Scaling Up Average Reward Reinforcement Learning by Approximating the Domain Models and the Value Function

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

Abstract

Almost all the work in Average-reward Reinforcement Learning (ARL) so far has focused on table-based methods which do not scale to domains with large state spaces. In this paper, we propose two extensions to a model-based ARL method called H-learning to address the scale-up problem. We extend H-learning to learn action models and reward functions in the form of Bayesian networks, and approximate its value function using local linear regression. We test our algorithms on several scheduling tasks for a simulated Automatic Guided Vehicle (AGV) and show that they are effective in significantly reducing the space requirement of H-learning and making it converge faster. To the best of our knowledge, our results are the first in applying function approximation to ARL.

1 Introduction

Most Reinforcement Learning (RL) methods optimize the discounted total reward received by an agent (Barto, Bradtke, & Singh, 1995; Watkins & Dayan, 1992). However, in many real-world domains, the natural criterion is to optimize the average reward received in unit time. Recently, there has been growing interest in Average-reward Reinforcement Learning (ARL) (Schwartz, 1993; Singh, 1994; Mahadevan, 1996; Boutilier & Puterman, 1995; Tadepalli & Ok, 1994). In this paper, we study two extensions to a model-based ARL method called H-learning, an undiscounted version of Adaptive Real-Time Dynamic Programming (ARTDP) (Barto et al., 1995).

The study of ARL so far has been restricted to table-based methods in which the value function is stored as a table. In the worst case, this table can become as large as the number of states in the state space. We are interested in using ARL in the task of scheduling industrial mobile robots, called Automatic Guided Vehicles (AGVs), for routine delivery tasks, with the goal of maximizing job throughput. To scale ARL to such domains with millions of states, it is essential to approximate its value function in a compact form.

There have been several function approximation methods studied in the discounted RL literature, including neural network learning, clustering, memory-based methods, and locally weighted regression (Lin, 1992; Boyan & Moore, 1994; Mahadevan & Connell, 1992; Moore, 1990; Schaal & Atkeson, 1994). Two characteristics of the AGV scheduling domain attracted us to local linear regression as the method of choice. First, the location of the AGV is one of the most important features of the state in this domain. Any function approximation scheme must be able to generalize specific locations of the AGV to large regions. Second, the value function for H-learning varies linearly over large geometric regions with uniform reward structure, where the values of other non-geometric features are fixed. In other words, the value function is locally linear in the location feature. We implemented an approximation method based on local linear regression and tested it on several AGV scheduling tasks. In each case, our method improved the performance of H-learning in storage space as well as in learning time.

Model-based methods like H-learning also have the additional problem of having to explicitly store their action models and the reward functions, which we call the "domain models." Dynamic Bayesian networks have been successfully used in the past to represent the domain models (Russell & Norvig, 1995). In many cases, it is possible to design these networks in such a way that a small number of parameters are sufficient to fully specify the domain models. We extended H-learning so that it takes the network structure as input and learns the conditional probabilities in the network. This method synergistically combines with local linear regression and leads to better space and time utilization. H-learning performs better than its discounted counterpart ARTDP when both use Bayesian network action models and local linear regression.

The next section briefly introduces H-learning. Section 3 describes learning actions models as dynamic Bayesian networks. Section 4 describes our value function approximation method. Section 5 presents experimental results in several domains. Section 6 summarizes our contributions and discusses future work.

2 H-Learning

We use the standard notion of Markov Decision Processes (MDP) described by a set of n discrete states S and a set of actions A. The actions which are applicable in a state i are denoted by $U(i)$ and are called *admissible*. The Markovian assumption implies that an action u in a given state $i \in S$ results in state j with some fixed probability $P_{i,j}(u)$. There is a finite immediate reward for executing an action u in state i resulting in state j, given by $r_i(u)$. Time is treated as a sequence of discrete steps $t = 0, 1, 2, \ldots$. A *policy* μ is a mapping from states to actions, such that $\mu(i) \in U(i)$. A stationary policy is a policy which does not change with time. By "policy" we mean stationary policy from now on.

Let an agent using a policy μ go through states $s_0, \ldots, s_t$ in time 0 thru t, with some probability. In ARL, we seek to optimize "the average expected reward of the policy μ starting with state s_0," denoted by $\rho^\mu(s_0)$, defined as the limit of the average expected reward per step over time t as t approaches ∞.

$$\rho^\mu(s_0) = \lim_{t\to\infty} \frac{1}{t} E(r_{s_0}(\mu(s_0)) + \ldots + r_{s_{t-1}}(\mu(s_{t-1})))$$

In what follows, we assume that the exploration strategy guarantees that every state in S is visited with some probability from every other state during training. Under these conditions, the long-term Average-reward per time step is independent of the initial state and is denoted by $\rho(\mu)$, the "gain" of the policy μ (Bertsekas, 1995; Puterman, 1994). The goal is to find a policy μ^* that maximizes the gain, a *gain-optimal policy*.

Even though the gain of a policy, $\rho(\mu)$, is the same for every starting state, the total expected reward in time t may not be the same for different starting states. The limit of the difference between this sum and $\rho(\mu)t$ for a state s as $t \to \infty$ (Cesaro limit in the case of periodic policies) converges to some finite $h(s)$, called the *bias* of state s. Intuitively, $h(i) - h(j)$ is the average relative benefit in long-term total reward when the MDP is started in state i as opposed to state j.

Theorem 1 *For a given MDP, if there exist a scalar ρ and a real-valued function h over S that satisfy*

$$\forall i \in S, h(i) = \max_{u\in U(i)} r_i(u) + \{\sum_{j=1}^{n} p_{i,j}(u)h(j)\} - \rho \quad (1)$$

then, ρ is its optimal gain. Further, every gain-optimal policy μ^* attains the above maximum for each state i.

H-learning computes the probabilities $p_{i,j}(a)$ and the expected rewards $r_i(a)$ by maximum likelihood estimates, and uses the current estimates as the true probabilities while updating the h values. It initializes all the h values to 0, and in each current state i, updates $h(i)$ with the right hand side of Equation (1). It is very similar to "Algorithm B" of Jalali and Ferguson (Jalali & Ferguson, 1989), and is an Average-reward version of the discounted RL method, Adaptive Real-time Dynamic Programming (ARTDP) (Barto et al., 1995). It can also be seen as a model-based version of R-learning (Schwartz, 1993).

The right hand side of Equation (1) also involves ρ, which is estimated as the gain of the current greedy policy. Like most RL methods, and unlike Algorithm B, H-learning makes random moves according to some exploration strategy. The effects of such "exploratory" moves on the immediate rewards have to be nullified when estimating ρ, because they are not part of the current greedy policy and add noise to the estimate. ρ is estimated by cumulatively averaging $r_i(u) - h(i) + h(k)$, whenever a greedy action u is executed in state i resulting in state k. H-learning improves its ability to automatically explore the state space by initializing the value of ρ to a value ρ_0 higher than the expected optimum gain and adjusting it using a decreasing learning rate α, initialized to a small value, α_0 (Ok & Tadepalli, 1996). Our experiments are based on this version of H-learning, denoted by H^{ρ_0,α_0}.

3 Model Generalization using Bayesian Networks

One of the disadvantages of model-based methods like H-learning is that explicitly storing its action and reward models consumes a lot of space. The space requirement for storing the models can be anywhere from $O(nm)$ to $O(n^2m)$ depending on the stochasticity of the domain, where n is the number of states and m is the number of actions. To scale the model-based learning methods to large domains, we need to compactly represent the action and reward models, called "domain models" from now on.

Dynamic Bayesian networks have been used in the previous literature to compactly represent stochastic domain models (Russell & Norvig, 1995; Dean & Kanazawa, 1989; Nicholson & Brady, 1992; Kjaerulff, 1992; Boutilier, Dearden, & Goldszmidt, 1995). In this section, we describe how we can adapt H-learning to learn the parameters for them, assuming that the network structure is given.

We assume that a state is described by a set of discrete valued features. A dynamic Bayesian network represents the relationships between the feature values and the action at time t, and the feature values at time $t+1$. A Bayesian network is a directed acyclic graph whose nodes represent random variables which can have values, along with a conditional probability table (CPT) associated with every node. The CPT at each node describes the probabilities of different values for a node conditioned on the values of its parents.

The probability of any event given some evidence is determined by the network and the associated CPTs, and there are many algorithms to compute this (Russell & Norvig, 1995). Since the network structure is given as prior knowledge in our case, learning action models reduces to learning the CPTs.

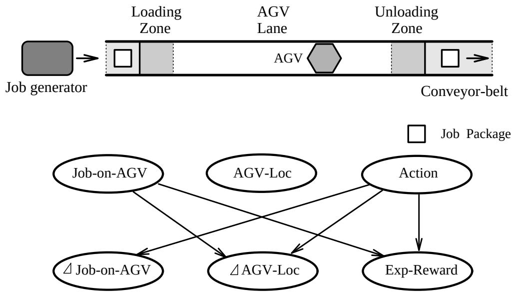

Figure 1: (a) the Slippery-Lane domain (top) and (b) its Bayesian network (bottom).

We illustrate the Bayesian network representation using the Slippery-Lane domain in Figure 1 (a). There is a job generator on the left and a conveyor-belt on the right. The job generator generates a new job immediately after the AGV loads a job. The goal of the AGV is to repeatedly load a job at the loading zone and unload it at the unloading zone. A state of this domain can be described by the job on AGV (Job-on-AGV which is 0 or 1) and the AGV location (AGV-Loc).

There are six actions: stay, load, unload, forward, backward, and move. The action forward is for entering the lane from the loading zone, and backward is for entering the lane from the unloading zone. If move is executed, it moves towards its correct destination with probability P_{move} and in the incorrect direction with probability $1 - P_{move}$ (since the lane is slippery). The other actions have the obvious meanings. The AGV gets a positive reward of 5 when it unloads a job, and gets a penalty of -0.005 whenever it moves with a job. In all other cases, the reward is 0.

Figure 1 (b) shows the Bayesian networks for this domain. ΔJob-on-AGV and ΔAGV-Loc represent the changes to the values of Job-on-AGV and AGV-Loc, respectively. In this domain, ΔJob-on-AGV is either +1 for loading, -1 for unloading, and 0 for other actions, and ΔAGV-Loc is either +1 for moving right, 0 for staying, and -1 for moving left. Because the load and unload actions are admissible only when Job-on-AGV has the appropriate value, ΔJob-on-AGV is independent of Job-on-AGV or AGV-Loc, given the Action. But the direction of move is also dependent on the job on AGV. Hence Job-on-AGV is connected to ΔAGV-Loc. Since the action move receives a negative reward only when the AGV has a job, the node Exp-Reward, which denotes the expected immediate reward, has both Job-on-AGV and Action as its parents.

Learning the CPTs of the dynamic Bayesian network from examples is straightforward. Consider a node f which has parents $f_1, \ldots, f_k$. The conditional probability $P(f = v | f_1 = v_1, \ldots, f_k = v_k)$ is approximated by the fraction of state-action pairs in which f has the value v out of all cases in which the parents have the desired values. For example, the probability of ΔAGV-Loc = +1 given Job-on-AGV = 1 and Action = move is the fraction of the cases in which the AGV moved right when it had a job and move was executed.

If n is the number of different AGV locations, the above Bayesian network representation reduces the space requirement of the domain models from $16(n-1)$ to 32. An important consequence of this reduction is that learning of domain models can now be much faster. In domains such as the Slippery-Lane, where accurate domain model learning is crucial for performance, Bayesian network-based models can demonstrably expedite policy learning (see Section 5).

4 Value Function Approximation

In this section, we describe our value function approximation method, which is based on local linear regression (LLR). We chose LLR for two reasons. First, in an AGV scheduling task such as the Slippery-Lane domain of the last section, one of the important features of the state is the location of the AGV. Since the AGV can typically be in one of a large set of locations, it is important to be able to generalize the locations to large regions to effectively approximate the h function. Second, in AGV domains the immediate reward is usually independent of the exact location of the AGV and is a function of other features of the state and the action. Under these conditions, the h function is locally linear with respect to the AGV location when the other features of the state are constant. In what follows, we assume that the value function is locally linear with respect to a small number of such features and can change arbitrarily with respect to other features.

4.1 The General Linear Regression Model

In linear regression, we fit a linear surface in k dimensions to a set of m data points so that the sum of the squares of the errors of these points with respect to the output surface is minimized.

Let $x_{i1}, \ldots, x_{ik}$ be the values of the k input variables for the i^{th} point, and let y_i be the value of its output. Then, we seek to find the linear parameters $\beta_0, \ldots, \beta_k$ to minimize the sum of squares of the error terms $e_1, \ldots, e_m$, where,

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik} + e_i, i = 1, \ldots, m.$$

Given a sample of m data points, the above general linear model can be expressed in matrix notation as $\mathbf{Y} = \mathbf{XB} + \mathbf{E}$, where

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1k} \\ 1 & x_{21} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{m1} & \cdots & x_{mk} \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix}, \mathbf{E} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_m \end{bmatrix}.$$

The values for parameters $\beta_0, \ldots, \beta_k$ that minimize the squared error can be obtained as the $(k+1) \times 1$ vector $\mathbf{B}$ (Canavos, 1984).

$$\mathbf{B} = (\mathbf{X}^{\mathsf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathsf{T}}\mathbf{Y}. \quad (2)$$

4.2 Local Linear Regression

Let us assume that the state is represented by a set of k "linear" features and $n-k$ "nonlinear" features. Our value function approximation is limited to generalizing the values of the k linear features. This is similar to Locally Weighted Regression (LWR) where the nonlinear features are given infinitely large weights (Schaal & Atkeson, 1994). It also means that the value function h may only be *locally* linear. We represent the value function using a select set of states and their h values, called the "exemplars."

Suppose we need an estimate of $h(p)$, which has values $x_{p1}, \ldots, x_{pk}, \ldots, x_{pn}$ for its n features, where the first k are the linear features. If p is one of the exemplars, its stored value is its estimate. Otherwise, the system selects one nearest exemplar of p in each of the 2^k quadrants centered at p, which has the same values as p for all its nonlinear features. If $k = 1$, for example, the nearest neighbor on each side of p in its linear dimension, which has values $x_{p(k+1)}, \ldots, x_{pn}$ for features $k+1, \ldots, n$, is selected. We found that selecting neighbors this way reduces the potentially large errors due to extrapolating the h value from states which are far off from p but are close to each other. In our experiments, it also resulted in $\mathbf{X}^{\mathsf{T}}\mathbf{X}$ always having an inverse, although, this is not theoretically guaranteed. Adding a diagonal matrix of small random numbers to $\mathbf{X}^{\mathsf{T}}\mathbf{X}$ is an inexpensive way to avoid singularities in general (Schaal & Atkeson, 1994). The Euclidean distance is used to find the nearest neighboring states. After selecting the 2^k neighbors, the system uses linear regression on the values of the first k features of these exemplars to find a least squares $\mathbf{B}$ from Equation (2) and uses it to predict its h value. If the predicted value is greater than the maximum of the values of all its selected neighbors, or is less than the minimum of their values, it is set to the maximum or minimum of their values, respectively. This step is useful in reducing the errors due to large differences in the distances of different neighbors from p.

If p does not have neighbors in all the 2^k quadrants which share its values for all their nonlinear features, then k is reduced by 1, and the nearest neighbors in 2^{k-1} quadrants are selected such that the above condition holds. This is continued until $k = 1$. If it also fails, then the h value is estimated to be 0. At any time, only those exemplars are stored whose h values cannot be estimated from the currently stored exemplars with a given tolerance. Since the h values of adjacent states

1. Take an exploratory action a or a greedy action a that maximizes $r_i(a) + \sum_{j=1}^{n} p_{i,j}(a)\hat{h}(j)$. Let l be the resulting state, and r_{imm} be the immediate reward received.
2. Update-BN-model (i, a, l, r_{imm})
3. If a is a greedy action, then
   (a) $\rho \leftarrow (1-\alpha)\rho + \alpha(r_i(a) + \hat{h}(l) - \hat{h}(i))$
   (b) $\alpha \leftarrow \frac{\alpha}{\alpha+1}$
4. $v \leftarrow \max_u \{r_i(u) + \sum_{j=1}^{n} p_{i,j}(u)\hat{h}(j)\} - \rho$
5. If $(i, h(i)) \in$ Exemplars, delete it. Let Neighbors be the nearest neighbors of i in the 2^k quadrants.
6. If $\mid v - \hat{h}(i) \mid > \epsilon\rho$ or $v > \max_{j \in \mathsf{Neighbors}} \hat{h}(j)$ or $v < \min_{j \in \mathsf{Neighbors}} \hat{h}(j)$
   (a) Add (i, v) to Exemplars.
   (b) For any $j \in$ Neighbors , if $\mid h(j) - \hat{h}(j) \mid \leq \epsilon\rho$ then delete $(j, h(j))$ from Exemplars.
7. $i \leftarrow l$; Goto 1.

Figure 2: LBH-learning which uses Bayesian networks for representing the action models and Local Linear Regression for approximating the value function.

in the state space differ by ρ when there is no immediate reward, the tolerance is normalized by multiplying a constant parameter ϵ with ρ. An exemplar is also stored if its updated h value is greater or less than the estimated h values of all its selected nearest neighbors.

Whenever a new exemplar (i, v) is stored, the system checks if any of its nearest neighbors (selected as described above), say j, can be safely deleted from the exemplar set, because it may now be possible to approximate the stored $h(j)$ value from j's nearest neighbors. Without this heuristic, a large number of exemplars will be stored in the beginning, and they are never deleted afterwards.

Figure 2 shows a modification of H-learning, which we call "LBH-learning," that uses Bayesian network models and local linear regression. Before entering the loop of Figure 2, the exemplar set is initialized to empty, ρ to a value higher than the expected optimum gain, and α to a low value. The explicitly stored h values of the exemplars are denoted by $h(.)$, and the values estimated by LLR are denoted by $\hat{h}(.)$. The immediate rewards $r_i(u)$ and the transition probabilities $p_{i,j}(u)$ are inferred from Bayesian networks using the standard Bayesian inference algorithms. The parameters of the Bayesian networks are updated incrementally by Update-BN-model as described in Section 3.

5 Experimental Results

In this section we show experimental results in three different AGV-scheduling domains. We compare the performance of LBH-learning with H-learning, "BH-learning" that uses the Bayesian network action models but without LLR, and "LH-learning" that uses LLR without Bayesian network models. The experiments demonstrate the synergy between Bayesian network models and LLR, the scalability of LBH-learning in both space and time, and its applicability to multi-dimensional domains.

5.1 Improving the performance of H-learning

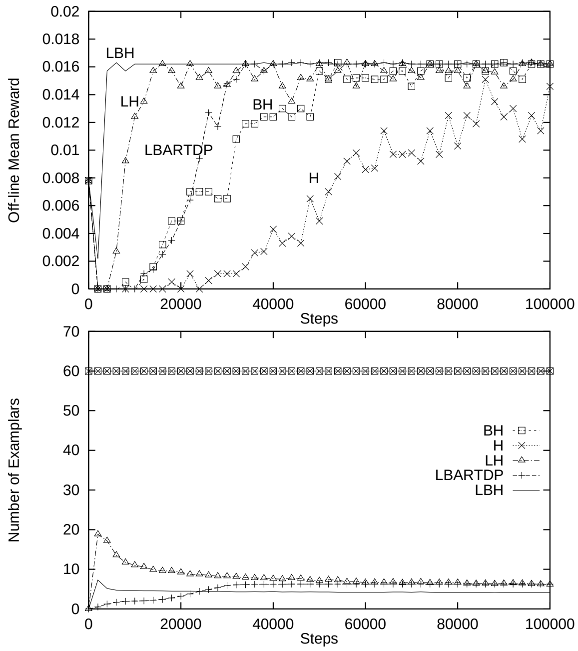

Figure 3: (a) off-line mean reward under 10% semi-uniform exploration (top) and (b) number of exemplars(bottom). Results are averaged over 30 trials of LBH$^{0.05,0.005}$, BH$^{0.01,0.001}$, LH$^{0.02,0.0001}$, H$^{0.01,0.001}$ and LBARTDP.

The goal of the first experiment is to demonstrate the synergy between LLR and the Bayesian network learning in scaling H-learning. We use the Slippery-Lane domain and the Bayesian network of Figure 1. The number of locations for the AGV was set to 30. To make the model learning significantly important, P_{move} was set to 0.6. The discounted method ARTDP with Bayesian network models and LLR (and $\gamma = 0.95$) was also used for comparison, and labeled LBARTDP in Figure 3. Since ARTDP does not have the parameter ρ, its tolerance was normalized to be $\epsilon\beta_i$, where β_i

is the smallest local slope of the value function among all the linear dimensions in state i.

The parameters of each method were tuned by trial and error. For LBH, LH and LBARTDP, ϵ was set to 1, 1 and 2 respectively. The experiments were repeated for 30 trials for each method, starting from a random initial state. In Figure 3 (a), we plotted the off-line mean reward over 30 trials, each estimate being based on 3 runs of 100K steps from 3 random start states. We chose off-line estimation because the convergence is too fast to reliably measure the on-line mean reward in this domain.

Since P_{move} is close to 0.5, model learning is important in this domain. This explains the performance difference between BH and H. LH converged faster than BH, but did suffer due to the lack of Bayesian network models. Most interesting was the performance of LBH which was clearly superior to both BH and LH, thus demonstrating the synergy between the two kinds of approximations used. LBARTDP was slower than LBH, although it was faster than BH.

Figure 3 (b) shows the average exemplar size of all the methods. In the end, LBH stores only 4 states that correspond to the end locations of the Slippery Lane, while LBARTDP stores 6 or 7 states because of its non-linear value function. LH also stores 6 or 7 states because its value function cannot be very smooth without the Bayesian network models. BH and H, on the other hand, store values for all 60 states. LBH and LH store more exemplars in the beginning than in the end because of incorrect value function and actions models in the beginning.

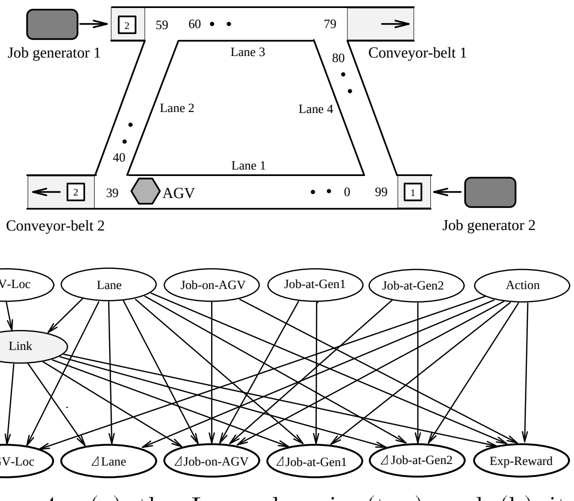

Figure 4: (a) the Loop domain (top) and (b) its Bayesian network (bottom).

5.2 Scaling of LBH-learning with Domain Size

The goal of the second experiment is to demonstrate the scaling of LBH-learning by varying the size of a domain. We use the "Loop domain" of Figure 4 (a).

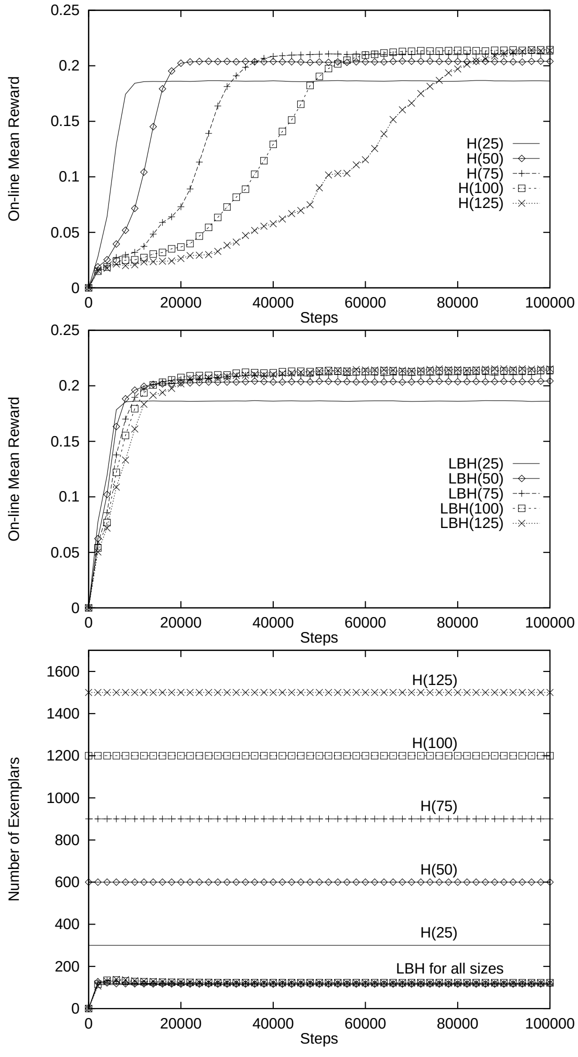

Figure 5: (a) on-line mean reward of H (top) and (b) LBH (middle) (c) number of exemplars (bottom) on the Loop domain. The results are averaged over 30 trials of $\mathrm{LBH}^{1,0.02}(25)$, $\mathrm{LBH}^{1.3,0.007}(50)$, $\mathrm{LBH}^{1.5,0.004}(75)$, $\mathrm{LBH}^{1.8,0.004}(100)$, $\mathrm{LBH}^{2,0.003}(125)$, $\mathrm{H}^{1,0.02}(25)$, $\mathrm{H}^{1,0.005}(50)$, $\mathrm{H}^{1,0.002}(75)$, $\mathrm{H}^{1.5,0.003}(100)$, and $\mathrm{H}^{1.5,0.002}(125)$. n is shown in parentheses.

There are one AGV, two job generators 1 and 2, and two conveyor belts 1 and 2. Each generator generates jobs for each destination belt with 50% probability. A state is described by 5 features AGV-Loc, Lane, Job-on-AGV, Job-at-Gen1, and Job-at-Gen2, with obvious meanings. Job-on-AGV is 0 if it has no job. AGV-Loc takes values from $1, \ldots, 2n$ for Lane 1 but from $1, \ldots, n$ for other three lanes. Therefore, the size of the state space is $5 \times n \times 3 \times 2 \times 2 = 60n$. The AGV has 4 actions move-forward, move-backward, load and, unload. To make the optimal Average-rewards the same for all n, the immediate reward for delivery is made proportional to n. If the AGV delivers a job to its correct belt, it receives a reward $+0.1n$. If it delivers it to the wrong belt, it gets a smaller reward of $+0.02n$.

Figure 4 (b) shows its Bayesian network. One new feature, Link, usefully abstracts the AGV-Loc feature. Link takes 3 values, distinguishing the end locations of each lane from the rest, which is useful to compactly model AGV's motion in the loop. The Bayesian network representation reduces the space requirements to store the domain models from $48n + 52$ to 1056.

We used random exploration with 10% probability in this experiment. We set ϵ to 1 and varied n from 25 to 125 in steps of 25. The performance comparisons are shown in Figure 5. The Y-axis indicates on-line mean reward for 30 trials, estimated over the last 5000 steps. As we can see, the performance of LBH is consistently better than that of H, and the difference increases with n. The convergence takes longer for larger values of n, because the AGV has to travel over longer distances to get new information. However, in LBH-learning, the number of steps for convergence grows much more slowly than in H.

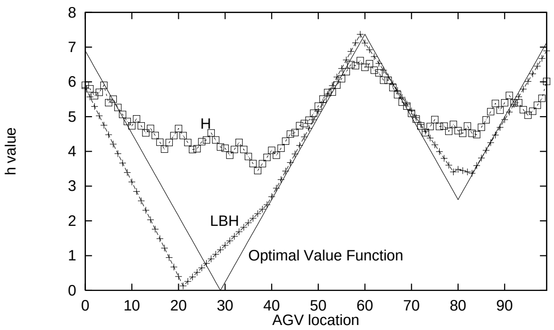

Figure 6: Final h value as a function of AGV location when AGV does not have a job, Generator 1 has Job 2 and Generator 2 has Job 1, and n=100. The AGV location is as shown on the lanes of Figure 4 (a).

The number of the exemplars of LBH, averaged from 30 trials, are shown in Figure 5 (c) in comparison to the exemplars stored by H-learning. For LBH, this number increases in the beginning, because the h values in the beginning are incorrect and non-linear; but it decreases later. The absolute number of exemplars almost remains constant even when n is increased.

The value function for H and LBH at the end of training are shown in Figure 6 in comparison to the optimal value function. It can be seen that LBH's value function is smoother than H's.

These results show that LBH scales better than H learning with the domain size, both in terms of learning speed and memory use.

5.3 Scaling to Multiple Linear Features

In this section, we demonstrate LBH-learning in a "Grid" domain shown in Figure 7 (a) which has two linear dimensions.

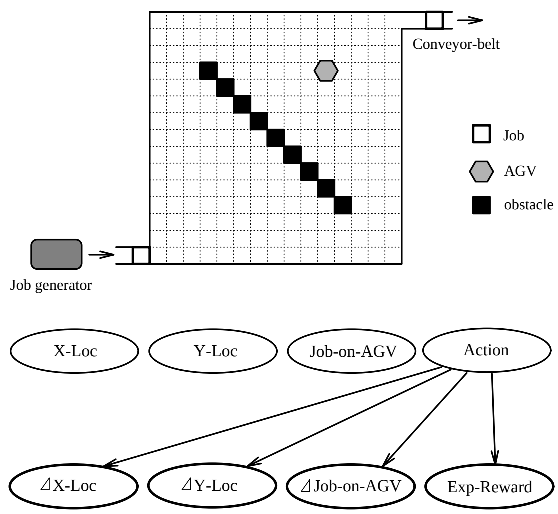

Figure 7: (a) the Grid domain (top) and (b) its Bayesian network (bottom).

At one corner of the 15×15 grid, there is a job generator and at the opposite corner, there is a destination conveyor-belt. The dark sqaures in the middle represent obstacles. The AGV can take actions move-north, move-south, move-east, or move-west without colliding with the walls and obstacles. It also has the load or the unload action available at the appropriate corner. It receives a reward of 58 when it delivers a job. Figure 7 (b) shows the dynamic Bayesian network for the Grid domain, which is very compact since the effect of any available action depends only on the Action variable. In fact, it reduces the space requirements for this domain from 3076 to 24.

Each point in Figure 8 (a) is the on-line mean reward for 30 trials calculated over the last 5000 steps. It shows the effect of setting ϵ to 0.4, 0.7, 1.0, and 1.3 on the performance of LBH with 10% random explo-

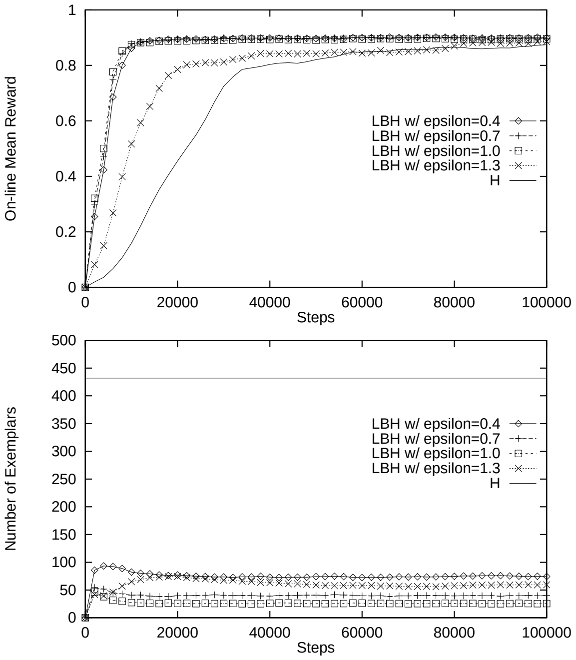

Figure 8: (a) on-line mean reward (top) and (b) average number of exemplars (bottom) of 30 trials of $H^{3,0.003}$ and $LBH^{3,0.002}$ on the Grid domain under 10% random exploration.

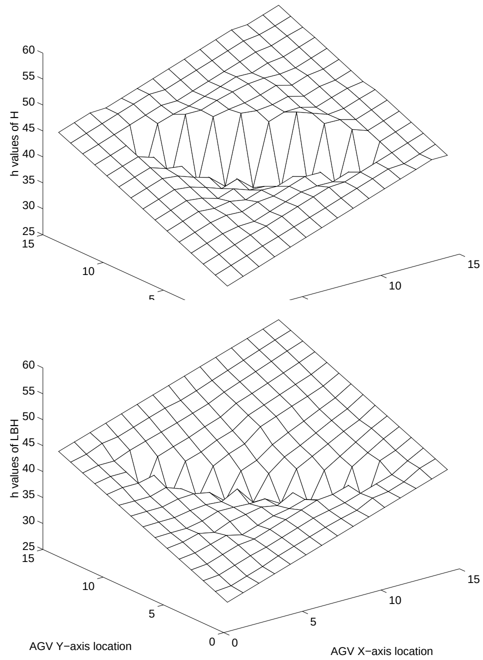

Figure 9: Final value function of $H^{3,0.003}$ (top) and $LBH^{3,0.002}$ (bottom) with $\epsilon = 1.0$ when AGV has load.

ration. With $\epsilon = 0.4$, 0.7, and 1.0, LBH finds the optimal policy much faster than H does. With $\epsilon = 1.3$, its speed decreases, because too high a value of ϵ sometimes makes it converge to a suboptimal policy. Figure 8 (b) shows the number of exemplars of LBH. LBH usually stores fewer exemplars with higher values of ϵ, except when ϵ is too high, 1.3 in this case.

The final value functions for H and LBH with $\epsilon = 1.0$ are shown in Figure 9. Both these functions are locally linear although LBH has a slightly smoother value function than H.

6 Conclusions and Future Work

Our results suggest that local linear regression (LLR) is a promising approach to approximation, and is especially effective in domains where the optimal value function is locally linear. We showed that the domain models can be compactly represented and learned using Bayesian networks. We demonstrated that our LBH algorithm performs significantly better than both H-learning and a version of ARTDP that uses LLR and Bayesian network models. We also showed that LBH scales well with domain size, and is applicable to domains with more than one linear feature. To the best of our knowledge, our results are the first in approximating the value function in Average-reward Reinforcement Learning (ARL).

Although our LBH algorithm scales with domain size in one or two dimensions, it still does not scale well with the number of features, especially when the features are nonlinear, e.g., the number of AGVs, machines, or the conveyor belts in our domains. We also ignored the computational cost of local linear regression in this paper, which is rather high. We plan to explore other approximation methods like neural networks to make it scale better with the number of features.

We assumed that the structure of the dynamic Bayesian networks is known. Learning the structure along with the conditional probability tables is an interesting future problem. Combining planning with on-line learning as in the DYNA architecture, is another promising direction to pursue (Sutton, 1990).

7 Acknowledgments

We gratefully acknowledge the support of NSF under grant number IRI-9520243. We thank Tom Dietterich, Sridhar Mahadevan, and Toshi Minoura for many interesting discussions on this topic. Thanks to Sridhar

Mahadevan and the AAAI reviewers for their thorough and helpful comments on the paper.

References

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. (1995). *Dynamic Programming and Optimal Control.* 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.*

Boutilier, C., & Puterman, M. L. (1995). Process-oriented planning and average reward optimality. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence.*

Boyan, J., & Moore, A. W. (1994). Generalizing reinforcement learning:safely approximating the value function. In *Proceedings of Neural Information Processing Systems.*

Canavos, G. C. (1984). *Applied Probability and Statistical Methods.* Little, Brown and Company, Boston, MA.

Dean, T., & Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence, 5(3)*, 142–150.

Jalali, A., & Ferguson, M. (1989). Computationally efficient adaptive control algorithms for markov chains. In *IEEE Proceedings of the 28'th Conference on Decision and Control* Tampa, FL.

Kjaerulff, U. (1992). A computational scheme for reasoning in dynamic probabilistic networks. In *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, pp. 121–129.

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

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

Mahadevan, S., & Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence, 55*, 311–365.

Moore, A. W. (1990). *Efficient Memory-based Learning for Robot Control.* Ph.D. thesis. Trinity hall, University of Cambridge, England.

Nicholson, A. E., & Brady, J. M. (1992). The data association problem when monitoring robot vehicles using dynamic belief networks. In *ECAI 92: 10th European Conference on Artificial Intelligence Proceedings*, pp. 689–693 Vienna, Austria. Wiley.

Ok, D., & Tadepalli, P. (1996). Auto-exploratory average reward reinforcement learning. In *Proceedings of AAAI-96.*

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Dynamic Stochastic Programming.* John Wiley.

Russell, S., & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Inc.

Schaal, S., & Atkeson, C. (1994). Robot juggling: An implementation of memory-based learning. In *IEEE Control Systems*, pp. 14:57–71.

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. (1994). Reinforcement learning algorithms for average-payoff markovian decision processes. In *Proceedings of AAAI-94.* MIT Press.

Sutton, R. S. (1990). Integrating architectures for learning, planning and reacting based on approximating dynamic programming. In *Proceedings of Machine Learning Conference.*

Tadepalli, P., & Ok, D. (1994). H-learning: A reinforcement learning method for optimizing undiscounted average reward. Tech. rep. 94-30-1, Dept. of Computer Science, Oregon State University.

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