AN ABSTRACT OF THE THESIS OF

Sandeep Seri for the degree of Master of Science in Computer Science presented on May 15th, 2002.

Title: Hierarchical Average Reward Reinforcement Learning

Redacted for Privacy

Abstract approved: _____

Prasad Tadepalli

Reinforcement Learning (RL) is the study of agents that learn optimal behavior by interacting with and receiving rewards and punishments from an unknown environment. RL agents typically do this by learning value functions that assign a value to each state (situation) or to each state-action pair. Recently, there has been a growing interest in using *hierarchical* methods to cope with the complexity that arises due to the huge number of states found in most interesting real-world problems. Hierarchical methods seek to reduce this complexity by the use of temporal and state abstraction. Like most RL methods, most hierarchical RL methods optimize the discounted total reward that the agent receives. However, in many domains, the proper criteria to optimize is the average reward per time step.

In this thesis, we adapt the concepts of hierarchical and recursive optimality, which are used to describe the kind of optimality achieved by hierarchical methods, to the average reward setting and show that they coincide under a condition called *Result Distribution Invariance*. We present two new model-based hierarchical RL methods, HH-learning and HAH-learning, that are intended to optimize the average reward. HH-learning is a hierarchical extension of the model-based, average-

reward RL method, H-learning. Like H-learning, HH-learning requires exploration in order to learn correct domain models and optimal value function. HH-learning can be used with any exploration strategy whereas HAH-learning uses the principle of "optimism under uncertainty", which gives it a built-in "auto-exploratory" feature. We also give the hierarchical and auto-exploratory hierarchical versions of R-learning, a model-free average reward method, and a hierarchical version of ARTDP, a model-based discounted total reward method.

We compare the performance of the "flat" and hierarchical methods in the task of scheduling an Automated Guided Vehicle (AGV) in a variety of settings. The results show that hierarchical methods can take advantage of temporal and state abstraction and converge in fewer steps than the flat methods. The exception is the hierarchical version of ARTDP. We give an explanation for this anomaly. Auto-exploratory hierarchical methods are faster than the hierarchical methods with ϵ-greedy exploration. Finally, hierarchical model-based methods are faster than hierarchical model-free methods.

Hierarchical Average Reward Reinforcement Learning

by
Sandeep Seri

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented May 15th, 2002
Commencement June 2003

Master of Science thesis of Sandeep Seri presented on May 15th, 2002

APPROVED:

Redacted for Privacy

Major Professor, representing Computer Science

Redacted for Privacy

Chair of the Department of Computer Science

Redacted for Privacy

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Sandeep Seri, Author

Approved by Committee:

Redacted for Privacy

Major Professor (Prasad Tadepalli)

Redacted for Privacy

Committee Member (Paul Cull)

Redacted for Privacy

Committee Member (Thomas Dietterich)

Redacted for Privacy

Graduate School Representative (Allen Wasserman)

Date thesis presented May 15th, 2002

ACKNOWLEDGMENTS

Without the constant stream of ideas and the infinite patience of my major advisor, Prasad Tadepalli, this work would not have been possible. Thanks, Prasad! I would also like to thank my minor advisor, Thomas Dietterich, and committee members Paul Cull and Allen Wasserman. Many thanks to all the faculty at Oregon State, from whose classes I learnt a lot, and to the staff, past and present, who were always very helpful. Many fellow graduate students enriched my learning experience. Finally, I would like to acknowledge the support of many friends and of my family, especially my mother and my elder brother, whose unconditional love has kept me going.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

DEDICATION

To the memory of my father, Sri S. Ram Kumar (1948 - 1998).

HIERARCHICAL AVERAGE REWARD REINFORCEMENT LEARNING

1. INTRODUCTION

1.1. Outline of the Thesis

The need for autonomous learning agents arises in a wide variety of domains. Robotics, game playing and scheduling all have problems where manually programming the agents is either tedious or not possible. Either the information needed to write the programs is not available or the number of different situations that need to be handled is too large. What is needed is an autonomous agent that learns from the feedback it gets by interacting with the environment.

Reinforcement learning (RL) agents do just that by representing the domain as a Markov Decision Problem (MDP). The agent has to learn the action to perform in each situation based on the feedback it gets. All the relevant information about the current situation of the domain environment is summarized as the *state*. The task of the agent is to learn a mapping from states to actions, referred to as the policy, that maximizes some measure of the external rewards it gets.

Reinforcement learning methods have been used successfully in a number of different domains. Tesauro (1992) produced a program that learned to play backgammon at the grandmaster level. Zhang and Dietterich (1996) created a job-shop scheduler that performed better than the best known heuristic method on a NASA space shuttle payload processing task. Crites and Barto (1996) applied RL methods to an elevator dispatching task. Their method was better than the

best heuristic method for the task. Singh and Bertsekas (1997) used an RL method for the task of dynamically allocating channels in a cellular telephone system.

Most reinforcement learning methods, such as ARTDP and Q-learning, optimize the *discounted* total reward received by the agent (Barto, Bradtke, & Singh, 1995; Watkins & Dayan, 1992). Reward is treated like money, with the value of a fixed amount of it going down with time. While this is mathematically convenient, there is no real basis for discounting in many domains, i.e., the value of reward doesn't change with time. The average reward received per time step is a natural criterion to optimize in most recurrent domains, including many problems in industrial engineering and operations research (Mahadevan, 1996).

Reinforcement learning methods can also be divided into two groups based on whether the domain model is explicitly learned. Model-free methods, which directly learn the value function, are simpler and take less space. Model-based methods learn the domain model and use it to compute the value function. While model-based methods take more space, they have been shown to be more efficient than model-free methods and and are amenable to sophisticated learning methods such as prioritized sweeping (Moore & Atkeson, 1993). They also facilitate planning and learning based on hypothetical experience. There are both model-based and model-free versions of RL algorithms that optimize the average reward - e.g. H-learning (Tadepalli & Ok, 1998) and R-learning (Schwartz, 1993), respectively.

One problem with standard RL methods is that very often the number of states of the domain environment can be huge. For example, the cellular telephone system cited above has 70^{49} states. Most RL methods learn an optimal policy indirectly by learning a value function that maps each state to a real number. A straightforward tabular representation of the value function will not work. RL methods typically cope with this problem by representing the value function

using a function approximator, e.g., as a neural network (Tesauro, 1992; Zhang & Dietterich, 1996; Crites & Barto, 1996), a linear function (Singh & Bertsekas, 1997; Tadepalli & Ok, 1998) or a decision tree (Boutilier, Dearden, & Goldszmidt, 1995).

Another approach to making RL methods work in large domains, that has been getting increasing attention (Dietterich, 2000; Ghavamzadeh & Mahadevan, 2001; Moore, Baird, & Kaelbling, 1999; Parr & Russell, 1998; Sutton, Precup, & Singh, 1999), is to use hierarchical methods. The idea is to transform a single learning problem into a hierarchy of smaller problems via the use of abstraction. Instead of viewing the problem as a single task with many primitive actions, we decompose the problem into a hierarchy of tasks, sub-tasks and primitive actions. Using hierarchical methods we can scale existing methods to solve larger problems.

Hierarchical methods typically constrain the set of policies that can be considered. Thus the best we can hope to do when using these methods is to find the *hierarchically optimal* policy, i.e., the policy that is the best among those that can be represented, given the constraints. Whether the hierarchically optimal policy is also optimal overall depends on the way the constraints are defined. RL methods that use the MAXQ framework (Dietterich, 2000), from which our framework is derived, are known to converge to lesser form of optimality called *recursive optimality.* A policy is recursively optimal when each subtask chooses the locally optimal policy given the policies of its children.

Most previous work in hierarchical reinforcement learning, including the original MAXQ work, is in the discounted or the episodic settings. Here, we are interested in undiscounted infinite horizon case, where the goal is to optimize the average reward received per time step or the "gain." Ghavamzadeh and Mahadevan (2001) extended and applied the MAXQ framework to the average reward

setting. In this thesis, we introduce the Hierarchical Average-reward Reinforcement Learning (HARL) framework, which is another adaptation of the 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 property 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. We also give a hierarchical version of R-learning, a model-free average reward RL method, called HR-learning. While our framework is not directly applicable to the discounted learning setting, we adapt it to give a hierarchical version of ARTDP, a model-based discounted RL method. We explain why we expect the hierarchical version of ARTDP to perform poorly.

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 ex-

ploration 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 also give the auto-exploratory version of HR-learning, called HAR-learning.

We compare the performances all the methods 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, compared to H-learning. HH-learning also outperforms HR-learning, its model-free counterpart. We also show that HH-learning and HR-learning, using ϵ-greedy exploration, are significantly outperformed by their auto-exploratory versions in a task where effective exploration is critical and challenging.

1.2. Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2 gives the background information on reinforcement learning and Semi-Markov Decision Problems (SMDPs). It describes SMDP versions of two standard average reward RL methods: H-learning, a model-based method and R-learning, a model-free method. It also gives an SMDP version of ARTDP, a model-based discounted RL method.

Chapter 3 explains the need for hierarchical methods and describes previous approaches to hierarchical reinforcement learning. We describe the HARL framework, our adaptation of the MAXQ framework to the average reward setting. We give the definitions of hierarchical and recursive optimality in the average re-

ward setting and prove that they coincide when the result distribution invariance condition holds.

Chapter 4 presents two new hierarchical RL methods: HH-learning, and HR-learning, the hierarchical versions of H-learning, and R-learning, respectively. We describe HARTDP, the hierarchical version of ARTDP, which is based on an adaptation of HARL to discounted learning setting. We present experimental results comparing the performance of the hierarchical and standard methods in the task of scheduling a simulated AGV.

Chapter 5 explains the need for exploration and illustrates its importance. We give auto-exploratory versions of HH-learning and HR-learning, called HAH-learning and HAR-learning, respectively. We give experimental results comparing the performance of the four methods.

Chapter 6 concludes the thesis with a summary of the results and outlines a few possible directions in which to extend them.

2. REINFORCEMENT LEARNING

In this chapter we present background information on reinforcement learning and Semi-Markov Decision Problems (SMDPs) which form the basis of many hierarchical methods, optimization criteria adapted to SMDPs, and SMDP versions of H-learning, R-learning and ARTDP.

2.1. Background

All standard RL methods assume the following scenario: the agent is embedded in an environment in which it can act. The actions cause the state of the environment to change and result in a reinforcement (reward or penalty) for the agent. Both the state changes and the reinforcement can be stochastic, i.e., an agent can take the same action in a state with different outcomes. The probabilities underlying the state changes and immediate reinforcements are assumed to be stationary. We also assume that the next state and the reinforcement depend only on the current state and the action chosen. This is called the Markov property. The agent is also assumed to be able to always detect the state it is in [1]. The reinforcement received by the agent can be delayed; a "good" action might not get rewarded immediately and a "bad" action might get some immediate reward but would eventually result in punishment. So the problem is to learn to make decisions under uncertainty and with delayed reinforcement. Unlike the supervised learning scenario, the RL agent is never presented with a input/output (or

---

[1] When the state is not always fully observable or there is *noise* in the state information, the Markov property is violated. The formal model that deals with this scenario is called a *Partially Observable Markov Decision Process* or POMDP.

state/good-action or state/bad-action) pairs. The task of the agent is to arrive at a policy, i.e., a mapping from states to actions, that maximizes some measure of the external reinforcement it gets.

The problem faced by the RL agent is modeled as a Markov Decision Process(MDP) (Puterman, 1994). An MDP consists of a set of states, S, a set of actions A, a reward function $r^a(s)$ which gives the reward for performing action a in state s, and a state transition matrix $P(s'|s,a)$ that gives the probability of reaching state s' upon executing action a in state s. States are assumed to be discrete. The actions are assumed to be "primitive" in the sense that they can be directly executed and take a single time step (in the next section we will see an extension of MDPs that does not have this limitation).

Given the domain model, there exist Dynamic Programming algorithms that find the optimal policy (Puterman, 1994). However, these algorithms require time that is polynomial in the number of states. Thus they are not practical for most real-world problems in which the number of states is huge and exponential in the number of relevant parameters. All the standard reinforcement learning methods (Kaelbling, Littman, & Moore, 1996; Sutton & Barto, 1998; Bertsekas & Tsitsiklis, 1996) are based on dynamic programming, but offer the advantages of scaling to large state spaces and not requiring a predefined domain model. Further, they can be used to make decisions while learning, i.e., the quality of the decisions improves with time.

2.2. Semi-Markov Decision Problems

Semi-Markov Decision Problems (SMDP) generalize MDPs 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. The agent chooses a new action at each decision epoch. The immediate reward is assumed to occur at the beginning of the action.

2.3. Optimization Criteria

As we mentioned above, the task of the RL agent is to find an *optimal* policy. A policy is optimal when it optimizes some long-term measure of the external reinforcement. But we have to decide exactly what measure or function of the long-term reinforcement we want to optimize. There are three choices that have been widely studied: the total reward, the discounted total reward and the average reward.

Before we look at the optimization criteria, there are a few of important points to be noted. First, given a policy there exists a value function that maps states to the expected value of starting in that state and following the policy. The value depends on the kind of optimization criteria we use, as described below. Second, only policies that are "greedy" with respect to their own value functions can be optimal. A greedy policy is one that always chooses an action that optimizes a local evaluation function which combines the value function of the policy and the immediate reward of the actions in some way. Finally, RL methods learn the optimal policy indirectly by learning its value function.

2.3.1. Total Reward Optimization

The simplest criteria is to optimize the total reward received. 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 total reward received starting in state s and following the policy π is given by:

$$V^{\pi}(s) = \lim_{N \to \infty} E(\sum_{j}^{N} R_j) \tag{2.1}$$

In episodic SMDPs there is an absorbing goal state that can be reached from every state, and the value function of a policy π is given by the following recurrence relation:

$$V^{\pi}(s) = \begin{cases} 0 & \text{is } s \text{ is a goal state} \\ r^{\pi(s)}(s) + \sum_{s' \in S} P(s'|s, \pi(s)) V^{\pi}(s') & \text{otherwise} \end{cases} \tag{2.2}$$

All the optimal policies, π^* share the same value function, given by (Puterman, 1994),

$$V^{\pi^*}(s) = \begin{cases} 0 & \text{is } s \text{ is a goal state} \\ \max_{a \in A(s)} [r^a(s) + \sum_{s' \in S} P(s'|s, a) V^{\pi^*}(s')] & \text{otherwise} \end{cases} \tag{2.3}$$

The optimal value function equations are often referred to as the Bellman equations.

2.3.2. Discounted Total Reward Optimization

If the SMDP we are interested in has an infinite horizon and there is no absorbing goal state, then the total reward could be infinite. To avoid this problem we can introduce a discounting factor, $\gamma < 1$, that is used to discount future rewards. The discounting factor can be interpreted either as an interest rate or as

the probability that the agent lives another time step. Assuming R_j and T_j are as previously defined, the discounted total reward received starting in state s and following policy π is given by:

$$f^{\pi}(s) = \lim_{N \to \infty} E(\sum_{j}^{N} \gamma^{T_j} R_j) \tag{2.4}$$

The value function of a policy π satisfy the following recurrence relation:

$$f^{\pi}(s) = r^{\pi(s)}(s) + \sum_{s' \in S} \gamma^{t^{\pi(s)}(s)} P(s'|s, \pi(s)) f^{\pi}(s') \tag{2.5}$$

The Bellman equations for discounted total reward are given by (Puterman, 1994),

$$f^{\pi^*}(s) = \max_{a \in A(s)} [r^a(s) + \sum_{s' \in S} \gamma^{t^a(s)} P(s'|s, a) f^{\pi^*}(s')] \tag{2.6}$$

2.3.3. Average Reward Optimization

The average reward or *gain* of π with respect to the starting state s is defined as:

$$\rho^{\pi}(s) = \lim_{N \to \infty} \frac{E(\sum_{j}^{N} R_j)}{E(\sum_{j}^{N} T_j)} \tag{2.7}$$

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 $h^{\pi}(s)$.

$$h^{\pi}(s) = \lim_{N \to \infty} E(\sum_{j}^{N}(R_j - \rho^{\pi} T_j)) \tag{2.8}$$

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:*

$$h(s) = \max_{a \in A(s)} [r^a(s) - \rho t^a(s) + \sum_{s' \in S} P(s'|s,a) h(s')] \tag{2.9}$$

Further, the greedy policy π^*, which selects actions that maximize the r.h.s. of Equation (2.9) attains the optimal average reward $\rho = \rho^{\pi^*} \geq \rho^{\pi}$ over all policies π. π^* is also known as the gain-optimal policy.

Only policies that are greedy with respect to their value function (in all recurrent states) can be optimal. This is formally stated in the following theorem (Puterman, 1994).

Theorem 2 *For any SMDP, all optimal actions, in any recurrent state maximize the Bellman equation (Equation 2.9).*

The proof of the second theorem is similar to that of Proposition 8.6.1 in Puterman (1994) which shows that if a policy improvement occurs in a recurrent state, the gain for the improved policy is greater than the gain for the previous policy.

2.4. H-LEARNING

The H-learning algorithm of (Tadepalli & Ok, 1998), adapted to SMDPs, is designed to optimize the average reward and 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 2.9 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. The complete algorithm is given in Figure 2.1. While H-learning has not been formally proved to converge[2], 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).

2.5. ARTDP

Barto et al. (1995) give a model-based discounted learning algorithm that works very similarly to H-learning. It can be adapted to SMDPs as follows. On-line experience is used to learn the next state, reward and time models. These are then used to update value of the current state using the following update equation:

$$f(s) \leftarrow \max_{a \in A(s)} [r^a(s) + \sum_{s' \in S} \gamma^{t^a(s)} P(s'|s,a) f(s')] \tag{2.10}$$

As in H-learning, exploration is needed to learn correct domain models. Actions are chosen using a mixture of greedy and exploratory policies. All the values are initialized to zero.

---

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

1. Take an exploratory action or a greedy action in the current state s. Let a be the action taken, s' be the resulting state, r_{imm} be the immediate reward received and t_{imm} be the time it takes for action a to complete.
2. $n(s,a) \leftarrow n(s,a) + 1$
3. $n(s,a,s') \leftarrow n(s,a,s') + 1$
4. $p(s,a,s') \leftarrow n(s,a,s')/n(s,a)$
5. $r^a(s) \leftarrow r^a(s) + (r_{imm} - r^a(s))/n(s,a)$
6. $t^a(s) \leftarrow t^a(s) + (t_{imm} - t^a(s))/n(s,a)$
7. $GreedyActions(s) \leftarrow \arg\max_{a \in A(s)}\{r^a(s) + \sum_{s' \in S} p(s,a,s')h(s')\} - \rho * t^a(s)$
8. If $a \in GreedyActions(s)$, then
   (a) $\rho \leftarrow (1-\alpha)\rho + \alpha((r^a(s) - h(s) + h(s'))/t^a(s))$
   (b) $\alpha \leftarrow \frac{\alpha}{\alpha+1}$
9. $h(s) \leftarrow \max_{a \in A(s)}\{r^a(s) + \sum_{s' \in S} p(s,a,s')h(s')\} - \rho * t^a(s)$
10. $s \leftarrow s'$
11. Go to step 1

FIGURE 2.1. The SMDP H-learning algorithm.

2.6. R-LEARNING

R-learning is a model-free, average reward method (Schwartz, 1993). Unlike H-learning and ARTDP it does not learn the domain models. Instead it learns evaluations for state/action pairs, known as R-values. We can define the R-value of a state s and action a, for SMDPs, as follows:

$$R(s,a) = r^a(s) - \rho t^a(s) + \sum_{s' \in S} P(s'|s,a)h(s') \tag{2.11}$$

Suppose the agent executes action a in state s and the resulting state is s', and the reward received and the transition times are r_{imm} and t_{imm}, respectively. R-learning updates the R-value using the following update equation:

$$R(s,a) \leftarrow (1-\beta)R(s,a) + \beta(r_{imm} + h(s') - \rho t_{imm}) \tag{2.12}$$

where $h(s)$ is defined as follows:

$$h(s) = \max_{a \in A(s)} R(s,a) \tag{2.13}$$

Here β is the learning rate for the R-values. The immediate reward r_{imm} is also used to incrementally revise ρ, as in H-learning, using a learning rate parameter, α.

3. HIERARCHICAL REINFORCEMENT LEARNING

In this chapter we present some background information on hierarchical RL, including the MAXQ framework. We give our adaptation of the MAXQ framework for the average reward setting, called HARL. Finally, we present a theorem about the kind of optimality achieved by our method.

3.1. Background

Although the standard RL methods have been successfully applied to many large problems, using function approximation, they suffer at least two major problems. First, they assume that all the information about the state is relevant to all decisions. This is, typically, not how humans make decisions. We tend to make decisions based on only a part of the available information, which we consider relevant. For example, we don't consider whether it is raining while deciding what kind of calculator to buy. Second, standard RL methods assume that all actions have the same granularity. They assume that all actions can be directly executed in one fixed way and take a single time step. Although the SMDP versions of the RL methods remove the second limitation, the first remains. RL methods lack the concept of making "big" decisions, such as buying a house, and then refining and optimizing them.

Abstraction and hierarchies are obvious solutions to the two problems. These have been studied in the AI field of planning for a long time (Erol, Hendler, & Nau, 1994; Knoblock, 1994; Korf, 1985; Sacerdoti, 1974). There has also been some early work in reinforcement learning using abstraction and hierarchies (Dayan & Hinton, 1993; Kaelbling, 1993; Singh, 1992). But these methods were designed for specific problems. More recently there have been some proposals that are more

general and also have formal convergence results. Sutton, Precup and Singh (1999) extend the notion of an action to include temporally extended courses of behavior called "options". An option differs from the temporally extended actions of SMDPs in the sense that it is not an indivisible unit. They give learning algorithms to learn both with fixed options and to improve options. Parr and Russell (1998) describe a way of decomposing a problem into a hierarchy of non-deterministic finite state machines, called HAMs. HAMs constrain the set of policies that can be learned and partially specify the policy. The learning task is to learn the appropriate actions in the non-deterministic or choice states of the HAMs. Parr and Russell (1998) provide a provably convergent learning algorithm for the task. The MAXQ framework (Dietterich, 2000) provides a third way of specifying a task hierarchy. Since our approach is based on it, we describe it in more detail in the next section.

3.2. MAXQ Framework

The MAXQ framework provides a decomposition of both the task into a hierarchy of subtasks and the value function of the task into the value functions of the subtasks. The hierarchy allows subtasks to be shared. The MAXQ framework decomposes a given MDP, M, to a set of n subtasks $M_0, M_1, \ldots, M_n$. We refer to the subtasks by their subscripts. Formally, a subtask is specified by a three-tuple $(T^i, A^i, \tilde{R}^i)$ defined as follows:

1. $T^i(s)$ is a termination predicate that specifies whether a subtask can continue in a given state. It partitions the set of states into a set of active state S^i and terminal states T^i

2. A^i is the set of "actions" that the subtask is allowed to choose. These can either be other (possibly shared) subtasks or "primitive" actions of the original MDP. The set A^i is referred to as the children of subtask i.

3. $\tilde{R}^i(s'|s,a)$ is a pseudo-reward function which specifies the a pseudo-reward for each transition into a terminal state. This is a way of specifying the desirability of a terminal state for a particular subtask.

MAXQ also provides a decomposition for the value function. For each task,i, the value of a state/subtask pair (referred to as the Q-value) is sum of the value of executing the subtask and the value of completing the task. Formally, the Q function is defined as follows:

$$Q^i(s,a) = V^a(s) + C^i(s,a) \tag{3.1}$$

where, $V^i(s)$ is defined as follows:

$$V^i(s) = \begin{cases} \max_{a \in A^i(s)} Q^i(s,a) & \text{if } i \text{ is composite} \\ \sum_{s'} P(s'|s,a) R(s'|s,a) & \text{if } i \text{ is primitive} \end{cases} \tag{3.2}$$

The completion function, $C^i(s,a)$, is the expected discounted total reward of completing task i after the execution of subtask a in state s.

$$C^i(s,a) = \sum_{s',N} P^i(s',N|s,a)\gamma^N V(i,s') \tag{3.3}$$

Dietterich (2000) gives a provably convergent learning methods for total reward and discounted total reward optimizations for the MAXQ framework.

3.3. HARL Framework

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. 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 descendants. 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 3.1 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.

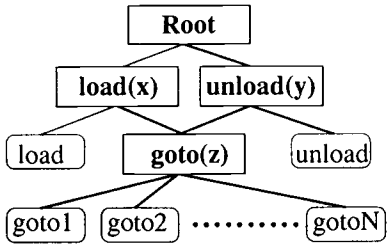
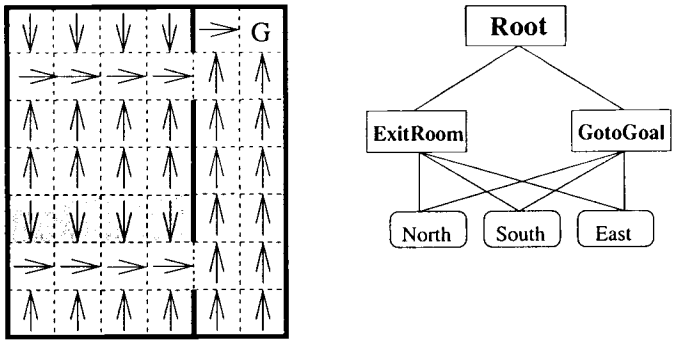


FIGURE 3.1. Task Hierarchy for the AGV Domain

The task hierarchy in Figure 3.1 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.

There are two related kinds of state abstraction that we use. One is ignoring the irrelevant state variables as in the *goto(x)* example above. Another form of state abstraction occurs when the range of values that a state variable can take is reduced. For example, both *load(x)* and *unload(x)* leave the AGV at one of



(a) (b)

FIGURE 3.2. A simple two room maze domain and its task hierarchy (adapted from (Dietterich, 2000))

machines or the destination stations. So the *root* task will always find the AGV to be in one of these four locations. Thus, we can abstract the state for the root task by reducing the range of values that the AGV location variable can take.

3.4. Result Distribution Invariance

In general, hierarchically optimal policies at any node depend on the policies used at all other nodes. For example, in the simple domain pictured in Figure 3.2(a) (taken from Dietterich (2000)), the task is to navigate to the goal state G in the right room starting from somewhere in the left room. The agent is allowed to move **North**, **South**, and **East**, which deterministically move the agent up, down, and left, respectively. Each move gets a reward of -1. We can define a simple task hierarchy for this domain as shown in Figure 3.2(b). The subtask **ExitRoom** terminates when the agent leaves the left room. The subtask **GotoGoal** terminates when the agent reaches the goal G. We assume that **ExitRoom** is admissible when the agent is in left room and **GotoGoal** is admissible when the agent is in the right room. The arrow marks in Figure 3.2(a) show the locally optimal policy within

each room. The problem is in the left room where, in the shaded region depicted in the figure, the locally optimal choice is not hierarchically optimal.

The optimal choice of which door to use to exit the left room depends on the goal location in the right room. 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 well-defined termination conditions, 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).

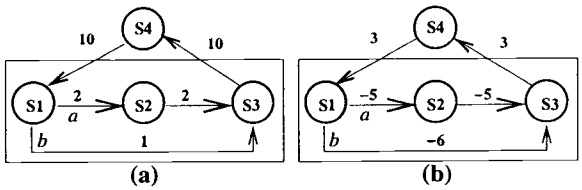


FIGURE 3.3. A 4-state MDP: to illustrate different criteria to optimize subtasks

The MDP in Figure 3.3(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 consequence 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, assuming that all its subtasks in turn use recursively gain-optimal policies. For state s and subtask i, we denote the total relativized reward by $h^i(s)$. 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 for all recurrent states.

$$h^i(s) = \begin{cases} r^i(s) - \rho t^i(s) & \text{if } i \text{ is a primitive action} \\ 0 & \text{if } s \text{ is a goal state for subtask } i \\ \max_{a \in A^i(s)} \{h^a(B^a(s)) + \sum_{s' \in S} P^i(s'|s,a) h^i(s')\}, & \text{otherwise} \end{cases} \tag{3.4}$$

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 $h^a(B^a(s)) = h^a(s)$. Upon executing

the primitive task i in state s, $r^i(s)$ is the expected immediate reward, and $t^i(s)$ is the expected completion time. Finally $P^i(s'|s,a)$ denotes the probability of reaching state s' after executing task a from state s under the recursively gain-optimal policy for a. 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 2.9, when the actions a are interpreted as its highest level subtasks. But since $(r^a(s) - \rho t^a(s))$ denotes the total expected relativized reward during action a, it is the same as $h^a(s) = h^a(B^a(s))$. Hence its solution also satisfies the above equations except it does not have a goal state.

The HH-learning algorithm we present in the next chapter is designed 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.,

$$P^i(s'|s,j,\Pi^j) = P^i(s'|s,j) \tag{3.5}$$

In other words, the result states reached after the execution of a subtask cannot be changed by altering the policy of the subtask. 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 3 *When the result distribution invariance condition holds, a policy is hierarchically gain-optimal iff it is also recursively gain-optimal.*

Proof: Part 1: Hierarchical $\rightarrow$ Recursive. Assume that a policy Π_1 is hierarchically gain-optimal but not recursively gain-optimal. Let j be a deepest node that does not satisfy the Bellman equations 3.4 for some recurrent state s, i.e.,

$$b = \pi_1^j(s) \neq arg\max_a(h^a(B^a(s)) + \sum_{s'} P^j(s'|s,a)h^j(s') = b' \tag{3.6}$$

Let Π_2 be the same as Π_1, except $\pi_2^j(s) = b'$. Clearly more expected total relativized reward is received during the execution of subtask j by following Π_2 rather than Π_1, i.e., the bias value $h^j(s)$ increases. Let i be a parent of j. Since RDI holds, the set of states at which the subtask i makes a decision remains unchanged and the expected total relativized 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 other s'. We can continue applying the same argument up the task hierarchy until the children of the *root* node are reached. Now, the h-value of one or more of the *root* node's children has increased and since RDI holds for the *root* node as well, the set of states at which the root task makes a decision does not change. For any policy π, let $P^\pi(s)$ give the distribution of states at which a subtask is chosen by the root task. Define $h_\pi(s,\rho)$ to be $R_{\pi(s)}(s) - \rho T_{\pi(s)}(s)$, i.e., the total relativized (relative to ρ) reward under policy π. From the definitions of $P^\pi(s)$ and the bias value function of policy π, $h_\pi(s)$, we have

$$\sum_{s \in S} P^{\pi}(s) h_{\pi}(s) = 0 \tag{3.7}$$

Therefore, $\sum_{s \in S} P^{\pi_1}(s) h_{\pi_1}(s) = 0$. Since $h_{\pi}(s, \rho^{\pi}) = h_{\pi}(s)$ and $P^{\pi_1} = P^{\pi_2}$

$$\sum_{s \in S} P^{\pi_2}(s) h_{\pi_1}(s, \rho^{\pi_1}) = 0 \tag{3.8}$$

Since, $h_{\pi_2}(s, \rho^{\pi_1}) \geq h_{\pi_1}(s, \rho^{\pi_1})$ for all states s and $h_{\pi_2}(s, \rho^{\pi_1}) > h_{\pi_1}(s, \rho^{\pi_1})$ for at least one state s,

$$\sum_{s \in S} P^{\pi_2}(s) h_{\pi_2}(s, \rho^{\pi_1}) > 0$$

$$\sum_{s \in S} P^{\pi_2}(s)(R_{\pi_2(s)}(s) - \rho^{\pi_1} T_{\pi_2(s)}(s))) > 0$$

$$\sum_{s \in S} P^{\pi_2}(s) R_{\pi_2(s)}(s) > \rho^{\pi_1} \sum_{s \in S} P^{\pi_2}(s) T_{\pi_2(s)}(s)$$

$$\sum_{s \in S} P^{\pi_2}(s) R_{\pi_2(s)}(s) / \sum_{s \in S} P^{\pi_2}(s) T_{\pi_2(s)}(s) > \rho^{\pi_1}$$

$$\rho^{\pi_2} > \rho^{\pi_1}$$

This contradicts the assumption that Π_1 is hierarchically gain-optimal.

Part 2: Recursive $\rightarrow$ Hierarchical. The proof of this part is very similar. We assume that a policy Π_1 is recursively gain-optimal but is not hierarchically gain-optimal. Let Π_2 be hierarchically gain-optimal. Hence there is a deepest level subtask j and recurrent state s such that $\pi_1^j(s)$ is not hierarchically gain-optimal. By Theorem 2, replacing $\pi_1^j(s)$ with $\pi_2^j(s)$ cannot increase $h^j(s)$ because Π_1 is recursively gain-optimal. 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 decreases or stays the same. Thus $h^i(s)$ does not increase for any state s. We can continue this argument until the children of the *root* node are reached. We can apply the similar argument as in Part 1 to show that the gain of the overall task remains the same or decreases using the new

policy Π_2, which violates the assumption that Π_1 is not hierarchically gain-optimal.

□

4. HIERARCHICAL ALGORITHMS

In this chapter, we present the two algorithms based on the HARL framework, HH-learning, which is model-based, and HR-learning, which is model-free. We adapt our framework to the discounted learning setting and give a hierarchical version of ARTDP, called HARTDP. We explain why we expect HARTDP to perform poorly. We give experimental results comparing the hierarchical versions with the standard or "flat" methods. Finally, we present a brief analysis explaining the improved performance of the average reward hierarchical methods.

4.1. HH-LEARNING

HH-learning can be viewed as a hierarchical extension of H-learning. The algorithm is presented in Figure 4.1. The h values of states are updated using the right hand sides of Equation 3.4 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 4.1), 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 3.4 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*).

---

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 4.1. The HH-learning algorithm.

4.2. HR-LEARNING

HR-learning is a hierarchical extension of R-learning based on the HARL framework. Unlike HH-learning it is model-free. It learns a value function, R, for state/subtask pairs. The value $R^i(s,a)$ is the bias resulting from executing action a in state s and then completing task i using the optimal policy. We can define the R-value of a state/subtask pair as follows:

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

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

HR-learning works as follows: A task i selects among its children using a combination of greedy and exploratory policies. The chosen subtask is executed until it reaches a goal state. Suppose subtask a is executed in state s resulting in state s'. The R-values the state/subtask pair are updated using the following update equation:

$$\begin{aligned} R^i(s,a) &\leftarrow (1-\beta) R^i(s,a) + \beta(h^a(B^a(s)) + h^i(B^i(s'))) \\ h^i(s) &= \max_{a \in A^i(s)} R^i(s,a) \end{aligned} \tag{4.1}$$

where β is the learning rate for R-values. When a primitive action a is chosen it terminates immediately after execution. Let r_{imm} be the immediate reward received and t_{imm} be the time taken. The h-value of the primitive action is updated using the following equation:

$$h^a(s) \leftarrow (1-\beta)h^a(s) + \beta(r_{imm} - \rho t_{imm}) \tag{4.2}$$

The immediate reward is also used to update the current estimate of ρ. As in HH-learning, we use a learning rate parameter α for ρ.

4.3. HARTDP

While the HARL framework is not directly applicable to the discounted learning setting, we can adapt it by using the following Bellman equations:

$$f^i(s) = \begin{cases} r^i(s) & \text{if } i \text{ is a primitive action} \\ 0 & \text{if } s \text{ is a goal state for subtask } i \\ \max_{a \in A^i(s)} \{ f^a(B^a(s)) + \sum_{s' \in S} \gamma^{t^a(s)} P^i(s'|s,a) f^i(s') \}, & \\ & \text{otherwise} \end{cases} \tag{4.3}$$

where γ is the discount parameter, which needs to be tuned for performance. $f^i(s)$ denotes the expected discounted total reward received by starting in state s and following the recursively optimal policy. Note that, unlike in the average reward setting, we require the expected completion times of all the subtasks, not just the primitive actions. Also, the aim in this modified framework is to find the hierarchical discounted-optimal policy.

HARTDP works very similarly to HH-learning. On-line experience is used to learn the next state and time models for subtasks, and time and reward models for primitive actions. These models are then used to update the value function using the above equations. Subtasks are chosen using a mixture of greedy and exploratory policies.

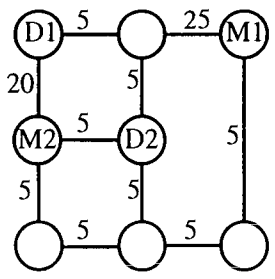
A key difference between HARL adapted to discounted learning setting and MAXQ is that the former lacks pseudo-reward functions (alternatively, we

can think of the pseudo-reward function always being zero). In subtasks without any non-zero rewards this causes methods such as HARTDP to do the equivalent of a random walk until the goal state is reached. Thus we expect HARTDP to perform poorly. The problem can be solved in MAXQ by giving appropriate non-zero pseudo-rewards. But this places an additional burden on the programmer. Further, the proper pseudo-rewards might depend on the goals at the higher levels of the task hierarchy.

4.4. Experimental Results

We compared the six methods, H-learning, R-learning, ARTDP, HH-learning, HR-learning, and HARTDP, in a number of AGV domains. A small AGV domain is depicted in Figure 4.2. 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 according to an exponential distribution with the means shown above the edges in Figure 4.2. 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.

In the first experiment, we compared the performance of the four methods in the AGV domain depicted in Figure 4.2 and described above. For this experiment, the domain parameters, K, p, q and u are 5, 0.5, 0.0 and 0.9 respectively. Both HR-learning and R-learning need the parameters, α and β to be tuned for better performance. We used six different values for β, 0.1, 0.3, 0.5, 0.7, 0.9, and 0.99, and three different values for α, 1, 0.1, and 0.01. ARTDP and HARTDP need the parameter γ to be tuned and we used four different values for it, 0.9, 0.95, 0.99, and 0.999 (We used the same range of values for all our experiments). H-learning and HH-learning do not have any parameters to tune. Figure 4.3 shows the average on-line reward for 30 trials. HR-learning performed best with the parameter values 1 and 0.3 for α and β, respectively. R-learning performed best with the parameter values 0.1 and 0.1 for α and β, respectively. The γ values that performed best for ARTDP and HARTDP were 0.99 and 0.999, respectively. Both the average reward hierarchical methods outperformed the corresponding "flat" methods, with HH-learning converging the fastest. As expected, HARTDP converged to a suboptimal policy and whereas ARTDP was able to find the optimal policy.



M1, M2: Producing Machine
D1, D2: Destination Station

FIGURE 4.2. Stochastic SMDP Domain

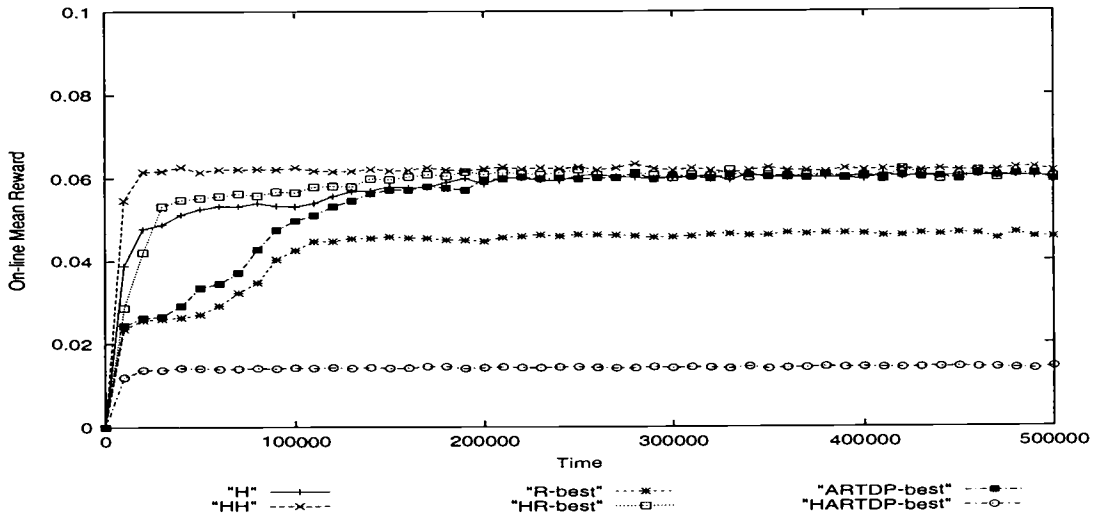
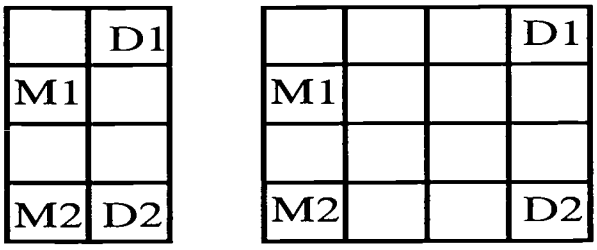


FIGURE 4.3. Stochastic SMDP Domain Learning Curves

In the next experiment, we tested how the six methods perform when the domain size is increased. We created two domains, one roughly twice the size of the other, depicted in Figure 4.4. The domain parameters 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 4.5 shows the mean on-line reward over 30 trials for the 4X2 domain. HR-learning performed best with the parameter values 1 and 0.1 for α and β, respectively. R-learning performed best with the parameter values 1 and 0.3 for α and β, respectively. The γ values that performed best for ARTDP and HARTDP were 0.95 and 0.99, respectively. Again both the average reward hierarchical methods outperform the "flat" methods and HARTDP converges to a sub-optimal policy.

Figure 4.6 shows the mean on-line reward over 30 trials for the 4X4 domain. In this domain, H-learning and R-learning take longer (roughly 50% more steps) to converge to an optimal policy compared to the 4X2 domain. On the



(a) 4X2 Domain (b) 4X4 Domain
M1, M2: Producing Machines
D1, D2: Destination Stations

FIGURE 4.4. Two AGV Domains: to show the effect of scaling

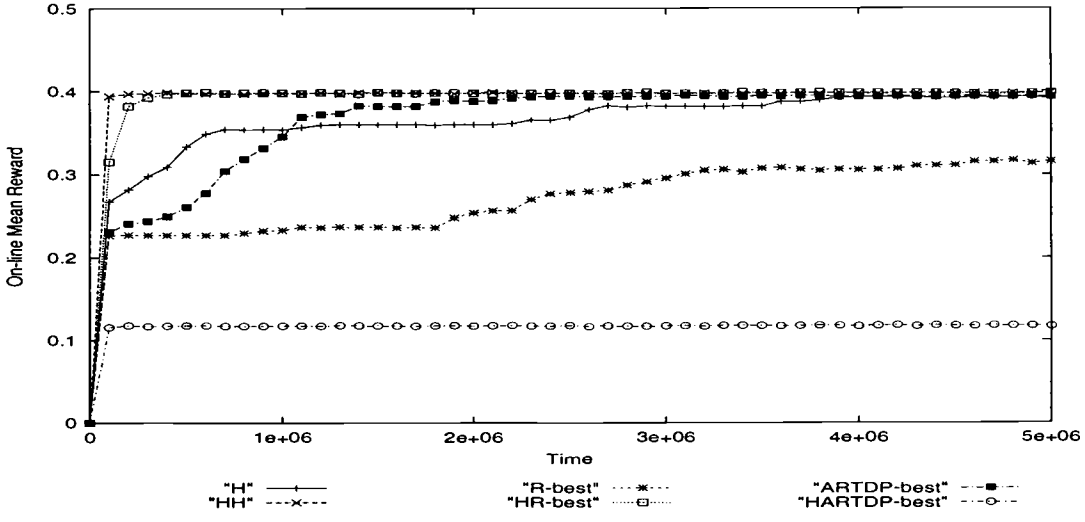


FIGURE 4.5. 4X2 Domain Learning Curves

other hand, HH-learning and HR-learning take about the same time. HR-learning performed best with the parameter values 0.1 and 0.3 for α and β, respectively. R-learning performed best with the parameter values 0.1 and 0.1 for α and β, respectively. The γ values that performed best for ARTDP and HARTDP were 0.95 and 0.99, respectively. The results of this experiment show that the average reward hierarchical methods scale better than the "flat" methods.

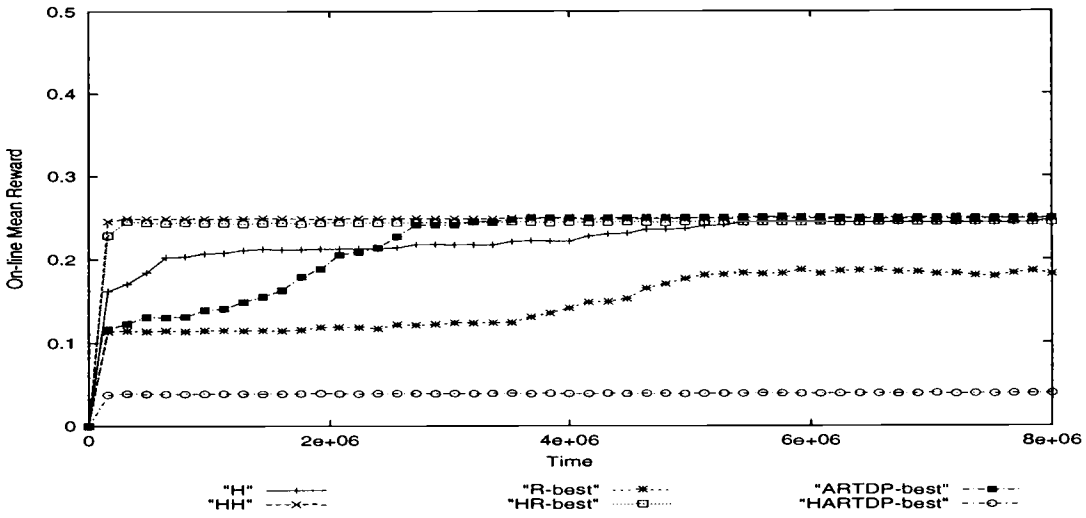


FIGURE 4.6. 4X4 Domain Learning Curves

4.5. Analysis

The reason why the hierarchical methods are 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.

4.6. Summary

From the experimental results presented in this chapter we can conclude that, (a) average reward hierarchical methods outperform and scale better than their "flat" counterparts, and (b) the model-based average reward methods outperform their model-free counterparts. The discounted version of our algorithm, namely HARTDP, fails to learn the hierarchical discounted-optimal policy when there are subtasks with no non-zero rewards for any state or appropriate pseudo-rewards for the terminal states.

5. AUTO-EXPLORATORY ALGORITHMS

In this chapter we describe an effective exploration method for hierarchical ARL based on the idea of optimism under uncertainty. We present two new algorithms HAH-learning and HAR-learning, auto-exploratory versions of HH-learning and HR-learning. We present experimental results comparing the four methods.

5.1. Exploration Vs. Exploitation

Effective exploration of the state space is an important issue for reinforcement learning. Without sufficient exploration RL methods could converge to suboptimal policies. Model-based methods could also learn incorrect models without sufficient exploration. However, RL agents are expected to perform, i.e. make decisions, while learning. Thus they need to exploit what they already know by acting greedily with respect to their value functions. This results a conflict between exploration and exploitation (Kaelbling et al., 1996), and requires an efficient exploration strategy. So far we have been using ϵ-greedy exploration, a simple strategy that we show is not very efficient when exploration is challenging and critical. 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. We describe this method in some detail in the next section and present an improvement in the subsequent section.

5.2. AH-LEARNING

AH-learning is designed for average reward reinforcement learning. In AH-learning, value of each state action pair, called the R-value, is stored separately. The initial value of ρ is set to be high and its learning rate α is set to be low. The best initial values of ρ and α are found through experimentation. To understand why these changes result in automatic exploration consider what happens in the initial stages of learning before any reward is received. All the values in H-learning are initialized to zero. Assume we selected action a in state s and ended up in state s'. Since no reward has been received and all other values, including ρ, are zero, the h-value of s is unchanged. Thus, state s, which has just been visited, appears as promising as any unvisited state. We can avoid this by setting ρ to a high value. Now, h-values are decreased every time a state is visited. Hence, states that haven't been visited will have better h-values and are more likely to be chosen as the greedy choices when compared to states that have been visited.

In H-learning, when making the greedy choice, all actions are equally likely to be chosen initially. This is irrespective of how many times a particular action has previously been selected. Suppose we store R-values and initialize ρ to a high value. Assume we are in state s for the first time and we chose action a. The corresponding R-value is now decreased. Thus, the next time we are in state s, we will select actions other than a while making the greedy choice. Now consider what happens when we follow the greedy policy π, i.e., always select the greedy action. If the current value of ρ is less than ρ^{π}, h-values of states in the current greedy policy will tend to increase or stay the same, ignoring changes to ρ itself. This is because the sum of the immediate rewards in any n steps ($n\rho^{\pi}$) is likely to be higher than $n\rho$. Since the h-values of states not in the current greedy policy

stay the same, we will never get out of the current set of states. If the optimal policy involves going to the unvisited states, H-learning will converge to a sub-optimal policy. If $\rho > \rho^{\pi}$, than the h-values of the states in the current greedy policy decrease on the average. Eventually states outside the current greedy policy will appear more promising and will be visited. In this case, H-learning will not converge to a sub-optimal policy. While we are initializing ρ to a high value. it will tend to decrease and could become lesser than ρ^{π}, for some sub-optimal policy π. To avoid this, we can set α, the rate at which ρ is changed, to a low initial value.

Unfortunately, AH-Learning requires the initial value of ρ and the learning rate α for updating ρ 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. We describe a way of avoiding having to tune these parameters in the next section.

5.3. HAH-LEARNING

HAH-learning is an auto-exploratory version of HH-learning. The ideas of AH-learning 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-values are defined as in HR-learning.

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

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

However, since HAH-learning is model-based, we learn the models, $P^i(s'|s,a)$, $r^i(s)$ and $t^i(s)$, from experience. We then use the above equations to update the R-value for the current state/subtask pair.

As mentioned in the previous section, the initial values of ρ and α need to be tuned. While it is difficult to guess the proper values for these parameters, 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}.

5.4. HAR-LEARNING

HAR-learning can be viewed both as the auto-exploratory version of HR-learning and as the model-free version of HAH-learning. HAR-learning stores and

updates the R-values exactly like HR-learning. However, the initial value of ρ is set to ρ^{max} and it is updated as in HAH-learning. The initial value of α is set to one and decayed gradually. It does not need to be tuned, unlike in HR-learning. β, the learning rate for R-values still needs to be tuned.

5.5. Experimental Results

In order to compare HH-learning, HAH-learning HR-learning and HAR-learning, we created two domains. In the "maze" domain, depicted in Figure 5.1, 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.

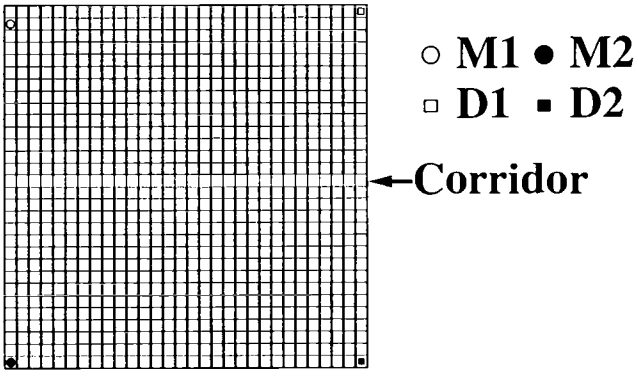


FIGURE 5.1. Maze Domain

The key difference between the two domains is the time a random walk will take to get to a particular location. Whitehead (1991) showed that for certain

idealized state spaces the time for a random walk to reach a goal depends on the ratio of the number of actions that lead toward the goal to the number of actions that lead away from the goal. If this ratio is greater that 1, then the expected time to reach the goal is exponential in the "depth" of the state space. In order for this result to hold the goal must be at the "center" of the state space with the boundary locations at a uniform distance to the goal.

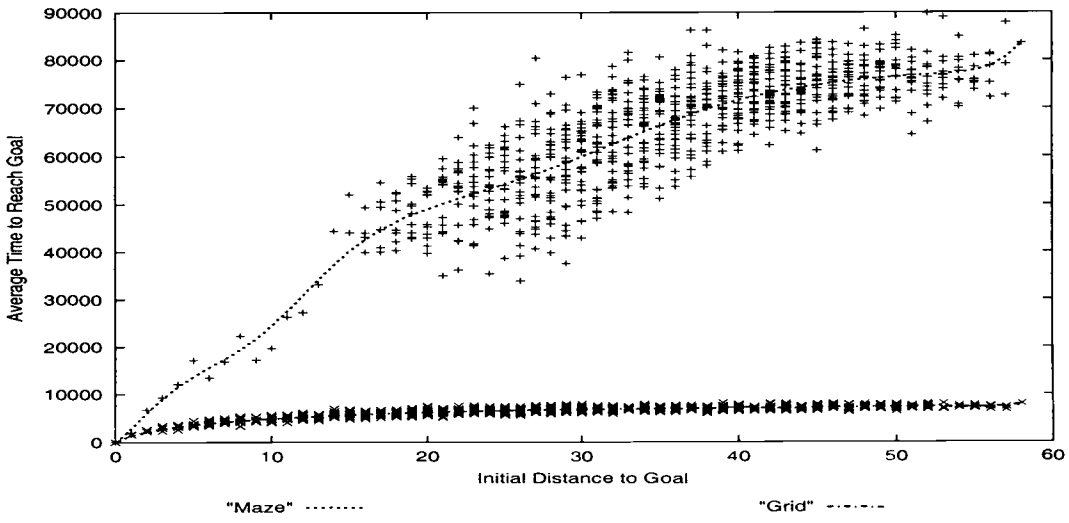


FIGURE 5.2. Random Walk in Maze and Grid Domains

We expect that the time a random walk will take to reach a particular location will vary more slowly with the initial distance to the goal state in the grid domain than in the maze domain. This is because along the corridor cells, in the maze domain, it is three times more likely to take a step that leads away from the goal than toward the goal. Since our domains do not have idealized state spaces, we ran an experiment to see the relationship between the time to reach a goal and the initial distance to the goal, while performing a random walk. In this experiment, we fixed the goal location at the upper right corner in both the

grid and maze domains. We then started a random walk from each of the cells. Figure 5.2 shows the average of 300 such trials. As can be seen from the figure a random walk takes much longer to reach a goal in the maze domain than in the grid domain. Hence, we expect exploration to be more critical in the maze domain than in the grid domain.

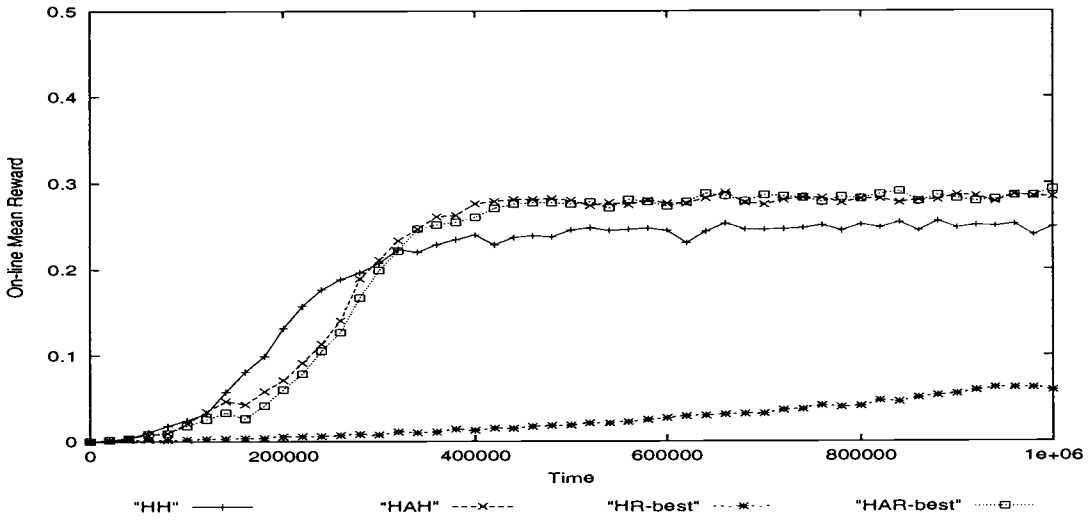


FIGURE 5.3. Grid Domain Learning Curves

Figure 5.3 gives the average on-line reward for 30 trials in the grid domain. The performance of HH-learning, HAH-learning and HAR-learning are comparable, whereas the performance of HR-learning is much worse. We could interpret this as exploration being much more critical to HR-learning than to HH-learning. In the maze domain, where exploration is critical, HAH-learning and HAR-learning converge to the optimal policy much faster than HH-learning (Figure 5.4), which uses ϵ-greedy exploration. Again the performance of HR-learning is much worse.

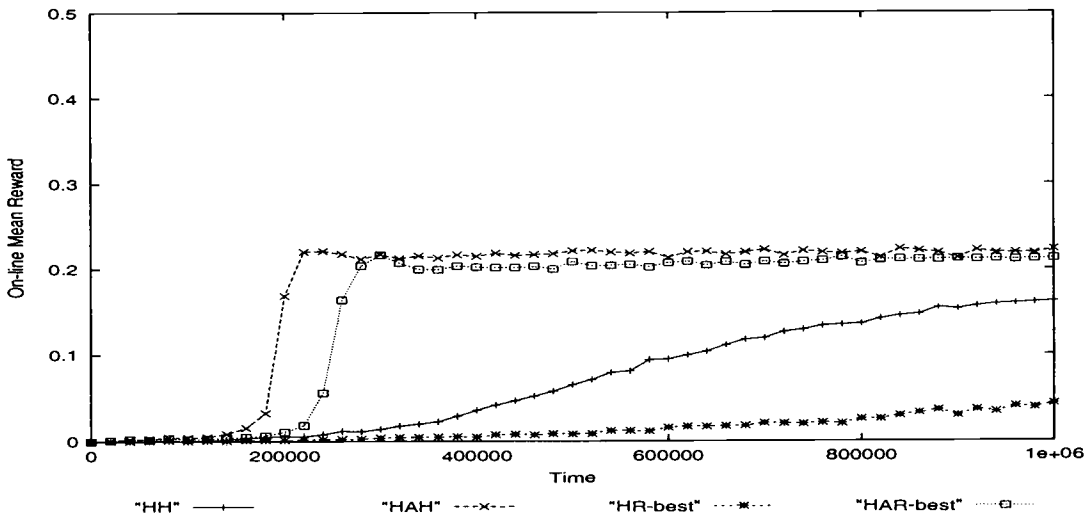


FIGURE 5.4. Maze Domain Learning Curves

5.6. Summary

From the results presented in this chapter we can conclude that, (a) the hierarchical auto-exploratory methods converge much faster than the hierarchical methods using ϵ-greedy exploration, and (b) the hierarchical model-based methods outperform their model-free counterparts.

6. CONCLUSIONS AND FUTURE WORK

In this chapter we present a summary of results and some directions to extend them.

6.1. Conclusions

In this thesis, 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. In the discounted learning setting, the actual discount rate depends on the completion time of the actions/subtasks. While RDI requires the distribution of result states of a subtask to be independent of the policy of the subtask, the completion time could still depend on the policy. Thus a recursively optimal choice could differ from the hierchically optimal one, even when RDI holds. 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.

One difference between MAXQ and HARL frameworks is in the kind of value functions learned. MAXQ learns the so called "completion functions" which correspond to the expected reward received in completing a task after completing one of its subtasks. There is one such function for each edge in the task graph. In order to decide which subtask to choose, MAXQ searches for all possible paths from the current task node to a leaf and chooses the one with the best total reward. In

HARL, value functions are learned for either each task (HH-learning) or for a task-subtask pair (HR-learning, HAH-learning and HAR-learning). The first represents a state-value function, and the second represents state-action value function like the Q-function. In both cases, it avoids searching through all possible paths of the task graph, choosing actions by greedy one step look ahead instead.

Further, our framework does not have (or need) the pseudo-reward functions. Non-zero pseudo-rewards are necessary in MAXQ for subtasks in which there is no "real" reward, such as the *goto(x)* subtask in the AGV domain. Without these pseudo-rewards the subtask will not converge to the optimal policy. However, introduction of non-zero pseudo-rewards requires algorithms based on the MAXQ framework, such as MAXQ-Q, to learn two completions functions. One completion function is used to make locally optimal decisions; the second to compute the value function of the parent task.

We presented two new hierarchical average reward learning algorithms HH-learning, which is model-based, and HR-learning, which is model-free. We showed their effectiveness in several simple AGV domains. These algorithms scale well with the physical size of the domain and find recursively gain-optimal policies. We presented two new auto-exploratory versions of the hierarchical algorithms, HAH-learning and HAR-learning, that are quite effective in domains where good exploration strategy is critical for performance, and eliminates the need for careful tuning of parameters.

While the model-free versions are easier to implement, our results show that they are slower to converge than the corresponding model-based methods. They also need their parameters to be tuned carefully for better performance. Moreover, they cannot be used for prediction and planning, unlike the model-based methods.

6.2. Future Directions

All the successful applications of RL methods to large problems involve the use of function approximation to compactly represent the value function. Model-based methods also need to compactly represent the domain models. Tadepalli and Ok (1998) report the use of local linear regression for function approximation and dynamic Bayesian networks for compactly representing action models. Combining hierarchical learning with function approximation and factored action models is an important area for future research.

We currently assume that the task hierarchy and abstractions are provided by the programmer. 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)

Our methods can only find the hierarchically optimal policy when result distribution invariance condition is met. When this assumption is violated, the algorithm may not find a hierarchically optimal policy. One could "solve" this problem by updating the values of the goal states of a subtask with their corresponding values in the higher level subtasks. However, this has the effect of making subtasks more context-sensitive and less amenable to abstraction and less reusable. We need to extend our methods to efficiently handle the case where the result distribution condition is not satisfied. Andre and Russell (2002) describe a method to learn hierarchically optimal policies in the MAXQ framework using a three part value function decomposition, which is adapted to ARL by Ghavamzadeh and Mahadevan (2002). In this approach a value function is attached to the terminal states of each subtask, and denotes the total expected relativized reward of those states with respect to the overall task.

Proving that our algorithms converge to the recursively optimal policy is an important task. Dietterich (2000) describes safe abstraction conditions for the MAXQ framework. We hope to be able to derive similar conditions for the HARL framework. Testing in more domains is needed to compare the performance of model-free and model-based methods. We also need to compare the performance of our methods with other hierarchical methods such as those of Ghavamzadeh and Mahadevan (2001).

BIBLIOGRAPHY

Andre, D., & Russell, S. (2002). State abstraction for programmable reinforcement learning. *To appear in the Proceedings of the Eighteenth AAAI.*

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence, 73(1)*, 81-138.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming.* Athena Scientific, Belmont, MA.

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting structure in policy construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence.*

Crites, R. H., & Barto, A. G. (1996). Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8,* Cambridge, MA. MIT Press.

Dayan, P., & Hinton, G. E. (1993). Feudal reinforcement learning. In *Advances in Neural Information Processing Systems, 5* San Mateo, CA. Morgan Kaufmann.

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

Erol, K., Hendler, J., & Nau, D. S. (1994). HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94),* Vol. 2, pp. 1123-1128 Seattle, WA. MIT Press.

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

Ghavamzadeh, M., & Mahadevan, S. (2002). Hierarchically optimal average reward reinforcement learning. *To appear in The Proceedings of the Nineteenth International Conference on Machine Learning.*

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

Kaelbling, L. P. (1993). Hierarchical reinforcement learning: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning* San Francisco, CA. Morgan Kaufmann.

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

Knoblock, C. A. (1994). Automatically generating abstractions for planning. *Artificial Intelligence, 68*(2), 243-302.

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.

Korf, R. E. (1985). Macro-operators: A weak method for learning. *Artificial Intelligence, 5(2)*, 35-77.

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.. In *Proceedings of the 18th International Conference on Machine Learning,*, pp. 361-368 San Mateo, CA. Morgan Kaufmann.

Moore, A. W., & Atkeson, A. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning, 13*, 103-130.

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

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. In *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.* John Wiley, New York.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence, 5(2)*, 115-135.

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

Singh, S. P., & Bertsekas, D. (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems 9,* Cambridge, MA. MIT Press.

Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning, 8*, 323-339.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning.* The MIT Press, Cambridge, MA.

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. In *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.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning, 8(3-4)*, 257-277.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*, 279-292.

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

Zhang, W., & Dietterich, T. (1996). High-performance job-shop scheduling with a time-delay TD(λ) network. In *Advances in Neural Information Processing Systems 8,* Cambridge, MA. MIT Press.