To appear in *The Seventh International Conference on Autonomous Agents and Multiagent Systems* (AAMAS 08), Estoril, Portugal, May 2008.

The Utility of Temporal Abstraction in Reinforcement Learning

Nicholas K. Jong
The University of Texas at Austin
1 University Station C0500
Austin, Texas 78712-0233
nkj@cs.utexas.edu

Todd Hester
The University of Texas at Austin
1 University Station C0500
Austin, Texas 78712-0233
todd@cs.utexas.edu

Peter Stone
The University of Texas at Austin
1 University Station C0500
Austin, Texas 78712-0233
pstone@cs.utexas.edu

ABSTRACT

The hierarchical structure of real-world problems has motivated extensive research into temporal abstractions for reinforcement learning, but precisely how these abstractions allow agents to improve their learning performance is not well understood. This paper investigates the connection between temporal abstraction and an agent's exploration policy, which determines how the agent's performance improves over time. Experimental results with standard methods for incorporating temporal abstractions show that these methods benefit learning only in limited contexts. The primary contribution of this paper is a clearer understanding of how hierarchical decompositions interact with reinforcement learning algorithms, with important consequences for the manual design or automatic discovery of action hierarchies.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms, Experimentation

Keywords

reinforcement learning, temporal abstraction

1. INTRODUCTION

The concept of hierarchy has strong intuitive appeal to artificial intelligence researchers. Humans cope with the extraordinary complexity of the real world in part by thinking hierarchically, and we would like to imbue our autonomous agents with the same faculty. In the reinforcement learning (RL) community, this idea has taken shape in work on temporal abstraction, in which abstract actions represent sequences of lower-level actions [1]. Early work demonstrated the potential of handcrafted temporal abstractions for improving the performance of RL agents in particular problems, but it raised the question of how to discover this hierarchical structure automatically [1]. This open question remains at the forefront of research into hierarchical RL. In this paper, we argue that efforts to enable agents to discover hierarchy automatically have been frustrated by a lack of clear understanding of how precisely temporal abstraction impacts learning performance.

Cite as: The Utility of Temporal Abstraction in Reinforcement Learning, Nicholas K. Jong, Todd Hester and Peter Stone, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX.
Copyright © 2008, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

One reflection of this uncertainty is the variety in motivations for applying temporal abstraction in RL. For example, the idea behind the popular options framework [14] is "to permit one to add temporally extended activities to the repertoire of choices available to an RL agent, while at the same time not precluding planning and learning at the finer grain of the core MDP. The emphasis is therefore on augmentation rather than simplification of the core MDP" [1]. In contrast, other researchers created temporal abstractions to constrain the choices available to an RL agent: "There are many reasons to introduce hierarchical reinforcement learning, but perhaps the most important reason is to create opportunities for state abstraction," so that "individual MDPs within the hierarchy can ignore large parts of the state space" [3]. We can't effectively design algorithms to discover temporal abstractions without knowing whether we're trying to augment or to abstract the learning problem!

The discovery algorithms proposed so far attempt to capture intuitions about what constitutes a useful abstract action. Most of these algorithms look for subgoal states and then define abstract actions that attempt to reach the discovered subgoals [7, 8, 10]. In principle, the creation of a subgoal decomposes the learning problem into at least two smaller problems: learning to attain the subgoal and learning optimal behavior from a subgoal state. In practice, prior work has not made clear how these approaches improve learning performance.

In this paper, we investigate and clarify the conditions under which temporal abstractions improve the performance of RL agents. Our experiments reveal that the conventional approach to formalizing and deploying abstract actions conflates the benefits of temporal abstraction with the benefits of other techniques. We suggest new directions for the study of temporal abstraction in general and hierarchy discovery in particular.

2. BACKGROUND

An RL agent assumes that its environment is a Markov decision process (MDP) [9]. An MDP $M = \langle S, A, T, R\rangle$ comprises a finite set of states S, a finite set of actions A, a transition function $T : S \times A \times S \to [0, 1]$, and a reward function $R : S \times A \to \mathbb{R}$. Given that the agent is in state $s \in S$ and executes action $a \in A$, it receives an expected immediate reward $R(s, a)$ and transitions to successor state $s' \in S$ with probability $T(s, a, s')$.

The goal of an RL agent is to learn a policy $\pi : S \to A$ that maximizes its expected cumulative reward [13]. To this end, the agent estimates the optimal value function, which satisfies the Bellman equations for each state $s \in S$:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s') \quad (1)$$

$$V(s) = \max_a Q(s,a), \quad (2)$$

where $\gamma \in [0, 1]$ is a discount factor that weights the value of future rewards. Intuitively, $V(s)$ is the expected cumulative reward given optimal behavior from state s; $Q(s, a)$ is the expected cumulative reward given the execution of action a in state s and optimal behavior thereafter. An agent can define an optimal policy from the optimal value function simply by choosing π such that $Q(s, \pi(s)) = V(s)$ for all $s \in S$.

An agent could compute the optimal value function directly if it knew the parameters of the MDP, but in the RL setting only the state space S and action space A are known a priori. Instead, the agent must estimate Q using data sampled from its environment. The seminal Q-learning algorithm [15] applies a simple update rule after a time step t in which the agent executed action a_t in state s_t and observed an immediate reward r_{t+1} and a transition to state s_{t+1}:

$$Q(s_t, a_t) \xleftarrow{\alpha_t} r_{t+1} + \gamma V(s_{t+1}), \quad (3)$$

where $\alpha \in [0, 1]$ is the learning rate and the notation $x \xleftarrow{\alpha} y$ denotes the assignment to x of the value $(1 - \alpha)x + \alpha y$, so that x is a stochastic approximation of the random variable sampled by y. Q-learning converges in the limit to the optimal value function, if the agent tries each state-action pair infinitely often and if α_t converges to 0 at a suitable rate.[1] A common exploration policy is the ϵ-greedy policy, which selects a random action with probability ϵ and otherwise chooses an action that maximizes $Q(s, \cdot)$.

2.1 Options

Q-learning remains the most popular RL algorithm due in large part to its simplicity and its theoretical convergence guarantees, but in practice it converges too slowly for most real-world autonomous agents. Research into hierarchical RL attempts to speed up learning by allowing agents to learn tasks by reasoning in terms of high-level abstract actions, which are in turn defined as the solutions to subtasks. In this paper, we adopt the options framework [14] for temporal abstraction, to remain consistent with the vast majority of prior work. An option $o = \langle I^o, \pi^o, \beta^o \rangle$ comprises an initiation set $I^o \subseteq S$, an option policy $\pi^o : S \to A$, and a termination function $\beta^o : S \to [0, 1]$. The initiation set I^o specifies the set of states in which the option is available as an (abstract) action. The option policy π^o specifies the primitive actions the option selects during its execution. The termination function β^o gives the probability $\beta^o(s)$ that option o will terminate upon transitioning into state s.

Options have become the most popular formulism for temporal abstraction in part due to their simplicity. An agent may simply treat an option as an action that may take more than one time step to execute. The addition of options transforms an MDP into a semi-Markov decision process (SMDP), in which the transition function specifies, for each state-action pair, a joint probability distribution over successor states and durations of execution [14]. RL algorithms designed for standard MDPs typically have trivial extensions to the SMDP case, so options are easy to deploy. For example, SMDP Q-learning simply revises the update rule in Equation 3 by appropriately discounting the value of the successor state and by replacing the immediate reward with the discounted reward accumulated during the option's execution:

$$Q(s_t, o_t) \xleftarrow{\alpha_t} \left(\sum_{i=1}^{k} \gamma^{i-1} r_{t+i} \right) + \gamma^k V(s_{t+k}), \quad (4)$$

where o_t is the potentially abstract action executed, k is the duration of the action's execution, and s_{t+k} is the successor state, in which the action terminated. However, only making one update to the value function each time an action terminates is too inefficient in practice. Most implementations of SMDP Q-learning use techniques such as intra-option learning [14] to generate additional updates to the value function.

3. LEARNING OPTION POLICIES

A thesis of this paper is that the options framework's simplicity and convenience belie subtleties in how precisely options improve agents' learning performance. One important subtlety arises from the formal definition of an option not as a subtask in a learning problem but as a solution to a subtask. For an illustration of how this distinction has impacted research in hierarchical RL, we consider the work on option discovery.

3.1 Option Discovery

Most existing algorithms for discovering temporal abstractions fit the same overall pattern. They identify certain states as subgoals, whether by finding states that frequently occur in successful episodes [8], that correlate with finding novel states [10], or that connect clusters of a state transition graph [7]. The agents then define options that transition to these subgoal states, obtaining the option policy using Experience Replay [6], a technique originally developed to speed the convergence of Q-learning. This technique works by simply applying the appropriate update rule (Equation 3 or 4) in batch fashion to saved trajectories of experience, so that each piece of data is used for more than one update. Given a newly discovered subgoal state, an agent can create an option by first defining an RL subproblem in which this state has high value. The agent then uses Experience Replay to propagate this value back through the option's state space, obtaining a local value function and the option policy. SMDP Q-learning [14] with the resulting options, discovered online, is then shown to improve upon standard Q-learning.

Although previous work demonstrated algorithms that outperform basic Q-learning, it offered at best an incomplete picture of the benefits of temporal abstraction. Given the procedure followed by most of the existing work on option discovery, one important question concerns the relative contributions of Experience Replay and of temporal abstraction to the performance improvements. In particular, could Experience Replay alone provide the same benefit as using

[1] $\sum_{i=t}^{\infty} \alpha_i = \infty$ and $\sum_{i=t}^{\infty} \alpha_i^2 = 0$

options whose policies are learned using Experience Replay?

McGovern and Barto briefly addressed this issue in their work on option discovery [8]. In their experiments, they included a condition that used Experience Replay without options, but they limited Experience Replay to the same number of value updates as they used to learn their option policies. If Experience Replay was thus applied to the entire learning problem, it performed worse that using the same number of backups to learn option policies. However, simply limiting Experience Replay to the states in the proposed option's initiation set provided the same benefit as learning the option policy, without actually creating an option.

This latter result alone should cast doubt over the contribution of temporal abstraction in this particular scenario, since the benefit seems to arise simply from focusing the efforts of Experience Replay to states near the discovered "subgoal" state. We argue that the problem is even deeper, since using subgoals to focus Experience Replay in this manner only makes sense if we assume that enough computation time exists to perform Experience Replay but not enough to apply it more globally. Most evaluations of RL performance, including those in the literature on option discovery, measure reward earned as a function of the amount of data used, not the amount of computation time. Nevertheless, recent work in temporal abstraction discovery continues to appeal to the unproven intuition that subgoal discovery helps to decompose learning tasks into simpler subtasks.

To clarify the interaction between Experience Replay and options, we conduct our own experiments with these two techniques in Section 3.4. First, we introduce in Section 3.2 the learning algorithm we use throughout this paper, and in Section 3.3 we describe the environment in which our experimental agents learn.

3.2 An SMDP Learning Algorithm

The experiments in this paper use the learning algorithm shown in Algorithm 1, adapted from Singh, Barto, and Chentanez [11]. The notation $O(s)$ denotes the set of options that include s among their initation sets: $O(s) = \{o \in O \mid s \in I^o\}$, and δ_s^x is 1 if $s = x$ and 0 otherwise. This algorithm augments Q-learning with a mechanism reminiscent of Dyna [13] to update the value function for options using learned models of those options. By learning models, the agent can predict the state to which each option will transition, as well as the expected rewards that will accrue during the execution of the option [14].

In each experiment, ϵ-greedy action selection was used with $\epsilon = 0.1$. The random action is selected from the available set of both primitive actions and options. The other parameters used were $\alpha = 0.3$ and $\gamma = 0.9$. Unless otherwise specified, the value function was initialized to 0.0.

Following Sutton, Precup, and Singh [14], we also allow the agent to interrupt the execution of an option if it enters a state where another action has higher value than the option being executed (see Line 25).

3.3 The Four-Room Gridworld

For consistency with prior work, we conduct our experiments in the simple four-room gridworld employed by Sutton, Precup, and Singh [14], shown in Figure 1. Each cell in the grid represents a state that the agent may occupy. From each cell, the agent can take one of four primitive actions: left, right, up, or down. Each primitive action is stochastic,

Algorithm 1

1: **loop**
2: Current state s_t, current primitive action a_t
3: Current option (or primitive action) o_t
4: Obtain reward r_{t+1} and observe next state s_{t+1}
5:
6: // Update option models
7: **for all** options $o \in O$ **do**
8: **if** $a_t = \pi^o(s_t)$ **then**
9: **for all** $x \in S$ **do**
10: $T(s_t, o, x) \overset{\alpha}{\leftarrow} (1 - \beta^o(s_{t+1}))\gamma T(s_{t+1}, o, x) + \beta^o(s_{t+1})\delta^x_{s_{t+1}}$
11: **end for**
12: $R(s_t, o) \overset{\alpha}{\leftarrow} r_t + \gamma(1 - \beta^o(s_{t+1}))R(s_{t+1}, o)$
13: **end if**
14: **end for**
15:
16: // Update value of primitive action
17: $Q(s_t, a_t) \overset{\alpha}{\leftarrow} r_t + \gamma V(s_{t+1})$
18:
19: // Update values of options
20: **for all** options $o \in O(s_t)$ **do**
21: $Q(s_t, o) \overset{\alpha}{\leftarrow} R(s_t, o) + \gamma \sum_{x \in S} T(s_t, o, x)V(x)$
22: **end for**
23:
24: // Choose option (or primitive action)
25: **if** o_t is primitive or $V(s_{t+1}) > Q(s_{t+1}, o_t)$ or with probability $\beta^{o_t}(s_{t+1})$ **then**
26: Choose $o_{t+1} \in A \cup O(s_{t+1})$ // ϵ-greedy
27: **else**
28: // Keep executing option o_t
29: $o_{t+1} \leftarrow o_t$
30: **end if**
31:
32: // Set next primitive action
33: **if** $o_{t+1} \in O$ **then**
34: $a_{t+1} \leftarrow \pi^{o_{t+1}}(s_{t+1})$
35: **else**
36: $a_{t+1} = o_{t+1}$
37: **end if**
38: **end loop**

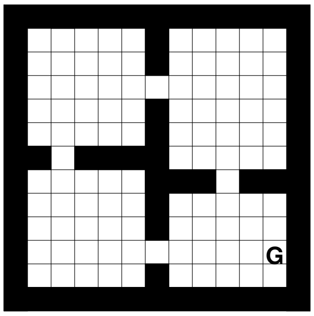

Figure 1: The four-room gridworld

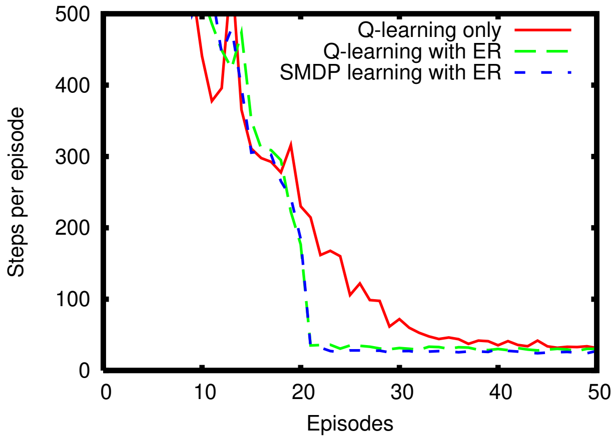

Figure 2: Learning performance of three agents that use standard Q-learning until episode 20, when one agent defines options using Experience Replay and another just uses Experience Replay to update its value function. Each line shows the number of steps required to reach the goal state, averaged over 50 independent runs.

taking the agent to the desired cell with probability 0.8 and in a perpendicular direction with probability 0.2. For example, when selecting the up action, the agent would move up 80% of the time, left 10% of the time and right 10% of the time. If the effect of the movement would place the agent in a wall, then the agent remains in its current cell. The agent starts each episode at a random location in the upper left room and the goal is a state near the lower right corner of the grid-world. The immediate reward is 0 at each state in the world, except at the goal, where the reward is 1.

This environment is clearly quite simple, and it was used in prior work presumably due to its suitability for learning with temporal abstraction. In particular, the doorway states are intuitive subgoals for an agent that must navigate this world. Nevertheless, we shall see in the experiments in this paper that even in this simple domain, the utility of temporal abstraction depends on numerous factors.

3.4 Options and Experience Replay

In this section we conduct experiments that reproduce the conditions used in recent work on option discovery. In particular, we compare standard Q-learning against an agent that, in the middle of learning, introduces options with policies obtained using Experience Replay. However, we also compare against an agent that simply applies Experience Replay without creating options, at the same point in the learning process.

Both agents that use Experience Replay save each experience $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$ at each time step. Instead of replicating the various subgoal discovery algorithms in past research, we give the option-learning agent the benefit of the doubt, and allow it access to the correct subgoal states, corresponding to the doorways between rooms, after 20 episodes of learning. Each subgoal thus defines an option that termintes only at that subgoal, has an initiation set that includes every other state, and whose policy is learned using Experience Replay.

We compared this method to Q-learning using Experience Replay only. For this agent, saved experience is simply played back in reverse order after 20 episodes to update the value function. The agent was allowed the same number of updates that were used to learn the four option policies, so it replayed the complete set of experiences four times.

Figure 2 compares the learning performance of the three agents. All three algorithms exhibit the same performance until episode 20, since they all use only standard Q-learning until then. The agent that defines options exhibits a marked improvement over the agent that simply continues to use Q-learning, but the agent that just applies Experience Replay exhibits the same improvement.

3.5 Dynamic Option Policies

Section 3.4 demonstrated that at least in one simple learning task, the benefits of introducing options defined using Experience Replay were subsumed by simply using Experience Replay alone. Many of the experiments in previous work intended to prove the effectiveness of various subgoal discovery algorithms may in fact have proven only the effectiveness of Experience Replay! This disconnect stems from the fact that introducing an option into an agent's action space conflates the contributions of at least two procedures: first, the identification of a subgoal that potentially decomposes the learning problem into subproblems, and second, the solution of one of those subproblems. This observation suggests that to isolate the benefit of using temporal abstraction, agents should be able to define subproblems without immediately solving them. Intuitively, we would like an agent to propose a subtask and then to learn how to perform that subtask concurrently with learning how to perform the overall task that may invoke the subtask.

To this end, we propose revising the option formalism to define an abstract action as a subproblem instead of as the solution to a subproblem. In the rest of this paper, we will use the term *subtask* to refer to a "partial option" $o = \langle I^o, A^o, G^o, \beta^o\rangle$. A subtask o thus corresponds to the following problem. From an initial state $s \in I^o$, select actions from A^o in such a way as to maximize the expected cumulative reward, given that a transition into state s' terminates the subtask with probability $\beta^o(s')$ and generates a "goal reward" of $G^o(s')$. A solution to a subtask defines an option, by specifying the option policy $\pi^o : S \to A^o$.

In general, an agent may learn the option policy by recursively applying RL to learn in the *subtask MDP* $M^o = \langle S, A^o, T^o, R^o\rangle$ where the state space S is inherited from the original MDP and the transition and reward functions are

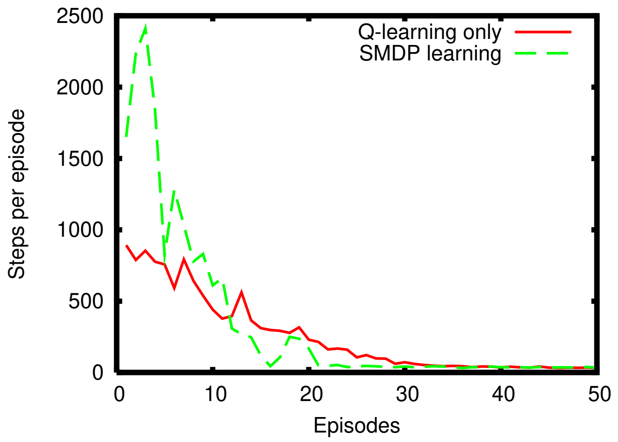

Figure 3: Performance of Q-learning and an algorithm that learns subtask policies in parallel with the overall task policy. Each learning curve is the average of 50 independent runs.

determined by the set of child actions A^o. Note that in general, the option may only be capable of visiting a subset of S, corresponding to the states reachable from I^o given the set of child actions A^o and the termination function β^o. The subtask may thus be thought of as an agent in its own right, attempting to learn only a part of the original problem. However, from the perspective of the agent that invokes the subtask, it is just an action whose behavior may change over time. This nonstationarity may complicate learning, but it permits the concurrent learning of subtasks with the overall task that must orchestrate those subtasks.

Figure 3 compares the performance of standard Q-learning against an algorithm that learns the option policies in parallel with the value function for the entire task. The option learner is given the four correct subtasks as prior knowledge, and it applies four instances of Q-learning in parallel with Algorithm 1 to learn the option policies. The results show a substantial decrease in performance for the agent that employs temporal abstraction, but perhaps surprisingly this poor performance does not seem to be a result of the evolving option policies. Instead, the agent exhibits some pathological initial exploratory behavior, as investigated in more detail in Section 4.1.

4. OPTIONS AND EXPLORATION

In this section, we will demonstrate than an important way in which options impact the performance of an RL agent is by affecting the agent's exploration behavior. When and where abstract actions become available can drastically change the structure of a learning problem.

4.1 The Initiation Set

From the perspective of evaluating the utility of temporal abstractions, prior work has also neglected to address the impact of when a discovered option becomes available during learning and how broadly applicable it is. For example, existing option discovery methods typically define the initiation set heuristically as the set of states historically visited a few time steps before the identified subgoal state [8, 10]. These methods then add the newly created options to the action space in the middle of learning the current task. Evaluations of temporal abstractions in this context are constrained in at least two ways. First, temporal abstraction is only applied when the value function is already partially learned. Second, the initiation set is in general only a subset of the possible states in which the option could reasonably be defined. These constraints have practical ramifications in situations when hierarchies are being learned in one task for use in a related task (in a transfer learning setting). Perhaps more importantly, they limit our theoretical understanding of the utility of temporal abstractions in general.

For an illustration of how these issues can lead to problems, consider the behavior of the subtask-learning agent whose performance is shown in Figure 3. It was given as prior knowledge four subtasks, corresponding to navigating to each of the four doorways from anywhere else in the environment. The addition of these four abstract actions drastically changes the apparent structure of the environment. Since the value function is initialized to 0 and the only nonzero reward occurs at the goal state, then the typical ϵ-greedy exploration policy will conduct a random walk until finding the goal at least once. However, every time the agent randomly selects one of the options, it will end up at one of the doorways, at least five steps from the goal. The agent cannot reach the goal until it randomly generates a sequence of actions that doesn't include any of the options but that does reach the goal, which becomes exponentially unlikely in the distance of the goal from the options' subgoal state.

To investigate this phenomenon, we compare four learning agents in Figure 4. The "Q-learning" agent performs standard Q-learning, as usual. The "options" agent is given four correct options at the very beginning of learning, including the optimal option policy for each subtask. These subtasks all include the entire state space (except for the one subgoal of each subtask) in their initiation sets. The "delayed options" agent receives the same four options, but only after the first 20 episodes of learning. Finally, the "limited options" agent receives options immediately, but the initiation sets of these options do not include any state in the room containing the goal state.

We see that the agent that has immediate access to the optimal option policies in all states performs the worst, for the reasons described above. The "delayed options" agent avoids the pathological exploration behavior by not invoking any options until it has a partially learned value function that allows it to learn not to select the options in states with higher value than the subgoal state. This agent performs identically to the Q-learning agent until after episode 20, at which point its performance rapidly improves to the optimal policy. Of course, this agent forfeits any possible benefit that temporal abstraction might provide in the initial stages of learning.

The "limited options" agent essentially uses prior knowledge to allow the options to execute only when navigating to a doorway might be helpful. Eliminating the state in the lower-right room from the initiation sets effectively prevents the agent from allowing an "earlier" subtask to interfere with completing the final steps of a solution. In a sense, existing option-discovery algorithms already apply this idea heuristically, by only adding to the initiation set the states experienced just before the candidate subgoal state.

Note that both of the agents that improve over Q-learning

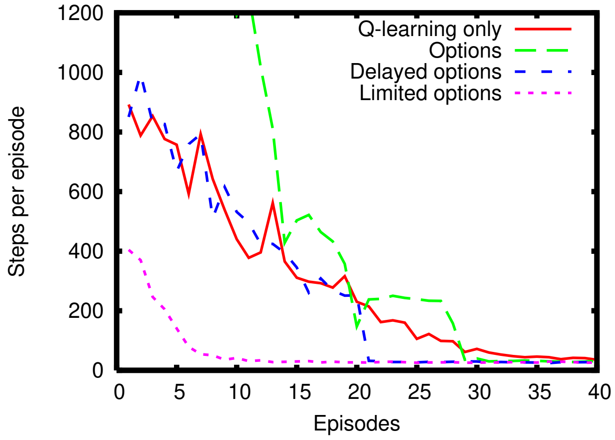

Figure 4: Comparison of learning agents with varying access to correct temporal abstractions. The Q-learning agent never uses options. The "options" agent gains immediate access to the correct options everywhere. The "delayed options" agent gains access to these options after 20 episodes. The "limited options" agent gains immediate access to these options except in the lower-right room. Each learning curve is the average of 50 independent runs.

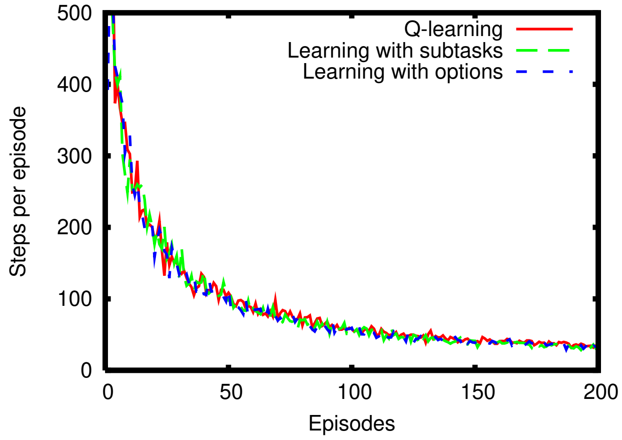

Figure 5: Performance of two learning agents that use optimistic initialization. The agents have identical performance, regardless of whether they use temporal abstraction. Learning curves are the average of 50 independent runs.

do so by artificially limiting the availability of the abstract actions. This phenomenon is nonintuitive to the extent that we can think of options as subroutines, which typically one would design to be as general as possible.

4.2 Optimistic Initialization

The previous section illustrated the perhaps nonintuitive possibility that introducing options might worsen learning performance, even when the agent retains direct access to all of the primitive actions. The reason is that the addition of such temporal abstractions changes the qualitative structure of the learning problem. The random exploration mechanisms employed in practice by most implementations of RL suffice in environments where the actions exhibit some degree of symmetry, so that random walks eventually reach every region of the state space. Introducing abstract actions that only terminate at subgoals biases random walks to states near those subgoals, since a single random selection of an option can erase the effort of several primitive actions that were carrying an agent away from the subgoal states.

One way to prevent options from interfering with the exploration behavior of an agent is to use an exploration mechanism that never relies on a random walk. A simple heuristic often used to encourage exploratory actions is optimism in the face of uncertainty, in which an agent assigns optimistic values to unfamiliar state-action pairs. This idea underlies the known finite-time convergence proofs for RL [2, 5, 12].

This heuristic can be applied to Q-learning simply by initializing the value function to some upper bound on the true optimal value function. This optimistic initialization causes most updates to the value function to decrease the value of state-action pairs just executed, making the agent more likely to select a different action the next time it revisits that state. This technique would prevent an agent from indefinitely returning to a subgoal state by driving the value of the corresponding option below the optimistic initial values of primitive actions that lead away from the subgoal.

Unfortunately, this technique also prevents the learning algorithm from continuing to select options when they are actually optimal! To converge to a stable optimal policy that selects an option in a certain state, the learning algorithm must first learn that every other primitive action (and option) has a lesser (or equal) value. Of course, at least one primitive action has the same value as the option: the action that the option's policy selects at that state.

Therefore, it's not clear that augmenting the action space of an agent with options can ever meaningfully impact learning performance. Regardless of the presence of options, the agent must execute every suboptimal state-action pair enough times to learn that they have smaller values than the optimal state-action pairs.

To demonstate this phenomenon, we conduct an experiment in which each agent uses optimistic initialization to probably approximately converge to the optimal policy in finite time. The first agent simply uses Q-learning, the second is given correct subtasks and learns the option policies dynamically, and the third is given optimal option policies as prior knowledge. To underscore our point, the last agent additionally receives as prior knowledge an option that navigates to the goal state optimally. Therefore, one optimal policy for this agent is simply always to select this option, in every single state. Nevertheless, we see in Figure 5 that all three agents exhibit the same learning performance.

4.3 Augmentation versus Abstraction

The preceding sections demonstrate that temporal abstractions benefit RL in only limited situations. Allowing the use of options in the wrong regions of the state space at the wrong stages of learning can lead to pathologically bad exploration. Much of the benefit when options do help might be more easily obtained by applying Experience Replay. The use of optimistic initialization, the only means for ensuring finite-time convergence to an optimal policy, completely precludes options from impacting learning performance at all.

Hauskrecht et al. consider another possible application of options: to abstract, rather than augment, the MDP [4]. Their work, in the context of planning given the MDP parameters, showed that planning with only an appropriate set of options can dramatically reduce the computational effort required to obtain an optimal policy. In contrast, they show that planning with both options and primitive actions can converge much more slowly than planning with primitive actions alone, given an optimistic initialization of the value function. These results do not translate directly to the RL context, where the emphasis is on sample complexity instead of computational complexity. For example, we have already seen that learning with options in addition to primitive actions does not hurt (or help) learning performance given optimistic value-function initialization.

One obvious danger in using only options to learn a task is the possibility that pruning away the primitive actions will remove the agent's ability to behave optimally. For example, an agent in our four-room gridworld that only selects from among the four subtasks that navigate to a doorway could only generate an optimal policy if the goal state were in one of the doorways. In the case of planning, analysis of the MDP can determine the goal state and create an appropriate option. For an RL agent, in the absence of prior knowledge about the goal state, the primitive actions must be available, at least in the vicinity of the goal. In our example domain, the agent only needs access to the primitive actions in the bottom-right room.

Figure 6 compares two agents that ignore primitive actions outside the bottom-right room against a standard Q-learning agent. One of these two agents uses only options until it reaches the last room, after which it uses both options and primitive actions (unless it leaves the room without reaching the goal). The other agent only uses primitive actions in the last room. We see that the agent that uses options in the last room exhibits the same pathological exploration behavior as the "options" agent in Figure 4: trying any of the options with an exploratory action takes the agent at least five steps away from the goal.

The agent that only uses primitive actions in the last room learns very quickly. Note that after executing any option, it can be in only one of four states. After executing a primitive action, it can be in only one of 24 states, since it only executes these actions in the bottom-right room (or its two doorways). As a result, this agent learns in a much smaller state space than the 104-state space explored by the agents that may always execute primitive actions.

This abstracted agent learns this simple task about as efficiently as the "limited options" agent in Figure 4. The primary difference between the two agents is that the options produce a "soft" bias in the "limited options" agent towards the subgoal states (except the in the bottom-right room). The abstracted agent uses the options to enforce a hard bias, so that it never needs to explore any actions in any states outside of the doorway states and the last room. A key theoretical benefit of this hard bias is that it allows an abstracted agent to employ techniques such as optimistic initialization without wiping out the benefit of temporal abstraction. Since the agent cannot explore from the states that were abstracted away, it can avoid a needlessly thorough exploration of every primitive action from every state.

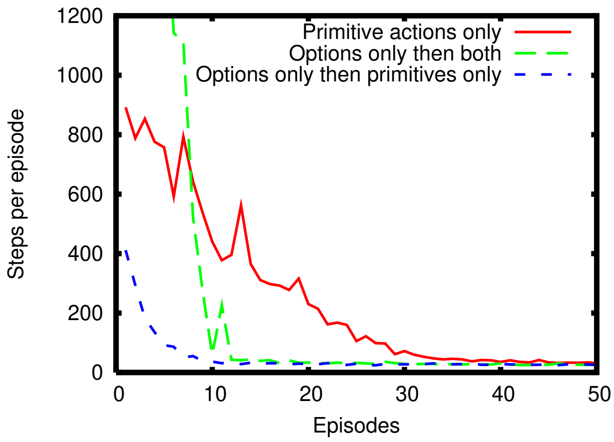

Figure 6: Performance of learning agents that remove primitive actions.

5. DISCUSSION

The benefit of employing temporal abstraction in RL depends on a surprising number of factors. These factors include the precise learning algorithm used, when and where the abstraction becomes available, and the availability of the primitive actions or other abstract actions. When simply augmenting an action space that includes primitive actions, temporal abstractions cannot hurt the asymptotic performance of an RL algorithm, but they are only likely to improve learning speed by reducing the number of backups to converge the value function. In algorithms that employ the theoretically motivated principle of optimism in the face of uncertainty, they cannot improve sample complexity; in other algorithms they may actually impede learning.

One way to construe the introduction of options into an agent's action space is as the introduction of bias in its random exploration. Random actions become much more likely to bring the agent to one of the indicated subgoal states. Introducing this bias too early in learning can prevent the agent from reaching the goal, which is the necessary first step in backing up positive values throughout the state space. Only later in learning can the options benefit, by backing value across multiple states at once and by biasing learning to important parts of the state space. Optimistic initialization, by removing much of the random element from exploration, eliminates any bias, beneficial or otherwise. By using options to replace primitive actions instead of to augment them, an agent can constrain exploration, including the effects of optimistic initialization.

This result highlights the difficulty in discovering truly useful options automatically. The agents described in Section 4.3 benefited from options because these options prevented them from ever thoroughly exploring three of the four rooms in the domain. How could an agent in this environment discover this temporal abstraction without first spending effort to explore those three rooms, thus negating the possible benefit? One possible answer is that a useful abstract action must somehow generalize from past experience. An agent that fully explores several empty rooms may soon learn just to pass through empty rooms in the future. By introducing an approriate abstract action, the agent dynamically adapts its generalization. This idea suggests an important connection between temporal abstraction and state abstraction or function approximation.

Another consequence of these results for algorithms that discover hierarchy is that candidate abstract actions may not always be evaluated in isolation. The benefit of adding one action may depend on the agent's ability to consequently remove others. This behavior depends on the action hierarchy as a whole, which determines the contexts in which each action executes and the possible state-action pairs that must be explored to learn each option's policy.

An important direction for future research is to investigate how temporal abstraction affects the computational complexity of RL. The fact that Experience Replay can subsume the benefits of using options in certain cases suggests that reducing the number of value function updates is one way in which options can help, but the work of Hauskrecht et al. indicates that options can actually increase the amount of computation required. More research is needed in the area of model-based algorithms, which more explicitly separate the gathering of data (exploration) from the computation of the value function from that data [5]. In particular, given the interaction between temporal abstraction and optimistic initialization, one open question is how to incorporate options into algorithms that currently offer finite-time convergence guarantees.

6. CONCLUSION

We investigated the utility of temporal abstraction using the standard options framework. Even in a simple environment designed to illustrate the effectiveness of options, the augmentation of the action space with options can easily worsen learning performance by leading to pathological exploration behavior. Using common heuristics that eliminate this pathological behavior also eliminates any potential benefit of using options. Finally, since an option is formally defined as the solution to a subtask, the benefit of using options is easily conflated with the benefit of the technique used to obtain the option policy. Nevertheless, temporal abstraction can help learning performance by biasing or constraining exploration. Future work in discovering options should be aware of how an individual abstract action interacts with other actions to shape these biases or constraints.

7. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 0237699 and the DARPA Bootstrap Learning program.

8. REFERENCES

[1] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete-Event Systems*, 13:41–77, 2003. Special Issue on Reinforcement Learning.

[2] R. I. Brafman and M. Tennenholtz. R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

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

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

[5] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 260–268, 1998.

[6] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning, and teaching. *Machine Learning*, 8:293–321, 1992.

[7] S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 560–567, 2004.

[8] A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 361–368, 2001.

[9] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[10] Ö. Şimşek and A. G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 751–758, 2004.

[11] S. Singh, A. G. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems 17*, 2005.

[12] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman. PAC model-free reinforcement learning. In *Proceedings of the Twenty-Third International Conference on Machine Learning*, pages 881–888, 2006.

[13] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

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

[15] C. Watkins. *Learning From Delayed Rewards*. PhD thesis, University of Cambridge, 1989.