDP level 1 actions) with 100 trials show a lower completion rate than the base-level planner, with 2.5 million backups when options are used exclusively. Options with base-level actions have an even lower completion rate, with 2.5 million backups. We noticed that options are spending many of these backups exploring actions that are not needed to complete the goal. Hence, we can confidently say that AMDPs obtain much better planning performance on the large Cleanup World domain.



(a) Cleanup world (b) Percentage of completion given a backup budget. (c) Average steps to move an object to a goal room.

Figure 3. Cleanup domain using AMDPs, base level planning. AMDPs have almost a 100% completion rates from 160, 000 backups.

5. Discussion

We notice that AMDP planning is orders of magnitude faster than planning at the ground level or options. We believe that hierarchical planning methods such as options and MAXQ would fail to plan in large domains because of the time spent on recursively decomposing the value functions to the base level. Hence, even though MAXQ and options provide a strong theoretical guarantees of being recursively (Dieterrich, 2000) and hierarchically optimal (Sutton et al., 1999), their time to plan might be too long for actual agents to act in the world. AMDPs, on the other hand, provide a weaker notion of optimality at every abstract level, allow faster planning in large domains. Moreover, the AMDP hierarchical structure without the SMDP transition function calculation allows us to be invariant to the small changes in stochasticity at the flat level. This generalization has useful applications in robotics, when we do not have accurate models for the base-level transitions.

We have many goals to achieve before we can fully realize the potential of AMDPs. First, we need planning time comparisons to the MAXQ algorithm given AMDP's local reward and transition functions. Secondly, 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. Further, we want to learn the AMDP hierarchies and their state abstractions automatically from data.

6. Conclusion

We introduced a novel planning method based on Abstract Markov Decision Processes (AMDPs). An AMDP hierarchy decomposes a planning problem into subtasks that have local reward and transition functions. Conventional hierarchical planning is slow because of their recursive value function decomposition. We demonstrated with Taxi and Cleanup World domains that AMDPs trade order of magnitude faster planning time for suboptimal solutions. We believe this planning speedup is crucial for planing in large domains like robotics and games like Minecraft.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. IIS -1426452, and by DARPA under grants W911NF-15-1-0503 and D15AP00102.

References

Bellman, R. A Markovian decision process. Technical report, DTIC Document, 1957.

Bollini, M., Tellex, S., Thompson, T., Roy, N., and Rus, D. Interpreting and executing recipes with a cooking robot. In *Proceedings of ISER*, 2012.

Dieterrich, T.G. Hierarchical reinforcement learning with the MAXQ value function decomposition. *JAIR*, 13:227–303, 2000.

Diuk, C., Littman, M.L., and Strehl, A.L. A hierarchical approach to efficient reinforcement learning in deterministic domains. In *AAMAS*, 2006.

Diuk, C., Cohen, A., and Littman, M.L. An object-oriented representation for efficient reinforcement learning. In *ICML*, 2008.

Jong, N.K. The utility of temporal abstraction in reinforcement learning. In *AAMAS*, 2008.

Jong, N.K. and Stone, P. Hierarchical model-based reinforcement learning: R-max+ MAXQ. In *ICML*, 2008.

Knepper, R.A., Tellex, S., Li, A., Roy, N., and Rus, D. Single assembly robot in search of human partner: Versatile grounded language generation. In *Proceedings of the HRI 2013 Workshop on Collaborative Manipulation*, 2013.

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

McGovern, Amy, Sutton, Richard S, and Fagg, Andrew H. Roles of macro-actions in accelerating reinforcement learning. *Grace Hopper celebration of women in computing*, 1317, 1997.

McMahan, H.B., Likhachev, M., and Gordon, G.J. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML*, 2005.

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