H-learning: A Reinforcement Learning Method to Optimize Undiscounted Average Reward

Prasad Tadepalli* and **DoKyeong Ok**
tadepall@research.cs.orst.edu okd@research.cs.orst.edu
Computer Science Department
Oregon State University
Corvallis, Oregon 97331-3202

12 May 1994

Abstract

In this paper, we introduce a model-based reinforcement learning method called *H-learning*, which optimizes undiscounted average reward. We compare it with three other reinforcement learning methods in the domain of scheduling Automatic Guided Vehicles, transportation robots used in modern manufacturing plants and facilities. The four methods differ along two dimensions. They are either model-based or model-free, and optimize discounted total reward or undiscounted average reward. Our experimental results indicate that H-learning is more robust with respect to changes in the domain parameters, and in many cases, converges in fewer steps to better average reward per time step than all the other methods. An added advantage is that unlike the other methods it does not have any parameters to tune.

1 Introduction

Automatic Guided Vehicles are robots which are used for routine transportation tasks in modern manufacturing plants, hospitals and office buildings [Minoura et al. 1993]. These robots, or AGVs as we shall call them henceforth, are more sophisticated than typical "industrial robots" used on the assembly lines, and yet, side-step many difficult AI problems such as general-purpose vision or autonomous navigation in unexplored territories. Usually, crude sonar sensors provide limited vision and depth perception, and bar-codes on the walls and markings on the floor enable landmark recognition and guided navigation.

However, there is one difficult problem that still needs to be addressed. Optimal scheduling of AGVs is a non-trivial task. At any given time requests might come from several locations to transport materials to specific destinations, and the scheduling system must decide in real-time what action to take in each instant for optimal utilization of resources. In general, there are multiple AGVs, with some routing constraints, such as the allowed direction and maximum speed on each

*This research was supported by the National Science Foundation under grant number IRI:9111231.

route segment, and capacity constraints such as the total weight and volume they can carry, and the total time they can work without recharging. More over, the dynamics of these environments is such that a fixed scheduling algorithm is unlikely to work for all problem distributions. The product lines and machine layouts in manufacturing plants are constantly changing, new patients and doctors move in and out of hospitals, and office buildings are in constant flux. Automatic learning of optimal scheduling algorithms for AGVs appears attractive and promising.

In this paper, we model the learning of scheduling algorithms for AGVs as a Reinforcement Learning (RL) problem. The system receives a reward after completion of each successful transportation task, and its goal is to learn to control the AGV in such a way as to maximize the average reward received in a time step. Learning occurs by systematic temporal propagation of rewards and punishments given by the environment for the actions of the AGV. Since we are assuming that the AGV is in a well-constrained and docile environment, it is reasonable to ignore the issues of low level perception and action, and instead focus on learning to schedule. Since this is essentially a high-level task, we study it using a simulator.

Most approaches to reinforcement learning, including Q-learning [Watkins and Dayan 92] and ARTDP [Barto et al. 1993], optimize the total discounted reward the learner receives. In other words, a reward which is received after one time step is considered equivalent to a fraction of the same reward received immediately. One advantage of discounting is that it yields a finite total reward even for an infinite sequence of actions and rewards. However, discounting encourages the learner to sacrifice long-term benefits for short-term gains, since the impact of long-term rewards on action selection decreases exponentially with time. As pointed out by Schwartz, many real world domains which we would like to apply RL to do not have a natural interpretation or need for discounting [Schwartz 93]. In spite of this, many researchers used techniques developed for optimizing discounted rewards for such problems, and evaluated them with respect to cumulative or average rewards without discounting [Kaelbling 90, Lin 92, Mahadevan and Connell 91]. Since the optimal policies for the discounted reward are not always the same as the optimal policies for the undiscounted reward, there is reason to believe that better results can be obtained by using learning techniques that directly optimize undiscounted rewards in these domains. Although optimizing undiscounted average rewards is a well-studied problem in dynamic programming literature [Bertsekas 87], its adaptation to RL is fairly recent, Schwartz's R-learning being one of the best known examples [Schwartz 93]. In this paper, we present a new RL method, called *H-learning*, that optimizes undiscounted average reward per step and compare it with three other previously published RL methods in the domain of AGV scheduling.

RL methods are divided into "model-free methods" that do not explicitly model the effects of actions, and "model-based methods" that learn and use action models simultaneously while learning optimal control [Barto et al. 1993]. Q-learning [Watkins and Dayan 92] and R-learning [Schwartz 93] are examples of model-free methods, while Adaptive Real-Time Dynamic Programming (ARTDP) is an example of a model-based method [Barto et al. 1993]. Previously it has been found that while model-free methods have simpler and more efficient update procedures, model-based methods converge in fewer steps [Barto et al. 1993]. Unlike Q-learning and R-learning, and like ARTDP, H-learning is a model-based method, and is similar to the "Algorithm B" of Jalali and Ferguson [Jalali and Ferguson 89]. In this paper, we derive H-learning from the classical successive approximation algorithm for optimizing undiscounted rewards [Bertsekas 87].

We evaluated H-learning in a simple version of AGV domain and compared its performance to ARTDP, Q-learning, and R-learning. We draw the following conclusions:

- All methods converge to the optimal values when the short term rewards coincide with the long term rewards, and the parameters are optimally tuned. Model-based methods (ARTDP and H-learning) converge in fewer steps than the model-free methods (Q-learning and R-learning) but take more time for each update.
- When the domain parameters are changed so that the optimal actions for short term rewards do not coincide with those for long-term rewards, the discounted methods (Q-learning and ARTDP) are either slower in converging or converge to the non-optimal values depending on the discount factor.
- On the whole, H-learning converges in fewer steps to better optimal values, and is more robust with respect to changes in the domain parameters. Unlike the other methods, it does not have any parameters to tune, and hence is to be preferred in cases where extensive exploration of the parameter space is prohibitive.
- All reinforcement learning methods including H-learning are sensitive to exploration, although this dependence is also a function of the reward system.

In the future, we plan to study scale-up issues in H-learning including function approximation, task decomposition, and abstraction in the AGV domain. In the rest of the paper, we summarize the four learning methods, describe the AGV domain, and report on experiments comparing the four methods. We conclude with a discussion of future research issues.

2 Reinforcement learning methods

Markovian Decision Problems (MDP) are described by a set of n discrete states S and a set of actions A available to an agent. The set of actions which are applicable in a state i are denoted by $U(i)$ and are called *admissible*. The actions are stochastic in that an action u in a given state $i \in S$ results in state j with a fixed probability $p_{ij}(u)$. We also have a finite immediate reward for executing an action u in state i, given by $r(i, u)$. Time is treated as a sequence of discrete steps $t = 0, 1, 2, \ldots$. A *policy* $\mu = \langle \mu(1), ..., \mu(n) \rangle$ is a mapping from states to actions, such that the controller executes action $\mu(i) \in U(i)$ when in state i. A stationary policy is a policy which does not change with time. By "policy," we mean stationary policy from now on.

2.1 The Role of Discounting

Let a controller using a policy μ take the agent through states $s_0, \ldots, s_t$ in time 0 thru t, with some probability. We call this a run, and its total reward $r^\mu(s_0, t) = \sum_{k=0}^{t-1} r(s_k, \mu(s_k))$.

The expected total reward, $E(r^\mu(s_0, t))$, is a good candidate to optimize; but if the controller has infinite horizon, i.e., as t tends to ∞, this value also approaches ∞. One way to make the sum finite is by exponentially discounting future rewards. In other words, we optimize $\lim_{t\to\infty} E(r_0 + \gamma r_1 + \ldots + \gamma^t r_t))$, where r_i is the immediate reward after time i in the future and γ is the discount factor between 0 and 1. A number of RL algorithms including Q-learning [Watkins and Dayan 92] and Adaptive RTDP [Barto et al. 1993] are designed to learn policies that maximize discounted total rewards.

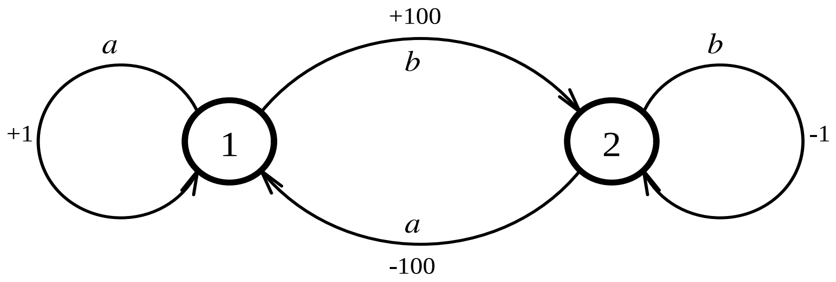

Figure 1: A problem that shows why discounting may cause sub-optimal control

While discounting solves the problem of infinite totals, it is not clear that discounted totals is what we *want* to optimize. As Schwartz pointed out, even researchers who use learning methods that optimize discounted totals evaluate their systems using a different, but more natural, measure – average reward per step [Schwartz 93]. The average expected reward of the policy μ starting with state s_0, denoted by $\rho^\mu(s_0)$, is the limit of the average expected reward per step over time t as t approaches ∞.

$$\rho^\mu(s_0) = \lim_{t\to\infty} \frac{1}{t} E(r^\mu(s_0, t))$$

If optimal average rewards is what we really want, optimizing the discounted totals could very well lead to bad policies, as the following example illustrates. In Figure 1, the controller has to choose between action a and action b. If the system is in state 1, choosing action a gives an average reward of 1, but choosing action b gives a one-time immediate reward of 100. If the future rewards are discounted, the system takes action b, gets an immediate reward of 100, and stays in state 2, since going back to 1 is costly. However, if the goal is optimizing the average reward, the system should stick to state 1 and keep executing action a. In other words, a discounted optimizing function makes it short-sighted, and it can be made to work arbitrarily poorly by increasing its short-term rewards at the cost of its long-term rewards. While this example might look unduly contrived, we will later show that this situation naturally arises in our domain and is the underlying cause of the poor performance of ARTDP and Q-learning in some cases.

2.2 H-learning

In this section, we introduce H-learning.

A set of states is ergodic with respect to a policy if every state is reachable from every other state in that set with some probability when the agent is using that policy, and there is no transition from any state in the set to one outside that set. In what follows, we assume that the exploration strategy guarantees that every state in S is visited with some probability, so that the total set of states S is ergodic with respect to every policy during training.

Under these conditions, the initial finite rewards obtained in going from a state s to s' do not contribute anything to the expected long-term average reward per time step. Hence, if μ is an intermediate policy used in training, then $\rho^{\mu}(s) = \rho^{\mu}(s')$. We denote this simply as $\rho(\mu)$, and consider the problem of finding an optimal policy μ^* that maximizes $\rho(\mu)$.

Even though $\rho(\mu)$ is the same for every starting state, the total expected reward in time t may not be the same for different starting states. The total reward for a starting state s in time t for a policy μ can be conveniently denoted by $\rho(\mu)t + \epsilon_t(s)$. As $t \to \infty$, $\epsilon_t(s)$ converges to a finite limit $h(s)$.[1] Hence $h(s)$ can be interpreted as the expected long-term *differential* reward (or *eld*-reward) for being in state s. The following theorem is proved in [Bertsekas 87].

Theorem 1 *If a scalar ρ and an n-dimensional vector h satisfy the recurrence relation*

$$h(i) = \max_{u \in U(i)} \{r(i,u) + \sum_{j=1}^{n} p_{ij}(u)h(j)\} - \rho, i = 1, \ldots, n. \tag{1}$$

then ρ is the optimal average reward $\rho(\mu^*)$, and μ^* attains the above maximum for each state i.

Intuitively, this can be explained as follows. In going from a state i to the best next state j, the system gained an immediate reward $r(i,u)$ instead of the average reward ρ. After convergence, the difference between these two must equal the difference between the *eld*-reward in state i and the expected value of the *eld*-reward of the state after executing u. This is also the recurrence relation used by Schwartz to derive R-learning [Schwartz 93], except he used σ to denote the h values. Note that if there is one solution to equation (1), infinitely many solutions can be generated by increasing all the h values by a fixed amount. However, all these sets of h values will result in the same set of optimal policies μ^*, since the optimal action in a state is only determined by the relative differences between the h values. To obtain a unique h vector as the solution, the h value of an arbitrarily chosen "reference state," is usually set to 0. For the system in Figure 1, $\rho = 1$ for the optimal policy of always executing action a. Setting $h(1) = 0$, $h(2) = -101$ satisfies the recurrence relations in (1).

Bertsekas shows that the above recurrence relation can be solved by synchronous successive approximation of the h vector [Bertsekas 87]. H-learning is an on-line asynchronous version of this algorithm that also learns action models simultaneously as it learns the h values and controls the system using them (see Figure 2). It is also similar to the "Algorithm B" of Jalali and Ferguson [Jalali and Ferguson 89]. It estimates the probabilities $p_{ij}(a)$ and rewards $r(i,a)$ by straightforward counting, and makes the so called "certainty equivalence assumption" that the current estimates are the true values while updating the h values [Bertsekas 87]. It updates the h value of the current state i using equation (1). Unlike the Algorithm B, which estimates the parameters of its action models before estimating the optimal policy, H-learning estimates them simultaneously. Since it is

[1] Strictly speaking, this is not always the case; but it does not seriously affect the argument. See [Bertsekas 87] for details.

not important to converge to a unique set of h values as long as the optimal policy is found, it does not use any reference state, also unlike the Algorithm B.

1. Let $N(i,u)$ be the number of times action u was executed from state i, and let $N(i,u,j)$ be the number of times it resulted in state j. Initialize these, the matrices $p_{ij}(u)$, $r(i,u)$, $h(i)$, and the scalar ρ to 0's. $U_{best}(i)$ is the set of optimal actions in state i and is initialized to $U(i)$. T is the total number of steps that an apparently optimal action was executed and is initialized to 0. Pick i to be some random current state.

2. Repeat

   (a) If the exploration strategy suggests a random action, take a random action from i, else execute the action $a \in U_{best}(i)$. Let k be the resulting state, and r' be the immediate reward received.

   (b) $N(i,a) \leftarrow N(i,a) + 1$

   (c) $N(i,a,k) \leftarrow N(i,a,k) + 1$

   (d) $p_{ik}(a) \leftarrow N(i,a,k)/N(i,a)$

   (e) $r(i,a) \leftarrow r(i,a) + (r' - r(i,a))/N(i,a)$

   (f) If the executed action $a \in U_{best}(i)$, then

      - $T \leftarrow T + 1$
      - $\rho \leftarrow \rho + (r' - h(i) + h(k) - \rho)/T$

   (g) Let $H(i,u) = r(i,u) + \sum_{j=1}^{n} p_{ij}(u)h(j)$

      - $U_{best}(i) \leftarrow \{v \mid H(i,v) = \max\limits_{u \in U(i)} H(i,u)\}$
      - $h(i) \leftarrow H(i,a) - \rho$, where $a \in U_{best}(i)$

   (h) $i \leftarrow k$

   Until convergence or MAX-STEPS times.

Figure 2: The H-learning Algorithm

The recurrence relation (1) also involves ρ which needs to be estimated. Like most RL methods, and unlike the Algorithm B, to ensure that all reachable states are explored with sufficient frequency, H-learning makes random moves with some fixed probability. Such "exploratory" moves make the estimation of ρ slightly complicated. Simply averaging over non-exploratory moves in the training run would not do, because the exploratory moves could make the system visit states that it never visits if it were simply following the optimal policy, and accumulate rewards received by optimal actions in these unusual states. Instead, we borrow the idea that Schwartz used to estimate the average reward [Schwartz 93]. From the recurrence relation (1), after convergence, for any best move u in any state i, $\rho = r(i,u) - h(i) + \sum_{j=1}^{n} p_{ij}(u)h(j)$. Hence, the current ρ can be estimated by cumulatively averaging $r(i,u) - h(i) + h(j)$, whenever an apparently optimal action u is executed in state i resulting in state j. In this algorithm the set of optimal actions in each state i is stored explicitly as an array $U_{best}(i)$. [Ok 94] describes a variant of H-learning where it is not explicitly represented, which has slightly better performance.

2.3 A summary of other Reinforcement Learning methods

H-learning can be seen as a cross between R-learning [Schwartz 93] and ARTDP [Barto et al. 1993]. Like R-learning and unlike ARTDP, H-learning optimizes undiscounted average reward per step. Like ARTDP and unlike R-learning, H-learning is model-based.

R-learning uses the following formula to update its R-values, assuming that the agent took the action u in state i, got an immediate reward r', and reached a state j.

$$R(i,u) \leftarrow R(i,u) + \beta(r' - \rho + U_R(j) - R(i,u)) \tag{2}$$

$U_R(j)$ is defined as $Max_{u \in U(j)} R(j,u)$ and corresponds to $h(j)$ of H-learning. They both denote the expected long-term differential reward of being in the state j. R-learning updates the ρ incrementally using

$$\rho \leftarrow \rho + \alpha(r' - U_R(i) + U_R(j) - \rho) \tag{3}$$

ARTDP is similar to H-learning in that they both use models, but different in that it optimizes the discounted total reward $f(i)$ for each state i [Barto et al. 1993]. Its update equation is given by

$$f(i) \leftarrow \max_{u \in U(i)} \{ r(i,u) + \gamma \sum_{j=1}^{n} p_{ij}(u) f(j) \} \tag{4}$$

Watkins's Q-learning differs from H-learning in both the above dimensions. Unlike H-learning, it is both discounted and model-free. Its update equation is given by

$$Q(i,u) \leftarrow Q(i,u) + \beta(r + \gamma U_Q(j) - Q(i,u)) \tag{5}$$

where $U_Q(j)$ is defined as $Max_u Q(j,u)$, and corresponds to $f(j)$ of ARTDP.

Since ARTDP uses action models to propagate more information in each step than Q-learning, it is shown to converge in fewer steps [Barto et al. 1993]. We will see that the same relationship holds between H-learning and R-learning.

All the above RL methods, except H-learning, have one or more parameters. ARTDP has γ, Q-learning has γ and β, and R-learning has α and β. The performances of all these methods are sensitive to their parameters, and hence it becomes necessary to tune them to optimize their performance.

3 AGV scheduling

To compare the various learning algorithms, a small AGV domain shown in Figure 3 was used. There are two job generators on the left, one AGV, and two destination conveyor belts on the

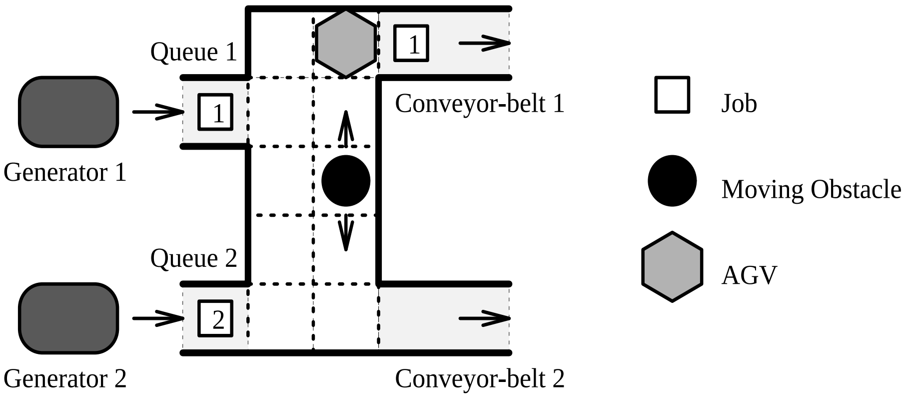

Figure 3: An AGV domain

right. Each job generator produces jobs and puts them on its queue as soon as it is empty. The AGV loads and carries a single job at a time to its destination conveyor belt.

Each job generator can generate either of two types of jobs when its queue is empty. Job 2, destined to belt 2, has a reward of 1 unit, while job 1, destined to belt 1, receives a reward K, when delivered. The probability of generating job 1 is p for generator 1, and q for generator 2.

The AGV moves on two lanes of 5 positions each, and can take one of six actions at a time: DO-NOTHING, LOAD, MOVE-UP, MOVE-DOWN, MOVE-ASIDE, and UNLOAD. To LOAD a job, the AGV must be in the position next to the queue. To UNLOAD a job, it must be next to the proper conveyor belt. To make this domain more interesting, a moving obstacle is introduced. It randomly moves up or down in each instant, but can only stay in the right lane and cannot stand still. The AGV and the obstacle can both move in a single time step. If the AGV collides with the obstacle, the positions of the AGV and the obstacle are not changed.

A state is specified by the two job numbers in the queues, the locations of the AGV and the obstacle, and the job number on the AGV. There are a total of 540 different states in this domain. The reinforcement for collision with the obstacle is -5. The goal of the AGV is to maximize the average reward received per unit time.

By varying the reward ratio of the jobs and/or the job mixes produced by the job generators, the optimal policy is changed. For example, when K =1, and both the job generators produce type 1 jobs with very low rates p and q, the AGV should unload jobs from queue 2 much more frequently than from queue 1 because the number of time steps needed to transport type 2 jobs from queue 2 to belt 2 is much smaller than that needed to move them from queue 1 to belt 2. But, when both the job generators produce jobs of type 1 with high rate, and $K = 9$, the AGV should unload jobs from queue 1 much more frequently than from queue 2, because the increased value of job 1 more than compensates for the extra distance. It is, in general, hard to predict the best policy given different values of p, q, and K.

4 Experimental results

In this section, we present the results of comparing H-learning with ARTDP, Q-learning, and R-learning in the AGV domain. But first, a few points about our evaluation method are in order. In reinforcement learning literature, the training and testing phases are not usually distinguished. Performance is measured on-line, i.e., during training. There are, however, two problems with this way of evaluation. One problem is that it makes it difficult to determine the actual performance of the system at any given time because the performance is constantly improving. The second problem is that during learning the system is still executing random exploratory actions, which gives a distorted picture of the average reward. To circumvent these two problems, we divided each trial into several alternating training and testing phases. Learning and exploration were frozen during the testing phase, and evaluation was stopped during the training phase. One consequence of this evaluation is that the performance increased or stayed the same until the exploration was very high. In Experiment 1 and Experiment 2, 50% of the training actions were randomly chosen in all the four algorithms. In Experiment 3, the performances of each method with various exploration rates are compared.

4.1 Experiment 1

In the first experiment, we compared H-learning with ARTDP, Q-learning, and R-learning in two situations: one is a conflict-free case where the long-term optimal policy is the same as the short-term optimal policy; the other is a conflict case where they are not the same.

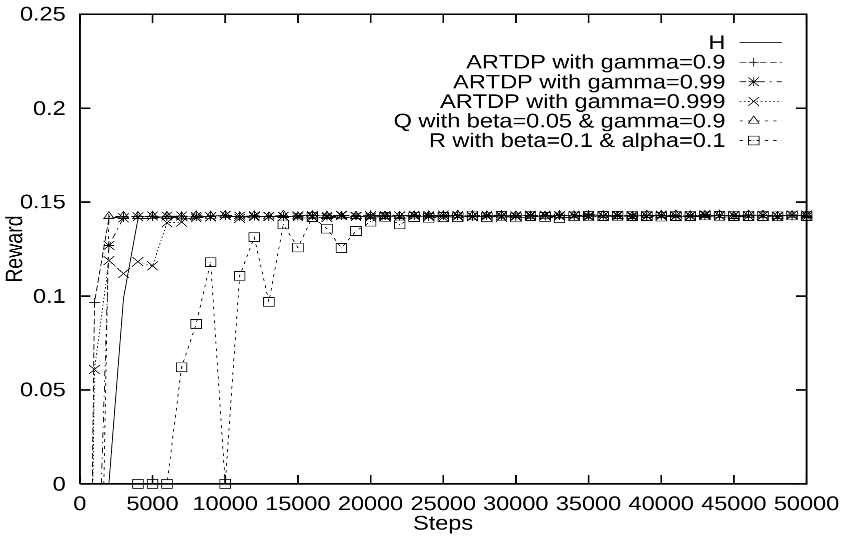

Figure 4: Median rewards per step of H-learning, ARTDP, Q-learning and R-learning during learning with p=0.5 and q=0.0 when K=1

Each experiment was repeated for 30 trials for each algorithm. Every trial started from a random initial state, and consisted of 50 sequential training phases of 1,000 actions each for the conflict-free case, and 100 sequential training phases of 20,000 actions each for the conflict case. Each training phase was followed by a testing phase of 100,000 actions in both cases. While testing a policy, the AGV was made to follow the learned policy without any random explorations, and the average reward per action was recorded. Since the trials consisted of a small number of outliers that distorted the means, following [Mahadevan 94], we plotted the *median* rewards of 30 trials against the number of training actions.

ARTDP was run with γ=0.9, 0.99, and 0.999. We denote these three versions with ARTDP$_{0.9}$, ARTDP$_{0.99}$, and ARTDP$_{0.999}$. The parameters for Q-learning and R-learning are tuned by trial and error to get the best performance. For the conflict-free case, the parameters for Q-learning are β=0.05 and γ=0.9, and for R-learning, β=0.1, α=0.1. For the conflict case, the parameters for Q-learning are β=0.2 and γ=0.99, and for R-learning, β=0.01, α=0.005.

The probability p of the job generator 1 producing job 1 is set to 0.5, and the probability q of the job generator 2 producing job 1 is set to 0.0. In other words, the generator 1 produces both types of jobs with equal probability, while the generator 2 always produces type 2 jobs.

For the conflict-free case the reward ratio K is set to 1(See Figure 4). So serving queue 2 is the short-term optimal policy as well as the long-term optimal policy. In this case, all methods found the optimal policy. ARTDP$_{0.9}$, ARTDP$_{0.99}$, and Q were slightly faster to converge than H. ARTDP$_{0.999}$ was slightly slower and R was the slowest.

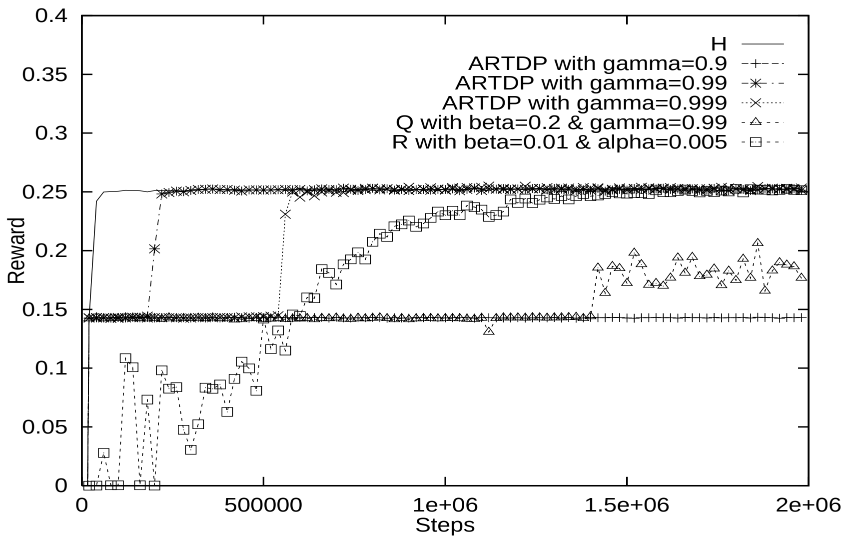

Figure 5: Median rewards per step of H-learning, ARTDP, Q-learning and R-learning during learning with p=0.5 and q=0.0 when K=5

For the conflict case, the reward ratio K is set to 5, making the unloading of jobs of type 1 five times more rewarding than the unloading of jobs of type 2. For this setting of parameters, optimizing the long-term reward conflicts with optimizing the short-term reward. Ignoring the obstacle, the long-term optimal policy in this case is to exclusively serve queue 1 because it generates high reward jobs with a higher probability than the other queue. If γ is sufficiently low, however, the short-term policy of exclusively serving queue 2 appears better because transporting a job from queue 2 to belt 2 takes less time.

As we can see in Figure 5, H-learning, ARTDP with γ=0.99 and 0.999, and R-learning found the optimal policy. H-learning converged to the optimal policy in the fewest steps, successively followed by $\text{ARTDP}_{0.99}$, $\text{ARTDP}_{0.999}$, and R-learning. But, Q-learning and $\text{ARTDP}_{0.9}$ could not converge to the optimal policy even after 2 million steps. Q-learning (with γ=0.99) learned a policy that served both the queues but sometimes chose non-optimal queues to load from. $\text{ARTDP}_{0.9}$ learned the least optimal policy of the group, which consisted of exclusively serving queue 2. While the variance of different trials of Q-learning at the end of training was 0.005, the variances of the other three methods were less than 0.00001.

The reason that H-learning converges in fewer steps is that it propagates more information in each step, by taking the *max* over the values of all the neighboring states before each update. This also requires H-learning to learn and store the action models explicitly, increasing the CPU-time for each update. We found that for the same number of steps, H-learning takes approximately 1.8 times the CPU time taken by ARTDP, 3.5 times that of R-learning, and 4.3 times that of Q-learning. However, the reduced number of steps more than compensates for this slower updating, making it converge in less CPU-time than all the other methods in the conflict case.

Compare the results of ARTDP for various γ values in Figure 5. The speed of convergence of $\text{ARTDP}_{0.99}$ is higher than that of $\text{ARTDP}_{0.999}$. Further reducing the γ value to 0.9 makes ARTDP converge even faster, but to a non-optimal policy in the conflict case! Thus, setting γ correctly is very important for ARTDP. An advantage of H-learning is that it has no parameters to tune.

The results of Experiment 1 show that when short-term optimization coincides with long-term optimization, H-learning performs slightly worse than ARTDP and Q-learning, but much better than R-learning. When short-term optimization conflicts with long-term optimization, discounting methods such as ARTDP and Q-learning either take too long to converge or, if γ is low, converge to a non-optimal policy. H-learning converges to the optimal policy in fewer steps than the other three methods in such cases.

4.2 Experiment 2

In the second experiment, we wanted to test the robustness of H-learning with respect to changes in the domain parameters p, q, and K. p and q are varied from 0.0 to 1.0 in steps of 0.25. 3 values are tried for K: 1, 5, and 9. Hence, there are a total of 75 different cases. The results for each case were obtained by taking the median over 10 trials of 300,000 training actions. Since ARTDP usually performs better than Q-learning and R-learning, and requires fewer parameters to tune, H-learning was compared only with ARTDP. Like Experiment 1, we ran $\text{ARTDP}_{0.9}$, $\text{ARTDP}_{0.99}$, and $\text{ARTDP}_{0.999}$.

Table 1 shows the methods that found the optimal policy with 300,000 steps for the 75 cases. H-learning found the best policies among the four methods for all 75 cases in less than 300,000 actions. $\text{ARTDP}_{0.9}$, $\text{ARTDP}_{0.99}$, and $\text{ARTDP}_{0.999}$ found the best policies for 48, 61, and 40 cases respectively. In 11 of the 75 cases none of the three versions of ARTDP could find the optimal policy.

Let's compare the convergence speed of each methods. Table 2 shows the methods that find the

K	$q \setminus p$	0.00	0.25	0.50	0.75	1.00
1	0.00	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
	0.25	H	H,$A_{0.9}$	H,$A_{0.9}$	H,$A_{0.99}$	H
	0.50	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$	H,$A_{0.9}$,$A_{0.99}$	H,$A_{0.9}$,$A_{0.99}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
	0.75	H,$A_{0.9}$,$A_{0.99}$	H,$A_{0.9}$,$A_{0.99}$	H,$A_{0.9}$,$A_{0.99}$	H,$A_{0.9}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
	1.00	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
5	0.00	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H	H,$A_{0.99}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
	0.25	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H	H	H,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
	0.50	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H	H,$A_{0.99}$	H	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
	0.75	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.99}$	H	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
	1.00	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.99}$	H,$A_{0.99}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
9	0.00	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.99}$	H,$A_{0.99}$	H,$A_{0.99}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
	0.25	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H	H,$A_{0.99}$	H,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
	0.50	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.99}$	H,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
	0.75	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H	H	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
	1.00	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.99}$	H,$A_{0.99}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$

Table 1: Methods that found the optimal policy within 300,000 steps for given K, p, and q, where $A_{0.9}$=ARTDP$_{0.9}$, $A_{0.99}$=ARTDP$_{0.99}$, and $A_{0.999}$=ARTDP$_{0.999}$.

optimal policy in the fewest steps for the 75 cases. In 54 out of the 75 cases, H-learning converged to the best policy in the fewest steps. ARTDP$_{0.9}$, ARTDP$_{0.99}$, and ARTDP$_{0.999}$ converged to the best policy in the fewest steps in 45, 20, and 6 cases respectively. Out of the 21 cases that H-learning was found to be not the fastest, in 16 cases K=1, in 1 case K=5, and in 4 cases K=9. In 13 of the 21 cases, it followed the best algorithm closely, i.e., in less than 20,000 steps. In the worst case, it lagged behind by 100,000 steps, and in only 4 cases, the lag was more than 50,000. The mean lag was 32,000 steps and the median was 20,000.

K	$q \setminus p$	0.00	0.25	0.50	0.75	1.00
1	0.00	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H,$A_{0.9}$,$A_{0.99}$
	0.25	H	$A_{0.9}$	$A_{0.9}$	H	H
	0.50	$A_{0.9}$	$A_{0.9}$	$A_{0.9}$	$A_{0.9}$	H,$A_{0.9}$
	0.75	$A_{0.9}$	$A_{0.9}$	$A_{0.9}$	$A_{0.9}$	$A_{0.9}$,$A_{0.99}$
	1.00	H,$A_{0.9}$	$A_{0.9}$	$A_{0.9}$	$A_{0.9}$	H,$A_{0.9}$,$A_{0.99}$
5	0.00	H,$A_{0.9}$,$A_{0.99}$,$A_{0.9}$	H	H	H	H,$A_{0.9}$
	0.25	H,$A_{0.9}$	H	H	H	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$
	0.50	H,$A_{0.9}$	H	H	H	H,$A_{0.9}$,$A_{0.99}$
	0.75	H,$A_{0.9}$	$A_{0.9}$	H	H	H,$A_{0.9}$,$A_{0.99}$
	1.00	H,$A_{0.9}$	H,$A_{0.9}$	H	H	H,$A_{0.9}$,$A_{0.99}$
9	0.00	H,$A_{0.9}$,$A_{0.99}$,$A_{0.999}$	H	H	H	H,$A_{0.9}$,$A_{0.99}$
	0.25	$A_{0.9}$,$A_{0.99}$	H	H	H	H,$A_{0.9}$,$A_{0.99}$
	0.50	H	$A_{0.99}$	H	H	$A_{0.9}$,$A_{0.99}$
	0.75	H,$A_{0.9}$	$A_{0.9}$	H	H	H,$A_{0.9}$,$A_{0.99}$
	1.00	H,$A_{0.9}$	H,$A_{0.9}$	H	H	H,$A_{0.9}$,$A_{0.99}$

Table 2: Methods that found the optimal policy in the fewest steps for given K, p, and q, where $A_{0.9}$=ARTDP$_{0.9}$, $A_{0.99}$=ARTDP$_{0.99}$, and $A_{0.999}$=ARTDP$_{0.999}$.

It is interesting to note that while γ=0.99 is preferable for ARTDP if finding the optimal value is important, a value of 0.9 is to be preferred if faster convergence is more important. It is difficult to guess the correct value of γ for each new situation, because if it is too low, ARTDP converges to a non-optimal policy and if it is too high, it converges very slowly.

In summary, our experiments indicate that H-learning is more robust with respect to changes in the domain parameters, and in many cases, converges in fewer steps to more optimal values than all the other methods at the cost of increased time per update.

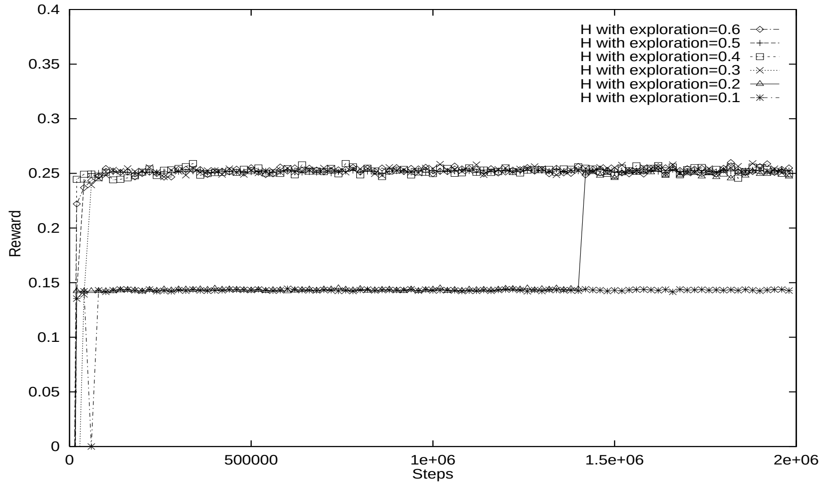

Figure 6: The performances of H-Learning with various exploration rates

4.3 Experiment 3

The performance of the reinforcement learning methods is sensitive to exploration and reward system. In the third experiment, we compared the performances of H-learning, ARTDP, Q-learning, and R-learning as a function of exploration rate in the AGV domain. We experimented with semi-uniform exploration strategy with random moves being chosen at rate 10%, 20%, 30%, 40%, 50% and 60%, with K=5, p=0.5, and q=0.0. The results were obtained by taking the mean over 10 trials of 2 million training actions.

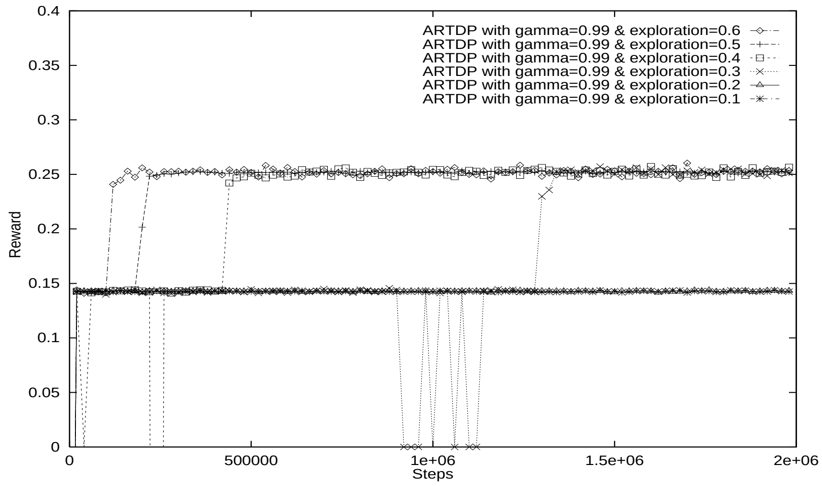

Figure 7: The performances of ARTDP with various exploration rates

Figure 6 shows the performances of H-learning. With more than 20% randomly chosen actions, H-learning always found the optimal policy very fast. But 20% exploration rate made H-learning slow to find the optimal policy, and 10% made it unable to find the optimal policy. With the reward system in AGV domain, H-learning performs better with a high exploration rate.

γ for ARTDP is set to 0.99 because it showed the best performance of ARTDP in the first experiment with the same K, p , and q. As Figure 7 shows, the higher ithe exploration rate, the better ARTDP's performance. 60% exploration rate made ARTDP find the optimal policy very fast. 30% exploration rate made ARTDP find the optimal policy even though the convergence was slow. However, Less exploration rate than 30% made ARTDP unable to find the optimal policy. ARTDP needs more explorations than H-learning for the same performance in this domain.

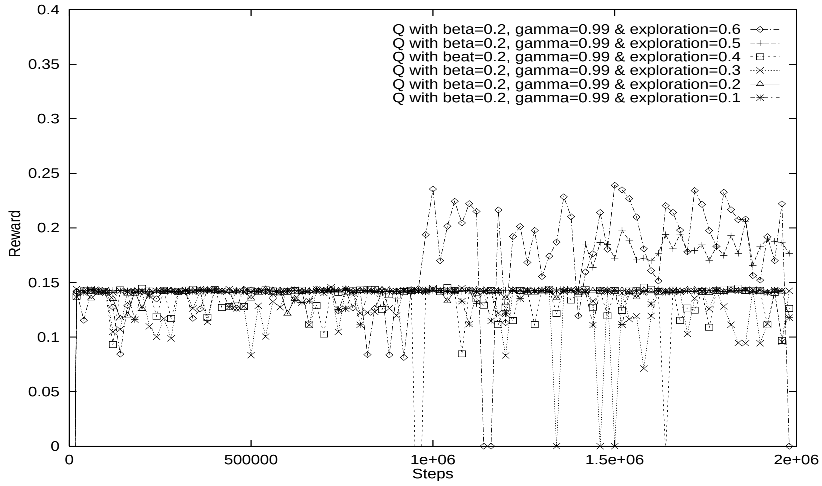

Figure 8: The performances of Q-Learning with various exploration rates

β and γ for Q-learning are 0.2 and 0.99 respectively. Figure 8 shows the results of Q-learning which look quite different from those of H-learning and ARTDP. The performance of ARTDP with 60% exploration rate was the best, but it could not find the optimal policy even though the AGV served Queue 1 as well as Queue 2. Q-learning with 50% exploration rate also made the AGV serve Queue 1 as well as Queue 2, but the performance was worse than that of Q-learning with 60% exploration rate. The performances of Q-learning with less exploration rate than 50% were very poor. Their policies were worse than a policy serving just Queue 2 only. Q-learning requires a very high exploration rate for the conditions of this experiments.

The performance of R-learning is shown in Figure 9. The parameters for R-learning are also the same as the parameters in the first experiment. Up to 1 million steps, the performances of R-learning varied a lot. But, after that, R-learning found the optimal policy for all exploration rates.

As we have seen, the performance of all learning methods is sensitive to exploration rates. H-learning and ARTDP with high exploration rate find the optimal policy even though there is a conflict. But, they can not find the optimal policy using a lower exploration rate. The sensitivity of the performance of H-learning and ARTDP to exploration is also demonstrated in [Ok 94]. Q-learning can not find the optimal policy using any exploration rate. However, R-learning finds the optimal policy with any exploration rate. Less exploration does not effect R-learning as it does

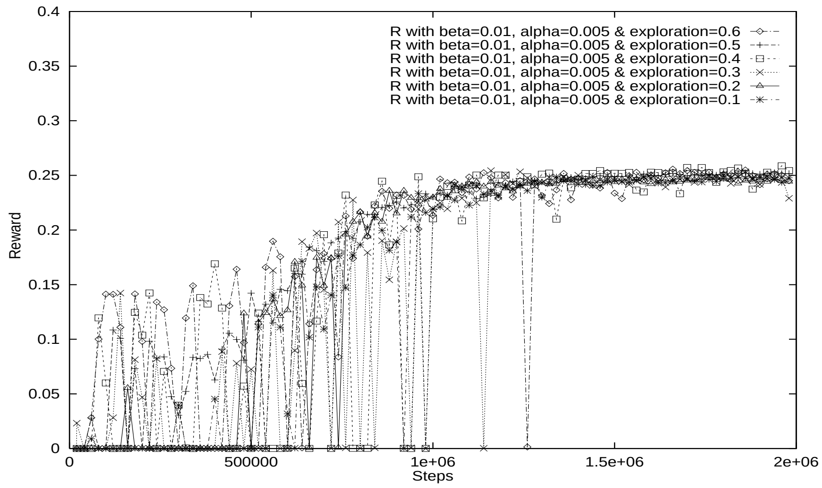

Figure 9: The performances of R-Learning with various exploration rates

the other learning methods. The reason that R-learning with a lower exploration rate finds the optimal policy is that R-learning has some inherent tendency to explore when many states have an immediate reward which is less than the average reward ρ. In AGV domain, most states have zero immediate rewards. R-learning reduces the R-values of state-action pairs executed under this condition. This enhances the chance of other actions to be selected in the future.

5 Discussion and Future Work

Our experimental results showed that optimizing undiscounted average rewards is a more robust strategy than optimizing discounted rewards, and yields better average reward when the short-term optimal policy is in conflict with long-term optimal policy. Our results are also consistent with the results of Barto et al. that showed that model-based methods converge in fewer steps than model-free methods [Barto et al. 1993]. However, unlike in the experiments of Barto et al., the updating cost for each step in H-learning was only 4.3 times higher than the corresponding model-free method, while the number of steps needed for convergence is at least an order of magnitude smaller. In the case where the long-term and short-term optimal policies conflict, H-learning converges not only in fewer steps, but also in less time. This, however, could change in other cases and other domains.

Recently, Mahadvan has shown that Q-learning works better than R-learning in his robot simulator domain, when compared with respect to R-learning's evaluation function [Mahadevan 94]. However, it appears that there is no conflict between the two evaluation functions in his domain. Our results also suggest that Q-learning converges faster than R-learning under such conflict-free conditions. It would be interesting to see the result of conflict cases in his robot simulator domain.

Singh derived some average-reward RL algorithms for policy evaluation and optimal control questions from the recurrence relation (1) [Singh 94]. All these algorithms are model-free, and hence different from H-learning.

Much remains to be done to scale H-learning to more realistic problem sizes. Our table-based approach to store the probability transition matrix and the H-values can be generalized by parameterization and function approximation. There is, indeed, no need to explicitly store the entire probability transition matrix in our domain. The action models can be easily decomposed in that each action effects only a small number of features. In particular, the probabilities in the transition probability matrix are completely determined by p and q, which represent the probabilities of generation of job 1 at the two queues. The obstacle can move in each direction with probability 0.5. The new position of the AGV is a stochastic function of its old position and the position of the obstacle. Similarly, the reward function is determined by the reward ratio K. Having this *a priori* knowledge can immensely simplify the learning of action models. The only reason for representing and learning all the probabilities explicitly in this paper is to make the comparisons to model-free learning methods fair. H-learning performs better than the other methods even without exploiting this obvious source of knowledge.

Like many real world domains, the AGV domain exhibits a lot of structure, which can be readily exploited by abstracting the state space. For example, if the AGV is moving down at time t, it is likely that it should be moving down at time $t+1$ as well. Similarly if an obstacle is at position x at time t, it is likely to be near x at time $t+1$. The function approximation methods can achieve better results by exploiting this temporal and spatial locality, which is predominant in many domains. One way to exploit this is to represent the positions of the AGV and the obstacle by real numbers rather than by arbitrarily discretized position labels. This opens up many interesting issues for future research such as dealing with actions that span more than one unit of time (e.g., going to the end of the corridor), and approximation methods for functions over real-valued attributes. [Ok 94] contains some specific proposals to do generalization of the value function using nearest neighbor approach. We also plan to explore the problems of controlling multiple AGVs and the role of teaching.

6 Conclusions

We introduced a model-based reinforcement learning method called H-learning that optimizes undiscounted average reward, and compared it to three previously published learning methods in the domain of Automatic Guided Vehicles. We showed that when the long-term optimal policy and the short-term optimal policy are in conflict, undiscounted learning converges to a better optimal value. The model-based methods usually converge in fewer steps, but consume several times more CPU-time in each step than the model-free methods. On the whole, it appears that H-learning is superior to the other three methods because it is more robust to changes in the domain parameters, gets better average rewards in fewer steps, and requires no parameter tuning. The increased CPU-time needed for each update appears to be modest in our domain. We plan to study the open problems in reinforcement learning such as function approximation, abstraction and teaching in the context of undiscounting learning.

Acknowledgments

We thank Toshimi Minoura for introducing us to the AGV domain, and Tom Dietterich for introducing us to Reinforcement Learning. We thank both of them and Sridhar Mahadevan for many helpful discussions.

References

[Barto et al. 1993] Barto, A. G., Bradtke, S. J., and Singh, S. P. Learning to Act using Real-Time Dynamic Programming, submitted to AI Journal special issue on Computational Theories of Interaction and Agency, 1993.

[Bertsekas 87] Bertsekas, D. P. *Dynamic Programming: Deterministic and Stochastic Models*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.

[Jalali and Ferguson 89] Jalali, A. and Ferguson, M. Computationally Efficient Adaptive Control Algorithms for Markov Chains. In *IEEE Proceedings of the 28'th Conference on Decision and Control*, Tampa, FL, 1989.

[Kaelbling 90] Kaelbling, L. P. *Learning in Embedded Systems*, MIT Press, Cambridge, MA, 1991.

[Lin 92] Lin, L. J. Self-improving Reactive Agents based on Reinforcement Learning, Planning, and Teaching. *Machine Learning*, 8:279-292, 1992.

[Mahadevan and Connell 91] Mahadevan, S. and Connell, J. Automatic Programming of Behavior-based Robots Using Reinforcement Learning. In *Proceedings of AAAI-91*, MIT Press, Cambridge, MA, 1991.

[Mahadevan 94] Mahadevan, S. To Discount or Not To Discount in Reinforcement Learning: A Case Study Comparing R-learning and Q-Learning. In *Proceedings of International Machine Learning Conference*, New Brunswick, NJ, 1994.

[Minoura et al. 1993] Minoura, T., Choi, S., and Robinson, R. Structural Active Object Systems for Manufacturing Control. *Integrated Computer-Aided Engineering*, 1(2), 121-136, 1993.

[Schwartz 93] Schwartz, A. A Reinforcement Learning Method for Maximizing Undiscounted Rewards. In *Proceedings of the Tenth International Conference on Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1993.

[Singh 94] Singh, S. P. Reinforcement Learning Algorithms for Average-Payoff Markovian Decision Processes. In *Proceedings of AAAI-94*, MIT Press, Cambridge, MA, 1994.

[Ok 94] Ok, D. A Comparative Study of Undiscounted and Discounted Reinforcement Learning Methods, Technical Report, 94-30-3, Dept. of Computer Science, Oregon State University.

[Watkins and Dayan 92] Watkins, C. J. C. H. and Dayan, P. Q-learning. *Machine Learning*, 8:279-292, 1992.