A Neuro-Dynamic Programming Approach to Call Admission Control in Integrated Service Networks: The Single Link Case [1]

Peter Marbach and John N. Tsitsiklis
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139
e-mail: marbach@mit.edu, jnt@mit.edu

Abstract: We formulate the call admission control problem for a single link in an integrated service environment as a Markov Decision Problem. In principle, an optimal admission control policy can be computed using methods of Dynamic Programming. However, as the number of possible states of the underlying Markov Chain grows exponentially in the number of customer classes, Dynamic Programming algorithms for realistic size problems are computationally infeasible. We try to overcome this so-called "curse of dimensionality" by using methods of Neuro-Dynamic Programming (NDP for short). NDP employs simulation-based algorithms and function approximation techniques to find control policies for large-scale Markov Decision Problems. We apply two methods of NDP to the call admission control problem: the TD(0) algorithm and Approximate Policy Iteration. We assess the performance of these methods by comparing with two heuristic policies: a policy which always accepts a new customer when the required resources are available, and a threshold policy.

[1] This research was supported by a contract with Siemens AG, Munich, Germany.

1 Introduction

Markov Decision Problems have been a popular paradigm for sequential decision making under uncertainty. Dynamic Programming [1] provides a framework for studying such problems, as well as algorithms for computing optimal decision policies. Unfortunately, these algorithms become computationally infeasible when the underlying state space is large. This so called "curse of dimensionality" renders the classical methods of Dynamic Programming inapplicable to most realistic problems. As a result, control policies for practical large-scale sequential decision problems often rely on heuristics.

In recent years, a new methodology called Neuro-Dynamic Programming (NDP for short) [2] has emerged. NDP tries to overcome the curse of dimensionality by employing stochastic approximation algorithms and function approximation techniques such as neural networks. The outcome is a methodology for approximating Dynamic Programming solutions with reduced computational requirements.

Over the past few years, methods of NDP have been successfully applied to challenging problems. Examples include a program that plays Backgammon [3], an elevator dispatcher [4], a job scheduler [5], and a call admission control policy in wireless communication networks [6]. Despite these successes, most algorithms proposed in the field are not well understood at a theoretical level. Nevertheless, the potential of these methods for solving systematically large-scale Markov Decision Problems and the successful experimental work in the field has drawn considerable attention.

In this paper, we apply methods of NDP to the call admission control problem in an integrated service environment. In particular, we consider a single communication link with a given bandwidth that serves several customer classes of different values. The customer classes are characterized by the following parameters: bandwidth demand, arrival rate, departure rate and a reward we obtain, whenever we accept a customer of that class. The goal is to find a call admission control policy which maximizes the long term reward. Related work has been done by Nordström et al. [7] and [8] .

The paper is structured in the following way: in Section 2, we state the call admission control problem. A brief review of Dynamic Programming is given in Section 3. Section 4 introduces two methods of NDP: the TD(0) algorithm and Approximate Policy Iteration. In Section 5, we describe two parametric forms used to approximate the reward (value) function of Dynamic Programming: multilayer perceptron and quadratic parameterizations. We formulate in Section 6 the call admission control problem as a Markov Decision Problem. Section 7 defines two heuristic control policies for the call admission control problem: a policy which always accepts a new customer when the required resources are available, and a threshold policy. Experimental results of two case studies are presented in Section 8.

2 Call Admission Control

We are given a single communication link with a total bandwidth of B units. We intend to support a finite set $\{1, 2, .., N\}$ of customer classes. Customers of the different classes request connections over the link according to independent Poisson Processes. The arrival rate of customers of class n is denoted by $\lambda(n)$. When a new customer requests a connection, we can either decide to reject that customer, or, if enough bandwidth is available, to accept it (*call admission control*). Once accepted, a customer of class n seizes $b(n)$ units of bandwidth for t units of time, where t is exponentially distributed with parameter $\nu(n)$, independently of everything else happening in the system. Furthermore, whenever we accept a customer of class n, we receive a reward $c(n)$. The goal is to exercise call admission control in such a way that we maximize long term reward.

In this problem formulation, the reward $c(n)$ could be the price customers of class n are paying for using $b(n)$ units of bandwidth of the link. This models the situation, where a telecommunication network provider wants to sell bandwidth to customers in such a way, that long term revenue is maximized.

The reward $c(n)$ could also be used to attach levels of importance/priority to the different service classes. This reflects the case where one wants to provide the different customer classes different qualities of service; e.g. customers of a class with a high reward (high importance/priority level) should be less likely to be blocked than customers of a class with with a low reward (low importance/priority level).

Furthermore, the bandwidth demand $b(n)$ could either reflect the demand associated with the peak transmission rate requested by customers of class n, or a so-called "effective bandwidth" associated with customers of class n. The concept of an effective bandwidth has been introduced and extensively studied in the context of ATM (see for example [9]). ATM (Asynchronous Transfer Mode) is a technology which implements integrated service telecommunication networks. In ATM, the notion of an effective bandwidth is used to encapsule cell-level behavior such as multiplexing. This separation of cell-level and call-level is important for the tractability of the call admission control problem.

Although one is ultimately interested in applying call admission control (combined with routing) to the case of an integrated service network, we focus here on the single link case. This allows us to test algorithms of NDP on a problem, for which good heuristic policies are available, and for which (if the instance is fairly small) an optimal control policy can be computed. Furthermore, results obtained for the single link case can be used as a basis for solving the network case.

3 Markov Decision Problems and Dynamic Programming

In this section, we give a brief review of Markov Decision Problems and Dynamic Programming. Markov Decision Problems have been a popular paradigm for sequential decision making problems under uncertainty. Dynamic Programming [1] provides a framework for studying such problems, as well as algorithms for computing optimal control policies.

We consider infinite-horizon, discrete-time, discounted Markov Decision Problems defined on a finite state space S and involving a finite set U of control actions. Although we formulate the call admission control problem as a continuous-time Markov Decision Problem, it can be translated into a discrete-time Markov Decision Problem using uniformization [1]. Therefore, we can without loss of generality limit our framework to discrete-time Markov Decision Problems.

For every state $s \in S$, there is a set of nonnegative scalars $p(s,u,s')$, such that for all control actions u in U, $\sum_{s' \in S} p(s,u,s')$ is equal to 1. The scalar $p(s,u,s')$ is interpreted as the probability of a transition from state s to state s' under control action u. With a state s and a control action u, we associate a real-valued one stage reward $g(s,u)$.

A stationary policy is a function $\mu : S \to U$. Let M be the set of all possible stationary policies. A stationary policy μ defines a discrete-time Markov Chain (S_k) with the transition probabilities

$$P\left\{S_{k+1} = s' \,|S_k = s\right\} = p\left(s, \mu(s), s'\right)$$

With a stationary policy $\mu \in M$ and a state $s \in S$, we associate the reward-to-go function

$$J_\mu(s) = \lim_{T \to \infty} E\left[\sum_{k=0}^{T} \alpha^k g\left(S_k, \mu(S_k)\right) |S_0 = s\right]$$

where the discrete-time process (S_k) evolves according to the Markov Chain defined by the policy μ and where $\alpha \in [0,1)$ is a discount factor.

Our objective is to maximize the the reward-to-go function $J_\mu(s)$ simultaneously for all states $s \in S$. A stationary policy μ^* such that

$$J_{\mu^*}(s) \geq J_\mu(s), \quad \text{for all } s \in S \text{ and for all } \mu \in M$$

is said to be an optimal policy.

We can think of the reward-to-go function associated with a policy μ as a mapping $J_\mu : S \to R$. It is well known that if μ is optimal then J_μ satisfies Bellman's equation

$$J(s) = \max_{u \in U} \left[g(s,u) + \alpha \sum_{s' \in S} p(s,u,s') J(s')\right], \quad \text{for all } s \in S$$

It is also known that Bellman's equation has an unique solution J^*, called the optimal reward-to-go function.

Given a function $J : S \to R$, we define a greedy policy for J to be a policy μ which has the following property

$$\mu(s) = \arg\max_{u \in U} \left[g(s,u) + \alpha \sum_{s' \in S} p(s,u,s') J(s') \right]$$

It is well known that a greedy policy for the optimal reward-to-go function J^* is an optimal policy. This means that the knowledge of the optimal reward-to-go function J^* allows us to derive an optimal policy μ^*.

For a function $J : S \to R$, a stationary policy $\mu : S \to U$, and for every state $s \in S$, we denote

$$\begin{aligned} T(J)(s) &= \max_{u \in U} \left[g(s,u) + \alpha \sum_{s' \in S} p(s,u,s') J(s') \right] \\ T_\mu(J)(s) &= g(s,\mu(s)) + \alpha \sum_{s' \in S} p(s,\mu(s),s') J(s') \end{aligned}$$

The optimal reward-to-go function J^* and the reward-to-go function J_μ of a policy μ have the following property

$$\begin{aligned} J^*(s) &= T(J^*)(s), \quad \text{for every state } s \in S \\ J_\mu(s) &= T_\mu(J_\mu)(s), \quad \text{for every state } s \in S \end{aligned}$$

.

There are several algorithms for computing J^* and determining an optimal policy μ^*, but we will only consider Value Iteration and Policy Iteration.

The Value Iteration algorithm generates a sequence of functions $J_k : S \to R$ according to

$$J_{k+1}(s) = T(J_k)(s), \quad \text{for all } s \in S$$

It is well known that for discounted, infinite-horizon Markov Decision Problems which evolve over a finite state space S and which involve a finite set U of control actions, the Value Iteration algorithm is guaranteed to converge to the optimal reward-to-go function J^*.

The Policy Iteration algorithm generates a sequence of policies $\mu_k : S \to U$ by letting

$$\mu_{k+1}(s) = \arg\max_{u \in U} \left[g(s,u) + \alpha \sum_{s' \in S} p(s,u,s') J_{\mu_k}(s') \right]$$

It is well known that for discounted, infinite-horizon Markov Decision Problems which evolve over a finite state space S and which involve a finite set U of

control actions, the Policy Iteration algorithm is guaranteed to terminate with an optimal policy μ^* in a finite number of iterations.

The reward-to-go function J_{μ_k} can be obtained by solving the linear system of equations

$$J_\mu(s) = g(s, \mu(s)) + \alpha \sum_{s' \in S} p(s, \mu(s), s') J(s')$$

(as discussed earlier), or by an algorithm which generates a sequence of functions $J_k : S \to R$ according to

$$J_{k+1}(s) = T_{\mu_k}(J_k)(s), \quad \text{for all } s \in S$$

Note that this iteration can be viewed as a special case of Value Iteration applied to a problem with no control options. Therefore, the convergence result for Value Iteration applies also here.

In principle, an optimal control policy can be obtained by means of the Value Iteration algorithm or the Policy Iteration algorithm. However, this requires the computation and storage of $J^*(s)$ and $J_{\mu_k}(s)$, respectively, for every state $s \in S$. For Markov Decision Problem evolving over a large state space, these algorithms become computationally infeasible.

4 Neuro-Dynamic Programming

Instead of computing the reward-to-go function for every state $s \in S$, methods of NDP use a parametric representation $\tilde{J}(\cdot, r)$ to approximate J^* and J_μ. In particular, we approximate $J^*(s)$ and $J_\mu(s)$ by a suitable approximation architecture $\tilde{J}(s, r)$, where r is a vector of tunable parameters (weights).

We present two NDP methods: TD(0) and Approximate Policy Iteration. TD(0) is related in structure to Value Iteration, whereas Approximate Policy Iteration emulates Policy Iteration. The TD(0) algorithm belongs to the class of Temporal Difference algorithms [10] (often referred to as TD(λ) algorithms), which are the most widely used NDP methods. Here, we will only state the iteration rule of the TD(0) algorithm; for a more comprehensive treatment of Temporal Difference methods, we refer to [2].

4.1 Temporal Difference Algorithm: TD(0)

The TD(0) algorithm is related to Value Iteration and can be used to find an approximation of J^*.

Let $\tilde{J} : S \times R^K \to R$ be a family of parametric representations such that $\nabla_r \tilde{J}(s, r)$ exists for every state $s \in S$ and every parameter vector $r \in R^K$. Choose an initial parameter vector $r_0 \in R^K$ and an initial state $s_0 \in S$. We generate a sequence (r_k) by the following recursive procedure:

1. Assume that we are given state s_k and parameter vector r_k; choose a control action u_k according to a greedy policy for $\tilde{J}(\cdot, r_k)$

$$u_k = \arg\max_{u \in U} \left[g(s_k, u) + \alpha \sum_{s' \in S} p(s_k, u, s') \tilde{J}(s', r_k) \right]$$

2. Choose the next state s_{k+1} at random, according to the transition probabilities $p(s_k, u_k, s_{k+1})$.

3. Update r_k by the following rule:

$$\begin{aligned} d_k &= \left(g(s_k, u_k) + \alpha \tilde{J}(s_{k+1}, r_k) - \tilde{J}(s_k, r_k) \right) \\ r_{k+1} &= r_k + \gamma_k d_k \nabla_r \tilde{J}(s_k, r_k) \end{aligned}$$

where $\gamma_k > 0$ is a small step size parameter and $\alpha \in [0, 1)$ is the discount factor. The scalar d_k is referred to as the temporal difference at iteration step k.

In general, the convergence results of Value Iteration do not apply to TD(0). However, for some classes of Markov Decision Problem, one can derive convergence results. We refer to [2] for a more detailed discussion of this issue.

4.2 Approximate Policy Iteration Using TD(0)

The general structure of Approximate Policy Iteration is the same as in ordinary Policy Iteration. Again, we generate a sequence of stationary policies $\mu_k : S \to U$. However, in Approximate Policy Iteration, we replace the exact reward-to-go function J_{μ_k} of the policy μ_k, by an approximation $\tilde{J}(\cdot, r_k)$

$$\mu_{k+1}(s) = \arg\max_{u \in U} \left[g(s, u) + \alpha \sum_{s' \in S} p(s, u, s') \tilde{J}(s', r_k) \right]$$

An approximation $\tilde{J}(\cdot, r)$ of $J_\mu(\cdot)$ for given policy μ can be obtained using TD(0), by setting

$$u_k = \mu(s_k)$$

in step 1 of its iteration rule.

Starting with an initial policy μ_0 and with the functional approximation $\tilde{J}(\cdot, r_0)$ of J_{μ_0}, we generate a sequence of policies in the following way:

1. Assume that we are given a policy μ_k and a functional approximation $\tilde{J}(\cdot, r_k)$ of J_{μ_k}. Choose μ_{k+1} to be a greedy policy for $\tilde{J}(\cdot, r_k)$

$$\mu_{k+1}(s) = \arg\max_{u \in U} \left[g(s, u) + \alpha \sum_{s' \in S} p(s, u, s') \tilde{J}(s', r_k) \right]$$

2. Compute the approximation $\tilde{J}(\cdot, r_{k+1})$ of $J_{\mu_{k+1}}$.

In general, Approximate Policy Iteration does not terminate, nor is the algorithm guaranteed to improve the reward-to-go function of the corresponding policies at each iteration step. Under certain assumptions however, bounds on its performance can be derived [2].

5 Architectures for Approximating J^* and J_μ

In this section, we describe the two parametric forms $\tilde{J}(\cdot, r)$ we use to approximate the optimal reward-to-go function J^* and the reward-to-go function J_μ of a stationary policy μ. They are multilayer perceptron and quadratic parameterizations.

A common nonlinear architecture for function approximation is the multilayer perceptron. Under this architecture, the input vector $s \in R^N$ is transformed by a matrix to give a vector $v \in R^L$:

$$v(l) = \sum_{n=1}^{N} r(l,n)s(n)$$

Each component of the vector v is then transformed by non-linear sigmoidal function $\sigma : R \to R$, which is differentiable, monotonically increasing, and has the property:

$$-\infty < \lim_{x \to -\infty} \sigma(x) < \lim_{x \to +\infty} \sigma(x) < +\infty$$

The sigmoidal function we use is given by the following rule:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The sigmoidal function σ is used to transform the vector $v \in R^L$ to a new vector $w \in R^L$ such that

$$w(l) = \sigma(v(l)) = \sigma\left(\sum_{n=1}^{N} r(l,n)s(n)\right)$$

The components of the vector $w \in R^L$ are then linearly combined using coefficients $r(l)$ to produce the output

$$\tilde{J}(s,r) = r(0) + \sum_{l=1}^{L} r(l)w(l) = r(0) + \sum_{l=1}^{L} r(l)\sigma\left(\sum_{n=1}^{N} r(l,n)s(n)\right)$$

The quadratic approximation for an input vector $s \in R^N$ is given by the following rule

$$\tilde{J}(s,r) = r(0) + \sum_{n=1}^{N} r(n)s(n) + \sum_{m=1}^{N}\sum_{n=1}^{N} r(m,n)s(m)s(n)$$

5.1 Input and Output Scaling

Input and output scaling can have a significant effect on the performance of training methods (such as TD(0)) for function approximations. Typically, the objective of input and output scaling is to force each component of the input and output vector to lie in the interval $[-1, 1]$.

Let $\tilde{J}(., r) : R^N \rightarrow R$ be an approximation architecture; let $s \in R^N$ be an input vector, let $m_x \in R^N, \sigma_x \in R^N, m_y \in R$, and $\sigma_y \in R$ be scaling vectors. Then the scaled input and output are given as follows

$$\begin{aligned} s'(n) &= \frac{s(n) - m_x(n)}{\sigma_x(n)} \\ y' &= \sigma_y J(s', r) + m_y \end{aligned}$$

6 Formulation of the Call Admission Control Problem in ATM networks as a Markov Decision Problem

In this section, we formulate the call admission control problem for a single link in an integrated service environment as an infinite-horizon, discounted Markov Decision Problem. We will initially adopt the framework of continuous-time Markov Decision Problems and later transform the problem into a discrete-time Markov Decision Problem through uniformization.

We describe the state of the single communication link by an N-tuple $s = (s(1), ..., s(N))$, where $s(n)$ denotes the number of customers of class n currently using the link. The state space S is given by

$$S = \left\{ s \in R^N \;\middle|\; \sum_{n=1}^{N} s(n)b(n) \leq B, \quad s(n) \in \{0, 1, 2, ...\} \right\}$$

where $b(n)$ is the bandwidth demand of a customer of class n, and B is the total available bandwidth of the communication link. Let s_t denote the state of the system at time $t \in [0, +\infty)$.

A control action $u = (u(1), ..., u(N))$ is an N-tuple such that each $u(n)$ equals either 0 or 1. Given a control action u, we accept a new connection request of a customer of class n, if $u(n)$ equals 1, and reject a new customer of class n, otherwise. Let U denote the set of all possible control actions:

$$U = \left\{ u \;\middle|\; u \in \{0, 1\}^N \right\}$$

We say that an event occurs at a certain time if a customer departs from the system or a new customer requires a connection over the communication link.

Let t_k be the time of the kth event. By convention, we start the system at time $t_0 = 0$; t_1 is the time when the first event occurs. We identify an event by the N-tuple $\omega = (\omega(1), ..., \omega(N))$ where $\omega(n)$ equals 1 if a new customer of class n requests a connection; $\omega(n)$ equals -1 if a customer of class n departs from the system and $\omega(n)$ equals 0 otherwise. Let Ω denote the sets of all possible events:

$$\Omega = \left\{ \omega \;\middle|\; \omega \in \{-1, 0, 1\}^N, \;\; \sum_{n=1}^{N} |\omega(n)| = 1 \right\}$$

Let s_k be the state of the system in the interval $(t_k, t_{k+1}]$. Note that if the system is in state s_k, the probability that the next event will be a specific event ω is determined by the arrival rates $\lambda(n)$ and the departure rates $\nu(n)$.

Given a state $s \in S$, an event $\omega \in \Omega$ and a control action $u \in U$, the next state s' is given by a function $f : S \times \Omega \times U \to S$ such that if s' equals $f(s, \omega, u)$, then the following holds:

$$s'(n) = \begin{cases} s(n) & \text{if } \omega(n) = 0 \\ s(n) & \text{if } \omega(n) = 1 \text{ and } u(n) = 0 \\ s(n) + 1 & \text{if } \omega(n) = 1 \text{ and } u(n) = 1 \\ s(n) - 1 & \text{if } \omega(n) = -1 \text{ and } s(n) > 0 \\ s(n) & \text{if } \omega(n) = -1 \text{ and } s(n) = 0 \end{cases}$$

We associate a one stage reward $g(s, \omega, u)$ with a state s, an event ω, and a control action u, according to the formula:

$$g(s, \omega, u) = \begin{cases} c(n) & \text{if } \omega(n) = 1 \text{ and } u(n) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Here, $c(n)$ is the reward associated with admitting a customer of class n.

A stationary policy is a function $\mu : S \to U$ and induces a Markov Process on the state space S.

With a stationary policy μ and a state s we associate the discounted reward-to-go $J_\mu(s)$:

$$J_\mu(s) = E\left[\sum_{k=0}^{\infty} e^{-\beta t_{k+1}} g(s_k, \omega_{k+1}, \mu(s_k)) \,|s_0 = s \right]$$

where the following condition is satisfied:

$$s_{k+1} = f(s_k, \omega_{k+1}, \mu(s_k))$$

and where β is a positive real number, called the discount rate. Note that in this formulation the first stage reward is discounted.

With every state s in the state space, we can associate a so called rate of transition $v(s)$ given by

$$v(s) = \sum_{n=1}^{N} [\lambda(n) + s(n)\nu(n)]$$

Using uniformization, the continuous-time Markov Decision Problem can be transformed into an equivalent discrete-time Markov Decision Problem with a state-dependent discount factor

$$\alpha(s) = \frac{\upsilon(s)}{\beta + \upsilon(s)}$$

In particular, Bellman's equation takes the form

$$J(s) = \alpha(s) \max_{u \in U} \left[\sum_{\omega \in \Omega} p(s,\omega) \left[g(s,\omega,u) + J(f(s,\omega,u)) \right] \right]$$

where the probabilities $p(s,\omega)$ are given as follows

$$p(s,\omega) = \begin{cases} \frac{\lambda(n)}{\upsilon(s)} & \text{if } \omega(n) = 1 \\ \\ \frac{s(n)\nu(n)}{\upsilon(s)} & \text{if } \omega(n) = -1 \end{cases}$$

Note that the cardinality of the state space increases exponentially with the number of customer classes. Therefore, for call admission control problems which involve a fair number of customer classes, the classical methods of Dynamic Programming are not feasible.

6.1 Formulation of the TD(0) algorithm for the Call Admission Control Problem

In this section, we describe how one implements the TD(0) algorithm for the call admission control problem.

Let $\tilde{J} : S \times R^K \to R$ be a family of parametric representations such that $\nabla_r \tilde{J}(s,r)$ exists for every state $s \in S$ and every parameter vector $r \in R^K$. Choose an initial parameter vector $r_0 \in R^K$ and an initial state $s_0 \in S$. We generate a sequence (r_k) by the following recursive procedure:

1. Assume that we are given state s_k and parameter vector r_k: choose the next event ω_{k+1} at random, according to the probabilities $p(s_k, \omega_{k+1})$.

2. If the next event ω_{k+1} corresponds to a departure of a customer, set the control action u_k equal to 0. If the next event ω_{k+1} corresponds to an arrival of a new customer of class n, choose the control action u_k as follows:

$$u_k = \begin{cases} e_n & \text{if } \tilde{J}(s_k, r_k) - \tilde{J}(s_k + e_n, r_k) \leq c(n) \\ & \text{and } \sum_{n=1}^{N} s_k(n) b(n) \leq B - b(n) \\ 0 & \text{otherwise} \end{cases}$$

where e_n is the nth unit vector.

3. Set s_{k+1} equal to $f(s_k, \omega_{k+1}, u_k)$.

4. Update r_k by the following rule:

$$\begin{aligned} d_k &= \left(\alpha(s)\left[g(s_k, \omega_{k+1}, u_k) + \tilde{J}(s_{k+1}, r_k)\right] - \tilde{J}(s_k, r_k)\right) \\ r_{k+1} &= r_k + \gamma_k d_k \nabla_r \tilde{J}(s_k, r_k) \end{aligned}$$

7 Heuristic

In order to assess the policies we obtain through NDP, we compare them with two heuristic control policies: one which we will call the "Always Accept" policy, and one to which we refer as the "Threshold" policy.

The Always Accept policy accepts a new customer of class n if the required bandwidth $b(n)$ is available, and otherwise rejects it; e.g. if at time t a customer of class n requires a connection over the link, the new customer is accepted to the system if $B(t)+b(n) \leq B$ and is otherwise rejected. Here, $B(t)$ is the bandwidth used at time t:

$$B(t) = \sum_{n=1}^{N} s_t(n) b(n)$$

The Threshold policy specifies for each customer class n a threshold parameter $h(n) \geq 0$, in units of bandwidth. If a customer of class n requests a new connection at time t over the communication link, we accept it only if $B(t) + b(n) + h(n) \leq B$.

8 Case Studies

In this section, we present two case studies: one involving 3 customer classes, and one involving 10 customer classes. A description of the parameters of the case studies is given in Section 8.1; a discussion of the results is provided in Section 8.2. Numerical results are reported in the Appendix.

For both cases, we implemented the TD(0) algorithm with the two approximation architectures described in Section 5 and Approximate Policy Iteration. The control policies obtained by these methods of NDP are compared with the Always Accept policy and Threshold policy. Simulating the policies of NDP, we could observe which customer classes get sometimes rejected, and what percentage of customers of a particular class gets rejected. This insight guided the tuning of the threshold parameters $h(n)$ of the Threshold policy, which was carried out manually. In particular, we don't expect our choices of $h(n)$ to be optimal.

As the state space in the first case study is relatively small, we are able to compute an optimal control policy using exact Dynamic Programming and to compare it with the policies obtained through NDP.

We use the average reward per unit time and the lost average reward per unit time as performance measures to compare the different policies. Based on a trajectory of N simulation steps, we obtain the average reward for a stationary policy μ as follows

$$\frac{1}{T_N} \sum_{k=0}^{N} g(s_k, \omega_{k+1}, \mu(s_k))$$

and the lost average reward by

$$\frac{1}{T_N} \sum_{k=0}^{N} [g(s_k, \omega_{k+1}, (1, 1, ..., 1)) - g(s_k, \omega_{k+1}, \mu(s_k))]$$

where

$$T_N = \sum_{k=0}^{N} \frac{1}{v(s_k)}$$

Note that the lost average reward refers to the additional reward that would have been obtained if no customer was ever rejected, i.e. if we had an infinite bandwidth link. To evaluate the average reward and the lost average reward, we used a trajectory of 4,000,000 simulation steps, which starts with an empty system. The trajectory was generated using a random number generator, initialized with the same seed for each evaluation.

8.1 Parameters

Here we give a brief description of the parameters used in the two case studies, the complete definition of the parameters can be found in the Appendix in Table 1, 3, 5 and 7.

In the first case study, we consider a communication link with a total bandwidth of 12 units. We intend to support 3 customer classes on that link. The discount rate was set to be 0.001. As a multilayer perceptron for TD(0) and Approximate Policy Iteration, we used an architecture with 5 hidden units.

In the second case study, we consider a communication link with a total bandwidth of 600 units. We intend to support 10 customer classes on that link. We varied the arrival rates of the different customer classes to yield three different scenarios: a highly loaded, a medium loaded and a lightly loaded link. The discount rate was set to be 0.1 for the highly loaded and medium loaded scenarios, and to 0.01 for lightly loaded scenario. As a multilayer perceptron for TD(0) and Approximate Policy Iteration, we used an architecture with 10 hidden units. The parameters which characterize the different service types are chosen in such a way, that there are pairs of service classes which differ only in the reward $c(n)$. Here, we interpret the reward $c(n)$ as a "virtual" reward. If the customer class n has a higher reward $c(n)$, but the same bandwidth demand $b(n)$, the same arrival rate $\lambda(n)$ and the same departure rate $\nu(n)$ as the customer

class m, then customers of class n should get a higher quality of service than customers of class m. This should be reflected by a good call admission control policy by blocking less customers of class n than customers of class m.

The stepsize parameter γ_k in the iteration rule of the TD(0) algorithm was chosen in the following way: we define an initial stepsize parameter γ, a multiplication factor δ and a positive integer Δ. Every Δ steps, we update γ to $\delta\gamma$.

We use in the Appendix "API" as an abbreviation of Approximate Policy Iteration; the parameters m_x, σ_x, m_y and σ_y refer to the scaling vectors defined in Section 5.1.

8.2 Results

The experimental results are given in the Appendix in Table 2, 4, 6, and 8, and in Figure 1 through Figure 3. Here, we will provide some more detailed information concerning the simulation runs and the experimental results.

In the iteration rule of the TD(0) for finding a functional approximation of J^* and J_μ; the initial state s_0 was set equal to $(0, 0, ...0)$, the state which corresponds to an empty system and the initial parameter vector r_0 was chosen at random.

In Figure 1 through Figure 3; for the TD(0) algorithm, the lost average reward was taken to be the corresponding average over a time window. The length of the time window is indicated in the label of the figures.

For the Approximate Policy Iteration algorithm, the performance results (Percentage of Customers Rejected, Percentage of Customers Rejected by Control, Average Reward and Lost Average Reward) correspond to the policy which attained the best performance among all the policies obtained through the iteration. Note that "Customers Rejected by Control" are those customers who are rejected even though the required bandwidth was available.

The main conclusions from the experimental results are the following:

- **Methods of NDP seem to be promising for the call admission control problem for a Single Link**. In the two case studies, methods of NDP lead to significantly better results than the heuristic Always Accept policy, except for the case of a lightly loaded link which supports 10 different customer types, where the performance of the two policies was the same. In particular, in all cases (except for the above mentioned case) the lost average reward due to rejection of customers, could be reduced by $10\% - 35\%$. Furthermore, for the case of 3 customer classes, essentially optimal performance was attained.

- **Methods of NDP can provide insight into a problem at hand, which allows to design good heuristic policies**. In our case, simulation of control policies obtained with NDP revealed that only a few customer classes get rejected by the control. This observation lead to the

formulation of the heuristic Threshold policy and guided the tuning of the threshold parameters.

- **Among the methods of NDP, Approximate Policy Iteration led to the best results.** However, Approximate Policy Iteration led to only slightly better policies than the ones obtained from TD(0) methods.

- **The performance of the policies obtained through Approximate Policy Iteration can oscillate significantly.** This is consistent with experience in other contexts, as well as with the analysis provided in [2].

- **The heuristic Threshold policy led to the best system performance in all cases.** However, this may not be the case for more complex problems, where good heuristic policies are hard to design. Furthermore, as mentioned earlier, NDP can be used in the design of powerful heuristics.

- **Exploration of the state space can be important.** We have noted that NDP methods did not lead to improvement for a lightly loaded system with 10 customer classes. The natural explanation is that a lightly loaded system rarely becomes full and therefore very little training takes place at these states where the control policy matters (near full system). How to sample the state space in places that have low probability under the policy employed, while preserving the stability of the algorithms employed is an intricate subject that is poorly understood at present.

Appendix

Customer Classes				
Customer Class	**Bandwidth,** $b(n)$	**Arrival Rate,** $\lambda(n)$	**Departure Rate,** $\nu(n)$	**Reward** $c(n)$
1	1	3.0	0.5	4
2	2	2.0	0.8	15
3	2	2.5	0.9	12

Scaling Vectors	
m_x	[0,0,0]
σ_x	[12,6,6]
m_y	40300
σ_y	50

Methods		
Method	**Architecture**	**Parameters**
Always Accept		
Threshold Policy		h = [11; 0; 0]
DP		
TD(0)	MLP	$\gamma = 0.01$ $\delta = 0.8$ Δ = 1,000,000
TD(0)	Quadratic	$\gamma = 0.01$ $\delta = 0.8$ Δ = 1,000,000
API	MLP	$\gamma = 0.01$ $\delta = 0.8$ Δ = 1,000,000

Table 1: Case study for 3 different customer classes.

Percentage of Customers Rejected			
	1	2	3
Always Accept	26.1	47.8	47.9
Threshold Policy	100.0	21.1	21.2
DP	100.0	21.1	21.2
TD(0): MLP	98.6	21.3	21.4
TD(0): Quadratic	98.6	21.3	21.4
API: MLP	100.0	21.1	21.2

Percentage of Customers Rejected by Control			
	1	2	3
Always Accept	0.0	0.0	0.0
Threshold Policy	78.8	0.0	0.0
DP	78.8	0.0	0.0
TD(0): MLP	78.7	0.0	0.0
TD(0): Quadratic	78.7	0.0	0.0
API: MLP	78.8	0.0	0.0

Performance		
	Average Reward	**Lost Average Reward**
Always Accept	40.09	31.86
Threshold Policy	47.23	24.72
DP	47.23	24.72
TD(0): MLP	47.19	24.76
TD(0): Quadratic	47.19	24.76
API: MLP	47.23	24.72

Table 2: Case study for 3 different customer classes.

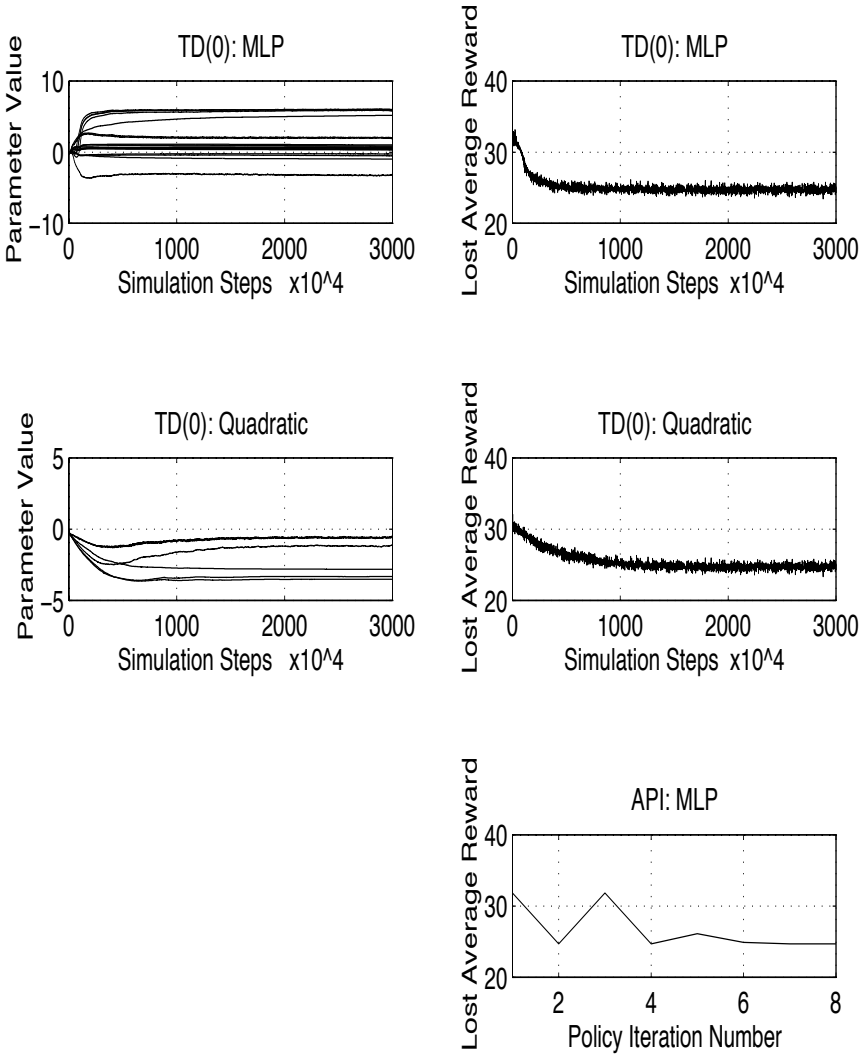

Figure 1: Case study for 3 different customer classes.

Customer Classes				
Customer Class	**Band-width,** $b(n)$	**Arrival Rate,** $\lambda(n)$	**Departure Rate,** $\nu(n)$	**Reward** $c(n)$
1	2	15.0	1.0	2.0
2	2	15.0	1.0	1.4
3	4	12.0	0.8	5.0
4	4	12.0	0.8	2.5
5	6	10.0	0.6	10.0
6	6	10.0	0.6	4.0
7	8	6.0	0.4	20.0
8	8	6.0	0.4	7.0
9	10	4.0	0.2	5.0
10	10	4.0	0.2	16.0

Scaling Vectors	
m_x	[0,0,0,0,0,0,0,0,0,0]
σ_x	[300,300,150,150,100,100,75,75,60,60]
m_y	4800
σ_y	10

Methods		
Method	**Architecture**	**Parameters**
Always Accept		
Threshold Policy		h = [0;0;0;79;0;594;0;592;0;140]
TD(0)	MLP	$\gamma = 0.001$ $\delta = 0.8$ Δ = 1,000,000
TD(0)	Quadratic	$\gamma = 0.1$ $\delta = 0.5$ Δ = 1,000,000
API	MLP	$\gamma = 0.001$ $\delta = 0.8$ Δ = 1,000,000

Table 3: Case study for 10 different customer classes: highly loaded link.

Percentage of Customers Rejected										
	1	2	3	4	5	6	7	8	9	10
Always Accept	13.88	13.78	25.73	25.83	35.95	36.05	45.21	45.04	53.05	52.81
Threshold Policy	1.04	1.03	2.16	69.18	3.23	100.00	4.45	100.00	5.57	98.96
TD(0): MLP	2.42	2.66	7.28	79.84	8.36	94.06	10.49	92.94	13.23	73.09
TD(0): Quadratic	0.92	0.90	1.89	99.99	2.86	99.99	3.90	99.99	4.88	99.97
API: MLP	1.03	1.01	2.13	71.71	3.22	99.99	4.39	99.99	5.43	99.99

Percentage of Customers Rejected by Control										
	1	2	3	4	5	6	7	8	9	10
Always Accept	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Threshold Policy	0.00	0.00	0.00	67.06	0.00	96.67	0.00	95.51	0.00	93.30
TD(0): MLP	0.00	0.26	2.26	74.82	0.77	86.43	0.00	82.42	0.00	59.82
TD(0): Quadratic	0.00	0.00	0.00	98.09	0.00	97.13	0.00	96.12	0.00	95.02
API: MLP	0.00	0.00	0.00	69.60	0.00	96.83	0.00	95.69	0.00	94.56

Performance		
	Average Reward	**Lost Average Reward**
Always Accept	412.77	293.15
Threshold Policy	518.17	187.67
TD(0): MLP	505.46	200.34
TD(0): Quadratic	511.45	194.71
API: MLP	517.38	188.54

Table 4: Case study for 10 different customer classes: highly loaded link.

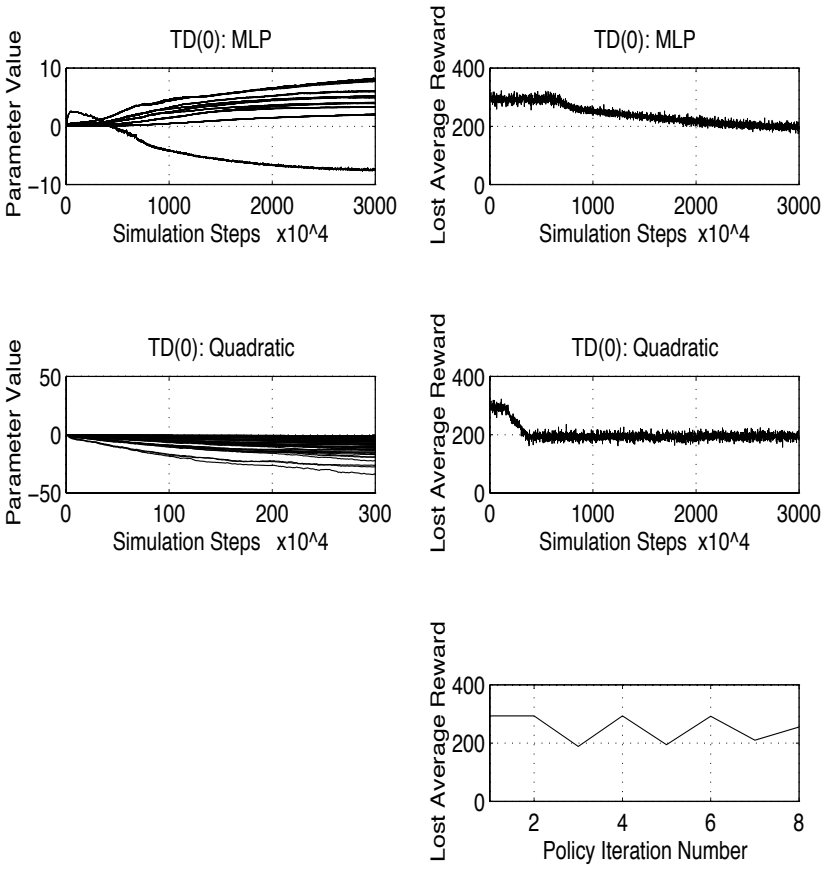

Figure 2: Case study for 10 different customer classes: highly loaded link.

Customer Classes				
Customer Class	**Bandwidth,** $b(n)$	**Arrival Rate,** $\lambda(n)$	**Departure Rate,** $\nu(n)$	**Reward** $c(n)$
1	2	15.0	1.0	2.0
2	2	15.0	1.0	1.4
3	4	10.0	0.8	5.0
4	4	10.0	0.8	2.5
5	6	7.0	0.6	10.0
6	6	7.0	0.6	4.0
7	8	3.0	0.4	20.0
8	8	3.0	0.4	7.0
9	10	1.8	0.2	5.0
10	10	1.8	0.2	16.0

Scaling Vectors	
m_x	[0,0,0,0,0,0,0,0,0,0]
σ_x	[300,300,150,150,100,100,75,75,60,60]
m_y	3800
σ_y	10

Methods		
Method	**Architecture**	**Parameters**
Always Accept		
Threshold Policy		h = [0;0;0;0;0;29;0;31;0;0]
TD(0)	MLP	$\gamma = 0.01$ $\delta = 0.5$ Δ = 1,000,000
TD(0)	Quadratic	$\gamma = 0.1$ $\delta = 0.5$ Δ = 1,000,000
API	MLP	$\gamma = 0.01$ $\delta = 0.5$ Δ = 1,000,000

Table 5: Case study for 10 different customer classes: medium loaded link.

Percentage of Customers Rejected										
	1	2	3	4	5	6	7	8	9	10
Always Accept	2.45	2.41	4.91	4.89	7.16	7.36	9.61	9.92	12.27	11.90
Threshold Policy	0.77	0.78	1.63	1.62	2.53	30.01	3.49	35.63	4.73	4.71
TD(0): MLP	1.43	1.39	2.93	2.89	4.43	20.86	6.13	18.79	8.00	11.21
TD(0): Quadratic	0.76	0.74	1.58	1.54	2.39	99.98	3.14	3.16	4.18	3.91
API: MLP	0.13	0.13	0.28	2.39	0.42	31.35	0.60	41.94	0.85	33.11

Percentage of Customers Rejected by Control										
	1	2	3	4	5	6	7	8	9	10
Always Accept	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Threshold Policy	0.00	0.00	0.00	0.00	0.00	27.47	0.00	31.71	0.00	0.00
TD(0): MLP	0.00	0.00	0.00	0.00	0.00	16.40	0.00	12.60	0.00	3.42
TD(0): Quadratic	0.00	0.00	0.00	0.00	0.00	97.68	0.00	0.00	0.00	0.00
API: MLP	0.00	0.00	0.00	2.14	0.00	30.91	0.00	41.31	0.00	32.27

Performance		
	Average Reward	**Lost Average Reward**
Always Accept	389.53	34.31
Threshold Policy	396.68	26.97
TD(0): MLP	393.68	29.87
TD(0): Quadratic	385.39	38.63
API: MLP	394.18	29.34

Table 6: Case study for 10 different customer classes: medium loaded link.

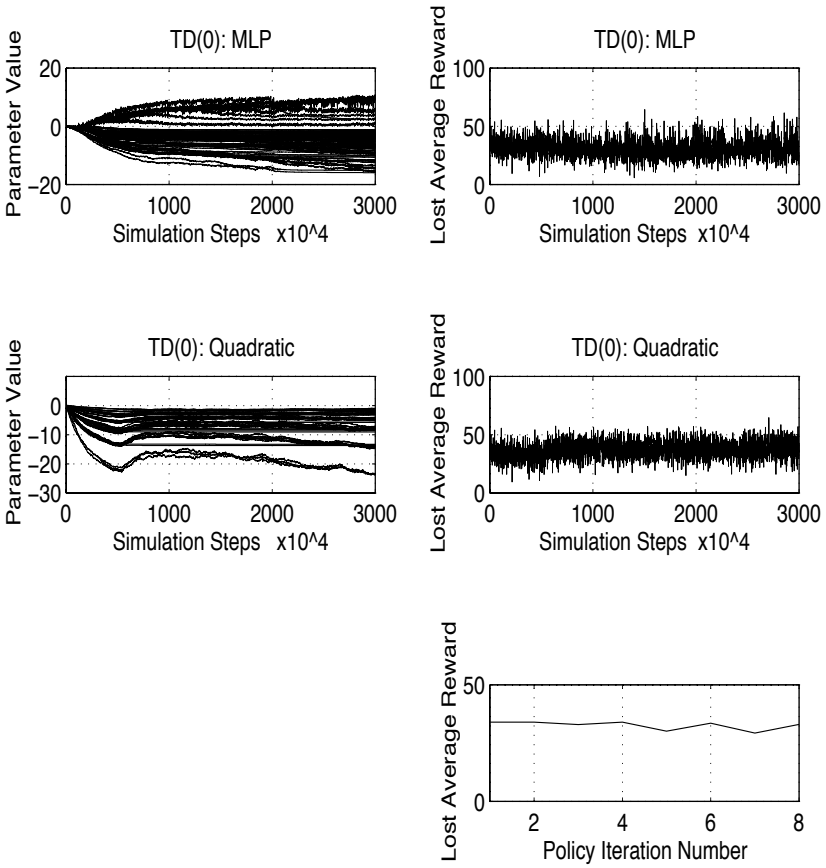

Figure 3: Case study for 10 different customer classes: medium loaded link.

Customer Classes				
Customer Class	**Band-width,** $b(n)$	**Arrival Rate,** $\lambda(n)$	**Departure Rate,** $\nu(n)$	**Reward** $c(n)$
1	2	16.0	1.0	2.0
2	2	16.0	1.0	1.4
3	4	12.0	0.8	5.0
4	4	12.0	0.8	2.5
5	6	7.0	0.6	10.0
6	6	7.0	0.6	4.0
7	8	2.4	0.4	20.0
8	8	2.4	0.4	7.0
9	10	1.1	0.2	5.0
10	10	1.1	0.2	16.0

Scaling Vectors	
m_x	[0,0,0,0,0,0,0,0,0,0]
σ_x	[300,300,150,150,100,100,75,75,60,60]
m_y	3580
σ_y	10

Methods		
Method	**Architecture**	**Parameters**
Always Accept		
Threshold Policy		h = [0;0;0;0;0;29;0;27;0;0]
TD(0)	MLP	$\gamma_0 = 0.01$ $\delta = 0.8$ Δ = 10,000,000
TD(0)	Quadratic	$\gamma_0 = 0.1$ $\delta = 0.5$ Δ = 10,000,000
API	MLP	$\gamma_0 = 0.01$ $\delta = 0.8$ Δ = 10,000,000

Table 7: Case study for 10 different customer classes: lightly loaded link.

Percentage of Customers Rejected										
	1	2	3	4	5	6	7	8	9	10
Always Accept	0.74	0.74	1.50	1.48	2.28	2.24	3.13	3.21	3.98	4.0
Threshold Policy	0.20	0.20	0.43	0.41	0.68	11.86	0.96	12.03	1.28	1.30
TD(0): MLP	0.74	0.74	1.50	1.48	2.28	2.24	3.13	3.21	3.98	4.0
TD(0): Quadratic	0.74	0.74	1.50	1.48	2.28	2.24	3.13	3.21	3.98	4.0
API: MLP	0.74	0.74	1.50	1.48	2.28	2.24	3.13	3.21	3.98	4.0

Percentage of Customers Rejected by Control										
	1	2	3	4	5	6	7	8	9	10
Always Accept	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Threshold Policy	0.00	0.00	0.00	0.00	0.00	11.21	0.00	11.10	0.00	0.00
TD(0): MLP	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TD(0): Quadratic	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
API: MLP	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Performance		
	Average Reward	**Lost Average Reward**
Always Accept	370.82	8.89
Threshold Policy	372.40	7.70
TD(0): MLP	370.82	8.89
TD(0): Quadratic	370.82	8.89
API: MLP	370.82	8.89

Table 8: Case study for 10 different customer classes: lightly loaded link.

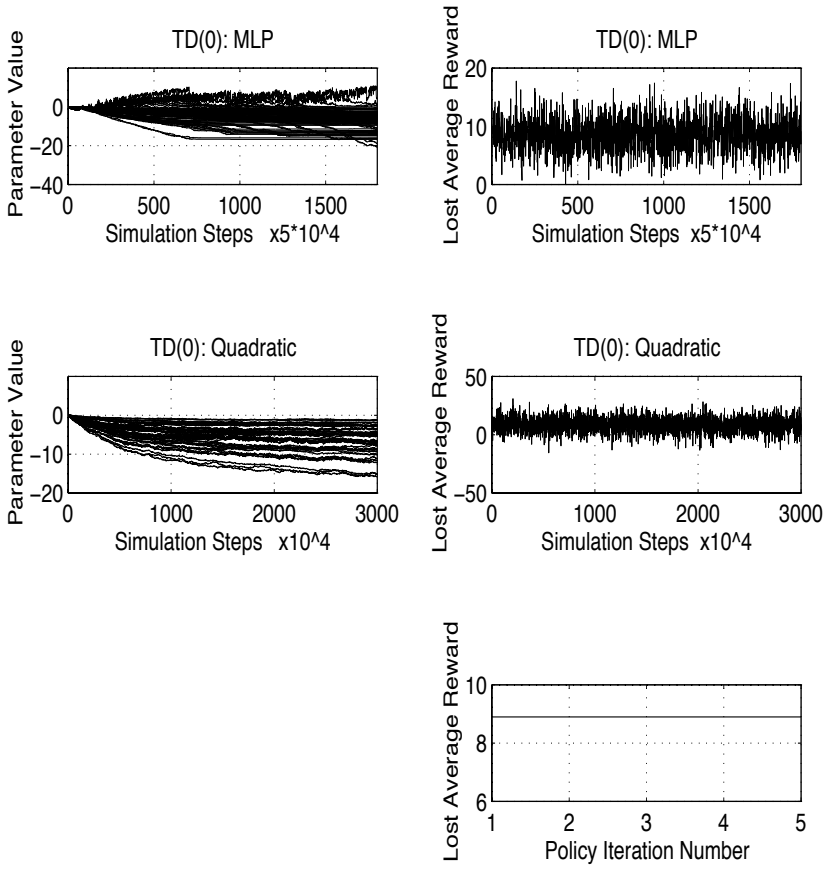

Figure 4: Case study for 10 different customer classes: lightly loaded link.

References

[1] D. P. Bertsekas, "*Dynamic Programming and Optimal Control*," Athena Scientific, 1995.

[2] D. P. Bertsekas and J. N. Tsitsiklis, "*Neuro-Dynamic Programming*," Athena Scientific, 1996.

[3] G. J. Tesauro, "*Practical Issues in Temporal-Difference Learning*," Machine Learning, vol. 8, 1988.

[4] R. H. Crites and A. G. Barto, "*Improving Elevator Performance Using Reinforcement Learning* ," Advances in Neural Information Processing Systems 8, MIT Press, 1996.

[5] W. Zhang and T. G. Dietterich, "*High Performance Job-Shop Scheduling with a Time-Delay TD(λ) Network*," Advances in Neural Information Processing Systems 8, MIT Press, 1996.

[6] S. Singh and D. P. Bertsekas, "*Reinforcement Learning for Dynamic Channel Allocation in Celluar Telephone Systems*," Advances in Neural Information Processing Systems 9, MIT Press, 1997.

[7] E. Nordström, J. Carlström, O. Gällmo and L. Apslund, "*Neural Networks for Adaptive Traffic Control in ATM Networks*," IEEE Communication Magazine, Oct. 1995.

[8] E. Nordström and J. Carlström, "*A Reinforcement Scheme for Adaptive Link Allocation in ATM Networks*," Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications 2, 1995.

[9] IEEE J. Select. Areas Commun., vol. 13, nos.6-7, Aug-Sept 1995.

[10] R. S. Sutton, "*Learning to Predict by the Methods of Temporal Differences*," Machine Learning, vol. 3, 1988.