Model-based Hierarchical Average-reward Reinforcement Learning

Sandeep Seri SERI@CS.ORST.EDU
Prasad Tadepalli TADEPALL@CS.ORST.EDU
Department of Computer Science, Oregon State University, Corvallis, OR 97331 USA

Abstract

There is a growing interest in using task hierarchies to tame the complexity of reinforcement learning. In this paper, we extend the MAXQ framework to hierarchical Average-reward Reinforcement Learning. We motivate and introduce the notions of recursive gain-optimality and hierarchical gain-optimality and show that these two coincide when a condition called *Result Distribution Invariance* holds. We present two model-based algorithms, *HH-learning* and *HAH-learning*, that use a predefined task hierarchy and abstraction functions, and learn recursively gain-optimal policies. HH-learning can be used with any exploration policy, whereas HAH-learning has a built-in "auto-exploratory" feature. We present empirical results that show that HH-learning converges in fewer steps than the corresponding "flat" algorithm and scales better with certain aspects of the domain size. The results on HAH-learning also show that it is more effective in exploring the state space than HH-learning using ϵ-greedy exploration.

1. Introduction

Hierarchies are widely used in a variety of settings from computer programs to complex organizations. Presumably, one reason for their popularity is the division of cognitive and computational labor that they facilitate. Hence it is not surprising that there is growing interest in task hierarchies as a way to tame the complexity of reinforcement learning (Dietterich, 2000; Ghavamzadeh & Mahadevan, 2001; Moore et al., 1999; Parr & Russell, 1998; Sutton et al., 1999). Indeed, in many domains, task hierarchies arise naturally when we formulate the problem according to how we humans tend to think about it.

Most previous work in hierarchical reinforcement learning, including the original MAXQ work, is in the discounted or the undiscounted finite horizon settings. In this paper we are interested in undiscounted infinite horizon case, where the goal is to optimize the average reward received per time step or the "gain." This is a natural criterion to optimize in most recurrent domains, including many problems in industrial engineering and operations research (Mahadevan, 1996). We introduce the Hierarchical Average-reward Reinforcement Learning (HARL) framework, which is an adaptation of MAXQ-framework to average-reward reinforcement learning (ARL). Like MAXQ, HARL assumes a predefined task hierarchy with the goal of finding an optimal policy for choosing appropriate subtasks for doing each task. As is typical in reinforcement learning, the policy is learned indirectly by learning a value function. As in the MAXQ-framework, we have task-specific value functions over predefined abstracted state spaces, facilitating quicker convergence.

We clarify and adapt the notions of *hierarchically optimal* and *recursively optimal* policies to the average reward setting, and show that these two coincide when the task hierarchy and the domain exhibit a property called *Result Distribution Invariance*. Intuitively, this holds when the result of each subtask is independent of how the subtask was achieved. We introduce an algorithm called Hierarchical H-learning (*HH-learning*), which learns action models and task-specific state-value functions that together determine a recursively optimal policy.

Effective exploration is an important issue for reinforcement learning. Tadepalli and Ok (1998) present an algorithm called AH-learning that uses the principle of "optimism under uncertainty" and automatically explores the state space while always taking greedy actions with respect to its current value function. We present a new algorithm called HAH-learning, which extends AH-learning to hierarchical setting and improves it by having it automatically readjust itself when more exploration is needed. The algorithm requires the user to supply a tolerable lower bound and an upper bound on the average reward, and finds a policy whose gain is between the two bounds. Thus, it can be used for “satisficing” rather than “optimizing,” which may sometimes be all that is needed.

We compare the performances HH-learning and H-learning in the task of scheduling a simulated Automated Guided Vehicle (AGV) in a number of different settings. We show that HH-learning converges in fewer steps and its performance scales better as the size of the problem is increased. We also show that HAH-learning significantly outperforms HH-learning with ϵ-greedy exploration strategy in tasks where effective exploration is critical and challenging.

The rest of the paper is organized as follows: Section 2 gives the background on ARL. Section 3 describes the HARL framework. Section 4 describes HH-learning with some empirical results. Section 5 motivates and describes HAH-learning. Section 6 concludes with a brief discussion of the future work.

2. Average-reward Reinforcement Learning

Semi-Markov Decision Problems (SMDP) generalize Markov Decision Problems (MDP) to temporally extended actions. An SMDP is described by a set of states S, actions $A(s)$ available in state s, a reward function $r^a(s)$ which is the expected immediate reward for executing action a in state s, a probability transition matrix $P(s'|s,a)$ that gives the probability of reaching a state s' upon executing action a in state s, and a transition time function $t^a(s)$ which is the expected completion time for action a when executed in state s. A policy is a mapping from states to actions. The agent chooses a new action at each decision epoch. Let R_j be the reward received during the j^{th} action and T_j be the time taken, when the agent started in state s and followed the policy π. The average reward or *gain* of π with respect to the starting state s is defined as:

$$\rho^\pi(s) = \lim_{N\to\infty} \frac{E(\sum_j^N R_j)}{E(\sum_j^N T_j)} \tag{1}$$

In the presence of sufficient exploration, the gain of a policy is independent of the starting state and is denoted by ρ^π. Given an SMDP, and a policy π with a gain ρ^π, we can define *relativized reward* of executing action a in state s as $r^a(s) - t^a(s)\rho^\pi$, which represents the expected “excess reward” over what is expected in the duration of the action on average. The limit (or the Cesaro's limit in the case of periodic SMDPs) of the total expected relativized reward starting from any state s and following policy π is called its bias and denoted by $h^\pi(s)$.

$$h^\pi(s) = \lim_{N\to\infty} E(\sum_j^N (R_j - \rho^\pi T_j)) \tag{2}$$

Intuitively, bias is the relative advantage (or disadvantage) to being in a given state over the long run. $h(s)$ is the bias of state s with respect to the optimal average reward policy or gain-optimal policy. The following theorem gives the Bellman equation for average reward SMDPs (Puterman, 1994).

Theorem 1 *For unichain SMDPs, there exist a scalar ρ and a real-valued function h that satisfy:*

$$h(s) = \max_{a\in A(s)} [r^a(s) - \rho t^a(s) + \sum_{s'\in S} P(s'|s,a)h(s')] \tag{3}$$

Further, the greedy policy π^*, which selects actions that maximize the r.h.s. of Equation (3) attains the optimal average reward $\rho = \rho^{\pi^*} \geq \rho^\pi$ over all policies π.

The H-learning algorithm of (Tadepalli & Ok, 1998), adapted to SMDPs, works as follows. Whenever an action a is executed in state s, the next state s', the time taken, and any immediate reward received are noted, and are used to estimate $P(s'|s,a)$, $t^a(s)$ and $r^a(s)$, respectively. Additionally, the immediate reward is also used to incrementally revise ρ, the estimated average reward of the current greedy policy. Finally, the h value of the current state is updated by Equation 3 using the current estimates of $P(s'|s,a)$, $t^a(s)$, $r^a(s)$, and ρ as the true values. Actions are chosen using a combination of greedy and exploratory policies. While H-learning has not been formally proved to converge[1], it was found to consistently converge to the optimal solution in every case it was tried and is usually faster than the corresponding model-free method, namely R-learning (Schwartz, 1993; Tadepalli & Ok, 1998).

3. HARL Framework

Recently there have been several proposals to exploit hierarchical task structure in reinforcement learning, including options (Sutton et al., 1999), hierarchical abstract machines or hams (Parr & Russell, 1998), and the MAXQ framework (Dietterich, 2000). Our work is most closely related to the MAXQ framework, but differs in details. To distinguish the two, we call ours the *Hierarchical Average-reward Reinforcement Learning* or HARL-framework.

[1] Although some progress has been made in proving it, technical problems remain due to simultaneous update of ρ and h.

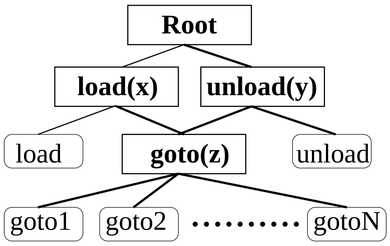

Figure 1. Task Hierarchy for the AGV Domain

Just as in the MAXQ-framework, we have a predefined task hierarchy. A task i is performed by invoking its immediate children in the hierarchy, that correspond to its subtasks and denoted by A^i. Each subtask i has a goal predicate G^i. We assume that we are in a recurrent task, so the root task has a goal condition "false" and it never terminates. All other tasks have some non-trivial goal predicates. When a subtask is invoked it runs until it reaches its goal. The lowest level subtasks are primitive actions that can be executed directly and terminate immediately. We also assume that each task i is associated with a predefined abstraction function B^i, which abstracts the state space by ignoring irrelevant state variables. Hence, each task i has an associated 3-tuple $\langle G^i, A^i, B^i \rangle$. A policy π^j at node j maps states to subtasks of j. Π^j denotes a *hierarchical policy* at node j, and defines a policy at j and each of its descendents. We use Π as a shorthand for Π^{root}. The goal of learning in the hierarchical average-reward RL setting is to find a policy Π that has the maximum gain among all hierarchical policies. This is called a *hierarchically gain-optimal policy*.

We use the problem of scheduling a simulated Automated Guided Vehicle (AGV) to illustrate our framework and present experimental results. AGVs transport parts between different machines on factory floors. We use the task decomposition in Figure 1 for all our domains. The tasks and primitive actions shown have the following meaning and goal predicates: The overall task (*Root*) is to deliver parts from producing machines to destination stations and it never terminates. The *load(x)* (*unload(x)*) task is to pick up (deliver) a load from (to) machine x and it terminates when the AGV is at location x and has (doesn't have) a load. The *goto(x)* is to move the AGV to machine x and it terminates when AGV is at x. The *load* and *unload* primitive actions pick up and deliver a load, respectively. The *goto*$\{i\}$ primitive action moves the AGV to location i.

The task hierarchy in Figure 1 should be seen as an abstraction of the real hierarchy, where the variable x is instantiated separately for every machine. Notice that the subtask *goto(x)* is shared by both *load(x)* and *unload(x)*. This allows the hierarchical learning agent to learn more from the same experience. Another advantage of the hierarchical decomposition is that the value functions for subtasks depend on fewer state variables, thus allowing strong abstraction. In the AGV domain, we will have state variables for the AGV location, the destination of the load on the AGV, and one variable for each of the loads waiting at the producing machines. Abstraction functions allows us to ignore irrelevant variables. For example, for the *goto(x)* task, the abstraction function B ignores all variables in the state other than the AGV location. For the load (and unload) task, the abstraction function ignores all variables other than the type (destination) of the load on the AGV, and the machine location the AGV is at. This encodes the knowledge that the only thing relevant about the location for the load action to succeed is whether the AGV is at the correct machine.

In general, hierarchically optimal policies at any node depend on the policies used at all other nodes. For example, Dietterich (2000) describes a subtask of exiting a room with two doors, where each door is closest to a different higher level goal location. The optimal choice of which door to use to exit the room depends on which higher level goal location one is trying to reach. Unfortunately, this makes optimal policies for the subtasks context-dependent, i.e., dependent on the policies at higher level nodes. To keep the optimal policies context-free, Dietterich (2000) defined a *recursively optimal policy* as an "optimal" policy for a task assuming that all its subtasks in turn use recursively optimal policies. But this leaves open the question of what optimality criterion should be used for each subtask in the average-reward RL setting so that, at least in some cases, recursive optimality is identical to hierarchical gain-optimality or closely approximates it. Since the subtasks have only finite horizons, one possibility is to simply optimize their total rewards. Another possibility, pursued in (Ghavamzadeh & Mahadevan, 2001), is to treat them also as average reward problems and optimize their reward rate (total expected reward / total expected time). Unfortunately, both of these approaches fail to find the hierarchically gain-optimal policy even in very simple MDPs, as we show below using a 4-state MDP (each action takes 1 time step).

The MDP in Figure 2(a) has an optimal average reward of 7 using the action b. The boxed part of the MDP is the only non-primitive subtask. The only action choice is at the initial state of this subtask. Interestingly, optimizing the subtask using either total reward or average reward criterion results in choosing action a over action b. The unfortunate conse-

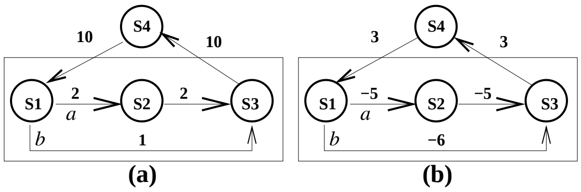

Figure 2. A 4-state MDP: to illustrate different criteria to optimize the subtask

quence of this is that now the subtask takes 2 units of time, which would contribute to a suboptimal average reward of 6 for the overall task. The reason for this paradoxical result is that the times for actions are not "charged" correctly. The time is not charged at all in total reward optimization, and it is charged incorrectly when using the average reward optimization. The obvious solution is to charge time at the rate of the current optimal gain ρ. Part (b) of the above figure shows the same MDP with the immediate rewards relativized by the optimal gain, which is 7. When optimized by the total relativized reward expected during a subtask, the subtask picks action b which is also optimal according to the overall task. Thus, we define a policy as *recursively gain-optimal* if the policy at each subtask i maximizes its expected total relativized reward with respect to the gain of the overall policy for each state s, denoted by $h^i(s)$, assuming that all its subtasks in turn use recursively gain-optimal policies. Note that $h^{Root}(s)$ is the same as $h(s)$ for the root task as defined in Definition 2. The value functions for the recursively gain-optimal policy must satisfy the following Bellman equations.

$$h^i(s) = \begin{cases} r^i(s) - \rho t^i(s) & \text{if } i \text{ is a primitive action} \\ 0 & \text{if } s \text{ is a goal state for subtask } i \\ \max_{a \in A^i(s)} \{h^a(B^a(s)) + \sum_{s' \in S} P^i(s'|s,a) h^i(s')\}, & \\ & \text{otherwise} \end{cases} \quad (4)$$

Recall that A^i is the set of subtasks of i and $B^a(s)$ is an abstraction of state s and is assumed to be "safe" in the sense that $h^a(B^a(s)) = h^a(s)$. Finally $P^i(s'|s,a)$ denotes the probability of reaching state s' after executing task a from state s. Note that all subtasks in the HARL hierarchy except the root tasks have goal predicates and hence are episodic. Hence the solution for the above equations optimize their expected total relativized reward, assuming that all their subtasks are recursively optimal. This is so because the first term of the last part of the equation, $h^a(B^a(s))$, denotes the expected total relativized reward during the subtask a, and the second term denotes the expected total relativized reward from then on until the completion of task i. The root task is non-episodic, and it should satisfy Equation 3, when the actions a are interpreted as its highest level subtasks. But since $r^a(s) - \rho t^a(s) = h^a(s) = h^a(B^a(s))$ when the subtasks are recursively gain-optimal, its solution also satisfies the above equation.

The algorithm we present in the next section is intended to find a recursively gain-optimal policy. As we will see shortly, when the following condition holds, it is also hierarchically gain-optimal.

Result Distribution Invariance (RDI) For all tasks i and state s, the distribution of states reached after the execution of any subtask (j) of i is independent of the hierarchical policy (Π^j) of that subtask, i.e.,

$$P^i(s'|s,j,\Pi^j) = P^i(s'|s,j) \quad (5)$$

Note that RDI holds for all tasks in the AGV domain. For example, upon executing the *goto(x)* subtask, in a given state s, the result state s' is the same irrespective of the policy used to execute the subtask. Also, upon executing the *load(x)* subtask the AGV location changes to x, its load is whatever the part at machine x at the start of the subtask was, and the new part at machine x is generated according to a probability distribution that is independent of the policy used for *load(x)*. Thus, RDI holds for the *load(x)* subtask.

Theorem 2 *When the result distribution invariance condition holds, a policy is hierarchically gain-optimal iff it is also recursively gain-optimal.*

Proof Sketch: Assume that a policy Π is hierarchically gain-optimal but not recursively gain-optimal. Hence there is a deepest level subtask j and state s such that $\pi^j(s)$ is not recursively gain-optimal. Replacing $\pi^j(s)$ with j's recursively gain-optimal subtask increases $h^j(s)$. Since RDI holds, the distribution of states at which the parent task of j, say i, makes a decision remains unchanged and the reward obtained by one or more of its subtasks increases. This in turn increases $h^i(s)$ for some s without decreasing $h^i(s')$ for any s'. We can continue this argument until the children of the root node. Since RDI also holds for the root node, and the h-values of at least one of the subtasks is increased, the gain of the new policy would increase. This violates the assumption that Π is hierarchically gain-optimal. We can similarly prove that a policy cannot be recursively gain-optimal without also being hierarchically gain-optimal. □

A complete proof is given in Seri (2002).

4. Hierarchical H-Learning

In this section, we present the HH-learning algorithm and describe some experimental results.

4.1 Algorithm

In the algorithm of Figure 3, the h values of states are updated using the right hand sides of Equation 4 in the fashion of value iteration (lines 13, 26 and 28). HH-learning estimates the $P^i(s'|s,a)$ for subtasks (lines 21–24 in Figure 3), and $r^i(s)$ and $t^i(s)$ for primitive actions (lines 3–6) from on-line experience. It uses these values to select a subtask that maximizes the right hand side of Equation 4 or an exploratory action (lines 16–17). We used the ϵ-greedy exploration strategy in our experiments, where we select an exploratory subtask at random a fixed percentage (we use 8% for all our experiments) of the time and a greedy subtask the rest of the time. The chosen subtask a is then executed recursively. When a subtask finishes, satisfying its goal condition, execution control returns to its parent task. The parent task updates its domain models and H-value (lines 19–26) and selects the next subtask. When a primitive action is selected, it executes immediately, updates its reward and time models (lines 2–6) and its H-value (line 13). The average reward ρ is estimated from the rewards received during the primitive actions, not counting steps which are part of executing a non-greedy subtask at some level in the hierarchy (lines 7–11). All values are initialized to zero, except α and $AverageTime$, which are initialized to 1. The initial call is to HH(s, $root$).

4.2 Experimental Results

We compared HH-learning to H-learning in a number of AGV domains. A small AGV domain is depicted in Figure 4. In this domain there are two machines that produce parts to be delivered to one of two destination stations. The AGV can carry one load at a time. It gets a reward K when a part is delivered to destination D1 and a reward 1 when a part is delivered to destination D2. A part produced at either machine is destined for one of the destination stations. The probability of part produced being destined for D1 is p for machine M1 and q for machine M2. The amount of time the AGV takes to move between adjacent nodes varies exponentially with the means shown above the edges in Figure 4. A move to an adjacent node succeeds with probability u and the rest of the time the AGV consumes one time unit and stays at the same location. The AGV takes one time step to load or unload a part. The state of the learning agent at any time can be given by specifying the AGV location, part on the AGV and the parts waiting at the machines.

procedure HH(state s, task i)
1: **if** (i is a primitive)
2: Execute i. Observe the reward r and elapsed time t.
3: //Update the model for primitive action i
4: $n^i(s) \leftarrow n^i(s) + 1$
5: $r^i(s) \leftarrow r^i(s) + (r - r^i(s))/n^i(s)$
6: $t^i(s) \leftarrow t^i(s) + (t - t^i(s))/n^i(s)$
7: **if** i and all its ancestors are a greedy actions
8: $AverageReward \leftarrow (1-\alpha)AverageReward + \alpha r^i(s)$
9: $AverageTime \leftarrow (1-\alpha)AverageTime + \alpha t^i(s)$
10: $\rho \leftarrow AverageReward/AverageTime$
11: $\alpha \leftarrow \alpha/(\alpha+1)$
12: //Update H-value for primitive action i
13: $h^i(s) \leftarrow r^i(s) - \rho t^i(s)$
14: **else**
15: **while** (**not** $G^i(s)$) //while task i has not finished
16: Choose a, an exploratory subtask or a greedy one with
17: $\arg\max_{a \in A^i(s)}\{h^a(B^a(s)) + \sum_{s' \in S} p^i(s,a,s')h^i(s')\}$
18: HH($B^a(s)$, a) //Execute the sub-task
19: Observe the state s'.
20: $s' \leftarrow B^i(s')$ //Abstract the state s'
21: //Update the model for task i
22: $n^i(s,a) \leftarrow n^i(s,a) + 1$
23: $n^i(s,a,s') \leftarrow n^i(s,a,s') + 1$
24: $p^i(s,a,s') \leftarrow n^i(s,a,s')/n^i(s,a)$
25: //Update H-value for task i
26: $h^i(s) \leftarrow \max_{a \in A^i(s)}[h^a(B^i(s)) + \sum_{s' \in S} p^i(s,a,s')h^i(s')]$
27: $s \leftarrow s'$
28: $h^i(s) \leftarrow 0$ //Set H-value for goal state s to zero

Figure 3. The HH-learning algorithm.

In the first experiment, we compared the performance of H-learning with HH-learning in the AGV domain depicted in Figure 4 and described above. For this experiment, the domain parameters, K, p, q and u are 5, 0.5, 0.0 and 0.9 respectively. Figure 5 shows the average on-line reward for 30 trials. In this domain HH-learning performs slightly better than H-learning.

In the next experiment, we tested how the two methods perform when the domain size is increased. We created two domains, one roughly twice the size of the other, depicted in Figure 6. The domain parameters

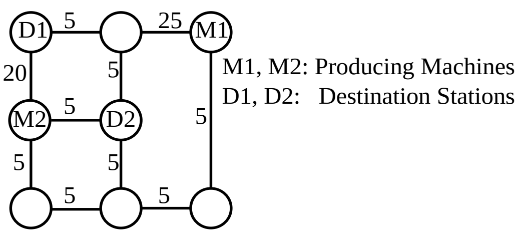

Figure 4. Stochastic SMDP Domain

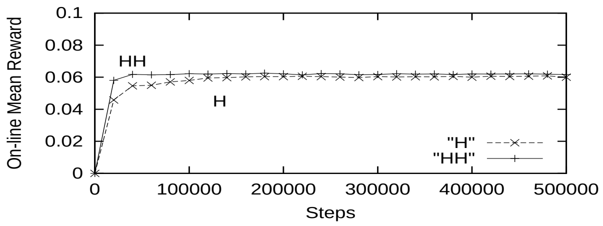

Figure 5. H Vs. HH in the Stochastic SMDP domain

in both domains, K, p, q and u are 5, 0.5, 0.0 and 1.0 respectively. The travel time between adjacent cells is always one time unit. Figure 7 shows the mean on-line reward over 30 trials for the 4X2 domain. Here, HH-learning converges faster to an optimal policy compared to H-learning.

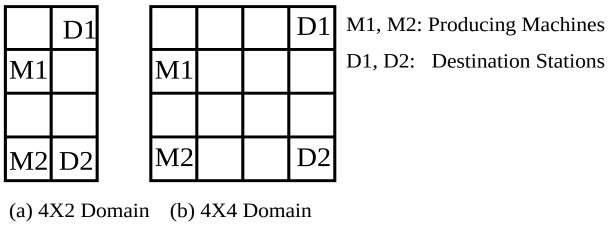

Figure 6. Two AGV Domains: to show the effect of scaling on H and HH

Figure 8 shows the mean on-line reward over 30 trials for the 4X4 domain. In this domain, H-learning takes roughly twice as many steps to converge to an optimal policy compared to the 4X2 domain. On the other hand, HH-learning takes about the same time. The results of this experiment show that HH-learning scales much better than H-learning.

4.3 Analysis

The reason why HH-learning is effective in the AGV domains can be understood using a crude estimation of the sizes of the flat and hierarchical spaces. If m is the number of machines and l is the number of locations, the size of the flat state space is $O(l(m+1)^{m+1})$. The hierarchy divides this space into the top level where the scheduling decisions are made (when the AGV is located near one of the machines) which is of size $O(m(m+1)^{m+1})$, and the navigational level, which is of size $O(lm)$. Comparing $O(l(m+1)^{m+1})$ with $O(m(m+1)^{m+1} + lm)$, we can predict that the speedup of the hierarchical learner increases with the ratio l/m. We expect large speed-ups when the number of possible locations is much higher than the number of machines.

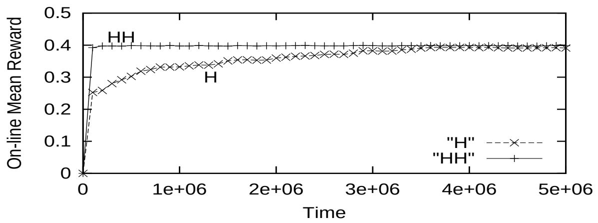

Figure 7. H Vs. HH in the 4X2 domain.

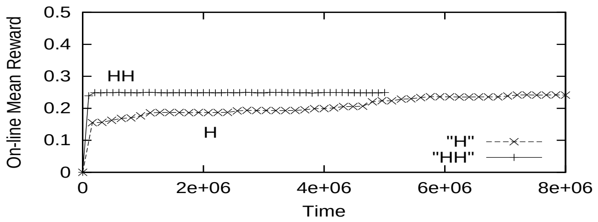

Figure 8. H Vs HH in the 4X4 domain.

5. Hierarchical AH-Learning

In this section, we present the HAH-learning algorithm and describe some experimental results.

5.1 Algorithm

Effective exploration of the state space is an important issue for reinforcement learning. Exploration methods that employ "optimism under uncertainty" are shown to be much more efficient than random exploration in many cases (Kaelbling et al., 1996; Koenig & Simmons, 1996). Tadepalli and Ok (1998) describe an algorithm called AH-learning that is based on this principle, and automatically explores the unexplored parts of the state space while always following the greedy policy.

AH-learning differs from H-learning in two ways: First, the value of ρ is initialized to a high initial value. Since ρ is subtracted from the h-value of a state whenever it is visited, this decreases the values of the states which are visited frequently, thus encouraging the system to visit unexplored states. To also encourage the system to take actions that are not taken frequently, instead of storing the state-value function h, AH-learning stores a state-action value function called R which is similar to the Q-function (Schwartz, 1993).

These ideas directly transfer to HH-learning in a straight-forward way. HAH-learning stores R-values for state action pairs instead of h-values. $R^i(s, a)$ represents the bias resulting from executing action a in state s and from then on following the optimal policy until the end of task i. The R-value of a state/subtask pair is updated using the following equation:

$$R^i(s,a) = h^a(B^a(s)) + \sum_{s' \in S} P^i(s'|s,a) h^i(B^i(s'))$$

$$h^i(s) = \begin{cases} r^i(s) - \rho t^i(s) & \text{if } i \text{ is a primitive action} \\ 0 & \text{if } s \text{ is a goal state for } i \\ \max_{a \in A^i(s)} R^i(s,a) & \text{otherwise} \end{cases}$$

Unfortunately, AH-learning requires the initial value of ρ and ρ's learning rate α to be tuned for best performance. In general, it is very difficult to tune these, because a high value of ρ may be needed for exploration, but too high a value wastes resources. Similarly too low a value of α makes convergence slower, and too high an α may result in suboptimal policies. On the other hand, we would generally have some idea about the maximum average reward we expect (ρ^{max}) and the minimum we would be satisfied with (ρ^{min}). For example, in the AGV domain we could estimate ρ^{max} from the physical constraints of the system and ρ^{min} from the performance of other scheduling methods.

Hence, instead of tuning ρ and α by trial and error, we set the initial value of ρ to ρ^{max}. α is initialized to one, as in HH-learning. ρ is normally updated as in HH-learning. Whenever the current value of ρ falls below ρ^{min} the algorithm sets $AverageReward$ to $\rho^{max} * AverageTime$, effectively setting ρ to ρ^{max}. The result of this change is that whenever the average reward of the current policy falls below our minimum expectations, the algorithm is forced to search again for a better policy. Thus HAH-learning tries to find the best policy with an average reward between ρ^{max} and ρ^{min}.

5.2 Experimental Results

In order to compare HH-learning with HAH-learning, we created two domains. In the "maze" domain, depicted in Figure 9, the AGV can move to all adjacent cells only in the row labeled "Corridor". In all other locations, the AGV can only move to the cell above and below the current cell. In the "grid" domain (not pictured), the AGV can move to all adjacent cells from any cell, not just in the corridor. The size of the domain and the placement of machines and destination stations are identical. The domain parameters in both domains, K, p, q and u are 50, 0.5, 0.0 and 1.0 respectively.

The key difference between the two domains is the time a random walk will take to get to a particular location. We expect this time to vary more slowly with the distance to the goal in the grid domain than in the maze domain; since, along the corridor cells, in the maze domain, it is 3 times more likely to take actions that lead away from the goal than toward the goal (Whitehead, 1991). Hence, we expect exploration to be more critical in the maze domain than in the grid domain.

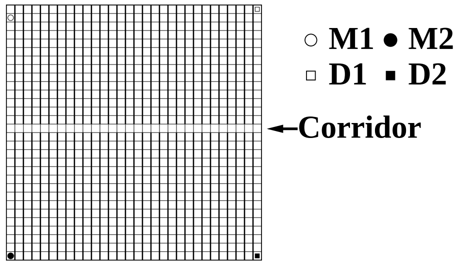

Figure 9. Maze Domain: The task of the AGV is to deliver parts from M1 and M2 to D1 and D2.

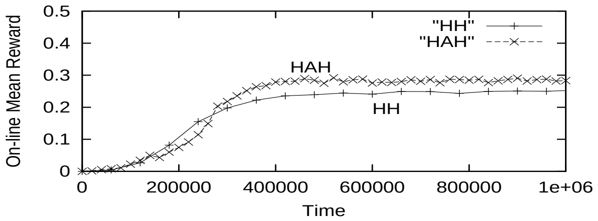

Figure 10. HH Vs. HAH in the Grid Domain

Figure 10 gives the average on-line reward for 30 trials in the grid domain. As expected, the performances of the two algorithms are comparable. In the maze domain, where exploration is critical, HAH-learning converges to the optimal policy much faster than HH-learning (Figure 11), which uses ϵ-greedy exploration.

6. Conclusion

In this paper, we explored a new framework for hierarchical average reward reinforcement learning. We refined the notions of recursive and hierarchical optimality and showed that they are equivalent when the Result Distribution Invariance holds. This result also applies to undiscounted total reward optimization, but does not apply to discounted learning. RDI is a constraint on the task hierarchy as well as the underlying MDP. For example, Dietterich's two-door task does not satisfy the RDI, but a hierarchy with two subtasks, one for each door does. Hence our algorithms can be made to learn a hierarchically gain-optimal policy in this domain using an appropriate task hierarchy.

We presented two learning algorithms HH-learning and HAH-learning and showed their effectiveness in several simple AGV domains. These algorithms scale well with the physical size of the domain and find

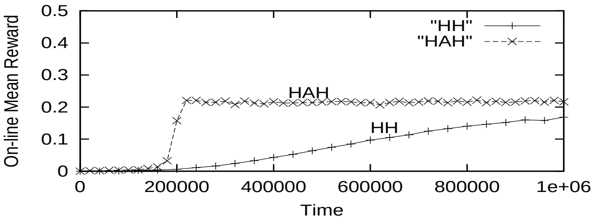

Figure 11. HH Vs. HAH in the Maze domain

recursively gain-optimal policies. HAH-learning is also quite effective in domains where good exploration strategy is critical for performance, and eliminates the need for careful tuning of parameters. While model-free versions of our algorithms are easier to implement, they are slower to converge than the corresponding model-based methods (Seri, 2002). Moreover, they cannot be used for prediction and planning, unlike the model-based methods. Combining hierarchical learning with function approximation and factored action models is an important area for future research. Deriving abstractions automatically is another fundamental problem that needs to be addressed. Some recent work in this direction is reported in (Jonsson & Barto, 2001; McGovern & Barto, 2001; Tadepalli & Dietterich, 1997)

7. Acknowledgements

We gratefully acknowledge the support of NSF under grant number IIS-0098050. We would also like to thank Tom Dietterich, Mohammad Ghavamzadeh, Sridhar Mahadevan, and the anonymous reviewers for their many useful comments.

References

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *9*, 227–303.

Ghavamzadeh, M., & Mahadevan, S. (2001). Continuous-time hierarchical reinforcement learning. *Proceedings of the Eighteenth International Conference on Machine Learning,*. San Mateo, CA: Morgan Kaufmann.

Jonsson, A., & Barto, A. G. (2001). Automated state abstraction for options using the U-Tree algorithm. *Advances in Neural Information Processing Systems 13* (pp. 1054–1060). Cambridge, MA: MIT Press.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Koenig, S., & Simmons, R. G. (1996). The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning*, *22*, 227–250.

Mahadevan, S. (1996). Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, *22*, 159–195.

McGovern, A., & Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. *Proceedings of the 18th International Conference on Machine Learning,* (pp. 361–368). San Mateo, CA: Morgan Kaufmann.

Moore, A. W., Baird, L. C., & Kaelbling, L. P. (1999). Multi-value-functions: Efficient automatic action hierarchies for multiple goal MDPs. *Proceedings of the 19th International Joint Conference on Artificial Intelligence.*

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems 10,* (pp. 1043–1049). Cambridge, MA: MIT Press.

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming.* New York: John Wiley.

Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. *Proceedings of the Tenth International Conference on Machine Learning,*. San Mateo, CA: Morgan Kaufmann.

Seri, S. (2002). Hierarchical average reward reinforcement learning. Master's thesis, Oregon State University.

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

Tadepalli, P., & Dietterich, T. G. (1997). Hierarchical explanation-based reinforcement learning. *Proceedings of the Fourteenth International Conference on Machine Learning,*. San Mateo, CA: Morgan Kaufmann.

Tadepalli, P., & Ok, D. (1998). Model-based average reward reinforcement learning. *Artificial Intelligence*, *100*, 177–224.

Whitehead, S. D. (1991). A complexity analysis of cooperative mechanisms in reinforcement learning. *proceedings of the AAAI-91* (pp. 607–613).