# Approximate and Data-Driven Dynamic Programming for Queueing Networks

Ciamac C. Moallemi
Graduate School of Business
Columbia University
email: ciamac@gsb.columbia.edu

Sunil Kumar
Graduate School of Business
Stanford University
email: skumar@stanford.edu

Benjamin Van Roy
Management Science & Engineering
Electrical Engineering
Stanford University
email: bvr@stanford.edu

Initial Version: December 13, 2006
Revised: September 8, 2008

**Abstract**

**We develop an approach based on temporal difference learning to address scheduling problems in complex queueing networks such as those arising in service, communication, and manufacturing systems. One novel feature is the selection of basis functions, which is motivated by the gross behavior of the system in asymptotic regimes. Another is the use of polytopic structure to efficiently identify desired actions from an intractable set of alternatives. Application to input-queued crossbar switch models with up to hundreds of queues and quadrillions of alternative actions yield scheduling policies outperforming a heuristic recently shown to have certain optimality properties in the heavy traffic scale. We also extend the approach to a setting where aspects of the queueing network are not modeled and we must rely instead on empirical data. This data-driven approach is useful, for example, when the statistical structure of arrivals is poorly understood but historical data traces are available.**

## 1. Introduction

In principle, optimal control problems can be solved by the methods of dynamic programming. However, the curse of dimensionality often gives rise to prohibitive computational requirements. Approximate dynamic programming (ADP) methods aim to circumvent this computational burden and yet obtain near optimal control policies.

ADP has a long history, dating back to early work in the 1950s [1, 2, 3], but the past two decades have witnessed a vibrant research activity in this area. Several books and papers review segments of the growing literature [4, 5, 6, 7, 8, 9], which covers algorithms, theoretical analyses, computational studies, and applications. Many ideas emerging from this line of research are useful to the design of solution methods for particular problems. However, there is no 'turn-key' ADP algorithm that

is effective across the entire range of optimal control problems. Rather, for a problem at hand, one can combine and customize ingredients from the literature to generate an effective algorithm. This process is somewhat of an art, requiring tinkering and iterative trial and error.

In this paper, we develop an ADP approach to address scheduling problems in complex queueing networks. This is a broad class of optimal control problems that subsumes many useful models of service, communication, and manufacturing systems. Practical instances typically give rise to enormous state and action spaces, which grow exponentially in the number of queues. Our aim is to provide a streamlined method that can be applied to problems in this class without significant custom design or trial and error.

As opposed to the broader class of general optimal control problems, queueing networks possess a lot of special structure. Further, this structure is well-studied, with a large literature of theoretical and empirical work. Many heuristics have been proposed and analyzed. Sophisticated mathematical tools such as fluid and diffusion limit models offer intuition that guide the design of scheduling policies. It is through leveraging this special structure and the wealth of knowledge about it that we hope to develop a turn-key ADP approach.

Our approach uses temporal difference learning (TD-learning) [10] to fit a linear combination of basis functions to the dynamic programming differential cost function. We propose a new method for selecting the basis functions, which is motivated by the gross behavior of the system in the asymptotic regimes. In particular, the asymptotic theory suggests use of a basis consisting of ridge functions with orientations given by workload vectors and individual queue lengths.

Scheduling decisions can be made by taking actions that are greedy with respect to an approximate differential cost function. However, action spaces in queueing problems are typically enormous, making it impossible to enumerate and evaluate each alternative to identify a greedy action. This presents an obstacle both to real-time control and the application of TD-learning, which requires simulation of state trajectories controlled by greedy policies associated with intermediate approximations to the differential cost function. To address this issue, we develop a scalable method that exploits polytopic structure to efficiently compute greedy actions.

As a case study, we apply our approach to models of input-queued crossbar switches. These models involve up to hundreds of queues and quadrillions of alternative actions. Scheduling policies generated by our approach reduce average delay by up to about 4.5% relative to MWM-0+, a well studied heuristic for which a notion of heavy traffic scale optimality has been established [11, 12]. This result is somewhat surprising considering a sentiment among some researchers in the networking community that this heuristic is near optimal.

We also extend our ADP approach to a setting where aspects of the queueing network are not modeled and the optimization process relies instead on empirical data. This data-driven approach can be viewed as integrating ideas from $Q$-learning [13] in a way that leverages the special structure of queueing problems to simplify representation of the $Q$-function. We anticipate that this data-driven approach will be useful in many applied contexts. For example, this approach alleviates the need to formulate a model when the statistical structure of arrivals is poorly understood. Instead, the ADP algorithm can work directly with historical data traces.

To summarize, contributions of this paper include:

1. a method for selecting basis functions, which is motivated by the gross behavior of queueing networks in the asymptotic regime of heavy traffic;

2. a scalable method that exploits polytopic structure to efficiently compute greedy actions;

3. application to crossbar switch models, leading to scheduling policies that reduce average delay by up to about 4.5% relative to MWM-0+;

4. a data-driven ADP method that can be applied when aspects of the queueing network are not modeled and the optimization process must rely instead on empirical data.

These contributions build on other recent work about ADP in queueing networks. This includes simple case studies carried out by de Farias and Van Roy [14, 15], Choi and Van Roy [16], and Adelman [17], as well as the work of Veatch [18]. The latter also leverages insights from asymptotic regimes, but this work differs from ours in a crucial way. The basis functions proposed there grow combinatorially in the size of the network, and computing policies for a particular problem instance requires solution of an LP that is exponential in the number of buffers. Hence the examples considered are quite small (up to six buffers).

The rest of the paper is organized as follows. In Section 2, we describe a prototypical mathematical model for a queueing network and the formulation of the associated control problem as a Markov decision process. In Section 3, we present ADP as a method for finding control policies and discuss properties of the queueing network which are important for efficient implementation. In Section 4, we present a method for basis function selection. In Section 5, we discuss a data-driven ADP methodology. Finally, in Section 6, we present empirical results in the case of an input-queued crossbar switch.

## 2. A Prototypical Queueing Network

In this section, we describe a prototypical mathematical model of a queueing network, and we represent the associated optimal control problem as a Markov decision process. At a high-level, this model describes an open queueing network operating in continuous-time, with multiple classes of jobs, and where processor sharing is not allowed. Scheduling decisions are allowed to be preemptive. Both controlled and uncontrolled routing is allowed, as are activities which must simultaneously process multiple inputs. Arrival processes are assumed to be Poisson and service times are assumed to be exponential.

This framework is not meant to be comprehensive in its scope—indeed, the case study we consider in Section 6 does not fall within this formulation. However, it is sufficiently general to cover many queueing networks of interest and to be illustrative in broader contexts.

### 2.1. System Evolution

Consider a queueing network consisting of a set $\mathcal{I} = \{1, \ldots, I\}$ of buffers (queues). The network operates in continuous time. Each buffer holds a set of jobs awaiting service and is of infinite capacity. Arrivals from outside the system occur according to independent Poisson processes, and we use $\lambda_i \geq 0$ to denote the arrival rate for buffer $i$.

There is a set $\mathcal{J} = \{1, \ldots, J\}$ of activities. Each activity $j \in \mathcal{J}$ processes inputs from some subset of the buffers simultaneously. This relationship is encoded in a matrix $B \in \mathbb{R}^{I \times J}$ by setting $B_{ij} = 1$ if jobs from buffer $i$ are processed by activity $j$, and setting $B_{ij} = 0$ otherwise. The set of buffers associated with an activity $j$ is referred to as the constituency of that activity and denoted by $\mathcal{B}_j$. The processing time required by activity $j$ is an independent exponential random variable with parameter $\mu_j \geq 0$.

Once a job is processed, it may be routed to a different queue or it may leave the system. In particular, upon a service completion for an activity $j \in \mathcal{J}$, one job is removed from each constituent buffer $i \in \mathcal{B}_j$. There is a sub-stochastic routing matrix $P^j \in \mathbb{R}^{I \times I}$ so that the departing job is routed to another buffer $i' \in \mathcal{I}$ with probability $P^j_{i'i}$. With the remaining probability $1 - \sum_{i' \in \mathcal{I}} P^j_{i'i}$, the job leaves the system.

At each point in time, the evolution of the system is controlled by a scheduling decision that specifics which activities are to be undertaken. This decision will result in an allocation (action) represented by a vector $u \in \mathbb{R}^J$. Here, $u_j = 1$ if activity $j$ is to be employed, and $u_j = 0$ otherwise. Thus,

$$u_j \in \{0, 1\}, \quad j \in \mathcal{J}. \tag{2.1}$$

This setting, where fractional allocation of activities is disallowed, is referred to as *non-processor sharing*.

The set of admissible allocations is further restricted by a set $\mathcal{K} = \{1, \dots, K\}$ of common resources (for example, processors or servers). It is assumed that each activity may require some subset of the resources to perform its service. The resource consumption matrix $A \in \mathbb{R}^{K \times J}$ is defined by setting $A_{kj} = 1$ if activity $j$ requires the resource $k$, and $A_{kj} = 0$ otherwise. We preclude any resource from simultaneously being used in multiple activities. Hence we require allocations $u$ to satisfy

$$Au \le \mathbf{1}. \tag{2.2}$$

Here, $\mathbf{1}$ is a vector with every component set to 1.

Finally, an activity cannot be employed if there is not an available job in each of its constituent buffers. This results in another constraint on allowable allocations $u$, when the buffer lengths are given by the vector $q \in \mathbb{Z}^I_+$,

$$u_j = 0, \quad \text{if } q_i = 0, \ \forall\ j \in \mathcal{J},\ i \in \mathcal{B}_j,. \tag{2.3}$$

We make the following definition.

**Definition 1. (Admissible Allocations)** Denote by $\mathcal{A}(q) \subset \mathbb{R}^J$ the set of all permitted allocations when the buffer lengths are given by $q \in \mathbb{Z}^I_+$, that is, those vectors $u \in \mathbb{R}^J$ satisfying (2.1)–(2.3). Denote by $\mathcal{A} \subset \mathbb{R}^J$ the set of all permitted allocations ignoring restrictions imposed by buffer lengths, that is, those vectors satisfying (2.1) and (2.2).

Let us denote the allocation applied at time $\tau$ by $u(\tau)$. If the vector of buffer lengths at time $\tau$ is $q(\tau)$, we require that $u(\tau) \in \mathcal{A}(q(\tau))$. Further, the process $\{u(\tau)\}$ must be non-anticipatory, and is allowed to be preemptive. We call such a process a policy. Define the average cost given a policy $u$ and an initial state $q(0) = q$ by

$$\limsup_{T \to \infty} \mathsf{E}_u \left[ \frac{1}{T} \int_0^T c \cdot q(\tau)\, d\tau \;\middle|\; q(0) = q \right]. \tag{2.4}$$

Here, the $c \in \mathbb{R}^I_+$ is a vector of holding costs. We assume that $c > 0$. Our objective is to find a policy which achieves a minimal average cost.

## 2.2. Stability

We wish to avoid situations where the average cost (2.4) is infinite under every policy. In order to address this, define the matrix $R \in \mathbb{R}^{I \times J}$ by

$$R_{ij} = \mu_j \left( B_{ij} - \sum_{i' \in \mathcal{I}} B_{i'j} P^j_{i'i} \right), \quad \forall\, i, i' \in \mathcal{I},\ j \in \mathcal{J}.$$

Given an allocation $u \in \mathcal{A}$, the vector $Ru$ is the vector of mean rates at which buffers are drained.

The *static planning linear program* [19] is given by

$$\begin{array}{ll} \text{minimize} & \rho \\ \text{subject to} & Rx = \lambda, \\ & Ax \leq \rho \mathbf{1}, \\ & x \geq 0. \end{array} \tag{2.5}$$

The optimal value $\rho^*$ of this LP is called the *load* of the system. Intuitively, for each activity $j \in \mathcal{J}$, we can interpret the decision variable $x_j$ as the fraction of time in which activity $j$ is employed under some policy over a long time horizon. The first constraint forces the average rate of buffer growth to be zero over the time horizon. The second constraint forces the decision variable $\rho$ to be an upper bound on the mean utilization of any resource. This interpretation can be made rigorous on the fluid scale (see [20], for example). It follows that, if $\rho^* > 1$, there exists no policy under which the system is stable.

We make the following assumption.

**Assumption 1.** The optimal value $\rho^*$ of the static planning LP (2.5) is strictly less than 1.

## 2.3. Markov Decision Process Formulation

It is convenient from an analytical perspective to reduce the optimal control problem from the continuous-time setting described in Section 2.1 to that of a discrete-time, countable state Markov decision process (MDP). Given the memory-less characteristic of the Poisson arrivals and exponential service times, this can be accomplished via the process of uniformization [9]. Specifically, assume, without loss of generality, that time is normalized so that

$$\sum_{i \in \mathcal{I}} \lambda_i + \sum_{j \in \mathcal{J}} \mu_j = 1.$$

We consider the state of the system at discrete times $t = 0, 1, \ldots$ corresponding to 'events', that is, arrivals of either new jobs or 'virtual service tokens' to the system.

Denote the buffer lengths in the system at time $t$ by $q(t) \in \mathbb{Z}^I_+$, and the scheduling decision at time $t$ by $u(t) \in \mathcal{A}(q(t))$. We restrict the policies under consideration to *stationary* policies. Under a stationary policy (or 'feedback law'), the scheduling decision is determined exclusively as a function of the buffer lengths. That is, there is a function $u : \mathbb{Z}^I_+ \to \mathbb{R}^J$ so that $u(t) = u(q(t))$, for all $t$. Given a stationary policy $u$, $q(t)$ evolves as a discrete-time, countable state Markov chain. Given an initial state $q(0) = q$, we can define the long-term average cost

$$\eta(q, u) = \limsup_{T \to \infty} \mathsf{E}_u \left[ \frac{1}{T} \sum_{t=0}^{T-1} c \cdot q(t) \;\middle|\; q(0) = q \right].$$

The classical dynamic programming approach to optimal control is to find a function $V^* : \mathbb{Z}_+^I \to \mathbb{R}$ and a scalar $\eta^* \in \mathbb{R}$ that solve the Bellman equation

$$V^*(q) + \eta^* = \min_{u \in \mathcal{A}(q)} c \cdot q + \mathsf{E}_u \left[ V^*(q(t+1)) \mid q(t) = q \right], \quad \forall\, q \in \mathbb{Z}_+^I. \tag{2.6}$$

If a solution exists and satisfies certain technical conditions (see [21], or [22, 23, 18] for a discussion in the context of queueing networks), then the following hold:

(a) The optimal average cost is equal to $\eta^*$ and is independent of the initial state, that is

$$\eta^* = \min_u \eta(q, u), \quad \forall\, q \in \mathbb{Z}_+^I.$$

(b) If $u^*$ is a stationary policy such that

$$u^*(q) \in \operatorname*{argmin}_{u \in \mathcal{A}(q)} \mathsf{E}_u \left[ V^*(q(t+1)) \mid q(t) = q \right], \quad \forall\, q \in \mathbb{Z}_+^I, \tag{2.7}$$

then $u^*$ achieves the optimal average cost.

(c) If $u^*$ is a stationary policy satisfying the conditions of part (b), then $u^*$ also achieves the optimal average cost in the original continuous-time setting (2.4).

We call the function $V^*(\cdot)$ the *differential cost function*.

# 3. Approximate Dynamic Programming

The analysis in Section 2.3 suggests a standard plan of attack for determining an optimal stationary policy:

1. Solve the Bellman equation (2.6) in order to obtain the differential cost function $V^*(\cdot)$.
2. Define the policy by acting greedily with respect to the differential cost according to (2.7).

Both of these steps may be intractable, however. Solving the Bellman equation is notoriously difficult when the state and action spaces are large. In the case of closed queueing networks, where jobs do not enter or leave the system and the state space is finite, the problem is known to be provably intractable [24]. The case we consider, with a countable state space, would seem to be more difficult. Further, even if the differential cost function was known, the policy selection equation (2.7) is a combinatorial optimization problem and it is not clear that this can be solved efficiently. In this section, we address these two issues.

## 3.1. Affine Expectations and the Allocation Polytope

Consider a function $V : \mathbb{Z}_+^I \to \mathbb{R}$. We wish to define a policy by acting greedily with respect to $V(\cdot)$ as an estimate of the relative cost of possible future states. That is, we would like to solve the combinatorial optimization problem

$$\min_{u \in \mathcal{A}(q)} \mathsf{E}_u \left[ V(q(t+1)) \mid q(t) = q \right]. \tag{3.1}$$

First, notice that the objective is an affine function of the allocation $u$. To see this, we first define some additional notation. For a vector $a \in \mathbb{Z}_+^I$, denote by $P^j(a)$ the probability that a service completion by activity $j \in \mathcal{J}$ results in exactly $a_i$ jobs being routed to each buffer $i \in \mathcal{I}$. For each activity $j \in \mathcal{J}$, denote by $B^j \in \mathbb{Z}_+^I$ the $j$th column of the constituency matrix $B$. Given the dynamics described in Section 2.3, we have the explicit expression

$$\mathsf{E}_u\left[V(q(t+1)) \mid q(t) = q\right] = \sum_{i \in \mathcal{I}} \lambda_i V(q + e^i) + \sum_{j \in \mathcal{J}} \mu_j \sum_{a \in \mathbb{Z}_+^I} P^j(a) V(q - B^j + a) u_j. \tag{3.2}$$

Here, $e^i \in \mathbb{Z}_+^I$ is a unit vector that is zero except in component $i$.

We make the following definition.

**Definition 2. (Affine Expectations)** A queueing network has the affine expectations property if there are functionals $\Delta$ and $\Xi$, so that for every $V : \mathbb{Z}_+^I \to \mathbb{R}$, $q \in \mathbb{Z}_+^I$, and $u \in \mathcal{A}(q)$,

$$\mathsf{E}_u\left[V(q(t+1)) \mid q(t) = q\right] = (\Delta V)(q) + (\Xi V)(q) \cdot u.$$

Examining (3.2), it is clear that the affine expectations property holds for the class of queueing networks under consideration. For other queueing networks, such as the example considered in Section 6, this may not be true.

Given the linear objective in the optimization problem (3.1), we may consider relaxing the discrete set of admissible allocations $\mathcal{A}(q)$ to a bounded convex polytope whose vertices lie in $\mathcal{A}(q)$. Consider the following definition:

**Definition 3. (Allocation Polytope)** Denote by $\bar{\mathcal{A}} \subset \mathbb{R}^J$ the bounded polytope consisting of vectors $u$ with

$$\begin{aligned} 0 \le u \le \mathbf{1}, \\ Au \le \mathbf{1}. \end{aligned}$$

For $q \in \mathbb{Z}_+^I$, denote by $\bar{\mathcal{A}}(q) \subset \bar{\mathcal{A}}$ the bounded polytope consisting of vectors further satisfying

$$u_j = 0, \quad \text{if } q_i = 0, \ \forall\ j \in \mathcal{J},\ i \in \mathcal{B}_j,.$$

Observe that the polytopes $\bar{\mathcal{A}}(q)$ and $\bar{\mathcal{A}}$ are obtained by relaxing the integrality constraint (2.1) imposed on $\mathcal{A}(q)$ and $\mathcal{A}$, respectively, in Definition 1. These polytopes can be described with at most $J + K$ constraints. However, in order to proceed, we must make the following assumption.

**Assumption 2.** The vertices of the polytope $\bar{\mathcal{A}}$ are contained in the set $\mathcal{A}$.

It is not difficult to see that, given Assumption 2, the vertices of the polytope $\bar{\mathcal{A}}(q)$ are contained in the set $\mathcal{A}(q)$, for every $q \in \mathbb{Z}_+^I$. Then, we can determine a policy that acts greedily with respect to some function $V(\cdot)$ by computing an optimal basic solution for the linear program

$$\min_{u \in \bar{\mathcal{A}}(q)} (\Xi V)(q) \cdot u. \tag{3.3}$$

Several remarks are in order. First, note that both the affine expectations property and Assumption 2 are assumptions on the *representation* of the control space of the problem. Indeed, *any* Markov decision process with a finite control space can be described in a way such that policy

decisions can be computed by solving an LP of the form (3.3). We can consider actions (allocations) as elements of a vector space by identifying them with vectors of transition probabilities, so that the one-step expectation is a linear function of the choice of action. We can then define an admissible polytope by taking the convex hull of the set of available actions at each state. In general, however, this trivial representation results in a polytope that may be of high dimension and may involve many linear constraints. This may result in a linear program with an intractable number of variables and constraints, even in a moderately-sized problem. Implicit in Assumption 2 is the idea that, for many queueing networks, a natural and compact description of the control space results in an equally compact description of its convex hull.

Second, observe that Assumption 2 holds for many queueing networks of interest. It holds for the class of reversed Leontief networks [20], and the further special case of unitary networks [25]. Reversed Leontief networks are defined by the property that each activity participates in a single resource constraint. That is, the matrix $A$ has a single non-zero entry in each column. More generally, the assumption holds when the resource constraints possess a 'network flow' structure [26, Chapter 7]. This includes the example considered in Section 6.

Finally, observe that if a queueing network allows *processor sharing*, then the integrality constraint (2.1) for admissible allocations does not apply. Hence, Assumption 2 is immediately satisfied. Assumption 2 has been previously noted by Dai and Lin [20] in this setting. In particular, recall that, from Section 2.2, analysis of the static planning LP (2.5) yields necessary conditions for the existence of a stabilizing policy. For queueing networks with processor sharing, sufficient conditions can similarly be developed. In the non-processor sharing case, this is also true, if Assumption 2 is satisfied.

## 3.2. Differential Cost Approximation

The intractability of solving the Bellman equation (2.6) suggests the following alternative approach. Instead of trying to exactly determine the differential cost function $V^*(\cdot)$, we may seek to find a function which is easier to compute and which serves as a good approximation to $V^*(\cdot)$. In this paper, we focus on *linear parameterizations*, that is, we attempt to find a good approximation for $V^*(\cdot)$ in the span of a finite set of basis functions,

$$V^*(q) \approx \tilde{V}(q,r) = \sum_{\ell=1}^{L} r_\ell \phi_\ell(q). \tag{3.4}$$

Here, for each $1 \le \ell \le L$, $\phi_\ell : \mathbb{Z}_+^I \to \mathbb{R}$ is a basis function which is chosen *a priori*, and $r_\ell \in \mathbb{R}$ is a scalar weight is to be computed algorithmically.

*Temporal difference learning* (TD-learning) [10] is a standard method to select a vector of weights $r$ in a manner that yields a good approximation to the differential cost function. To motivate this algorithm, take as given a weight vector $r$. Let $u^r$ be the associated stationary policy, that is

$$u^r(q) \in \operatorname*{argmin}_{u \in \mathcal{A}(q)} \mathsf{E}_u \left[ \tilde{V}(q(t+1), r) \;\middle|\; q(t) = q \right].$$

Given an estimate of average cost $\eta$, consider the optimization problem

$$\min_{r'} \; \frac{1}{2} \left\| c \cdot q - \eta + \mathsf{E}_{u^r} \left[ \tilde{V}(q(t+1), r) - \tilde{V}(q(t), r') \;\middle|\; q(t) = q \right] \right\|_{2,r}^2. \tag{3.5}$$

Here, $\|\cdot\|_r$ is the weighted $\ell_2$-norm defined by

$$\|f\|_{2,r}^2 = \sum_{q \in \mathbb{Z}_+^I} \pi^r(q) f(q)^2,$$

where $\pi^r(\cdot)$ is the stationary distribution under the policy $u^r$ and $f : Z_+^I \to \mathbb{R}$ is a function on the state space. Define $T$ to be the operator that maps $r$ to a parameter vector $r'$ that is optimal for (3.5). $T$ is sometimes called a projected Bellman operator. Intuitively, a fixed point of this operator has small Bellman error on the most relevant portions of the state space, where relevancy is determined by the stationary distribution.

TD-learning can be viewed as a stochastic approximation method for incrementally solving the fixed point equation $r = Tr$. The algorithm maintains a set of weight vectors $\{r(t)\}$ and average cost estimates $\{\eta(t)\}$ which are updated over time. In place of an optimal policy, the current weight vector is used to select an allocation. In particular, at time $t$, the algorithm proceeds as follows:

1. The current state $q(t)$ is observed.

2. An allocation decision $u(t)$ is made as follows. With probability $\epsilon(t) \in [0,1]$, an allocation in $\mathcal{A}(q(u(t)))$ is chosen uniformly at random. This is referred to as *exploration*. Otherwise, the allocation is selected according to

$$u(t) \in \operatorname*{argmin}_{u \in \mathcal{A}(q(t))} \mathsf{E}_u \left[ \tilde{V}(q(t+1), r(t)) \;\middle|\; q(t) \right]. \tag{3.6}$$

   The exploration probability $\epsilon(t)$ is typically chosen to be small and non-increasing.

3. The *temporal difference* is a scalar $d(t) \in \mathbb{R}$ is computed according to

$$\begin{aligned} d(t) = c \cdot q(t) - \eta(t) \\ + \mathsf{E}_{u(t)} \left[ \tilde{V}(q(t+1), r(t)) - \tilde{V}(q(t), r(t)) \;\middle|\; q(t) \right]. \end{aligned} \tag{3.7}$$

   This quantity represents an estimate of the Bellman error at the current state $q(t)$ given weights $r(t)$ and average cost estimate $\eta(t)$.

4. The weights and average cost estimate are updated according to

$$r(t+1) = r(t) + \gamma_1(t) d(t) \nabla_r \tilde{V}(q(t), r(t)). \tag{3.8}$$

$$\eta(t+1) = (1 - \gamma_2(t))\eta(t) + \gamma_2(t) c \cdot q(t). \tag{3.9}$$

   Here, $\gamma_1(t), \gamma_2(t) > 0$ are step-sizes that is chosen to be small and non-increasing. The vector $d(t)\nabla_r \tilde{V}(q(t), r(t))$ represents a stochastic estimate of a descent direction in the optimization problem (3.5).

TD-learning is one of a number of stochastic approximation methods for approximate dynamic programming. We have chosen it because it is simple to describe and is commonly used. Different algorithms have been explored in the literature, and we refer the curious reader to recent surveys on this topic [4, 5, 6, 7, 8, 9].

# 4. Basis Function Architecture

The most difficult and crucial aspect of the ADP methodology is the selection of basis functions. One would like to select a broad set of basis functions whose span is likely to be close to the true differential cost function on relevant portions of the state space. For example, in a queueing network under light traffic, one may consider a set of basis functions including indicator functions for all 'small' states and functions describing asymptotic behavior for 'large' states. This way, the differential cost function can be approximated to a fine level of detail on the 'small' states, where the process is most often expected to live. However, this architecture quickly proves to be cumbersome since the number of basis functions will grow exponentially in the dimension of the state space. Further, it may fail to yield good policies under higher levels of load.

A more useful approach is to attempt to identify the salient features in the problem which are most important to good decision-making, and hence are likely to yield policies with good performance. This is most often accomplished through structural intuition and specific analysis of the problem at hand.

In the queueing network context, significant structural intuition can be obtained via analysis of the problem in the asymptotic fluid and diffusion limit regimes. The key insight of these methods is the identification of critical bottlenecks or 'workloads', which have the most impact on performance. By selecting basis functions which capture this insight, we can hope to design a tractable set of basis functions that yield good performance. In this section, we demonstrate this with one particular asymptotic method, and the performance of the resulting policies can be seen in Section 6. This also suggests an alternative perspective on ADP for queueing networks: we can view asymptotic methods as providing the insights necessary to construct a manageable class of good policies, and ADP as an algorithmic methodology to optimize within this class on a non-asymptotic scale.

## 4.1. The Fluid Control Problem

One class of differential cost approximations that have been proposed for queueing networks involve using the value function for the fluid optimal control problem [23, 18]. We digress briefly to give a formal description of this problem (see [22, 27], for example, for a more complete account).

A fluid path with initial condition $x \in \mathbb{R}_+^I$ is a continuous and almost everywhere differentiable map $\tilde{q}(\cdot, x) : [0, \infty) \to \mathbb{R}^I$. The dynamics of this path are constrained by

$$\tilde{q}(0, x) = x, \tag{4.1}$$

$$\tilde{q}(t, x) \geq 0, \quad \forall\ t \geq 0, \tag{4.2}$$

$$\frac{q(t, x) - q(s, x)}{t - s} \in \mathcal{V}, \quad \forall\ t > s \geq 0. \tag{4.3}$$

Here, $\mathcal{V}$ is a polytope describing the set of allowed 'velocities' and is given by

$$\mathcal{V} = \{Ru + \lambda \in \mathbb{R}^I \mid u \in \mathbb{R}^J,\ u \geq 0,\ Au \leq 1\}.$$

**Definition 4. (Fluid Optimal Control Problem)** The fluid optimal control problem is the optimization problem

$$V^f(x) = \min \int_0^\infty c \cdot \tilde{q}(t, x)\ dt, \tag{4.4}$$

where the minimization is constrained to paths $\tilde{q}(\cdot, x)$ satisfying (4.1)–(4.3). The function $V^f(\cdot)$ is referred to as the fluid value function.

For certain classes of queueing networks [22], the fluid value function is related to the differential cost function to the stochastic queueing network by

$$\limsup_{\|x\|\to\infty} \left| \frac{V^f(x) - V^*(x)}{\|x\|^2} \right| = 0.$$

This suggests that for large states (heavily congested networks), the fluid value function may be useful as a differential cost approximation. The ADP method proposed by Veatch [18] considers such an approach by defining a set of basis functions which are piecewise quadratic over regions of the state space corresponding to the structure of the optimal fluid control policy. However, outside of small examples or special cases, it is difficult to compute the fluid value function or the associated optimal control policies (see [28] for recent progress in this area). Further, the number of basis functions with such an approximation will grow exponentially in the number of buffers. Hence, approximations based on the structure of the optimal fluid control policy will prove intractable for large queueing networks.

## 4.2. The Relaxed Fluid Control Problem and Workload

Observe that, by Assumption 1, the polytope $\mathcal{V}$ contains the origin, and hence can be described by linear constraints as

$$\mathcal{V} = \{v \in \mathbb{R}^I \mid \zeta^i \cdot v \geq -(1-\rho_i),\ 1 \leq i \leq N\}.$$

Here, for each $i$, the vector $\zeta^i$ is called a *workload vector*, and scalar $\rho_i < 1$ is the associated *load*. We define the $i$th component of the *workload process* $w(t,x)$ by $w_i(t,x) = \zeta^i \cdot q(t,x)$. Note that the velocity constraint (4.3) is equivalent to

$$\frac{w_i(t,x) - w_i(s,x)}{t-s} \geq -(1-\rho_i), \quad \forall\ t > s \geq 0,\ 1 \leq i \leq N.$$

The workload relaxation (see [23] for a full account) is based on the idea of replacing the complex fluid control problem (4.4) with a simpler problem. This simpler problem seeks to retain the important features of the fluid control problem, but allows for both easier identification of good policies and visualization of network behavior. The simplification is based on the idea of relaxing the velocity set $\mathcal{V}$ to obtain a lower-dimensional and hopefully simpler control problem. In particular, $n$ distinguished workload vectors are selected, with $n \ll \min\{I, N\}$. Typically these are the workload vectors with the highest loads and thus represent bottlenecks in the system. As we will see shortly, the workload relaxation reduces the fluid optimal control problem from an optimization over paths in an $I$-dimensional space to an optimization over paths in an $n$-dimensional space.

To begin, we will assume the workload vectors are numbered so that these critical workload vectors are $\zeta^1, \ldots, \zeta^n$, and that these vectors are linearly independent. We define $\Lambda \in \mathbb{R}^{n \times I}$ to be a matrix with rows $(\zeta^1, \ldots, \zeta^n)^\top$. The relaxed set of allowable velocities is defined by

$$\hat{\mathcal{V}} = \{v \in \mathbb{R}^I \mid \zeta^i \cdot v \geq -(1-\rho_i),\ 1 \leq i \leq n\}.$$

The relaxed queue length process process $\hat{q}(\cdot, x) :\ [0,\infty) \to \mathbb{R}^I_+$ has its velocity constrained to satisfy

$$\frac{\hat{q}(t,x) - \hat{q}(s,x)}{t-s} \in \hat{\mathcal{V}}, \quad \forall\ t > s \geq 0. \tag{4.5}$$

**Definition 5. (Relaxed Optimal Control Problem)** The relaxed optimal control problem is defined by

$$\min \int_0^\infty c \cdot \hat{q}(t,x)\ dt. \tag{4.6}$$

Here, the path $\hat{q}(\cdot, x)$ is required to satisfy (4.1)–(4.2) and the relaxed velocity constraint (4.5).

The key insight of the relaxed formulation is that the dimension of the problem (4.6) can be reduced via a change of coordinates. In particular, define the space

$$\mathcal{W} = \{w \mid w = \Lambda x,\ x \in \mathbb{R}_+^I\} \subset \mathbb{R}^n,$$

We define a relaxed workload process $\hat{w}(\cdot, x) :\ [0,\infty) \to \mathbb{R}^n$ to be a path which satisfies the constraints

$$\hat{w}(0,x) = \Lambda x, \tag{4.7}$$

$$\hat{w}(t,x) \in \mathcal{W}, \quad \forall\ t \geq 0, \tag{4.8}$$

$$\frac{\hat{w}_i(t,x) - \hat{w}_i(s,x)}{t-s} \geq -(1-\rho_i), \quad \forall\ t > s \geq 0,\ 1 \leq i \leq n. \tag{4.9}$$

We define the *effective cost* $\hat{c}(w)$ for a workload $w \in \mathcal{W}$ as the solution of the linear program

$$\begin{array}{rll} \hat{c}(w) = & \text{minimize} & c \cdot x \\ & \text{subject to} & \Lambda x = w, \\ & & x \geq 0. \end{array} \tag{4.10}$$

**Definition 6. (Workload Relaxation)** The workload relaxation is defined to be the optimization problem

$$\min \int_0^\infty \hat{c}(\hat{w}(t,x))\ dt, \tag{4.11}$$

over paths $\hat{w}(\cdot, x)$ satisfying (4.7)–(4.9).

We can provide an equivalence between the relaxed optimal control problem (4.6) and the workload relaxation (4.11). Assume that $\hat{q}(\cdot, x)$ is an optimal solution to the relaxed optimal control problem (4.6), and define

$$\hat{w}(t,x) \triangleq \Lambda \hat{q}(t,x), \quad \forall\ t \geq 0.$$

Then, $\hat{w}(\cdot, x)$ is optimal for the workload relaxation (4.11).

Conversely, define a map $\mathcal{X} :\ \mathcal{W} \to \mathbb{R}_+^I$ by

$$\begin{array}{rll} \mathcal{X}(w) \in & \text{argmin} & c \cdot x \\ & \text{subject to} & \Lambda x = w, \\ & & x \geq 0. \end{array} \tag{4.12}$$

Assume that $\hat{w}(\cdot, x)$ is optimal for the workload relaxation (4.11), and define

$$\hat{q}(t,x) \triangleq \mathcal{X}\left(\hat{w}(t,x)\right), \quad \forall\ t \geq 0.$$

Then, $\hat{q}(\cdot, x)$ is an optimal solution to the relaxed optimal control problem (4.6).

## 4.3. Proposed Basis Functions

The discussion in the previous section suggests the following procedure for solving the relaxed optimal control problem (4.6):

(i) Determine the optimal path $\hat{w}(\cdot, x)$ for draining workload from the system, according to the workload relaxation (4.11).

(ii) While following the optimal workload trajectory, rearrange the queue lengths in the system so as to minimize cost at a given level of workload. This is done by implementing an optimal *lifting map* $\mathcal{X}(\cdot)$ of the form (4.12).

We will identify features of control policies that allow the execution of the two-step procedure. By proxy, identifying basis functions that capture such features will enable us to characterize a set of policies for the *original*, stochastic queueing network, that contains policies with good performance. ADP can then be used to optimize within this policy class.

First, consider the problem of implementing an optimal lifting map. The program (4.12) greedily minimizes the instantaneous cost subject to velocity constraints. Literal implementation in the stochastic queueing network of this would imply the scheduling policy

$$\begin{aligned} u(q) &\in \operatorname*{argmin}_{u \in \mathcal{A}(q)} \mathsf{E}_u \left[ \sum_{i \in \mathcal{I}} c_i q_i(t+1) \;\middle|\; q(t) = q \right] \\ &= \operatorname*{argmin}_{u \in \mathcal{A}(q)} \mathsf{E}_u \left[ \sum_{i \in \mathcal{I}} c_i \big(q_i(t+1) - q_i(t)\big) \;\middle|\; q(t) = q \right], \quad \forall\, q \in \mathbb{Z}_+^I. \end{aligned} \tag{4.13}$$

This greedy rule is not sensible. It depends only on the identity of non-empty queues and otherwise completely ignores queue lengths. This can result in inefficient allocation of server effort when some queues are near zero. The greedy rule will always give priority to the same queues given a particular configuration of non-emptiness, even if this results in allocating server effort towards queues which are close to empty, only to result in future idleness. In fact, the greedy rule can often fail to stabilize the network. Therefore, we want to implement an optimization problem that is "close" to (4.12) but guarantees some prioritization of longer queues so as to avoid idleness. One way to do this is via the greedy minimization of the function

$$\tilde{V}(q) = \sum_{i \in \mathcal{I}} c_i q_i^{1+\alpha}, \tag{4.14}$$

for small $\alpha > 0$. As $\alpha \downarrow 0$, the solutions of the this family of problems will converge to those of (4.12). For each fixed $\alpha > 0$, however, the function (4.14) will offer some priority to the service of longer queues. Moreover, policies that greedily minimize (4.13) are closely related to so-called MaxWeight-0+ policies, which been shown to have good asymptotic properties in some critically loaded settings [29, 11, 12].

Next, consider the structure of the workload relaxation (4.11). The fact that the velocity constraints (4.9) are separable in terms of individual workload components, suggests a greedy policy where each component is drained as fast as permitted by (4.9). This is optimal under certain technical conditions, when, for example, the workload dimension $n$ is 1 or 2. In general, however, the greedy path may fail to be optimal for two reasons. First, the cost function $\hat{c}(\cdot)$ can be non-monotonic. In such cases, the optimal workload trajectory can counterintuitively involve allowing some components of the workload to increase. Second, the greedy path as described above

may not be well-defined on the boundary of the workload space $\mathcal{W}$. On the boundary, which corresponds to configurations where some servers are forced to idle, it may not be possible to drain each workload component simultaneously at the maximum rate.

We consider differential cost approximations of the form

$$\tilde{V}(q) = \sum_{i \in \mathcal{I}} F_i(q_i) + \sum_{i=1}^{n} G_i(\zeta^i \cdot q). \tag{4.15}$$

Here, the functions $\{F_i(\cdot), G_i(\cdot)\}$ are one-dimensional functions. The functions $\{G_i(\cdot)\}$ capture the desire to greedily minimize each component of the workload process. The functions $\{F_i(\cdot)\}$ seek to implement an optimal lifting map, by spanning policies of the form (4.14). These functions can linearly parameterized by picking a set of one-dimensional basis functions (for an example, see Section 6). Assuming a fixed number of linear parameters per basis function, the complexity of the resulting approximation scales according to $I + n$. In many problem instances, the number of relevant workload vectors $n$ is constant, or (as in the example in Section 6) sub-linear in the number of buffers $I$. In particular, the complexity does not scale with the size of the control space and hence this method can be applied to very large queueing networks.

Note that, in addition, the basis function architecture defined by (4.15) is sufficiently flexible to capture the general class of 'back pressure' or MaxWeight scheduling policies [30, 31, 20, 29]. This guarantees, for example, that stabilizing policies exist within the span of these basis functions.

ADP can be used to fit the precise values of the linear parameters in a particular problem instance. This allows algorithmic determination of, for example, which workload components are most important, or how to trade off the conflicting goals of greedily minimizing workload and shifting the work currently in the system to a lower cost configuration.

## 5. A Data-Driven Methodology

Most approaches to optimal control require knowledge of an underlying probabilistic model of the system dynamics. This typically requires that certain assumptions be made, and entails a separate estimation step to estimate the parameters of the model.

In a queueing network, it is often the case that the service and routing processes are well understood. For example, they may be deterministic, or they may be under the control of the system designer and easy to model probabilistically. The exogenous arrival processes, on the other hand, are a different matter. Their structure may be poorly understood, and there may be correlations which are difficult to estimate. However, it is usually the case that significant historical data is available in terms of data traces. In this section, we will describe how the ADP algorithm described in Section 3 can be extended in a way that can operates directly with such data traces, without making explicit probabilistic assumptions regarding the arrivals or requiring a separate estimation step.

In Section 3, the fundamental object under consideration was the differential cost function $V^*(\cdot)$, which captures the relative desirability of states assuming an optimal control policy. An alternative and equivalent representation is to consider the *Q-function*. Intuitively, the $Q$-function captures the relative cost of the choice of a particular allocation for the next time-step at a given state, assuming that an optimal policy is used for all future time steps. Specifically, given the differential cost $V^*(\cdot)$ and the optimal average cost $\eta^*$, we define the $Q$-function $Q^*(\cdot, \cdot)$ by

$$Q^*(q, u) = c \cdot q - \eta^* + \mathsf{E}_u \left[ V^*(q(t+1)) \mid q(t) = q \right], \quad \forall\, q \in \mathbb{Z}_+^I,\ u \in \mathcal{A}(q).$$

Minimizing this equation over $u \in \mathcal{A}(q)$ and comparing to the Bellman equation (2.6), we have

$$V^*(q) = \min_{u \in \mathcal{A}(q)} Q^*(q,u), \quad \forall\ q \in \mathbb{Z}_+^I.$$

In fact, the Bellman equation (2.6) can be written in an equivalent form that only involves $Q$-functions,

$$Q(q,u) + \eta = c \cdot q + \mathsf{E}_u \left[ \min_{u' \in \mathcal{A}(q(t+1))} Q(q(t+1), u') \;\middle|\; q(t) = q \right], \quad \forall\ q \in \mathbb{Z}_+^I,\ u \in \mathcal{A}(q). \tag{5.1}$$

Further, optimal stationary policies can be defined by

$$u^*(q) \in \operatorname*{argmin}_{u \in \mathcal{A}(q)} Q^*(q,u), \quad \forall\ q \in \mathbb{Z}_+^I.$$

$Q$-learning [13] is an approximate dynamic programming technique that seeks to find a good approximation to the $Q$-function $Q^*(\cdot,\cdot)$. Analogous to the approximations described in Section 3.2, we consider approximations that are linearly parameterized over the span of a finite set of basis functions. In particular, consider, for each $1 \le \ell \le L$, a basis function $\psi_\ell(\cdot,\cdot)$ which maps pairs of states and admissible allocations to real numbers. Given a vector of weights $r \in \mathbb{R}^L$, we allow approximations of the form

$$Q^*(q,u) \approx \tilde{Q}(q,u,r) = \sum_{\ell=1}^{L} r_\ell \psi_\ell(q,u).$$

Given such an approximation, the allocation decision at time $t$ can then be made according to the rule

$$u(t) \in \operatorname*{argmin}_{u \in \mathcal{A}(q(t))} \tilde{Q}(q(t),u,r). \tag{5.2}$$

In particular, this rule does not involve any knowledge of the underlying system dynamics.

## 5.1. Basis Function Selection

$Q$-learning is typically considered to be intractable for MDPs with large control spaces. This is for two reasons. First, the selection of basis functions for the $Q$-function is much more difficult than the selection of basis functions for the differential cost functions, because of the fact that the $Q$-function $Q^*(\cdot,\cdot)$ we seek to approximate inhabits a much bigger space. Second, given a $Q$-function approximation, the decision rule (5.2) may be intractable to solve.

The special structure of the queueing network under consideration allows us to overcome these difficulties. In particular, we can impose a particular functional form on the basis functions. To see this, note that

$$\begin{aligned} Q^*(q,u) &= c \cdot q - \eta^* + \mathsf{E}_u \left[ V^*(q(t+1)) \mid q(t) = q \right] \\ &= c \cdot q - \eta^* + (\Delta V^*)(q) + (\Xi V^*)(q) \cdot u, \end{aligned}$$

where $\Delta$ and $\Xi$ are the functionals defined in Section 3.1. Thus, we need only consider basis functions that are affine in the choice of allocation. That is,

$$\psi_\ell(q,u) = \psi_\ell^1(q) + \psi_\ell^2(q) \cdot u,$$

where $\psi_\ell^1 : \mathbb{Z}_+^I \to \mathbb{R}$ and $\psi_\ell^2 : \mathbb{Z}_+^I \to \mathbb{R}^J$ are, respectively, real- and vector-valued functions on the state space. For basis functions with such affine structure, the decision rule (5.2) can be efficiently implemented using the LP relaxation proposed in Section 3.1.

The class of basis functions can be further restricted by requiring that $\psi_\ell^2(\cdot)$ lie in the range of the functional $\Xi$. Note that, up to scaling, the functional $\Xi$ has no dependence on arrival rates. A natural way to do this is as follows:

1. Select a set of basis functions $\{\phi_\ell(\cdot) \mid 1 \le \ell \le L_0\}$ for the differential cost function. This can be done, for example, using the techniques described in Section 4. Define corresponding basis functions for the $Q$-function by

$$\psi_\ell^1 = 0, \quad \text{and} \quad \psi_\ell^2 = \Xi\phi_\ell, \quad \forall\ 1 \le \ell \le L_0.$$

2. For the remaining basis functions $\{\psi_\ell(\cdot,\cdot) \mid L_0 < \ell \le L\}$, allow $\psi_\ell^1(\cdot)$ to be arbitrary and set $\psi_\ell^2 = 0$.

## 5.2. Parameter Optimization

Similar to the TD-learning procedure described earlier, $Q$-learning attempts to find a set of parameters $r$ that are a fixed point for a projected version the Bellman equation (5.1) by a method of stochastic approximation. The method we describe here differs from the method in Section 3.2 in two ways. First, it is a *batch* gradient variation, as opposed to the *incremental* gradient variation of TD-learning described earlier. A batch method updates the parameter vector only after processing a series of time-steps, rather than after every time-step. Batch methods are more natural to describe when considering a finite trace of historical data, rather than an infinite simulated sample path, however the difference between these methods is mainly an implementation concern. Second, since the arrival rates are not known, the Markov chain can no longer be uniformized. Hence, the evolution of system will be considered in continuous-time in a discrete event framework.

Assume there is some long time horizon $[0, T]$ for which a data trace of arrivals is available. We will track the system at discrete event times $0 = t_0 < t_1 < \cdots < t_K = T$. These event times correspond to arrivals or service completions. We take as given initial values for the parameter vector $r$ and an average cost estimate $\eta$. We can compute a sample trajectory of buffer lengths $\{q(t_0), \ldots, q(t_K)\}$ and of allocation decisions $\{u(t_0), \ldots, u(t_K)\}$ inductively as follows. As time $t_0 = 0$, set $q(0) = 0$. Then, at each event index $k \ge 0$:

1. With probability $\epsilon \in [0, 1]$, the allocation decision $u(t_k)$ is chosen uniformly at random from the set $\mathcal{A}(q(t_k))$. Otherwise, $u(t_k)$ is chosen according to

$$u(t_k) \in \operatorname*{argmin}_{u \in \mathcal{A}(q(t_k))} \tilde{Q}(q(t_k), u, r). \tag{5.3}$$

2. Random exponential samples are generated for the remaining service times of all the activities that are employed under the allocation $u(t_k)$. Denote by $\tau_k$ the minimum of the corresponding service completion times and the next arrival as specified by the historical trace data. If $\tau_k \ge T$, we set $K = k + 1$, $q(t_{k+1}) = q(t_k)$, and end the simulation. Otherwise, set $t_{k+1} = \tau_k$. If the next event is an arrival, set $q(t_{k+1})$ by adding the corresponding arrival to $q(t_k)$. If the next event is a service completion, then random samples are generated to determine the routing of the completed jobs, and $q(t_{k+1})$ is set appropriately.

Consider the least squares minimization

$$\min_{r'} \frac{1}{2T} \sum_{k=0}^{K-1} \Big( (c \cdot q(t_k) - \eta)(t_{k+1} - t_k) + \tilde{Q}(q(t_{k+1}), u(t_{k+1}), r) - \tilde{Q}(q(t_k), u(t_k), r') \Big)^2. \tag{5.4}$$

This can be thought of as a sample path variation of the optimization problem (3.5). We can adjust the parameter vector $r$ using the gradient of the objective in (5.4), and update the average cost estimate $\eta$ according to the realized average cost along the sample path. Specifically, we have

$$r := r + \frac{\gamma_1}{T} \sum_{k=0}^{K-1} d(t_k) \nabla_r \tilde{Q}(q(t_k), u(t_k), r),$$

$$\eta := (1 - \gamma_2)\eta + \frac{\gamma_2}{T} \sum_{k=0}^{K-1} c \cdot q(t_k)(t_{k+1} - t_k),$$

where $d(t_k)$ is the temporal difference

$$d(t_k) = (c \cdot q(t_k) - \eta)(t_{k+1} - t_k) + \tilde{Q}(q(t_{k+1}), u(t_{k+1}), r) - \tilde{Q}(q(t_k), u(t_k), r),$$

and $\gamma_1, \gamma_2 > 0$ are step-sizes. The process is then repeated with a new sample path generated using the same historical trace data for arrivals, but with the allocation decisions made using the new parameter vector $r$, and with new random samples for service completions and routing events.

There are a few implementation details to consider. Typically, the exploration probability $\epsilon$ and the step-sizes $\gamma_1$ and $\gamma_2$ are taken to be decreasing from iteration to iteration. Further, it may be computationally more efficient to incrementally adjust $r$ and $\eta$ along the sample trajectory within each batch, in the style of the TD-learning algorithm described by (3.6)–(3.9). In particular, incremental algorithms typically converge faster. A hybrid approach may also be considered. Here, the historical data trace is divided into chunks, and the iterations of the algorithm cycle through these chunks, updating $r$ and $\eta$ after processing each chunk.

## 6. Case Study: The Crossbar Switch

The input-queued crossbar switch is an architecture for moving data packets from a collection of input ports (or sources) to a collection of output ports (or destinations). It is a common design paradigm used, for example, in many routers for packet switched networks such as the Internet. Making scheduling decisions for a crossbar switch can be viewed as an optimal control problem for a queueing network. This control problem does not fall in the canonical formulation described in Section 2, however. This is because a crossbar switch is most naturally modeled in a discrete time setting and with deterministic service times. Nevertheless, in this section, we demonstrate that many of the ideas from Section 3 can still be applied in this setting.

An $n \times n$ switch has $n$ input ports, labeled from the set $\{1, \ldots, n\}$, and $n$ output ports, also labeled from the set $\{1, \ldots, n\}$. See Figure 1 for a schematic diagram of a $3 \times 3$ switch . A packet arrives at an input port, and must be transported to a certain output port. The switch fabric is the mechanism by which the packets are transported. At each time-step, an allocation decision is made to connect input ports to output ports. This decision subject to a *matching* condition imposed by the switch fabric: at most one input port is connected to each output port, and at each input ports is connected to at most one output port. Once the decision is made, for each connected

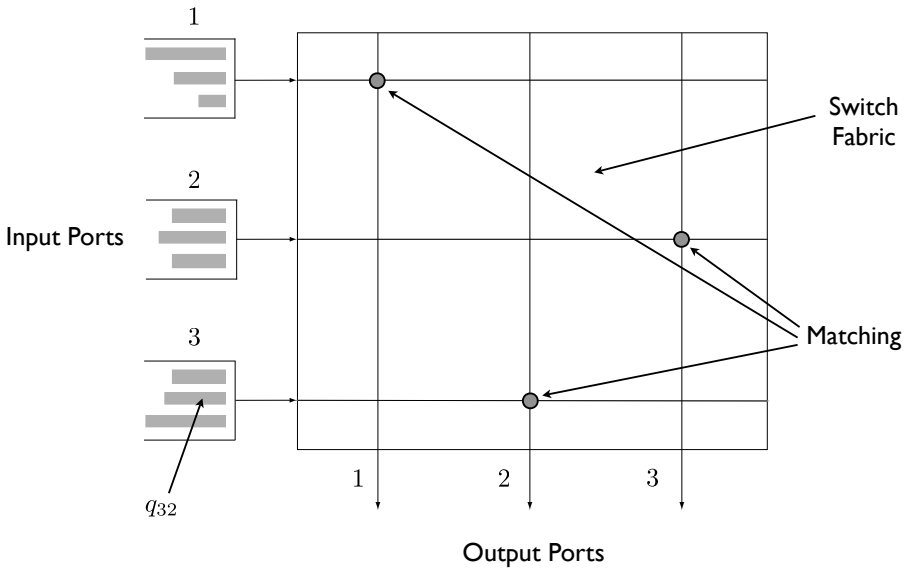



Figure 1: A $3 \times 3$ input-queued crossbar switch.

input-output pair $(i, j)$, if there is a packet waiting at the input port $i$ for the output port $j$, it is transported in a single time-step. Thus, at each time-step, a maximum of $n$ data packets are processed and leave the system.

The evolution of a switch can be modeled as a queueing network as follows. The switch proceeds in discrete time-steps $t = 0, 1, \ldots$. There are $n^2$ buffers. We denote the set of buffers by the set of input-output pairs

$$\mathcal{I} = \{(i, j) \mid 1 \leq i, j \leq n\}.$$

At each time $t$, we denote by $q_{ij}(t)$ the length of the buffer of packets awaiting transport from input port $i$ to output port $j$. Let $q(t) \in \mathbb{R}^{n^2}$ be the vector of buffer lengths.

An allocation decision is represented by a vector $u \in \mathbb{R}^{n^2}$. Here, $u_{ij} = 1$ if a packet from input port $i$ to output port $j$ is to be transported, and $u_{ij} = 0$ otherwise. We define the set $\mathcal{A}(q) \subset \mathbb{R}^{n^2}$ to be the set of admissible allocations given buffer lengths $q$. From the above discussion, $u \in \mathcal{A}(q)$ if and only if:

$$u_{ij} \in \{0, 1\}, \quad \forall\ 1 \leq i, j \leq n, \tag{6.1}$$

$$\sum_{j=1}^{n} u_{ij} \leq 1, \quad \forall\ 1 \leq i \leq n, \tag{6.2}$$

$$\sum_{i=1}^{n} u_{ij} \leq 1, \quad \forall\ 1 \leq j \leq n, \tag{6.3}$$

$$u_{ij} \leq q_{ij}, \quad \forall\ 1 \leq i, j \leq n. \tag{6.4}$$

Note that (6.2) and (6.3) enforce the matching condition. Also, this set is of the form described in Definition 1.

For each input-output pair $(i,j) \in \mathcal{I}$, we assume that arrivals occur according to independent Bernoulli processes with rate $\lambda_{ij} \in [0,1]$. Service times are deterministic and always equal to a single time-step. An analysis of the dual of the static planning LP (2.5) reveals that for this system, a necessary condition for the existence of a stable policy is[1]

$$\lambda_{i\oplus} = \sum_{j=1}^{n} \lambda_{ij} \leq 1, \quad \forall\ 1 \leq i \leq n,$$
$$\lambda_{\oplus j} = \sum_{i=1}^{n} \lambda_{ij} \leq 1, \quad \forall\ 1 \leq j \leq n.$$

The load of the system is

$$\rho^* = \max_{i,j} \{\lambda_{i\oplus}, \lambda_{\oplus j}\}.$$

We assume that $\rho^* < 1$.

Our objective is to find an allocation policy $u$ minimizing the long-term expected average delay,

$$\limsup_{T\to\infty} \mathsf{E}_u \left[ \frac{1}{T} \sum_{t=0}^{T-1} \mathbf{1} \cdot q(t) \right].$$

## 6.1. Maximum Weight Matching

The input-queued crossbar switch has been studied extensively in the literature (see, for example, [32, 33, 34, 35]). One class of heuristic policies that has been proposed is the class based on *maximum weight matching*. Here, a weight function $W : \mathbb{Z}_+^I \to \mathbb{R}^{n^2}$ is chosen, and a policy decision is made at time $t$ given buffer lengths $q(t) = q$ by solving the optimization

$$\max_{u \in \mathcal{A}(q)} W(q) \cdot u.$$

Because of the structure of the constraint set, this optimization problem is a maximum weight matching problem on a bipartite graph, and can be efficiently solved in $O(n^3)$ time. From the discussion in Section 5, it is clear a choice of $W(\cdot)$ is equivalent to a choice of an approximate $Q$-function.

MWM-$\alpha$ is a particular class of maximum weight matching policies parameterized by a scalar $\alpha \geq 0$. The MWM-$\alpha$ policy uses the weight function

$$W_{ij}(q) = q_{ij}^{\alpha} \mathbb{I}_{\{q_{ij}>0\}}.$$

[1] For a vector $x \in \mathbb{R}^{n^2}$, we use the notation

$$x_{i\oplus} = \sum_{j=1}^{n} x_{ij}, \quad \forall\ 1 \leq i \leq n,$$
$$x_{\oplus j} = \sum_{i=1}^{n} x_{ij}, \quad \forall\ 1 \leq j \leq n.$$

For example, where $\alpha = 0$, the policy is greedy and seeks an allocation that serves as many data packets in the next time-step as possible. As $\alpha \to \infty$, the policy seeks to give priority to serving packets from the longest buffers. When $\alpha = 1$, the policy is a variant of a scheduling method for wireless networks first proposed by Tassiulas and Ephremides [30, 31]. Subsequent work has generalized these policies to larger classes of queueing networks [20, 29].

For $\alpha \in (0, \infty)$, the MWM-$\alpha$ policy is known to be throughput optimal. That is, for any vector of arrival rates $\lambda$ with load less than 1, the MWM-$\alpha$ policy is stable [32, 34]. Further, it has been observed that the average delay improves as $\alpha \downarrow 0$ [34]. The MWM-0 policy, however, is something of a singularity. In particular, it can be unstable [35]. This is because MWM-0 is in some sense underspecified. Imagine, for example, a switch state where every buffer is non-empty. Here, there are $n!$ admissible allocations matching each input to an output, each of these allocations serves exactly $n$ packets in the next time-step, and hence any of these allocations is optimal for MWM-0.

One can consider a sequence of MWM-$\alpha$ policies, with $\alpha \downarrow 0$. This sequence of policies is known to have certain optimality properties in heavy traffic [29, 11, 12]. A limiting policy MWM-0+ can be defined as follows. Note that, for $\alpha$ small,

$$W_{ij}(q) \approx (1 + \alpha \log q_{ij}) \mathbb{I}_{\{q_{ij} > 0\}}.$$

The MWM-0+ policy selects an allocation by first considering the set of allocations computed by MWM-0, and then breaking ties within this set using a secondary weight function $\log q_{ij}$. Intuitively, MWM-0+ seeks to greedily serve as many packets as possible, but breaks ties by favoring service of longer buffers. MWM-0+ is conjectured to be optimal in the heavy traffic regime [11, 12].

## 6.2. Affine Expectations and the Allocation Polytope

The ADP method of Section 3 suggests computing an approximate differential cost function $\tilde{V} : \mathbb{Z}_+^{n^2} \to \mathbb{R}$, and then selecting an allocation decision $u(t)$ at time $t$ by

$$u(t) \in \operatorname*{argmin}_{u \in \mathcal{A}(q(t))} \mathsf{E}_u \left[ \tilde{V}(q(t+1)) \,\middle|\, q(t) \right]. \tag{6.5}$$

In general, this is a combinatorial optimization problem with a decision space that is exponential in size as a function of the switch size $n$. However, unlike the system described in Section 2, the expectation in (6.5) is *not* an affine function of the allocation $u$, in general.

Fortunately, there is a specific class of functions $\tilde{V}(\cdot)$ whose one-step expectations are affine as a function of the allocation. Consider functions of the form

$$\tilde{V}(q) = \sum_{C \in \mathcal{C}} F_C(q_C). \tag{6.6}$$

Here, $\mathcal{C}$ is a collection of subsets of the input-output pairs $\mathcal{I}$. That is, each $C \in \mathcal{C}$ is a subset $C \subset \mathcal{I}$. We restrict these subsets to correspond to at most a single input or output port. In other words, if $C \in \mathcal{C}$, then either there exists an input port $i$ so that

$$C \subset \{(i, j) \mid 1 \le j \le n\},$$

or there exists an output port $j$ so that

$$C \subset \{(i, j) \mid 1 \le i \le n\}.$$

Given a set $C \in \mathcal{C}$, $q_C$ is the vector of corresponding buffer lengths, and $F_C(\cdot)$ is a real-valued function that only depends on the buffer lengths described in $q_C$.

To verify the affine expectation property, consider a single subset $C$ and the corresponding function $F_C(\cdot)$. For a binary vector $a \in \mathcal{S} = \{0,1\}^{n^2}$, denote by $P_\lambda(a)$ the probability that, in a single time-step, there is an arrival to an input-output pair $(i,j) \in \mathcal{I}$ if and only if $a_{ij} = 1$. That is,

$$P_\lambda(a) = \prod_{(i,j)\in\mathcal{I}} \left(\lambda_{ij} a_{ij} + (1-\lambda_{ij})(1-a_{ij})\right).$$

Then, for a state $q \in \mathbb{Z}_+^{n^2}$ and an allocation $u \in \mathcal{A}(q)$,

$$\begin{aligned}
\mathsf{E}_u\left[F_C(q(t+1)) \mid q(t) = q\right] &= \sum_{a\in\mathcal{S}} P_\lambda(a) F_C\left(q_C + a_C - \sum_{(i,j)\in C} u_{ij} e_C^{ij}\right) \\
&= \sum_{a\in\mathcal{S}} P_\lambda(a) F_C(q_C + a_C) \\
&\quad + \sum_{a\in A} \sum_{(i,j)\in C} P_\lambda(a) \left[F_C\left(q_C + a_C - e_C^{ij}\right) - F_C(q_C + a_C)\right] u_{ij}.
\end{aligned}$$

Here, $e^{ij} \in \mathbb{Z}_+^{n^2}$ is a unit vector that is zero except in the $(i,j)$th component, and for a vector $x \in \mathbb{Z}_+^{n^2}$, we denote by $x_C$ its restriction to the subset $C$. The second equality relies on the fact that, because of the matching restriction, at most a single packet is served from a given input port or to a given output port in any time-step.

Restricting the class of approximate differential cost functions under consideration to those of the form (6.6), we can proceed as in Section 3.1. Define the allocation polytope $\bar{A}(q) \subset \mathbb{R}^{n^2}$ by relaxing the integrality condition (6.1) in the definition of $\mathcal{A}(q)$. That is, $\bar{A}(q)$ is the set of vectors $u \in \mathbb{R}_+^{n^2}$ satisfying (6.2)–(6.4), and with $0 \le u \le \mathbf{1}$. Note that for every $q \in \mathbb{Z}_+^{n^2}$, the vertices of $\bar{\mathcal{A}}(q)$ are contained in $\mathcal{A}(q)$. This follows from the fact that the constraints defining $\bar{\mathcal{A}}(q)$ possess a 'network flow' structure [26, Chapter 7]. Hence, Assumption 2 holds, and the policy decision (6.5) can be efficiently computed by linear programming or maximum weight matching methods.

## 6.3. Empirical Performance

In this section, we compare the performance of ADP-derived policies to those from the MWM-$\alpha$ class. We consider an $n \times n$ switch, for varying choices of $n$, under uniform arrival rates, for varying system loads $\rho^*$.

The basis function architecture used is of the following form:

$$\tilde{V}(q,r) = r_0 + \sum_{(i,j)\in\mathcal{I}} F(q_{ij}, r_1) + \sum_{i=1}^{n} G(q_{i\oplus}, r_2) + \sum_{j=1}^{n} G(q_{\oplus j}, r_2). \tag{6.7}$$

Note that the workloads for this system are $\{q_{i\oplus}, q_{\oplus j}\}$, hence this architecture is of the type suggested in Section 4. It is also of the form (6.6) than admits affine expectations. The functions $F(\cdot, r) :\ \mathbb{Z}_+ \to \mathbb{R}$ and $G(\cdot, r) :\ \mathbb{Z}_+ \to \mathbb{R}$ are parameterized according to

$$F(x, r_1) = \sum_{\ell=0}^{k-1} r_{1,\ell}(x-\ell)^+ + r_{1,k} x \log x, \quad G(x, r_2) = \sum_{\ell=0}^{k-1} r_{2,\ell}(x-\ell)^+ + r_{2,k} x \log x.$$

Intuitively, these functions are allowed to be arbitrary for small values of the argument $x$, up to some constant $k$. The tail behavior is a linear combination of the functions $x$ and $x \log x$. This tail behavior is motivated by the structure of the MWM-$\alpha$ policy. MWM-$\alpha$ is qualitatively similar to a policy which acts greedily with respect to a differential cost approximation of the form

$$\tilde{V}(q) = \sum_{(i,j)\in\mathcal{I}} q_{ij}^{1+\alpha} \approx \sum_{(i,j)\in\mathcal{I}} q_{ij} + \alpha q_{ij} \log q_{ij}.$$

Here, the approximation holds for $\alpha$ small.

Note also that our approximation architecture is invariant to permutations of input or output labels or the transposition of inputs and outputs. This is because the dynamics of the system have these same symmetries under uniform arrivals. In a more general setting, it is sensible to allow, for example, the parameter set $r_1$ to vary as a function of $(i, j)$ in the first summation in (6.7).

The parameter set $r$ was optimized using a discounted variation of the TD-learning update equations (3.6)–(3.9). A constant step-size was used and no exploration was employed.

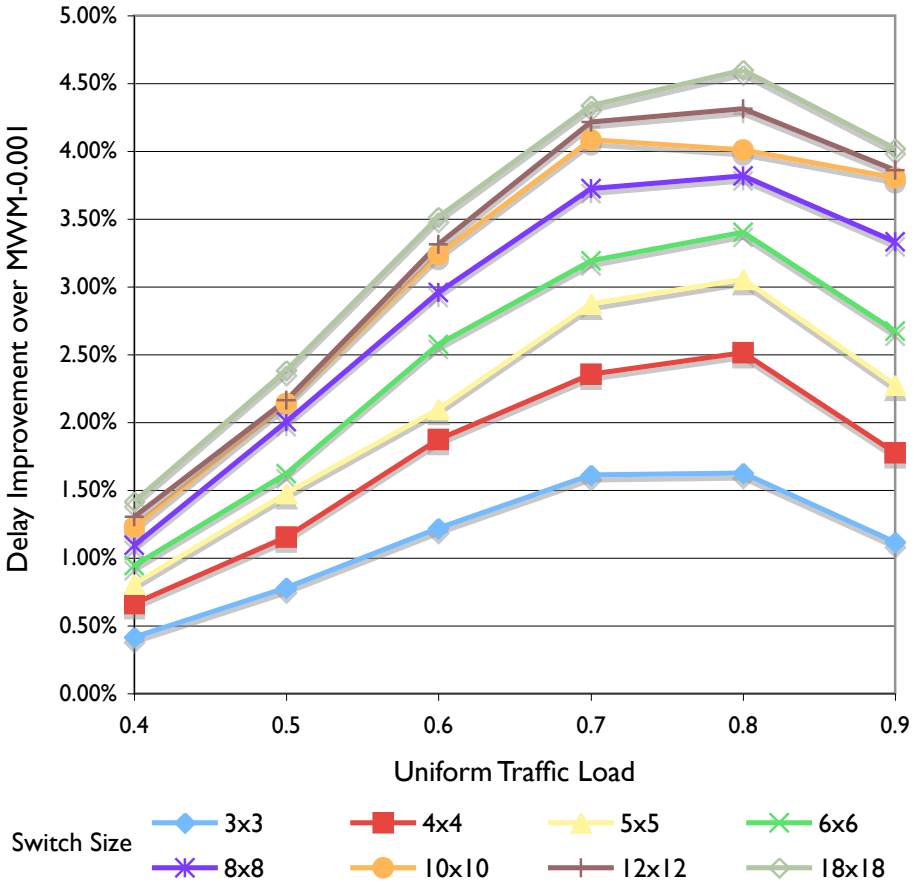
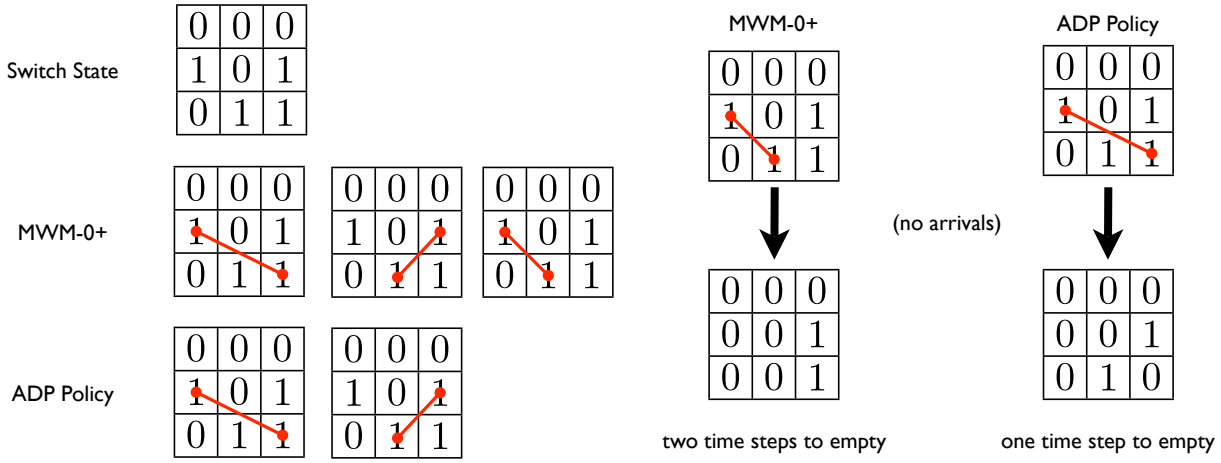



Figure 2: Average delay performance of ADP-derived policies versus MWM-0.001, over varying traffic load and switch size.

In Figure 2, we see the average delay performance of the ADP policies relative the MWM-0.001

policy. The performance improvement is measured across a variety of problem instances obtained by varying the switch size and the system load.

Note that in all instances, the ADP polices perform better. The improvement is most pronounced at moderate loads of 0.7 to 0.8. In light traffic, the improvements are more modest. Intuitively, this is because the scheduling problem is easier in light traffic where resources are less constrained. For a higher load of 0.9, the performance improvement also slightly decreases. This is consistent with the optimality properties that have been established for MWM-0+ in heavy traffic. More interestingly, the delay improvement increases as the switch size increases. Modern commercial routers typically employ switches with 16 to 64 ports, so the large switch regime is the most relevant. Further, in limited experiments, greater improvements have been observed with non-uniform arrival traffic.




(a) The switch state and corresponding allocations selected by MWM-0+ and ADP policies. The red circles indicate buffers that are selected for service under the various allocations.

(b) The evolution of the switch state over one time-step, assuming no arrivals.

Figure 3: A comparison of MWM-0+ and ADP policies for a particular state of a $3 \times 3$ switch.

In examining the policy decisions made by the ADP policies, we see that they typically act greedily with respect to the one-step cost. That is, they attempt to serve as many packets as possible, as in the case of the MWM-0 policy. However, ties are broken in a different way than the MWM-0+ policy. In Figure 3(a), a particular switch for a $3 \times 3$ switch is shown. MWM-0+ selects one of 3 possible allocations, while the ADP policy limits itself to the first 2 (note that these two allocations are equivalent under symmetry, so it is impossible to further distinguish them). All of these allocations are greedy with respect to the one-step cost, but, in Figure 3(b), we see that the 3rd choice made by MWM-0+ is inferior to the others. In particular, if there are no arrivals, this choice results in a switch state in the following time-step with a larger time-to-empty. It is precisely the use of workload in the ADP basis function architecture that allows this type of decision to be made. A similar structure exists in the 'longest-port-first' heuristic policy proposed in [33].

Finally, note that the input-queued crossbar switch is an example that possesses considerable structure and symmetry. This has be exploited in the design of good heuristic policies. In large, complex, and unstructured examples, it is can very difficult to design good heuristics. It is in such cases that ADP-derived policies are likely to be most advantageous.

## Acknowledgments

The first author was supported by a Benchmark Stanford Graduate Fellowship. The first author wishes to thank Devavrat Shah for helpful discussions.

## References


[1] C. E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.

[2] R. E. Bellman and S. E. Dreyfus. Functional approximation and dynamic programming. *Math. Tables and Other Aids Comp.*, 13:247Ð251, 1959.

[3] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, pages 210–229, 1959.

[4] D. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[5] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[6] B. Van Roy. Neuro-dynamic programming: Overview and recent trends. In E. A. Feinberg and A. Shwartz, editors, *Handbook of Markov Decision Processes: Methods and Applications*, pages 305–346. Kluwer, Boston, MA, 2002.

[7] J. Si, A. G. Barto, W. B. Powell, and D. Wunsch, editors. *Learning and Approximate Dynamic Programming: Scaling up to the Real World*. John-Wiley and Sons, New York, 2004.

[8] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley-Interscience, Hoboken, NJ, 2007.

[9] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, Belmont, MA, 3rd edition, 2006.

[10] R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

[11] D. Shah and D. J. Wischik. Optimal scheduling algorithms for input-queued switches. In *Proceedings of IEEE INFOCOM*, Barcelona, Spain, April 2006.

[12] D. Shah and D. J. Wischik. Heavy traffic analysis of optimal scheduling algorithms for switched networks. Submitted, 2007.

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

[14] D. P. de Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, 2003.

[15] D. P. de Farias and B. Van Roy. A cost-shaping linear program for average-cost approximate dynamic programming with performance guarantees. *Mathematics of Operations Research*, 31(3):597–620, 2006.

[16] D. S. Choi and B. Van Roy. A generalized Kalman filter for fixed point approximation and efficient temporal-difference learning. *Discrete Event Dynamic Systems*, 16(2):207–239, 2006.

[17] D. Adelman. A price-directed approach to stochastic inventory/routing. *Operations Research*, 52(4):499–514, July-August 2004.

[18] M. H. Veatch. Approximate dynamic programming for networks: Fluid models and constraint reduction. preprint, 2005.

[19] J. M. Harrison. Brownian models of queueing networks with heterogeneous customer populations. In *Stochastic Differential Systems, Stochastic Control Theory and Applications (Minneapolis, Minn., 1986)*, volume 10 of *IMA Vol. Math. Appl.*, pages 147–186. Springer, New York, 1988.

[20] J. G. Dai and W. Lin. Maximum pressure policies in stochastic processing networks. *Operations Research*, 53:197–218, 2005.

[21] S. P. Meyn. Stability, performance evaluation and optimization. In E. A. Feinberg and A. Shwartz, editors, *Handbook of Markov Decision Processes: Methods and Applications*, pages 305–346. Kluwer, Boston, MA, 2002.

[22] S. P. Meyn. Sequencing and routing in multiclass queueing networks. Part I: Feedback regulation. *SIAM Journal on Control and Optimization*, 40(3):741–776, 2001.

[23] S. P. Meyn. Workload models for stochastic networks: Value functions and performance evaluation. *IEEE Transactions on Automatic Control*, 50(8):1106–1122, 2005.

[24] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of optimal queueing network control. *Mathematics of Operations Research*, 24(2):293–305, May 1999.

[25] M. Bramson and R. J. Williams. Two workload properties for Brownian networks. *Queueing Systems*, 45:191–221, 2003.

[26] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, Belmont, MA, 1997.

[27] S. P. Meyn. Sequencing and routing in multiclass queueing networks. Part II: Workload relaxations. *SIAM Journal on Control and Optimization*, 42(1):178–217, 2003.

[28] G. Weiss. A simplex based algorithm to solve separated continuous linear programs. *Mathematical Programming*, 115(1):151–198, 2008.

[29] A. L. Stolyar. MaxWeight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic. *Annals of Applied Probability*, 14(1):1–53, 2004.

[30] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936–1948, December 1992.

[31] L. Tassiulas and A. Ephremides. Dynamic server allocation to parallel queues with randomly varying connectivity. *IEEE Transactions on Information Theory*, 39(2):466–478, March 1993.

[32] N. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *Proceedings of IEEE INFOCOM*, pages 296–302, San Francisco, CA, March 1996.

[33] A. Mekkittikul and N. McKeown. A practical scheduling algorithm to achieve 100% throughput in input-queued switches. In *Proceedings of IEEE INFOCOM*, 1998.

[34] I. Keslassy and N. McKeown. Analysis of scheduling algorithms that provide 100% throughput in input-queued switches. In *Proceedings of the 39th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, October 2001.

[35] I. Keslassy, R. Zhang-Shen, and N. McKeown. Maximum size matching is unstable for any packet switch. *IEEE Communications Letters*, 7(10):496–498, October 2003.