A Reinforcement Learning Approach for Product Delivery by Multiple Vehicles

Scott Proper[+], Prasad Tadepalli[+], Hong Tang[+], and Rasaratnam Logendran[*]

[+]Department of Computer Science
Oregon State University
Corvallis, OR 97331-3202

[*]Department of Industrial and Manufacturing Engineering
Oregon State University
Corvallis, OR 97331-2407

Abstract

Real-time delivery of products in the context of stochastic demands and multiple vehicles is a difficult problem, as it requires the *joint* investigation of the problems in inventory control and vehicle routing. We model this problem in the framework of Average-reward Reinforcement Learning (ARL) and present experimental results on a model-based ARL algorithm called H-Learning with piecewise linear function approximation. We also show that a version of hill climbing yields at least as good performance as exhaustive search with an order of magnitude speedup in execution time.

Keywords
machine learning, average reward reinforcement learning, vehicle routing, product delivery

1. Introduction

Reinforcement Learning (RL) provides a nice framework to model a variety of real-world stochastic optimization problems [1]. In this paper, we use the RL framework to study the problem of real-time delivery of products using multiple vehicles when the demands are stochastic. While RL has been applied separately to inventory control [2] and vehicle routing [3,4] in the past, we are not aware of any applications of RL to the integrated problem of real-time delivery of products that includes both.

The delivery problem we investigate is as follows: There is one distribution center and several shops. Products are delivered from the distribution center to the shops by several trucks. We assume that there is a single type of product. The current status of the inventory levels of the shops, and the loads and locations of the trucks at any time are known to the scheduling system. The task of the system is to decide at each time-step the next action for each truck, including going to a different location, unloading, and waiting. Since the system is only given the levels of the inventories and the current load levels and locations of the trucks, and is required to determine the best actions for all the trucks, it addresses the combined problem of inventory control and vehicle assignment and routing simultaneously. While the actions of the trucks themselves only have deterministic effects, they happen concurrently with customer actions of consuming the products from the shops, which are highly stochastic. As a result, our problem is a stochastic Markov Decision Problem (MDP) with two kinds of costs: 1) the cost of moving the truck from one location to the other, which combines the fuel, vehicle, and the driver costs; and 2) the *stockout cost*, which is the lost profit due to not satisfying the customer when a customer finds the shop empty.

While it is natural to minimize the expected total cost in stochastic domains where the planning horizon is finite, in most recurrent domains including real-time delivery, the natural optimization criterion is to maximize the expected *average reward per time step*. In this paper we investigate the effectiveness of a model-based Average-reward Reinforcement Learning (ARL) method called H-Learning in solving a moderately difficult instance of the real-time delivery problem [5]. H-Learning works by learning explicit models of the actions and the environment as well as value functions over the states. The models represent how the state of the world changes with the actions chosen by the system. In our case, since the direct effects of actions by the trucks are known already, the distribution of consumption at each shop is learned. The value function represents the *bias* of each state, defined as the best expected reward when starting from that state over and above what would be expected on the average over the infinite horizon. Given the models and the optimal value function over all states, a one-step look-ahead search over all available actions of all trucks would yield the optimal decision.

Unfortunately, H-Learning, or any RL method that depends on storing the optimal value function and action models as tables, does not scale to large state-spaces such as our delivery domain. The table-based methods require space proportional to the number of states, which is exponential in the number of shops and trucks in our domain. In this paper we introduce a function approximator which stores the value function as a piecewise linear function, which achieves a high degree of compression and good performance (as measured by the average reward per unit time). The action models are also stored compactly as a function of a small set of expected inventory levels. Further optimization and abstraction of the truck identity makes it possible to scale the approach to 5 shops and 5 trucks from 5-shops and 2-trucks. Since at each time-step we have to consider the actions of all the trucks, even one-step look-ahead search is computationally expensive. To reduce this computational cost, we used a stochastic hill climbing algorithm that runs significantly faster and scales to 5 shops and 5 trucks without sacrificing the quality of the solutions.

In the next section, we discuss Average-reward Reinforcement Learning (ARL) and H-learning. In Section 3 we describe our method of function approximation and several other optimizations. In Section 4, we present experimental results and in Section 5 we describe possibilities for future work.

2. Average-Reward Reinforcement Learning

We assume that the learner's environment is modeled by a Markov Decision Process (MDP). An MDP is described by a discrete set S of N states, and a discrete set of actions, A. The set of actions that are applicable in a state i are denoted by $U(i)$ and are called *admissible*. The Markovian assumption means 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 $r_{i,j}(u)$ for executing an action u in state i resulting in state j. Time is treated as a sequence of discrete steps. A *policy* μ is a mapping from states to actions, such that $\mu(i) \in U(i)$. We only consider policies that do not change over time, which are called "stationary policies" [6]. In Average-reward Reinforcement Learning (ARL), we seek to optimize the average expected reward per step over time t as $t \longrightarrow \infty$, which is called the *gain.* For a given starting state s_0 and policy μ, the gain is given by Equation (1) where $r^u(s_0,t)$ is the total reward in t steps when policy u is used starting at state s_0, and $E(r^u(s_0,t))$ is

```
1. Take an exploratory action or a greedy action in the current state $i$. Let $a$ be the action taken, $k$ be the
   resulting state, and $r_{imm}$ be the immediate reward received.
2. $N(i,a) \leftarrow N(i,a)+1;\ \ N(i,a,k) \leftarrow N(i,a,k)+1$
3. $p_{i,k}(a) \leftarrow N(i,a,k)/N(i,a)$
4. $r_i(a) \leftarrow r_i(a) + (r_{imm} - r_i(a))/N(i,a)$
5. $GreedyActions(i) \leftarrow$ All actions $u \in U(i)$ that maximize
      a. $\{r_i(u) + \theta_{0,v} + \sum_{l=1}^{n} \theta_{l,v} E(\phi_{l,i} \mid i,u)\}$ (If using function approximation)
      b. $\{r_i(u) + \sum_{j=1}^{N} p_{i,j}(u) h(j)\}$ (otherwise)
6. If $a \in GreedyActions(i)$, then
      a. $\rho \leftarrow (1-\alpha)\rho + \alpha(r_i(a) - h(i) + h(k))$
      b. $\alpha \leftarrow \frac{\alpha}{\alpha+1}$
7. If using function approximation: a. $\vec{\theta}_v = \vec{\theta}_v + \beta\left(\max_{u \in U(i)}\{r_i(u) + \theta_{0,v} + \sum_{l=1}^{n} \theta_{l,v} E(\phi_{l,i} \mid i,u) - \rho\} - h(i)\right)\vec{\phi}_i$
   Otherwise: b. $h(i) \leftarrow \max_{u \in U(i)}\{r_i(u) + \sum_{j=1}^{N} p_{i,j}(u) h(j)\} - \rho$
8. $i \leftarrow k$
```

Figure 1: The H-learning algorithm. The agent executes steps 1-8 when in state i.

its expected value:

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

In most common MDPs, the gain of the optimal policy ρ^* is independent of the starting state [6]. The goal of ARL is to control the MDP in a way that achieves near-optimal gain by executing actions and receiving rewards and learning from them. Ideally, we would like to find a policy that optimizes the gain, i.e., a *gain-optimal policy*. The expected total reward in time t for optimal policies depends on the starting state s and can be written as the sum $\rho \cdot t + h_t(s)$. The Cesaro's limit of the second term h_t(s) as $t \to \infty$ is called the *bias* of state s and is denoted by *h(s).* The optimal gain ρ^* and the biases of the states under the optimal policy satisfy the following Bellman equation:

$$h(i) = \max_{u \in U(i)} \left\{ r_i(u) + \sum_{j=1}^{N} p_{i,j}(u) h(j) \right\} - \rho^* \tag{2}$$

The optimal policy chooses actions that maximize the right hand side of the above equation. The classical dynamic programming (DP) methods to solve the above equation, including *value iteration* and *policy iteration*, depend on knowing the probability transition models $p_{i,j}(u)$ and the immediate rewards $r_i(u)$. Reinforcement Learning (RL) methods are not given these models and need to learn them on-line by executing actions, observing the next states, and receiving rewards [1]. We use an ARL method called "H-Learning" which is "model-based" in that it computes the probabilities $p_{i,j}(a)$ and rewards $r_i(a)$ by straightforward Monte Carlo sampling, as shown in steps 2-4 in Figure 1. It then employs the "certainty equivalence principle" by using the current estimates as the true values while updating the *h*-value of the current state *i* in step 7.b. of Figure 1 according to the equation:

$$h(i) \leftarrow \max_{u \in U(i)} \left\{ r_i(u) + \sum_{j=1}^{N} p_{i,j}(u) h(j) \right\} - \rho \tag{3}$$

Unlike the classical DP methods that update all states in a sweep, at any time the RL algorithms such as ours only update the value of the current state *i*, and not the values of other states. To make sure that all state-values are updated until they converge, Reinforcement Learning methods require "exploration actions" that ensure that every action in every state is executed with some non-zero probability (step 1 of Figure 1). However, to get a good online reward, it should not deviate too much from the greedy policy as computed in step 5 of the algorithm in Figure 1. One issue that still needs to be addressed in Average-reward RL is the estimation of ρ^*, the optimal gain. Since the optimal gain is unknown we use ρ, an estimate of the average reward of the current greedy policy, instead. From Equation (2), it can be seen that $\max_{u \in U(i)} \left\{ r_i(u) + \sum_{j=1}^{N} p_{i,j}(u) h(j) \right\} - h(i)$ gives an unbiased estimate of ρ^*. This is what is estimated in Step 6 of Figure 1. In the next section, we describe a variety of enhancements to the basic H-Learning to make it scale to the product delivery problem.

3. On-line Product Delivery

Consider the supplier of a product such as bread that needs to be supplied to several shops from a warehouse using several delivery trucks. What is an efficient policy for ensuring the stores remain supplied? Trucks must not move unnecessarily to save on transportation costs, but must still make deliveries on time to prevent stockouts. We developed a simple model and experimented with an instance of the problem shown in Figure 2. We considered 5 shops, one centrally located depot, and 4 other intermediate locations where trucks can change their routes. We assumed it takes one unit of time to go from any location to its adjacent location or to execute an unload action. The shop inventory levels and truck load levels are discretized into 5 levels 0-4. With 2 trucks and 10 locations, this gives us a state space size of $(5^5)(5^2)(10^2)$= 7,812,500, and with 5 trucks and 10 locations, it gives us a size of $(5^5)(5^5)(10^5)$ = 976,592,500,000. State space size grows exponentially in the number of trucks and the number of shops. Each truck has 9 actions available at each time step: unload 25%, 50%, 75%, or 100% of its load, move in one of the four available directions, or do nothing. This is a total of 9^2 = 81 actions for 2 trucks, or as many as 9^5 =

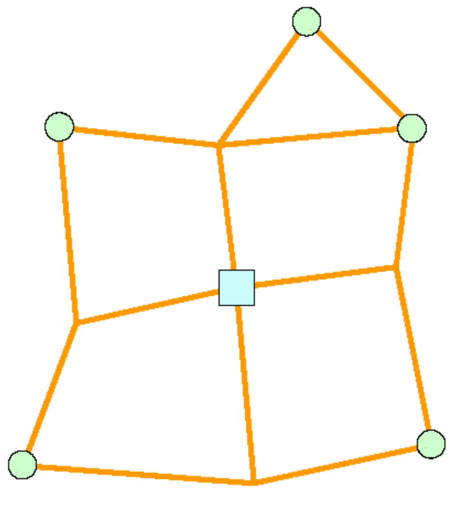
Figure 2: The Product Delivery Domain: The square in the center is the depot and the circles are the shops.

59,049 actions for 5 trucks. Trucks are loaded automatically upon reaching the depot. There is a small probability of a visit by a customer at every time step, which varies from shop to shop. If a customer enters a store and finds the shelves empty, the system receives a penalty of -20. A small negative reward of -.1 is given for every "move" action a truck may take to reflect the fuel , maintenance, and the driver costs.

3.1 Piecewise Linear Function Approximation

Table-based methods such as H-Learning do not scale to such large state spaces as the delivery domain both due to limitations of space and convergence speed. The value function needs to be represented more compactly using function approximation to make it scale to large domains. While there exist many function approximation methods such as neural networks and regression trees, linear function approximators are the simplest. Unfortunately, however, in many cases including ours, the optimal value function is highly nonlinear and cannot be captured by linear value functions. Instead, we used a function approximation scheme based on piecewise linear functions. In particular, we hypothesize that the value function can be broken down into several pieces, each of which has the same set of values for some of the so-called "non-linear" features. Each such piece of the value function is linear in the remaining set of features.

For example, in the delivery domain, it can be argued that the optimal value function is monotonically non-decreasing in certain variables of the state such as the inventory levels of the shops and the trucks, for any given set of locations of the trucks. This is true because the chance of a stockout is smaller when the inventories are higher than when they are lower. Similarly one has more options when the trucks have more load than when they have less load. We made a further simplifying assumption that the value function is *linear* in all these primitive features of the state, i.e., $i_1,\ldots,i_5$, that represent the shop inventory levels, and $t_1,\ldots,t_5$, that represent the truck loads. A third assumption is that the value function does not have cross-terms such as $i_1 i_2$. Since the locations of the trucks are discrete entities, we do not make a similar assumption about them. Instead, we assume a different linear function for each possible 5-tuple of locations $l_1,\ldots,l_5$ for the 5 trucks. Thus the entire value function is represented as a piecewise linear function, with one piece covering the function for one set of locations. Only m^k linear functions are needed to represent the h-function, where k is the number of trucks and m is the total number of locations. Since each linear function now has up to 11 weights (5 for shop inventory levels, 5 for truck inventory levels, and one weight, θ_o , that represents the bias), and there are 10^5 different linear functions, this reduces the total number of parameters to be learned to 1.1 million, nearly a million-fold reduction from the table-based approach.

The h-value is represented as a parameterized functional form with parameter vectors $\vec{\theta}_v$, one for each possible set v of values of the "nonlinear features," i.e., locations of the trucks. Corresponding to every state i, there is a vector of n "linear" features, $\vec{\phi}_i = (1, \phi_{1,i}, \phi_{2,i}, ..., \phi_{n,i})^T$. For each set of values of v, the parameter vectors, $\vec{\theta}_v = (\theta_{0,v}, \theta_{1,v}, \theta_{2,v}, ...\theta_{n,v})^T$, have the same number of components as $\vec{\phi}_i$. The approximate h-value function is given by:

$$h(i) = h(\vec{\theta}_v, \vec{\phi}_i) = \vec{\theta}_v^T \vec{\phi} = \sum_{i=0}^{n} \theta_{i,v} \phi_i \qquad (4)$$

where v is the set of values of the nonlinear features in state s. Then $\vec{\theta}_v$ is updated using the following equation:

$$\vec{\theta}_v \leftarrow \vec{\theta}_v + \beta \left(\max_{u \in U(i)} \{ r_i(u) + \sum_{j=1}^{N} p_{i,j}(u) h(j) - \rho \} - h(i) \right) \nabla_{\vec{\theta}_v} h(\vec{\theta}_v, \vec{\phi}_i) \text{, where } \nabla_{\vec{\theta}_v} h(\vec{\theta}_v, \vec{\phi}_i) = \vec{\phi}_i \qquad (5)$$

and β is the learning rate. The above update suggests that the value of *h(i)* would be adjusted toward the result of one-step backed up value of the next state.

3.2 Storing and using the action models

One of the drawbacks of the model-based RL methods is that they require stepping through all possible next states of a given action to compute the expected next state. In domains like ours where the state is decomposed into several parts, one for each shop, steps 1 and 5 of the H-learning algorithm (Figure 1) are very time-consuming. This is so because the number of possible next states is exponential in the number of variables such as the shop inventory levels. Anything that can be done to optimize these steps improves the speed of the algorithm considerably. As it turns out, using linear function approximation allows us a method of doing this. These steps both contain the term $\sum_{j=1}^{N} p_{i,j}(u)h(j)$ where N is exponential in the number of stores. (in our case, with 5 stores and 2 possible customer actions, $N = 2^5$). Note that the possible next states of an action at any state differ only in the shop inventory levels, since they are the only stochastic parts of the system.

Since the value of *h(j)* is assumed to be linear in the shop inventory levels, we can write this as $\sum_{j=1}^{N} p_{i,j}(u)(\theta_{0,v} + \sum_{l=1}^{n} \theta_{l,v}\phi_{l,i})$ which can be rewritten as $\theta_{0,v} \cdot \sum_{j=1}^{N} p_{i,j}(u) + \sum_{l=1}^{n} \theta_{l,v} \sum_{j=1}^{N} p_{i,j}(u)\phi_{l,i}$ and be simplified to $\theta_{0,v} + \sum_{l=1}^{n} \theta_{l,v} E(\phi_{l,j} \mid i,u)$, where $E(\phi_{l,j} \mid i,u) = \sum_{j=1}^{N} p_{i,j}(u)\phi_{l,j}$ and is directly estimated by on-line sampling. Instead of taking a number of steps proportional to the number of expected next states, this would only take steps proportional to the number of shops, which is exponentially smaller. For example, if the current inventory level of shop *l* is 2, and the probability of inventory going down by 1 in this step is 0.2, then $E(\phi_{l,i} \mid i,u)$ = 2-1*.2 = 1.8. Thus, we obtain 5.a by substituting $\theta_{0,v} + \sum_{l=1}^{n} \theta_{l,v} E(\phi_{l,j} \mid i,u)$ for $\sum_{j=1}^{N} p_{i,j}(u)h(j)$ in Step 5.b of the H-Learning algorithm. In Step 7.a., we combine this equation with (5) above.

3.3 Ignoring Truck Identity

A second possible optimization allows us to reduce the number of linear functions necessary to represent the h-function. The figure of m^k linear functions stated in section 3.1 assumes that a specific identity is associated with each truck: that is, truck 1 is at location l_1, truck 2 is at location l_2, and so on. This is more information than is actually necessary. If we no longer care about the identity of each truck, but use only the fact that there are 2 trucks, one of which is located at l_1, the other located at l_2, then we can reduce space requirements for representing the h-function, which in turn leads to faster convergence. Instead of m^k linear functions, we now require $\binom{m-1+k}{k}$. For 2 trucks, which previously required 100 linear functions, we now require only 55. For 5 trucks, previously requiring 10^5 functions, we only need 2002 linear functions. This reduces the number of learnable weights to 22,022 from 1.1 million in the 5-truck case.

3.4 Hill Climbing Search

Another problem that had to be addressed was the size of the action space. With 5 trucks, we have 9^5 joint actions. It is usually too expensive to consider each possible joint action at each step. Although we could do this in principle, it can take over a day of simulation time to finish learning using this method. We implemented a simple hill climbing strategy which significantly sped up the process by as much as an order of magnitude. Note that every action *a* is a vector of individual "truck actions" $\vec{a} = (a_1, a_2, ..., a_t)$. This vector is initialized with all "wait" actions. Starting at a_1, each of the 9 possible actions for that truck is considered, and $\vec{a}$ is set to the best action. This process is repeated for each truck $a_2 \ldots a_n$. The process then starts over at a_1, repeating until $\vec{a}$ has converged to a local optimum.

4. Experimental Results

We conducted several experiments testing the methods discussed in section 3. Two of these are presented here. All tests are averaged over 20 runs of 10^6 time steps each. Figure 3 compares the performance of the agent using piecewise linear function approximation vs. using simple table-based approaches. Two trucks and five shops were modeled. As can be seen from these results, the state space is so large, even for just 2 trucks, that table-based approaches converge too slowly to be practical. Using function approximation allows for much faster convergence.

Figure 4 compares the performance of using hill climbing to search the action space vs. using the more typical fully exhaustive search of all possible actions during each iteration. 4 trucks and shops, and 5 trucks and 5 shops are considered. In both cases, exhaustive search seems to perform better at first, but the hill climbing method quickly catches up and converges to a more stable result. The reason for this result appears to be that the hill climbing method allows more robust exploration of interesting parts of the state space than what is allowed by simple exploration strategies such as ε-greedy (take random actions with probability ε and the greedy action with probability 1- ε). This is an extremely encouraging result since it suggests that it may be possible to do less search and obtain better results in the end. The greatest benefit is the time to compute the 10^6 iterations plotted here. For purposes of comparison, exhaustive search takes about half a day for one run in the 5-truck case, hill climbing about half an hour.

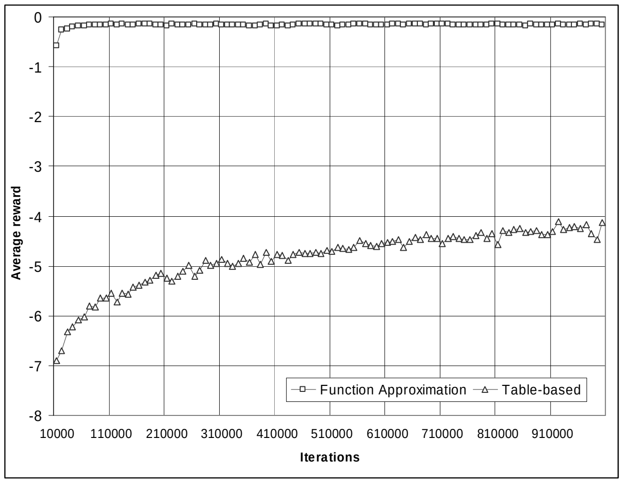

Figure 3: Piecewise linear function approximation vs. table-based approach

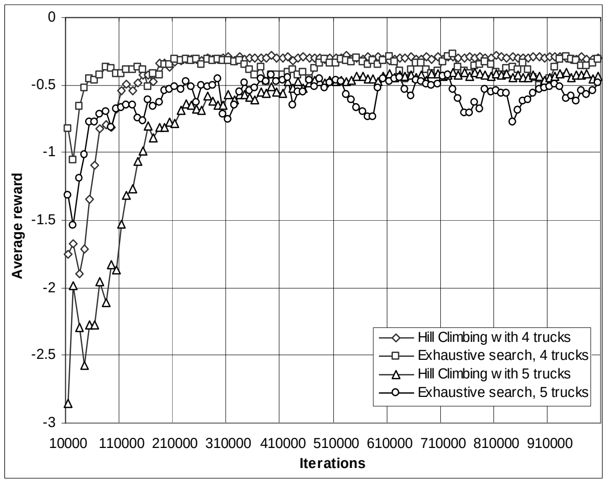

Figure 4: Hill climbing vs. exhaustive search for 4 and 5 trucks

Note that the average reward decreases with the number of trucks. This is because the percentage chance a customer will enter a shop to purchase an item is increased proportionally to the number of trucks in our experiments. The increase in the number of trucks was insufficient to combat the increased demand introduced in order to keep the problem challenging.

5. Future Work

There are numerous ways this work can be further extended. Scaling up these results to more realistic settings and improving the computation time are important open problems. We will continue to scale up the domain, adding more trucks, more locations, more shops, and more depots. Even more challenging would be to add more items: currently each truck carries only one type of item in its inventory. Scaling this up to two or more items creates a more difficult problem. Instead of trucks moving between 10 locations in discrete "jumps", allowing trucks to move with variable speed along highways interconnecting each intersection would be more realistic. This would require an event-based model of time, to allow trucks to arrive at shops and intersections at any time. Currently inventory and load levels are represented by 5 discrete values. Using real-valued shop inventory and truck load levels would be another interesting challenge. The function approximation method has room for improvement. Further experimentation with exploration strategies such as "optimism under uncertainty" may prove more useful.

In summary, we conclude that Average-reward RL is a promising approach for real-time product delivery. While we made many simplifying assumptions, the experimental results in our domain are encouraging, and give us confidence that we can make our approaches scale to larger and more practical problems.

Acknowledgements

This work was supported by the National Science Foundation under grant number: IIS-0098050. The authors would like to thank Neville Mehta and Jason Tracy for many interesting discussions on the topic of the paper.

References

1. R. S. Sutton and A. G. Barto. Reinforcement learning: an introduction. MIT Press, 1998.
2. B. Van Roy, D. P. Bertsekas, Y. Lee, and J. N. Tsitsiklis, "A Neuro-Dynamic Programming Approach to Retailer Inventory Management," Proceedings of the IEEE Conference on Decision and Control, 1997.
3. N. Secamondi, "Comparing Neuro-Dynamic Programming Algorithms for the Vehicle Routing Problem with Stochastic Demands," Computers and Operations Research September, Volume 27 Issue 11-12 , 2000.
4. N. Secamondi, "A Rollout Policy for the Vehicle Routing Problem with Stochastic Demands," Operations Research, 49(5), pp 796-802, 2001.
5. P. Tadepalli and D. Ok. Model-based Average Reward Reinforcement Learning. *Artificial Intelligence*, 100, 177-224, 1998.
6. M. L. Puterman. *Markov Decision Processes: Discrete Dynamic Stochastic Programming.* John Wiley, 1994.