Approximate Linear Programming for Networks: Average Cost Bounds

Michael H. Veatch
Department of Mathematics
Gordon College
mike.veatch@gordon.edu

May 19, 2010

Subject classifications: Dynamic programming/optimal control: approximations/large-scale problems.

Abstract

This paper uses approximate linear programming (ALP) to compute average cost bounds for multiclass queueing network control problems. The method requires only solving an LP and approximates the differential cost by a linear form. A largely empirical approach is taken to identifying approximating functions, though guided by fluid analysis. For several choices of functions, the structure of the constraints allows the infinite constraint set to be reduced to a more manageable, though still exponentially large, set. For other functions, constraint sampling and truncation methods are developed and justified by analysis of the constraints. On examples with two to six buffers, bounds on average cost were 12 to 40% below optimal using quadratic approximating functions. These LPs are easily solved on examples with up to 17 buffers. Using additional functions reduced error to 1 to 5% at the cost of larger LPs, most of which were solved quickly by a commercial solver. The method computes bounds much faster than standard value iteration. Policies are also constructed from the ALP.

1 Introduction

Multiclass queueing networks (MQNETs) are a common modeling framework for manufacturing, computer, and communication systems. Even under the simplest assumptions of exponentially distributed service and
interarrival times and linear holding costs, MQNET control problems are NP-hard so that we cannot hope to solve large problems exactly [28]. Standard policy iteration or value iteration are too computationally intensive to use even on moderate size problems, particularly in heavy traffic, due to the well-known “curse of dimensionality.”

This paper uses approximate linear programming (ALP) to compute lower bounds on average cost for MQNETs. More specifically, a sequence of bounds with increasing accuracy are computed by solving LPs with an increasing number of variables. The method does not scale in the sense of a polynomial size LP; however, it is efficient enough that moderately tight bounds can be computed for problems that are beyond the reach of exact dynamic programming (DP). For smaller problems, optimal average cost can generally be approximated much faster using ALP than DP. ALP has been used on a number of challenging Markov decision processes (MDPs); see [7, Section 6.3], [10] and the references therein. Some of the algorithmic strategies in this paper are general enough that they should be useful for other problems.

A primary use of such bounds is benchmarking the many heuristics for controlling these systems. The justification of these heuristic policies has been less than satisfactory. Many heuristic policies have been shown to be stable, but relatively little is known about suboptimality. Certain heuristic policies have been shown to be asymptotically optimal for the limiting Brownian control problem in heavy traffic or the fluid control problem, which essentially considers large buffer contents. However, asymptotic optimality is generally too loose a criteria for designing near-optimal policies.

The primary challenge in applying any approximate DP method, including ALP, is selecting a compact but accurate class of approximating functions for the cost-to-go or, in our average cost setting, the differential cost. The first contribution of this paper is identifying new approximating functions. Some of the functions are motivated by analysis of the MQNET and its fluid limit; others are general function approximation methods on a multidimensional space. Based on numerical testing, we develop a sequence in which functions are added to the approximation architecture to improve accuracy. The combination of several types of functions performs much better than functions of a single type. Moreover, the sequencing of the functions allows a trade-off between accuracy and size of the LP, which is needed in practice given the intractability of these problems.

The second challenge in solving the ALP is that it contains one constraint for every state-action pair. An open network with uncontrolled arrivals has an infinite state space. Using the standard truncation, where buffer sizes are limited, can be more efficient than DP but is not a large improvement. The second contribution of this paper is to provide new methods of reducing the number of constraints that exploit the
structure of the constraints. We use the method in [26] for quadratic functions and extend the method in [27] for piece-wise quadratics. A new method is presented for reducing certain exponential functions. Finally, for functions of one or two state variables, we use the notion of a factored MDP to reduce the number of constraints. For each type of function, the number of constraints is still exponential in the number of buffers; however, the reduction extends considerably the size of problem for which the ALP can be solved.

To accommodate general functions and larger problems, the alternative used by several authors is constraint sampling. Although constraint sampling has theoretical support [8], it has serious limitations in our numerical tests: a very large number of constraints must be sampled to obtain a small sampling error on some moderate size problems (see Section 6.2). The poor performance could be due to the more complex functions we use. In [7, Section 6.3] constraint sampling is tested on quadratic functions; it is also used for the game of tetris, with mostly linear functions, in [10]. The limitations of constraint sampling make constraint reduction a useful approach when it is available. We also use a hybrid approach: Motivated by the fact that the constraints are a smooth function of the queue lengths, we supplement randomly sampled constraints with certain limiting constraints to improve accuracy.

The third contribution of this paper is an extensive numerical study of the accuracy of the ALP bound for various approximation architectures. Tests were conducted on problems with up to six buffers that could be solved using value iteration. The quadratic approximation, which is the easiest LP to solve, gave an average cost 12 to 40% below optimal. By testing various combinations of other functions, relatively small approximation architectures were found that achieved errors of 1 to 5%.

We also briefly address the policies associated with the differential cost approximations. The approximation architectures are compact enough that one could implement these policies. However, testing has shown that they do not always have good performance and may not even be stabilizing, as demonstrated in [6] for a related control problem involving a single queue. We report some cases where the ALP gives a useful policy and relate the approximation architecture to the form of policy obtained.

An average cost objective is used throughout. Although stronger theoretical results are available for a discounted cost objective, there are several reasons to consider average cost. For most dynamic scheduling problems, a long time horizon seems more realistic and avoids having to choose a discount rate. Average cost is also much easier to simulate for a given policy. In the average cost problem, the optimality equations can be written using difference operators, facilitating constraint reduction and making the resulting LP more sparse. The average cost problem can also be related to the fluid model, giving some guidance in the choice of approximating functions.
Several criteria were used in identifying approximating functions. Some functions are appealing because they allow constraint reduction or create sparse columns in the LP. After solving a given ALP, Bellman error can be examined to give some indication of where an approximation should be improved. We consider the following functions.

(i) **Quadratics**.

(ii) **Exponential** decay functions, which focus on states with small queue lengths.

(iii) **Piece-wise quadratics**, with the regions taken from the associated fluid model. For networks of more than a few queues, identifying these regions becomes difficult; instead, a simplified fluid model or some heuristic method could be used.

(iv) **Rational** functions. A set of rational functions is constructed that attempts to fit higher-order interactions in a region of the state space where each queue length is below a specified value.

(v) **Piece-wise quadratics on rectangular regions**. Using the basic insight that important states in the approximation generally have small queue lengths, we define rectangular regions using intervals that expand with queue length.

(vi) **Functions of one or two variables**. A common approach to high-dimensional state spaces is to map the state into a single variable, or feature, and define functions of this new variable. We consider functions of a single queue length or station workload. Here workload is defined as the expected time that a station will spend serving all jobs currently in the network. For series systems, we also use echelon inventory as a feature and consider functions of two variables: queue length and echelon inventory at a class. These functions allow constraint reduction using a factored MDP approach.

Constraint reduction is possible for (i), (iii), (vi), and, to a lesser extent, (ii) and (v). The functions (iii), (v), and (vi) create sparsity.

Although our main goal is developing ALPs that perform well numerically, we formulate the average cost ALP for the countable state space MQNET. Most bounds for general MDPs, such as [7, Section 6.3], [8], and [10], assume a finite state space. The approach is similar to [26], which considers a less general problem. We also relate error in the differential cost approximation, as measured by expected Bellman error, to the accuracy of the ALP average cost. This relationship helps motivate fitting the differential cost to improve the ALP average cost.
The ALP approach was originally proposed by [32]. It is applied to discounted network problems in [7] using quadratic value function approximations. For MDPs on a finite state space, they provide an error bound for the ALP value function. In particular, a suitable weighted norm of the error is bounded by the minimum of this error norm over all functions in the approximating class, multiplied by a constant that does not depend on problem size. Similar bounds are given on performance of the policy implied by the ALP value function. Instead of constraint reduction, they use constraint sampling, which is shown to be probabilistically accurate in [8]. The results in [7] and [8] are extended in [10] to a Lagrangian form of the ALP that is potentially more accurate. For average cost problems, two modifications of the ALP approach are proposed in [6] and [9]. Although the latter provides a performance bound, it is not clear how to apply it to networks. Also, [9] addresses countable state spaces. In [37], column generation methods are used to solve average cost ALPs more efficiently. Approximate dynamic programming is applied to a specific communication network—a crossbar switch—in [25] using “ridge” functions of a single queue length or workload and to inventory problems in [1] and [34] using linear functions. A major difference between our work and [25] is that we identify different approximating functions which give a better trade-off between speed and accuracy for a test suite of MQNETs.

Constraint reduction for piece-wise quadratic functions is used in [26] and [27]. They consider a different quadratic on each set of states defined by which buffers are empty, which is a special case of (v). Our paper develops additional constraint reduction methods for (iii). Another difference is that they consider specific policies, rather than bounding average cost for a class of policies. Factored value functions and MDPs are introduced in [16]. Constraint reduction for them is presented in [12], where the LP arises in max-norm projection, and is applied to ALP in [31] and [13].

Average cost bounds have also been obtained for queueing networks using the achievable region method [4], [18] and Lyapunov functions [3]. These results are elegant but not very satisfactory from a numerical viewpoint. For example, the achievable region bound is quite loose in balanced heavy traffic, i.e., when more than one server has a traffic intensity near one. A duality relationship between the achievable region method and ALPs with quadratic approximation is noted in [17]. This relationship leads to the result that the quadratic approximation ALP gives a tighter bound than the achievable region LP; see [26, Appendix A] and [30].

The rest of this paper is organized as follows. Section 2 defines the MQNET sequencing problem and the associated fluid control problem and Section 3 describes average cost ALPs. In Section 4 the various approximating functions are introduced. Constraint reduction for some of these functions is presented in
Section 5. Numerical results are presented in Section 6, including a description of the constraint sampling and truncation method. Conclusions and future work are discussed in Section 7.

2 Open MQNET sequencing: Discrete and fluid models

In this section we describe the standard MQNET model and the fluid model associated with it. There are $n$ job classes and $m$ resources, or stations, each of which serves one or more classes. Associated with each class is a buffer in which jobs wait for processing. Let $x_i(t)$ be the number of class $i$ jobs at time $t$, including any that are being processed. Class $i$ jobs are served by station $\sigma(i)$. The topology of the network is described by the routing matrix $P = [p_{ij}]$, where $p_{ij}$ is the probability that a job finishing service at class $i$ will be routed to class $j$, independent of all other history, and the $m \times n$ constituency matrix with entries $C_{ji} = 1$ if station $j$ serves class $i$ and $C_{ji} = 0$ otherwise. If routing is deterministic, then $p_{i,s(i)} = 1$, where $s(i)$ is the successor of class $i$. If, in addition, routes do not merge then either $p_{p(i),i} = 1$, where $p(i)$ is the unique predecessor of class $i$, or $i$ has no predecessor.

Exogenous arrivals occur at one or more classes according to independent Poisson processes with rate $\alpha_i$ in class $i$. Processing times are assumed to be independently exponentially distributed with mean $m_i = 1/\mu_i$ in class $i$. To create an open MQNET, the routing matrix $P$ is assumed to be transient, i.e., $I + P + P^2 + \ldots$ is convergent. As a result, there will be a unique solution to the traffic equation

$$\lambda = \alpha + P'\lambda$$

given by

$$\lambda = (I - P')^{-1}\alpha.$$ 

Here $\lambda_i$ is the effective arrival rate to class $i$, including exogenous arrivals and routing from other classes, and vectors are formed in the usual way. The traffic intensity is given by

$$\rho = C \text{ diag}(m_1, \ldots, m_n)\lambda$$

that is, $\rho_j$ is the traffic intensity at station $j$. Stability requires that $\rho < 1$.

The network has sequencing control: each server must decide which job class to work on next, or possibly to idle. Preemption is allowed. Let $u_i(t) = 1$ if class $i$ is served at time $t$ and 0 otherwise. Admissible
controls are nonanticipating and have

\[ Cu(t) \leq 1 \]
\[ u_i(t) \leq x_i(t). \]

The first constraint states that a server’s allocations cannot exceed one; the second prevents serving an empty buffer.

The objective is to minimize long-run average cost

\[ J(x, u) = \lim_{T \to \infty} \frac{1}{T} E_{x,u} \int_0^T c'x(t) dt. \]

Here \( E_{x,u} \) denotes expectation given the initial state \( x(0) = x \) and policy \( u \). Consider only stationary Markov policies and write \( u(t) = u(x(t)) \). We use the uniformized, discrete-time Markov chain and assume that the potential event rate is \( \sum_{i=1}^n (\alpha_i + \mu_i) = 1 \). Let \( P_u = [p_u(x,y)] \) be the transition probability matrix under policy \( u \) and use the notation

\[ (P_u h)(x) = \sum_y p_u(x,y)h(y). \]

Let \( A(x) \) be the set of feasible controls in state \( x \) and \( A \) be their union.

Under the condition \( \rho < 1 \), the control problem has several desirable properties:

1. An optimal policy exists and its average cost is constant, \( J_* = \min_u J(x, u) \) for all \( x \).

2. There is a solution \( J_* \) and \( h_* \) to the average cost optimality equation

\[ J + h(x) = c'x + \min_{u \in A(x)} (P_u h)(x). \]  \hspace{1cm} (1)

3. Under the additional condition that \( h \) is bounded below by a constant and above by a quadratic, there is a unique solution \( J_* \) and \( h_* \) to (1) satisfying \( h_*(0) = 0 \). Furthermore, \( J_* \) is the optimal average cost, any policy

\[ u_*(x) = \arg \min_{u \in A(x)} (P_u h_*) (x) \]

is optimal, and \( h_* \) is the differential cost of this policy,
\[ h_*(x) = \lim_{T \to \infty} \sup (E_{x,u_*} \int_0^T c'(x(t))dt - E_{0,u_*} \int_0^T c'(x(t))dt). \]  

(2)

Properties (1) and (2) can be established using general results for MDPs as in [33, Theorems 7.2.3 and 7.5.6]. For networks, properties (1) and (2) are shown in [20, Theorem 7]; (3) is obtained by applying standard verification theorems to networks; see, e.g., [19, Theorem 2.1 and Section 7] and [21, Theorem 10.7].

A natural starting point in approximating the differential cost function is the associated fluid model. In this model all transitions are replaced by their mean rates and a continuous state \( q(t) \in \mathbb{R}_+ \) is used. In a fluid control problem, for each initial state there is a time horizon \( T \) such that \( q(t) = 0 \) for all \( t \geq T \). The fluid control problem corresponding to (1) is

\[
\text{(FCP)} \quad V(x) = \min_u \int_0^T c'q(t)dt \\
\dot{q}(t) = Bu(t) + \alpha \\
Cu(t) \leq 1 \\
q(0) = x \\
q(t) \geq 0, \quad u(t) \geq 0
\]

where \( \alpha = (\alpha_1, \ldots, \alpha_n)' \) and \( B = (P' - I)\text{diag}(\mu_1, \ldots, \mu_n) \). An optimal \( u_*(t) \) can be chosen so that it is piece-wise constant, making \( q(t) \) piece-wise linear with \( \dot{q}(t) \) existing except on a set of zero measure. The fluid “cost to drain” \( V(x) \) guides some of our approximations of \( h(x) \). The motivation for this approximation is [20, Theorem 7(iv)], based on [19, Theorem 5.2]. It establishes the following connection between the discrete and fluid cost functions:

\[
\lim_{\theta \to \infty} \frac{h_*(\theta x)}{V(\theta x)} = 1
\]

(3)

for any \( x \neq 0 \); see [36] for discussion of the policy implications.

A scaling argument shows that \( V \) grows quadratically in any direction: \( V(\theta x) = \theta^2 V(x) \). In fact, \( V \) is piece-wise quadratic and can be written

\[ V(x) = \frac{1}{2} x'Q_k x, \quad x \in S_k \]  

(4)
for some partition $S_1, \ldots, S_\kappa$ of $\mathbb{R}^n_+$. Combining (3) and (4),

$$h_*(x) = \frac{1}{2} x' Q_k x + o(|x|^2), \quad x \in S_k.$$  \hspace{1cm} (5)

Because $V$ is continuous, we can assume that each $S_k$ has full dimension.

The quadratic regions depend on the optimal fluid policy, which partitions $\mathbb{R}^n_+$ into control switching sets where the control is constant. The scaling property of $V$ implies a scaling property for the switching sets; however, if the optimal policy is not unique, ties must be broken appropriately. For some optimal fluid policy, each switching set is a cone emanating from the origin, i.e., if a switching set contains $x$, it also contains $\theta x$, $\theta > 0$. Assume that the switching sets are convex and polyhedral (we conjecture that this is true for some optimal fluid policy; again, ties must be broken appropriately). It is straightforward to show from its definition that $V$ is quadratic on the set of $x$ for which the trajectory from initial state $x$ visits a specified (finite) sequence of switching sets. By definition, this set is a subset of a switching set, so $\dot{q}(x)$ is constant in this set. An affine geometry applies, and this set is also a convex polyhedral cone emanating from the origin. Some sequences of switching sets visited will define sets that are not full dimension; these can be eliminated. Also, some sequences may share the same quadratic $V$; these can be combined. The rest are used as the $S_k$. Thus, the quadratic regions $S_k$ are convex polyhedral cones of full dimension, emanating from the origin, that generally subdivide (but may also combine) the switching sets of full dimension.

3 Approximate LP: Average cost bounds

In this section we describe a general method for constructing a linear program in a small number of variables that approximates the differential cost and places a lower bound on average cost. For problems with finite state spaces and $J_*$ independent of the initial state, an inequality relaxation of Bellman’s equation gives an equivalent LP in the same variables (see, e.g., [2]):

$$\begin{align*}
\text{(LP)} \quad & \max_{J, h} J \\
\text{s.t.} \quad & J + h(x) \leq c' x + (P_u h)(x) \quad \text{for all } x \in \mathbb{Z}^n_+, \ u \in \mathcal{A}(x) \\
& h(0) = 0.
\end{align*}$$
An additional condition on $h$ is needed because of the countable space: for some $L_1, L_2 > 0$,

$$-L_1 \leq h(x) \leq L_2(1 + |x|^2). \quad (6)$$

Appendix A shows that (1) is equivalent to (LP) and (6). This exact LP has one variable for every state. To create a tractable LP, the differential cost can be approximated by a linear form

$$h_\ast(x) \approx \sum_{k=1}^{K} r_k \phi_k(x) = (\Phi r)(x) \quad (7)$$

using some small set of basis functions $\phi_k$ and variables $r_k$. Assume that $\phi_k(0) = 0$. The resulting approximate LP is

$$(ALP) \quad \bar{J} = \max_{J, r} J$$

s.t. $J + (\Phi r)(x) \leq c'x + (P_u \Phi r)(x)$ for all $x \in \mathbb{Z}_+^n$, $u \in \mathcal{A}(x)$

$$-L_1 \leq (\Phi r)(x) \leq L_2(1 + |x|^2) \quad \text{for all } x \in \mathbb{Z}_+^n.$$

The bounds $L_1$ and $L_2$ may depend on $r$; all that is needed is that the bound applies to each $\phi_k$. Since (ALP) is formed by adding the constraints $h = \Phi r$ to the exact LP, the exact LP is a relaxation. Hence, (ALP) gives a lower bound, $J_\ast \geq \bar{J}$. (ALP) is feasible, so it has an optimal solution, say $r_\ast$.

By reversing the first inequality in (ALP) and restricting the class of policies, one can compute an upper bound on the average cost of these policies. This upper bound is used to check stability of, say, all nonidling policies. The upper bound for a single policy $u$ is considered in [26]. Of course, a policy $u$ can also be simulated to estimate its average cost $J_u$.

Any differential cost approximation $h$ defines an $h$-greedy policy

$$u_h(x) = \arg \min_{u \in \mathcal{A}(x)} (P_u h)(x).$$

The approximation architecture $\Phi$ restricts the greedy policy to a certain class of policies. For example, a quadratic approximation architecture implies linear boundaries between control regions. The greedy policy from (ALP) may have poor performance or not even be stabilizing. However, in some cases it has good performance, as reported in Section 6.5.

It is not obvious that (ALP) will provide a good approximation $\Phi r_\ast$ to the differential cost $h_\ast$, particularly
since the objective does not involve $\Phi r$. In contrast, discounted cost ALPs use a weighted sum of the value function approximations as the objective. One response to this apparent shortcoming of (ALP) has been to modify its objective [6], [9]. These papers show that their modified ALPs satisfy certain error bounds. We use (ALP) because we are primarily interested in estimating $J^*$. However, the proposition below provides some justification for trying to “fit” $h_*$ when searching for additional approximating functions $\phi_k$. Given any $J$ and $h$, Bellman error is defined as

$$B(x) = \min_{u \in A(x)} (P_u h)(x) - h(x) + c' x - J.$$  \hfill (8)

If $J, h$ satisfy the constraints (ALP) then $B(x) \geq 0$ is the minimum slack of constraints for that $x$.

**Proposition 1** Let $J, r$ be feasible for (ALP), $h = \Phi r$, and $\tilde{u}$ be an $h$-greedy policy. Assume

A1. $\tilde{u}$ is stabilizing. Let $E_{\tilde{u}}$ denote expectation with respect to a stationary distribution for policy $\tilde{u}$.

A2. $J_{\tilde{u}} < \infty$ and $E_{\tilde{u}} (B) < \infty$.

Then

$$J = J_{\tilde{u}} - E_{\tilde{u}} (B).$$ \hfill (9)

**Proof.** Since $\tilde{u}$ achieves the minimum in (8),

$$B(x) = (P_{\tilde{u}} h)(x) - h(x) + c' x - J.$$ \hfill (10)

Taking expectations,

$$E_{\tilde{u}} (B) = E_{\tilde{u}} [(P_{\tilde{u}} h)(x) - h(x) + c' x - J]$$

$$= E_{\tilde{u}} [c' x - J]$$

$$= J_{\tilde{u}} - J.$$  

The crucial second equality above holds by Proposition 8.2.5 of [24], which requires A2 and the growth condition (6).

Interpreting (8) as (1) for the perturbed problem with cost $c' x - B(x)$ and (10) as Poisson’s equation for this problem, Proposition 8.2.5 asserts that $h$ is a valid solution (the differential cost). The significance of
(9) is that
\[ J_\ast - J = (J_\ast - J_\tilde{u}) + E_\tilde{u}(B). \] (11)

In words, the average cost error decomposes into the suboptimality of policy \( \tilde{u} \) and the expected Bellman error. Given an approximation architecture for which the suboptimality is relatively small, this suggests a criteria for adding functions to the approximation: They should address Bellman error in states with a large contribution to its expectation. Bellman error for quadratic \( h \) is discussed further in Section 5.1.

The approximate LP is still not a manageable size because it has one constraint for each state-action pair. In Section 5, various constraint sets are algebraically reduced or approximated by a smaller set of constraints.

4 Differential cost approximation

This section defines and motivates the functions \( \phi_k \) used to approximate the differential cost.

(i) Quadratic functions

Consider the quadratic differential cost approximation
\[ h(x) = \frac{1}{2}x'Qx + px \] (12)
where \( Q = [q_{ij}] \) is symmetric. For convenience, the variable names \( q_{ij} \) and \( p_i \) will be kept rather than mapping them into \( r_k \) as in (7). This approximation is motivated by (5). It is also interesting to note that for a single uncontrolled queue
\[ h_\ast(x) = \frac{1}{2(\mu - \alpha)}(x^2 + x). \] (13)
The quadratic term in (13) matches the fluid value function; the effect of randomness is to shift the fluid value function 1/2 unit to the left. Constraint reduction for quadratics is covered in Section 5.1.

(ii) Exponential functions

Numerical experience suggests that a quadratic approximation misses important features of \( h_\ast \) in states where certain queue lengths are small or zero. Figure 1 shows the residual when a quadratic is fit to \( h_\ast \) over the region graphed for the series queue with \( \mu_1 > \mu_2 \) of Section 6. The residuals, plotted on the \( z \) axis, are small compared to \( h_\ast \); the largest value of \( h_\ast \) on this grid is over 500. The percent residual is larger when \( x \) is small and particularly when \( x_2 \) is small. The optimal stationary distribution also has larger probabilities in these states. In fact, above a switching curve where server 1 idles the probabilities are zero. In this example,
the switching curve limits \( x_2 \leq 5 \) when \( x_1 = 1 \) and \( x_2 \leq 9 \) when \( x_1 = 10 \). Thus, additional functions to approximate the complex shape of \( h_* \) when \( x_2 \) is small appear important.

For this example and using the quadratic approximation, (ALP) can be solved analytically for \( r_* \). The resulting Bellman error is zero when \( x_2 > 0 \); for \( x_2 = 0 \) it has the form \( B(x_1,0) = a + (b/\rho_2)x_1 \). Different parameter values lead to different Bellman errors, but only when \( \mu_1 < \mu_2 \) is Bellman error nonzero at \( x_2 > 0 \). In light of (11), it is reasonable to seek functions that reduce \( B(x) \). Of course, using functions that only adjust \( h(x_1,0) \) will introduce Bellman error in the adjacent states \((x_1,1)\). A better choice is functions that decrease with \( x_2 \). Although it would be difficult to repeat this analysis for larger networks, we suspect that the complex shape of \( h_* \) and the importance of states with at least one small queue length are typical.

For general networks, to emphasize states with small \( x_i \), we will use the functions

\[
\phi_i(x) = \beta_i^{x_i} \quad \text{(14)}
\]
\[
\phi_{ij}(x) = x_j\beta_i^{x_i} \quad \text{(15)}
\]

where \( \beta_i < 1 \). There is a natural choice of \( \beta \) in (14) for some classes. If class \( i \) has a unique predecessor

Figure 1: Residual \((z)\) in the best quadratic fit to \( h_* \) for the series queue.
class, no arrivals, and \( \mu_{p(i)} > \mu_i \) set \( \beta = \mu_i / \mu_{p(i)} \). Then

\[
[(P_u - I)\phi_i](x) = \left[ (\alpha_i + u_{p(i)}\mu_{p(i)})(\beta_i - 1) + u_i\mu_i(1/\beta_i - 1) \right] \phi_i(x) \\
= \left[ u_{p(i)}(\mu_i / \mu_{p(i)} - 1) + u_i\mu_i(\mu_{p(i)}/\mu_i - 1) \right] \phi_i(x) \\
= (u_{p(i)} - u_i)(\mu_i - \mu_{p(i)})\phi_i(x).
\]

Note that if the two classes have different servers, \( \sigma(p(i)) \neq \sigma(i) \), then in states where both are being served, \( u_{p(i)} = u_i = 1 \) and (16) is equal to 0, i.e., (16) is in the kernel of the generator \( P_u \). This has two advantages. First, as shown in Section 5.2, it allows constraint reduction. Second, it can influence Bellman error in states where class \( i \) is not being served without affecting it in states where classes \( i \) and \( p(i) \) are being served. For the switching curve policies typical of these networks, this would mean influencing Bellman error in states with \( x_{p(i)} \) at or above the class \( i \) switching curve.

Another advantage of (14) is that the greedy policies for this approximation include more realistic policies than the linear switching curve policies that result when \( h \) is quadratic. For example, consider adding just \( r\phi_2 \) to the quadratic (12) for the series queue. Server 1 is busy in the greedy policy when

\[
h(x-e_1+e_2) - h(x) = (-q_{11} + q_{12})x_1 + (q_{22} - q_{12})x_2 + \frac{1}{2}q_{11} + \frac{1}{2}q_{22} - q_{12} - p_1 + p_2 - r(1 - \beta)\beta x_2 < 0. \quad (17)
\]

Solving this ALP numerically often leads to \( q_{22} = q_{12} \) and \( r_1 < 0 \); then (17) reduces to

\[
x_2 < \ln(x_1 + A)/\ln \beta + B
\]

for some \( A \) and \( B \). Numerical experience and [23] suggest a logarithmic form to the optimal switching curve. Thus, the approximation architecture has the potential to produce realistic switching curves; see Section 6.5.

(iii) **Piece-wise quadratic on cones**

As in (4) for the fluid model, let \( S_1, \ldots, S_\kappa \) be a partition of \( R^n_+ \) into convex polyhedral cones of full dimension, emanating from the origin. We approximate \( h \) as quadratic on each of these regions:

\[
h(x) = \frac{1}{2}x'Q^k x + p'^k x + f^k, \ x \in S_k. \quad (18)
\]

Note that we do not require \( h \) to be continuous, so the assignment of boundaries to the \( S_k \) can affect \( h \). Constraint reduction for (18) is addressed in Section 5.3.
The only information being used from the fluid model is the quadratic regions. For some examples the fluid policy is greedy, in which case the control depends only on which \( x_i \) are nonzero. A greedy fluid policy gives less information about the optimal policy \([35]\) and tends not to perform as well \([36]\). Also, in this case we expect \( V \) to be quadratic instead of piece-wise quadratic. Thus, the approximation (18) is of interest when the fluid policy is not greedy.

The regions \( S_k \) are found in Section 5.3 for an example with three classes. For larger problems, finding the fluid policy becomes intractable. An alternative is to analyze the two-station fluid workload relaxation in \([22]\), for which an optimal policy can easily be found. A “greedy, workload constrained” translation of this policy from the workload space to the original state space is also given in \([22]\). It appears that the quadratic regions for this policy could be determined algorithmically by working backward from the origin and determining all sequences of control switching sets that can be visited, though we have not done so. One could also use (18) with the regions \( S_k \) defined by some method other than the fluid model.

(iv) Rational functions

To generate a larger set of basis functions, higher-order terms e.g., \( x_1 x_2 x_3 \) or \( x_1^2 x_2^2 \), would be a natural choice. However, the functions need to be bounded by a quadratic, so we divide by a suitable polynomial of degree two less than the numerator. Also, instead of using different powers, i.e., 1, \( x_i \), \( x_i^2 \), we use \( f_{i,1}(x_i) = x_i^2 \), \( f_{i,2}(x_i) = x_i(N_i - x_i)^+ \), \( f_{i,3}(x_i) = [(N_i - x_i)^+]^2 \), where \( x^+ = \max\{x, 0\} \). Note that \( f_{i,2} \) and \( f_{i,3} \) are zero beyond \( N_i \), emphasizing the interval \( 0 \leq x_i \leq N_i \) and increasing the sparsity of the constraint matrix. The numerical tests in Section 6 use \( N_i = 6/(1 - \rho_{\sigma(i)}) \). The three choices for \( f_{i,j} \) lead to \( 3^n \) rational functions

\[
\phi(x) = \frac{\prod_{i=1}^n f_{i,j(i)}(x_i)}{(1 + \sum_{i=1}^n x_i/N_i)^{2(n-1)}}, \quad j(i) = 1, 2, 3. \tag{19}
\]

(v) Piece-wise quadratic on rectangular regions

A common function approximation is to use a separate polynomial on different regions, as done in spline functions or local polynomials. However, a uniform grid is not appropriate here because of the importance of states with small queue lengths. Instead, we define rectangular regions that collect states into sets that grow exponentially as total queue length grows. Let

\[
S(y) = \{x : 2^{y_i - 1} - \frac{1}{2} \leq x_i < 2^{y_i}, i = 1, \ldots, n\} \tag{20}
\]

for all \( y \in \mathbb{Z}_+^n \) such that \( z \neq 0 \) and \( 0 \leq y_i < M \) for some \( M \). These \( n^M - 1 \) sets cover the states (excluding the origin) with \( x_i < 2^{M-1} \), dividing each \( x_i \) into the sets \( \{0\}, \{1\}, \{2, 3\}, \{4, 5, 6, 7\}, \) etc. The piece-wise
quadradic approximation on these sets is

\[ h(x) = \frac{1}{2} x^T Q^y x + p^y x + f^y, \quad x \in S(y). \]  

(21)

The number of basis functions, or variables, in (21) is roughly \( n^M n^2 / 2 \), which limits its use. However, they allow some constraint reduction as in [26]. Also, not all of these variables are needed. For example, if \( y_i = 0 \) or 1, then we can set \( q_{ij}^y = p_i^y = 0 \) because there is only one value of \( x_i \) in \( S(y) \). The numerical results in Section 6 also modify (20) by removing the upper bound when \( y_i = M - 1 \), so that the sets cover all of \( Z_n^+ \). With this modification, \( M = 2 \) corresponds to the functions in [26].

We also consider the piece-wise linear approximation on these regions. An even smaller set of functions is the piece-wise constant approximation \( h(x) = f^y, \ x \in S(y) \), which can be thought of as state aggregation.

**vi) Functions of one or two variables**

Consider a differential cost approximation of the form

\[ h(x) = \sum_{i=1}^n h_i(x_i) \]  

(22)

using functions of one variable. To parameterize \( h_i \), use the indicator functions \( \phi_j(x_i) = 1_{\{x_i = j\}} \) and let

\[ h_i(x_i) = \sum_{j=0}^{M-1} r_{ij} \phi_j(x_i) + r_{i0} \phi_0(x_i). \]  

(23)

Note that \( h_i(x) \) is arbitrary for \( x < M \) and has the form \( \phi_0 \) for \( x \geq M \). This parameterization is used in [25] with \( \phi_0(x_i) = x_i \ln x_i \), which we include in our tests. The number of functions in (22) is linear in the number of classes, so that it is suitable for large problems. We use an ad hoc modification of (22) to improve accuracy: replace the indicator functions \( \phi_j \) with

\[ \phi_j(x) = 1_{\{x_i = j\}} \left( 1 + \gamma |x_{(i)}|^2 \right) B |x_{(i)}| \]  

(24)

where \( |x_{(i)}| = \sum_{k \neq i} x_k \) and \( \gamma \geq 0, B \leq 1 \) are parameters to be chosen. Setting \( \gamma = 0 \) and \( B = 1 \) recovers the indicator functions. Because (24) depends on all the state variables, it does not allow constraint reduction.

Our numerical tests also use functions of a single workload, proposed in [25]. Define the workload \( w_j(t) \) of station \( j \) as the expected remaining time that it will spend serving jobs currently in the system. The mapping from queue length to workload is \( w(t) = CM^{-1}(I - P^r)^{-1}x(t) \), where \( M = \text{diag}(\mu_i) \) and \( C \) is the
constituency matrix from Section 2. Because \( w \) is not integer, a different parameterization is needed. We use a piece-wise constant \( h_j(w_j) \), with \( M \) intervals of width \( \delta \) plus \( \phi_0(w_j) = w_j \ln w_j \); \( \delta \) is set experimentally.

Finally, we consider functions of two variables. For a series or reentrant line, we use the pair of variables \( x_i \) and \( \sum_{j=i}^n x_j \), known as the echelon inventory. These two variables are commonly used in heuristic control policies in manufacturing [11]. Constraint reduction for these functions is presented in Section 5.4.

5 Constraint reduction

For certain approximating functions, the constraints of the resulting ALP can be algebraically reduced to a small, or at least more easily approximated, set. Section 5.1 addresses quadratic functions, Section 5.2 addresses exponential functions, Section 5.3 addresses piece-wise quadratics on cones, and Section 5.4 addresses functions of one or two variables using ideas from factored MDPs.

5.1 Quadratic functions

The constraints (ALP) can be reduced to a finite set for quadratic \( h \) [26, Appendix A]. To simplify notation, consider only deterministic routing. First, we write the constraints as

\[
J \leq \sum_{i=1}^n (c_ix_i + \alpha_i[h(x + e_i) - h(x)] + u_i\mu_i[h(x - e_i + e_{s(i)}) - h(x)]) \quad \text{for all } x \in \mathbb{Z}_n^+, \; u \in \mathcal{A}(x). \tag{25}
\]

Unlike a discounted model, only differences in \( h \) appear in these constraints, simplifying the analysis. It is convenient to use the substitution \( x = z + u \), so that a control \( u \) is feasible for all \( z \in \mathbb{Z}_n^+ \). Substituting (12) into (25) yields

\[
J \leq d^u + (c^u)'z \quad \text{for all } z \in \mathbb{Z}_n^+, \; u \in \mathcal{A} \tag{26}
\]

where

\[
c_i^u = c_i + \sum_{j=1}^n [\alpha_j q_{ij} + u_j \mu_j (q_{i,j} - q_{ij})]
\]

\[
d^u = \sum_{i=1}^n [u_i(c_i^u + \mu_i(\frac{1}{2} q_{ii} + \frac{1}{2} q_{s(i),s(i)} - q_{i,s(i)} + p_{s(i)} - p_i)) + \alpha_i(\frac{1}{2} q_{ii} + p_i)]
\]

17
and \( c^u = [c^u_i] \). For (26) to hold for all \( z \), the right hand side must be nondecreasing in \( z_i \). Hence, (26) is equivalent to

\[
J \leq d^u \quad \text{for all } u \in \mathcal{A} \tag{27}
\]

\[
c^u_i \geq 0 \quad \text{for } i = 1, \ldots, n, \ u \in \mathcal{A}. \tag{28}
\]

If the optimal policy is nonidling, then for a given control \( u \), (28) is only needed for \( i \) in

\[
N(u) = \left\{ i : \sum_{j: \sigma(i) = \sigma(j)} u_j = 1 \right\}
\]

i.e., the classes served by busy stations. Under nonidling there are only \(|N(u)| + 1\) constraints for each \( u \) instead of \( n + 1 \).

The number of constraints in (27) and (28) is typically exponential in \( n \) because of the number of actions \(|\mathcal{A}|\). Nevertheless, the reduced ALP can be solved for fairly large \( n \).

For quadratic \( h \), Bellman error has the form

\[
B(x) = \min_{u \in \mathcal{A}(x)} d^u + (c^u)'(x - u) - J
\]

which is the minimum of linear functions.

### 5.2 Exponential functions

Constraint reduction for quadratic functions relied on the fact that the constraints are linear in \( x \). Depending on which exponential functions are used, constraints will be linear in some \( x_i \) and uni-min in others, allowing some constraint reduction and efficient truncation. Assume that each class has at most one predecessor, i.e., routing is deterministic and routes do not merge, and that there are no arrivals at classes that have a predecessor class. If class \( i \) has no predecessor, set \( p(i) = 0 \). Consider the approximation using quadratics and (14)

\[
h(x) = \frac{1}{2} x^T Q x + px + \sum_{i: p(i) \neq 0} r_i \beta_i x_i.
\]

Also assume \( \beta_i = \mu_i / \mu_{p(i)} < 1 \) so that (16) holds.

The constraints (25) are:
\[ \Lambda J \leq d^u + e^u z + \sum_{i:p(i) \neq 0} r_i (u_{p(i)} - u_i) (\mu_i - \mu_{p(i)}) \beta_i^{z_i + u_i} \quad \text{for all } z \in \mathbb{Z}_+^n, \ u \in \mathcal{A} \]  

(29)

where \( d^u, e^u, \) and \( z = x - u \) are from Section 5.1. Consider (29) for a given action \( u \) and let \( \mathcal{C}(u) \) be the set of classes \( i \) for which there is no \( \beta_i^{z_i} \) term. For (29) to hold for all \( z \), we must have \( c_i^u \geq 0 \) for \( i \in \mathcal{C}(u) \). Given these constraints, (29) is only needed at \( z_i = 0, \ i \in \mathcal{C}(u) \), where it is tightest. Hence, the constraints are equivalent to \( c_i^u \geq 0 \) for \( i \in \mathcal{C}(u) \) and (29) for all \( z \in \mathbb{Z}_+^n \) such that \( z_i = 0, \ i \in \mathcal{C}(u) \).

This set of constraints is still infinite, but in fewer dimensions. The reduction in dimension, \( \min_{u \in \mathcal{A}} |\mathcal{C}(u)| \), is at least the number of classes with no predecessor and usually is larger if nonidling is assumed. We will approximate (ALP) using the constraints \( c_i^u \geq 0 \) for \( i \in \mathcal{C}(u) \) and (29) at \( z_i = 0, \ i \in \mathcal{C}(u) \) and \( z_i = 0, \ldots, N - 1 \) for \( i \notin \mathcal{C}(u) \) for some \( N \). Call this relaxation ALP(N). The total number of constraints is \( \sum_{u \in \mathcal{A}} (|\mathcal{C}(u)| + N^{n-|\mathcal{C}(u)|}) \). In our experiments a very small truncation often suffices—the solution to ALP(N) is the same for all \( N \geq M \) for some small \( M \). This behavior appears to be due to the fact that if the sign of \( r_i \) is such that the coefficient of the \( \beta_i^{z_i} \) term is positive, then (29) is uni-min in \( z_i \) and the minimum typically occurs at small \( z_i \).

Constraint reduction is also possible when some of the functions (15) are used. For example, consider the series queue in Figure 2 with \( \mu_2 < \mu_1 \) and the approximation

\[ h(x) = \frac{1}{2} x' Q x + px + r_1 x_1 \beta_2 x + r_2 x_2 \beta_2 \beta_2 + r_3 x_3 \beta_2 \beta_2 \]  

(30)

i.e., the functions (14) and (15) with \( i = 2 \) are added to the quadratic. Set \( \beta_2 = (\mu_2/\mu_1) \). Appendix B shows that the constraints for this problem reduce to a one-dimensional set and that ALP(N) contains \( 2N + 8 \) constraints. For the three-class reentrant line in Figure 3, the analogous \( h \) approximation includes (14) and (15) for \( i = 2, 3 \). The constraint set reduces to a two-dimensional set (one-dimensional if only the functions (14) are used).

5.3 Piece-wise quadratic on cones

This section addresses constraint reduction for the piece-wise quadratic functions (18). The method is applied to a three-class reentrant line. A dual method similar to [27, Section 3.4] will be used to reduce the constraints to a finite set; however, certain approximations are needed to make the constraints tractable. For each action, this approach defines sets of states in which the form of the constraints (25) is constant. Given
the action, the transition probabilities are translation invariant. Let \( \{ \delta_l \}, l = 1, \ldots, q \) be the transitions, i.e., \( p_u(x, x + \delta_l) > 0 \) for some \( u \) and all \( x \) such that \( u \in \mathcal{A}(x) \). Let \( \psi = (u, k_0, k_1, \ldots, k_q) \) and define

\[
X^\psi = \{ x \in S_{k_0} \cap Z^n_+ : x + \delta_l \in S_{k_l} \text{ for all } l \text{ such that } p_u(x, x + \delta_l) > 0 \}.
\]

There is one index \( \psi \) for each combination of action \( u \), quadratic region \( S_{k_0} \) of the current state, and quadratic region \( S_{k_l} \) of possible next states. If transition \( \delta_l \) does not occur under \( u \), then \( k_l = k_0 \).

Again using \( x = z + u \), let \( Z^\psi = \{ z : z + u \in X^\psi \} \). The constraints have the form

\[
J \leq d^\psi + c^\psi z + \frac{1}{2} z' M^\psi z, \quad z \in Z^\psi
\]

(31)

where \( d^\psi \), \( c^\psi \), and \( M^\psi \) are linear functions of \( Q^k \), \( p^k \), and \( f^k \). The quadratic term \( M^\psi \) is symmetric. It appears because of transitions between regions \( S_k \). The first approximation is to remove the integer restriction by allowing \( z \in \overline{Z}^\psi \), where \( \overline{Z}^\psi \) is a polyhedron, say \( \{ z \in R^n : A^\psi z \geq b^\psi, z \geq 0 \} \), whose lattice points are (nearly) the set \( Z^\psi \). For simplicity, we allow lattice points on the boundary of \( \overline{Z}^\psi \) that are not in \( Z^\psi \). This overlap could be avoided by adding more cutting planes. Also, because \( S_k \) has full dimension and the action \( u \) is feasible at all \( z \geq 0 \), there is no need for equality constraints in \( \overline{Z}^\psi \). If nonidling controls are desired, constraints of the form \( z_i = 0 \) can be enforced by removing these variables.

Checking (31) exactly for a given \( d^\psi \), \( c^\psi \), and \( M^\psi \) is related to determining if \( M^\psi \) is copositive; instead, following [27], we impose the stronger, simpler conditions

\[
M^\psi \geq 0
\]

(32)

and

\[
J \leq d^\psi + c^\psi z
\]

\[
A^\psi z \geq b^\psi
\]

(33)

\[
z \geq 0.
\]

The key observation is that these constraints are colinear in \( z \) and the ALP variables. A dual can be
constructed that separates $z$. Fixed values of $J$, $Q^k$, $p^k$, and $f^k$ satisfy (33) if and only if, for each $\psi$, the LP

$$\begin{align*}
\min & \quad c^{\psi'}z \\
\text{s.t.} & \quad A^{\psi}z \geq b^{\psi} \\
& \quad z \geq 0
\end{align*}$$

has optimal value $w^{\psi} \geq J - d^{\psi}$, or equivalently, so does its dual

$$\begin{align*}
\max & \quad b^{\psi'}y^{\psi} \\
\text{s.t.} & \quad A^{\psi'}y^{\psi} \leq c^{\psi} \\
& \quad y^{\psi} \geq 0
\end{align*}$$

Thus, (34) and $w^{\psi} \geq J - d^{\psi}$ for all $\psi$ are equivalent to (33). Reintroducing $J$, $Q^k$, $p^k$, and $f^k$ as variables, the dual form of (ALP) is

$$(\text{ALPD}) \quad \max \quad J \\
\text{s.t.} \quad A^{\psi'}y^{\psi} \leq c^{\psi} \\
\quad b^{\psi'}y^{\psi} \geq J - d^{\psi} \\
\quad M^{\psi} \geq 0 \\
\quad y^{\psi} \geq 0.$$
the origin, the ones bounding $\mathbf{Z}^\psi$ pass within roughly one transition of the origin (there are points in $\mathbf{Z}^\psi$ within one transition of the $S_k$ boundary). Thus, in a certain sense, the extreme points of $\mathbf{Z}^\psi$ lie near the origin. Also, finding the extreme directions is made easier by the fact that the extreme directions of $\mathbf{Z}^\psi$ are a subset of the extreme directions of $S_{k_0}$. In particular, $\mathbf{Z}^\psi$ has the ones contained in the common boundary of $S_{k_0}$ and all $S_{k_i}$ (because there are transitions into $S_{k_i}$ from $\mathbf{Z}^\psi$). Checking extreme directions in the linear constraints (33) is an exact method; however, we will apply it as an approximate check of (31). Requiring (31) at $z = t\beta$ for all $t \geq 0$ results in quadratic constraints. For simplicity, we use the stronger conditions $c^\psi \beta \geq 0$ and $\beta' M^\psi \beta \geq 0$.

These observations suggest the following approximation to (31). Find the extreme directions $\{\beta^\psi,l\}$ of $\mathbf{Z}^\psi$. The relaxation ALP(N) contains the constraints

\begin{align}
(31) \text{ for } z \in \mathbf{Z}^\psi \text{ and } z_i \leq N - 1 \\
c^\psi \beta^\psi,l \geq 0 \\
(\beta^\psi,l)' M^\psi \beta^\psi,l \geq 0 \tag{35} \tag{36}
\end{align}

for all $\psi$ and all directions $\beta^\psi,l$. The constraints (31) address the extreme points, while the limiting constraints (35) and (36) in the extreme directions allow faster convergence over $N$. ALP(N) has two limiting constraints for every extreme direction $\beta^\psi,l$, which is generally exponential in $n$, plus $N^n$ constraints (31). However, it avoids the dual variables $y^\psi$, making it potentially more tractable than (ALPD).

Notice that ALP(N) is based on the exact constraints, not the linearization (33), suggesting that ALP(N) might give a tighter bound than (ALPD). However, because of the approximate treatment of limiting constraints ALP(N) may not converge to (ALP).

To illustrate these definitions, consider the three-class, two-station reentrant line in [5] and [39] (Figure 3). Jobs arrive at rate $\alpha$ to class 1. Station 2 serves only class 2 and is the bottleneck, $m_2 > m_1 + m_3$, where $m_i = 1/\mu_i$ is the mean service time for class $i$. Costs are constant, $c_i = 1$, so the only decision is whether to serve class 1 or 3 at station 1. As [39] shows, when $x_2 = 0$ the fluid policy makes a trade-off between serving class 3, which starves the bottleneck, and serving class 1, feeding the bottleneck. Class 3 is given priority when $x_3 \leq \gamma x_1$, where

$$
\gamma = \frac{1}{1 - \alpha/\mu_2} \left( \frac{m_2 - m_1 - m_3}{m_1 + m_3} \right).
$$

When $x > 0$ class 3 is served.
Switching set visited next

Quadratic regions

\( x > 0 \)

State

Station 1 serves \( \hat{q} \)

<table>
<thead>
<tr>
<th>Quadratic region ((x &gt; 0))</th>
<th>State</th>
<th>Station 1 serves</th>
<th>( \hat{q} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_1 : x_3 &gt; \gamma x_1 + \frac{\alpha + \mu_3 - \mu_2}{\mu_2} x_2 )</td>
<td>( x_2 = 0, x_3 &gt; \gamma x_1 )</td>
<td>3</td>
<td>((\alpha, 0, \mu_2 - \mu_3))</td>
</tr>
<tr>
<td>( S_2 : x_3 \leq \gamma x_1 + \frac{\alpha + \mu_3 - \mu_2}{\mu_2} x_2 )</td>
<td>( x_2 = 0, x_3 \leq \gamma x_1 )</td>
<td>1 and 3</td>
<td>((\alpha - \mu_2, 0, \mu_2 - \mu_3(1 - \frac{\mu_2}{\mu_1})))</td>
</tr>
<tr>
<td>and ( x_3 &gt; \frac{\mu_3 - \mu_2}{\mu_2} x_2 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( S_3 : x_3 \leq \frac{\mu_3 - \mu_2}{\mu_2} x_2 )</td>
<td>( x_2 &gt; 0, x_3 = 0 )</td>
<td>3 and idle</td>
<td>((\alpha, -\mu_2, 0))</td>
</tr>
</tbody>
</table>

Table 1: Quadratic regions of the fluid cost in the reentrant line example

<table>
<thead>
<tr>
<th>Region</th>
<th>Extreme directions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_1 )</td>
<td>((0, 0, 1), (0, \mu_2, \alpha \gamma + \mu_3 - \mu_2), (1, 0, \gamma))</td>
</tr>
<tr>
<td>( S_2 )</td>
<td>((1, 0, 0), (0, \mu_2, \alpha \gamma + \mu_3 - \mu_2), (1, 0, \gamma)), ((0, \mu_2, \mu_3 - \mu_2))</td>
</tr>
<tr>
<td>( S_3 )</td>
<td>((1, 0, 0), (0, 1, 0), (0, \mu_2, \alpha \gamma + \mu_3 - \mu_2))</td>
</tr>
</tbody>
</table>

Table 2: Edges of the quadratic regions in the reentrant line example

Although the control is constant on \( x > 0 \), \( V \) has \( \kappa = 3 \) quadratic regions, depending on which of three actions will be used next on a trajectory starting from \( x \). The correspondence between quadratic regions and switching sets is shown in Table 1. One can verify that these are the three quadratic regions by following the trajectories. Trajectories in the first switching set in Table 1 enter the second switching set next; the second and third region feed into the switching set \( x_2 = 0 \) and \( x_3 = 0 \), which leads to \( x = 0 \). Note that this optimal policy idles station 1 in the third switching set, but this is just for convenience; a nonidling optimal policy also exists. The assignment of boundaries to \( S_k \) in Table 1 was chosen for consistency with the control regions. The extreme directions of the quadratic regions are listed in Table 2.

To illustrate the definition of \( X^\psi \), number the service transitions \( l = 1, 2, 3 \) and the arrival transition \( l = 4 \). Consider, for example, \( u = (0, 1, 1) \) and \( k_0 = 3 \), i.e., \( x \in S_3 \) (see Table 1). Then \( k_1 = 3 \) because class 1 is not served and \( k_3 = k_4 = 3 \) because these transitions cannot leave \( S_3 \). However, a class 2 service completion could stay in \( S_3 \) (\( k_2 = 3 \)), enter \( S_2 \) (\( k_2 = 2 \)), or, for certain parameter values, enter \( S_1 \) (\( k_2 = 1 \)). Specifically, if \( \mu_3 \geq 2\mu_2 \) and \( x = (0, 1, 1) \) then \( x \in S_3 \) but \( x + \delta_2 = (0, 0, 2) \in S_1 \), i.e., \( x \in X^\psi \) where \( \psi = (u, 3, 3, 1, 3, 3) \). Because \( S_1 \) and \( S_3 \) only meet at the origin, \( X^\psi \) can contain only states near the origin. In general, if \( \psi \) contains \( k_l \neq k_0 \) then \( X^\psi \) lies within one transition of the hyperplane separating \( S_{k_0} \) and \( S_{k_l} \).
5.4 Functions of one or two variables

When the $h$ approximation contains only functions of one variable (22), the constraints in (ALP) with the truncation $x_i < N$ have the form

$$\sum_{i=1}^{n} f_i(x_i, u(i)) \geq 0 \text{ for all } 0 \leq x_i < N, \ u \in A(x)$$

(37)

where $u(i)$ is a vector containing some of the components of $u$. For example, if each class has at most one predecessor, (25) implies that $u(i)$ contains $u_i$ and, if class $i$ has a predecessor, $u_{p(i)}$. Introduce the variables $f_{i+n}(u(i))$, representing the minimum of $f_i(x_i, u(i))$ over $x_i$. Then (37) is equivalent to

$$\sum_{i=1}^{n} f_{i+n}(u(i)) \geq 0 \text{ for all } u \in A$$

(38)

$$f_{i+n}(u(i)) \leq f_i(x_i, u(i)) \text{ for all } u_i \leq x_i < N, \ u(i) \text{ feasible for } i = 1, \ldots, n.$$  

(39)

Constraint (39) is needed for those $u(i)$ appearing in (38), namely, those $u(i)$ that can be augmented to give a feasible action $u \in A$. The constraint reduction is considerable. For a series line, $u_{(1)}$ and $u_{(n)}$ have two feasible values and the other $u(i)$ have four feasible values (two components), giving less than $(4n - 4)N$ constraints (39) and $2^{n-1}$ constraints (38). For comparison, (37) contains nearly $2^{n-1}nN$ constraints.

Further reduction is achieved by eliminating certain $u_i$. Note that (39) eliminates $x_i$ because it only appears in $f_i$. If a variable $u_j$ appears in $u(i)$ but does not appear with any other variables $x_k$, $k \neq i$, then $u_j$ can be eliminated in the same manner as $x_i$. Indeed, this elimination can be done sequentially, replacing $f_i$ one at a time by $f_{i+n}$ and considering only the remaining variables. For example, the constraints

$$f_1(x_1, u_1) + f_2(x_2, u_1, u_2) + f_3(x_3, u_2, u_3) \geq 0, \ u_i \leq x_i < N, \ u_i = 0, 1$$

are equivalent to

$$f_0(u_2) \leq f_3(x_3, u_2, u_3) \text{ for all } x_3, \ u_2, \ u_3$$

(40)

and

$$f_1(x_1, u_1) + f_2(x_2, u_1, u_2) + f_0(u_2) \geq 0 \text{ for all } x_1, \ x_2, \ u_1, \ u_2.$$  

Now that $u_2$ does not appear with $x_3$, eliminate $x_2$ and $u_2$, giving the equivalent constraints (40),

$$f_5(u_1) \leq f_2(x_2, u_1, u_2) + f_0(u_2) \text{ for all } x_2, \ u_1, \ u_2$$

24
There are $6N-3$ reduced constraints.

Note that the feasible $u$ in this example are for a single-class network. If the network is multiclass, a modified approach is needed in order to determine the feasible actions in constraints such as (40). At some point in the elimination process, suppose $u_j$ appears in $u_{(i)}$ but does not appear with any other variables $x_k$, $k \neq i$. Before eliminating $u_j$, augment $u_{(i)}$ with all other classes $k$ that use the same server, $\sigma(k) = \sigma(j)$, and appear in the constraint after $f_i$ is eliminated.

This method can be extended to functions of two or more variables using ideas from factored MDPs [16], [31], [12], [13]. An MDP on a state space $X$ is factored if

$$X = X_1 \times X_2 \times \ldots \times X_s$$

$$p_u(x, y) = \prod_{j=1}^s p_{u^{(j)}}(x^{(j)}, y^{(j)})$$

and cost is additive over $x^{(j)}$. Here $X_j$ contains a subset of “local” state variables of $X$ and $x^{(j)}$, $u^{(j)}$ are vectors containing some of the components of $x$ and $u$. The factored assumption allows a more compact representation of the MDP, but it is no easier to solve because the differential cost needed in (1) does not factor. Turning to ALP, if each $\phi_k$ is a function of some $x^{(j)}$, then $h$ is also “factored” and the constraints in (ALP) can be written as a sum of functions of the factored state variables and actions:

$$\sum_{j=1}^s f_j(x^{(j)}, u^{(j)}) \geq 0 \text{ for all } x \in X, u \in A(x). \quad (41)$$

Returning to the MQNET problem, no factored representation of the transition probabilities appears possible, particularly after uniformization. However, computing the expectation $(P_u h)(x)$ is not difficult and it suffices that the cost $c'x$ and the $h$ approximation “factor.” Using the basic idea from the example above, one can sequentially eliminate $f_j$’s and their unique variables. For brevity, we only describe functions of overlapping pairs of variables, $x^{(j)} = (x_j, x_{j+1})$, $j = 1, \ldots, n-1$. Eliminating just the $x$ variables, (41) is equivalent to

$$\sum_{i=1}^{n-1} f_{i+n}(u_i, u_{i+1}) \geq 0 \text{ for all } u \in A \quad (42)$$

$$f_{i+n}(u_{(i)}) \leq f_i(x_i, x_{i+1}, u_{(i)}) \quad (43)$$
for all \( u_i \leq x_i < N, u_{i+1} \leq x_{i+1} < N \), and \( u_{(i)} \) feasible for \( i = 1, \ldots, n-1 \). As described above for functions of one variable, certain \( u_j \) can also be eliminated after augmenting \( u_{(i)} \) with other classes that use the same servers. There are \( O(N^2) \) constraints in (43), with the dependence on \( n \) determined by the problem structure, and \( |A| < 2^n \) constraints in (42).

6 Numerical Results

Average cost error for various ALPs was computed by solving the ALP and finding optimal average cost using DP value iteration on a truncated state space. The ALPs were solved using QNET Approximator, available at www.math-cs.gordon.edu/qna, and CPLEX. Most of the DPs were solved using the program at www.math-cs.gordon.edu/~senning/qnetdp. The networks used are described in Section 6.1. Constraint sampling and truncation methods are presented and tested in Section 6.2. Results for the smaller approximation architectures are given in Section 6.3, including size and solution time of the ALP for some larger networks that cannot be solved by DP. Section 6.4 shows the accuracy achieved by larger approximation architectures. The performance of the ALP policy is computed for a few cases in Section 6.5.

6.1 Examples

A variety of examples from the literature were used. Their parameter values are shown in Table 3. Series queues with two classes (Figure 2) and four classes are listed. Longer series queues are also used; like the four-class example, \( \mu = (1 + 0.1(n - 1), \ldots, 1) \) and \( c = (1, \ldots, 2) \), with the \( \mu_i \) and \( c_i \) equally spaced. In the parallel server “N” network from [15], there are two classes and two servers. Server 1 can only serve class 1. Server 2 can serve class 1 or class 2. In the arrival routing problem from [14], arrivals must be immediately routed to one of two classes, each with its own server. The previous two examples do not fit the formulation of Section 2, but the required modifications are easily made. The reentrant line of Figure 3 is studied in [5] and [39]. The Rybko-Stoylar network, shown in Figure 4, is considered a challenging example because some static priority policies are not stabilizing. The six-class, two-station network of Figure 5 is a modification of [38] studied in [29], where DP results are given.

An 11-class, four-station network with reentrant flow and rework, typical in manufacturing applications, is taken from [24, Figure 7.1 and Section 7.2.1] and adapted to our problem setting. Station 1 serves classes 1 to 4, station 2 serves 5 to 9, station 3 serves 10, and station 4 serves 11. Station 5 in [24], which serves one class and has low traffic intensity, is omitted. Job type 1 follows a route through classes 1, 5, 9, 4. Job type
2 is routed 2, 6, 10, 11 or 8, 3, 7. The routing from class 10 is probabilistic, with $p_{10,8} = 1 - p_{10,11} = 3/19$. There is also probabilistic routing from class 7 to class 10, with $p_{7,10} = 0.2$ representing rework. The other parameters are $\alpha_1 = \alpha_2 = 19$,

$$ \mu = (65, 130, 65, 130, 75, 150, 75, 150, 75, 25, 37.5) $$

$c = 1$, and $\rho = (0.95, 0.975, 0.95, 0.533)$.
6.2 Constraint sampling

The results in the following sections only use constraint reduction for quadratic approximation. For other approximations, constraints on a truncated state space or constraint sampling is used, combined with the “limiting constraints” introduced in this section. Truncation is used for examples with two or three classes; for the larger examples, constraint sampling is used. The truncations were set by checking the sensitivity of average cost to the truncation in an effort to keep truncation error less than 1 to 2% and similarly for sample size. Sensitivity was checked for each network and many approximation architectures; however, once truncations or sample size were set, they were used for similar approximation architectures. Truncations for the DP were also set by checking sensitivity, with an accuracy goal of 0.1%. For the larger examples, distinct truncations were set for each variable $x_i$.

In constraint sampling, states are sampled according to the probability distribution

$$\pi(x) = \prod_{i=1}^{n} (1 - \tilde{\rho}_i) \bar{\rho}_i^{x_i}.$$  

A natural choice for the parameter $\tilde{\rho}_i$ is the traffic intensity of station $\sigma(i)$. The value 0.5 was also tested.
However, because it may be difficult to generate enough unique states, \(1 - \Phi_i\) is repeatedly reduced by a constant factor during sampling to spread the distribution. When a state \(x\) is sampled, constraints for all actions \(u \in A(x)\) are generated. Also, constraints for the \(2^n\) states with \(x_i = 0\) or \(1\) are generated before sampling because of their potential importance. Note that both of these procedures can generate exponentially large constraint sets, since the number of actions is also usually exponential in \(n\). We omit the \(x_i = 0\) or \(1\) constraints for the 11-class example.

Table 4 shows that even for the two-class series queue, the sample size required for a small average cost error due to sampling depends significantly on the approximation architecture. Exponential functions (14) and (15) with \(\beta = (.9, .8333)\), the rational functions (19) with \(N_i = 12\), and functions of one queue length (23) with \(M = 10\) and no \(\phi_0\) are used. The sampling distribution was initialized with \(\Phi_i = 0.5\). There is no sampling error with quadratic approximation; the \(x_i = 0\) or \(1\) constraints and a few randomly sampled constraints are sufficient.

Table 5 shows sampling error for two larger examples. For both examples \(\beta_i = 0.9\) and \(N_i = 12\). For the four-class series queue, \(M = 10\) and \(\Phi_i = 0.5\). For the eight-class series queue, the data are \(\alpha = 1\), \(\mu = (1.75, \ldots, 1.25)\), and \(c = (1, \ldots, 2)\), with the \(\mu_i\) and \(c_i\) equally spaced, and the parameters are \(M = 5\) and \(\Phi_i = \alpha/\mu_i\). Again the sample size required is highly dependent on the approximation architecture. Exponential functions require two million constraints for a sampling error of less than 1% in the four-class series queue. Actual errors are somewhat larger than reported for the non-quadratic cases because error is calculated in comparison to the best available run (the smallest average cost). For example, the four-class quadratic plus exponential row is in comparison to the minimum of the 10 runs made with two million constraints. Although only the mean percent error is reported in Tables 4 and 5, the coefficient of variation of the error is small, generally less than 0.2, so that the mean of five trials is a reasonable measure.

To reduce sampling error, we used a hybrid sampling and truncation method. The rationale for this method is that constraints tend to vary slowly and consistently with \(x\) for a given \(u\). Sampling and truncation both tend to miss the effect of constraints at large \(x\). The limiting constraints results in Table 5 add constraints at the \(2^n - 1\) states \(x \neq u\) such that \(x_i = u_i\) or \(N\). The idea is that for large \(N\), these constraints approximate a limiting constraint in various directions in the state space. To avoid numerical issues, \(N = 500\) was used (10,000 for quadratic functions and eight-class with quadratic plus one variable). The limiting constraints were not included in the number of constraints reported, but their number is generally negligible. Table 5 shows that limiting constraints can reduce the sample size required by an order of magnitude or more. The limiting constraints virtually eliminate sampling error for the quadratic approximation because...
### Table 4: Accuracy of constraint sampling for the series queue ($\mu_1 > \mu_2$). Reported error is the mean of five runs compared to the ALP without sampling.

<table>
<thead>
<tr>
<th>Constraints</th>
<th>200</th>
<th>500</th>
<th>1,000</th>
<th>2,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h$ approximation</td>
<td>Sampling error, %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quadratic</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Quad+exp</td>
<td>10.8</td>
<td>3.4</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>Quad+rat'1</td>
<td>2.5</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quad+exp+rat'1</td>
<td>1.5</td>
<td>0.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quad+one var</td>
<td>7.6</td>
<td>3.0</td>
<td>1.4</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 5: Accuracy of constraint sampling with limiting constraints, series queue. Reported error is the mean of five runs compared to the most accurate run available. *Mean of two runs.

### 6.3 Quadratics and functions of one variable

The ALP with quadratic approximation (Section 4.1) is the smallest and has been solved on series lines of up to 17 classes; see Figures 6 and 7. Even larger multiclass examples can be solved because they have fewer actions. For comparison, the upper bound ALP using the policy found by (ALP) was also solved. Because $h$ is quadratic, the ALP policy has polyhedral control regions (i.e., linear switching surfaces). We used the constraint reduction method in [27]; however, the reduced upper bound ALP is much larger than the lower bound ALP. Parameters from Section 6.1 were used except $\alpha = 0.9$ for the lower bound and $\alpha = 0.5$ for the upper bound ALP. Run time is using CPLEX version 10 on a 64-bit, 3.4 GHz processor with 6 GB of RAM and includes generating and solving the LP. The lower bound ALP requires much more memory than time.

Accuracy of the average cost bound for quadratic approximation and for functions of one queue length is shown in Table 6. The quadratic approximation is not very accurate, particularly for the series queues. Figure 11 suggests that accuracy improves in light traffic or single-bottleneck heavy traffic. Adding functions of one queue length, (23) and (24), improves accuracy to 12 to 30%. For the 11-class network, optimal average cost is not available, but it appears that functions of one variable are not very effective. We used $M = 10$
Figure 6: Size of the ALPs with quadratic approximation, series queue.

Figure 7: Run time for the ALPs with quadratic approximation, series queue.
functions $\phi_j$ and $\phi_0(x_i) = x_i \ln x_i$ for each class and parameters $\gamma = .05$ and $B = .95$ in (24). Tuning $\gamma$ and $B$ for each example could improve accuracy. An LP size of $13 \times 6$ indicates 13 constraints and 5 basis functions plus the variable $J$. The large number of constraints in the ALPs with functions of one variable reflects the truncation or constraint sample size used. Limiting constraints with $N = 500$ were used in the last five examples. For the 11-class network, there were 200,000 random constraints and 246,000 limiting constraints; the $x_i = 0$ or 1 constraints were too numerous to include. This ALP, with 199 variables, was solved in 238 seconds (excluding generating the LP) using CPLEX version 12 on a 1.86 GHz dual-core processor with 6 GB of RAM.

### 6.4 Larger approximation architectures

Extensive tests were conducted using the additional approximating functions in Section 4 to investigate the trade-off between size and accuracy achievable using ALP. Results for three progressively larger examples are shown in Figures 8, 9, and 10. Of the many approximations tested, only those with good accuracy for their size are graphed. For comparison, a simple approximation architecture is shown that uses quadratic functions and indicator functions on individual states with total queue length up to $M$, i.e.,

$$
\phi_y(x) = 1_{\{x = y\}} \quad \text{for all } y \in Z^n_+ \text{ such that } \sum_{i=1}^n y_i \leq M
$$

(44)

for some $M$. Using these indicator functions is equivalent to solving (1) exactly on these states. The accuracy of using DP on a truncated state space, $x_i < N$ for some buffer limit $N$, is also shown. The DP has one variable for each state, so the horizontal axis compares the number of variables.

For these examples, the best ALPs shown achieve the same accuracy as the truncated DP with roughly
two orders of magnitude fewer variables. The ALP does not do quite as well in the six-class example, where the smaller traffic intensity of 0.6 makes the problem easier for DP. The ALPs have the greatest advantage when the accuracy requirement is modest, such as 10 or 20 percent error. The ALPs with indicator functions also require fewer variables than DP. Since both methods solve (1) exactly on a set of states with small queue lengths, the difference can be attributed to the quadratic terms in the ALP, which impose a form of penalty at the boundary of these states.

Table 7 shows the solution time (excluding generating the LP), again using CPLEX version 12 on a 1.86 GHz dual-core processor with 6 GB of RAM. In comparison, the DP for the four-class series line took 96 seconds for the smallest run with an error less than 15%, 352 seconds for an error less than 5%, and 1168 seconds for an error less than 1%. The DP code uses value iteration with a stopping tolerance of .001. It was run using a parallel implementation on a cluster of eight workstations; the reported time is clock time multiplied by eight. Generally, the ALP is much faster at low levels of accuracy but loses its advantage as the accuracy requirement increases. The main reason is that the LP solution time increases more rapidly with the number of variables than does DP. For the four and six-class examples, ALPs with higher accuracy could not be solved with the available memory without reducing the sample size (or truncations) to the point where sampling (or truncation) error might be considerably larger than the 1 to 2% goal.
Figure 9: Accuracy and size of ALPs, four-class series queue.

Figure 10: Accuracy and size of ALPs, six-class network.
Table 7 also lists the approximation architectures with the best size-accuracy trade-off, including one ALP for the 11-class network. Adding exponential functions and functions of one variable increased the average cost bound 6% compared to just quadratics for the 11-class network. The other approximating functions were not tried on this example because of memory limitations. The parameters listed are $\beta_i$, the base for the exponential functions (14) and (15); $N_i$, the maximum queue length for the rational functions (19); $M$, which controls the number of functions of one or two variables (20) or piece-wise quadratic and linear (23); and $\gamma, B$ for the functions of one or two variables (24). They were set by testing two or three values, not by exhaustive search, so more improvement is possible. These results suggest an order for adding functions from Section 4 to the approximation architecture:

- quadratics
- exponential functions
- rational functions (on smaller examples)
- functions of one or two variables
- piece-wise linear (PWL) or piece-wise quadratic (PWQ) functions on rectangular regions.

Quadratics are used first because the LP is the easiest to solve. The set of $3^n$ rational functions is too large to use with larger networks. The functions of one variable used in Table 7 are functions of queue length. For the six-class network, functions of workload also improved accuracy and are used in one run in Figure 10. The functions of two variables, or “pairs,” used in Table 7 are functions of queue length and echelon inventory (defined in Section 4). Thus, it may be beneficial to try queue length, workload, and echelon inventory. The set of piece-wise quadratic functions is too large to use on large examples, which is why the piece-wise linear functions were used in the four-class and six-class examples. Piece-wise quadratics on cones were not included in the tests.

6.5 Performance

Performance of the $h$-greedy policy for the ALP solution $h_1$, found using value iteration for this fixed policy, was tested for the two-class series queue with $\mu_1 > \mu_2$. The quadratic and exponential approximations (30), called EXP3, were used. In Figure 11, traffic intensity $\rho = \alpha/\mu_2$ was varied while keeping $\mu_1$ fixed. The data below the line at 1.0 shows the accuracy of the lower bound, i.e., $J/J_*$, and the data above 1.0 shows the performance, i.e., $J_u/J_*$. The percent error vanishes in light traffic. The data suggests that percent
error also vanishes in heavy traffic. This is not surprising, since there is a single bottleneck at station 2. It is shown in [37] that the quadratic ALP gives a tighter bound than the achievable region method, which is known to have a vanishing percent error in heavy traffic [4]. Performance of the ALP policy is within 10% of optimal except in heavy traffic. Performance of the quadratic approximation is not shown at traffic intensity above 0.75 because its policy is unstable. Surprisingly, the exponential approximation (30) sometimes has poorer performance than the quadratic, even though (30) includes quadratic functions.

The quadratic ALP policy for this example has a simple form: server 1 is busy when $x_2 \leq 1$ and $x_2 > 0$, and this policy happens to perform fairly well. As noted in Section 4, using just a quadratic and the function $\beta x_2$ has the potential to produce realistic switching curves. Here $\beta = \mu_2/\mu_1 = .8333$. The policy for this approximation at $\alpha = 1$ is shown in Figure 12. It has an average cost just 1% above optimal. However, such close fits do not appear to be the norm; using (30), which has two more exponential terms, has a nearly flat switching curve at $x_2 = 5$, which does not track the optimal switching curve as closely, and an average cost 9% above optimal.

The performance of both ALP policies is better, with error under 5%, for the series line with $\mu_1 < \mu_2$ and with $\mu_1 = \mu_2$. Traffic intensity affects error in a similar way when the piecewise quadratic approximation (18) is used for the reentrant line, again with a single bottleneck station. However, for the series line with $\mu_1 = \mu_2$ which has two bottlenecks, as traffic intensity increases, the accuracy of the ALP bound continues to degrade.
Figure 11: Performance of the ALP policy, two-station series line, $\mu_1 > \mu_2$.

Figure 12: Policy for the ALP with quadratic and $\beta^2$ functions, series line ($\mu_1 > \mu_2$). Station 1 idles below the switching curve.
7 Conclusion

We have demonstrated the feasibility of using ALP to compute lower bounds on optimal average cost for small to moderate size networks. The method requires only the solution of an LP. A policy can also be obtained which is useful for some networks. The ALPs have the following features:

- The simplest, quadratic ALP gives tighter bounds than the LP bounds in [4] and [18], has moderate accuracy, and is fairly tractable because of constraint reduction. The number of constraints is typically exponential in the number of buffers because of the exponential number of actions; however, the ALP was solved quickly for up to 17 buffers—much larger than can be solved exactly by DP.

- New constraint reduction techniques were developed that make ALPs with certain exponential functions, piece-wise quadratic functions, and functions of one or two variables more tractable.

- A hybrid constraint sampling and truncation method was developed that allows a smaller number of sampled constraints to be used. Using this method, ALPs for networks with up to 11 buffers were solved. However, the large number of constraints required for accuracy make constraint reduction an important approach when it is possible.

- Accuracy of 1 to 5% was achieved for networks with up to six buffers by adding other approximating functions. An order for systematically adding basis functions to improve accuracy was developed.

- Using the best approximation architectures found, the ALP requires much less computation than using DP value iteration on a truncated state space; however, this advantage diminishes when more accuracy is desired because the LP run time grows more rapidly with the number of variables. The memory requirement also limits the ability to solve ALPs with high levels of accuracy on large problems.

These approximation architectures could also be used with other approximate dynamic programming techniques. Although ALP is simpler and has been shown to work well on these examples, simulation-based methods avoid enumerating the exponential number of actions.

The constraint reduction techniques can be extended to stochastic processing networks, which allow general mappings between servers and classes. In fact, the software described in Section 6 is written for stochastic processing networks. A simple example is the “N” network in Section 6.1, where both of the servers can serve class 1. Some of the functions that were effective for queueing networks are likely to be useful for value function approximation in other problem settings, particularly the piece-wise quadratic...
functions and functions of one or two variables. The use of Bellman error to guide the selection of additional functions should also be useful in other problem settings.

It seems likely that other approximating functions could be found to refine the ones proposed here. In particular, the rational functions and functions of two variables are rather ad hoc. Another possibility for improvement is to use a customized algorithm to solve the LP. Although CPLEX handles these LPs fairly well, they have some challenging characteristics. The number of constraints is very large and many of them are only incrementally different from other constraints. In [37], ALPs with quadratic approximation and up to 40 buffers are solved using column generation on the dual. Delayed column generation reduces the memory requirement. An open question is whether column selection can be done efficiently for other approximation architectures. The number of constraints could also be reduced by implementing the restriction to nonidling policies (Section 5.1), when appropriate. Heuristic policy restrictions could also be explored. Implementing the other constraint reduction methods of Section 5 would allow certain larger ALPs to be solved. In particular, for piece-wise quadratic functions on cones it would be interesting to compare ALP(N) to (ALPD) in Section 5.3.

This paper has focused on developing and testing algorithms. It would be desirable to extend the error bounds in [7], [8], and [10] to the average cost setting. Although some results for average cost are given in [9], it is not clear how they apply to queueing networks. Applying a “vanishing discount rate” to the aforementioned papers might give a clearer bound. Finally, performance of the policies recovered from the ALP remains a major open question. Numerical tests show that the policies can have poor performance or be unstable. However, using different approximation architectures or combining the ALP policies with heuristics such as safety stocks might lead to improved policies and bounds on their performance, analogous to the discounted cost bounds in [7] and [10].

Appendix A Equivalence of LP form of the optimality equation

This appendix shows the equivalence of (1) to (LP) and (6). The argument is similar to, e.g., [26]. Using the constraints for the optimal policy and letting $x_k$ denote the state after $k$ transitions (including self-transitions of the uniformized chain),

$$J \leq c^T x_k + E_{u_*} [h(x_{k+1}) | x_k] - h(x_k).$$
After summing and telescoping, taking expectations yields

\[ J \leq \frac{1}{N} \sum_{k=0}^{N-1} E_{x,u} c' x_k + \frac{1}{N} E_{x,u} h(x_N) + \frac{1}{N} h(x_0). \]

We need to show that

\[ \lim_{N \to \infty} \frac{1}{N} E_{x,u} h(x_N) = 0 \]  \hspace{1cm} (45)

so that taking the limit as \( N \to \infty \) leaves \( J \leq J^* \) for any feasible \( h \). Then, since \( (J^*, h^*) \) are feasible, they are optimal for (LP) and (6).

To show (45), use the fact that, for all policies \( u \) with finite \( J(x, u) \),

\[ \lim_{N \to \infty} \frac{1}{N} E_{x,u} |x_N|^2 = 0 \]  \hspace{1cm} (46)

[17, Theorem 1]. Although they assume nonidling policies, (46) also holds for \textit{weakly nonidling} policies where \( u(t) \neq 0 \) if \( x(t) \neq 0 \), which includes \( u_* \). Their result also assumes \( x \) is in the recurrent class, but for the optimal policy this extends easily to all states. Combining (46) and (6) gives (45).

**Appendix B ALP Constraints for the Series Queue**

This appendix derives and reduces the ALP constraints for the series queue in Figure 2 and the differential cost approximation (30) which we repeat here:

\[ h(x) = \frac{1}{2} x' Q x + px + r_1 \beta_1^2 + r_2 x_1 \beta_2^2 + r_3 x_2 \beta_2^2. \]

Assume \( c_1 < c_2 \), so that only station 2 is nonidling, and \( \mu_2 \leq \mu_1 \). Set \( \beta \equiv \beta_2 = (\mu_2/\mu_1) \), \( \alpha \equiv \alpha_1 \) (the arrival rate), and recall that \( x = z + u \).

For this problem, (25) is

\[ J \leq c' x + \alpha [h(x + e_1) - h(x)] + u_1 \mu_1 [h(x - e_1 + e_2) - h(x)] + u_2 \mu_2 [h(x - e_2) - h(x)]. \]  \hspace{1cm} (47)
Substituting (30) into (47), (47) has the form

\[ J \leq d^u + c^u z + (\zeta^u + \xi^u z) \beta^{z^2 + u^2} \]  

(48)

for all \( z \in \mathbb{Z}_+^2 \) and all \( u \) that are nonidling at station 2. Next we express \( d^u, c^u, \zeta^u \) and \( \xi^u \) as linear functions of the variables \( p, Q, \) and \( r \).

The terms in (47) are

\[
\begin{align*}
    h(x + e_1) - h(x) &= q_{11}x_1 + q_{12}x_2 + \frac{1}{2}q_{11} + p_1 + r_2 \beta^{x_2} \\
    h(x - e_1 + e_2) - h(x) &= -(q_{11} + q_{12})x_1 + (q_{22} - q_{12})x_2 + \frac{1}{2}q_{11} + \frac{1}{2}q_{22} - q_{12} \\
    &- p_1 + p_2 - r_1(1 - \beta)\beta^{x_2} - r_2[(1 - \beta)x_1 + \beta]\beta^{x_2} - r_3[(1 - \beta)x_2 - \beta]\beta^{x_2} \\
    h(x - e_2) - h(x) &= -q_{12}x_1 - q_{22}x_2 + \frac{1}{2}q_{12} - p_2 + r_1(\frac{\mu_1}{\mu_2} - 1)\beta^{x_2} \\
    &+ r_2x_1(\frac{\mu_1}{\mu_2} - 1)\beta^{x_2} + r_3 \left[ (\frac{\mu_1}{\mu_2} - 1)x_2 - \frac{\mu_1}{\mu_2} \right] \beta^{x_2}. 
\end{align*}
\]

For the control \( u = (1, 1) \), \( \xi^{(1,1)} = 0 \) and

\[
\begin{align*}
    c_1^{(1,1)} &= c_1 - (\mu_1 - \alpha)q_{11} + (\mu_1 - \mu_2)q_{12} \\
    c_2^{(1,1)} &= c_2 - (\mu_1 - \alpha)q_{12} + (\mu_1 - \mu_2)q_{22} \\
    d^{(1,1)} &= c_1^{(1,1)} + c_2^{(1,1)} + \frac{1}{2}(\alpha + \mu_1)x_1 - \mu_1x_1 + \frac{1}{2}(\mu_1 + \mu_2)x_2 - \mu_1x_2 + (\mu_1 - \mu_2)p_2 \\
    \zeta^{(1,1)} &= -r_2(\mu_2 - \alpha) - r_3(\mu_1 - \mu_2). 
\end{align*}
\]

For \( u = (0, 1) \),

\[
\begin{align*}
    c_1^{(0,1)} &= c_1 + \alpha x_1 - \mu_2x_1 \\
    c_2^{(0,1)} &= c_2 + \alpha x_2 - \mu_2x_2 \\
    d^{(0,1)} &= c_2^{(0,1)} + \frac{1}{2}q_{11}x_1 + \frac{1}{2}\mu_2x_2 - \alpha x_1 - \mu_2x_2 \\
    \zeta^{(0,1)} &= r_2(\mu_1 - \mu_2) \\
    \xi^{(0,1)} &= r_3(\mu_1 - \mu_2) \\
    \zeta^{(0,1)} &= \xi^{(0,1)} + r_1(\mu_1 - \mu_2) + r_2\alpha - r_3\mu_1. 
\end{align*}
\]
For \( u = (1, 0) \), we must have \( z_2 = 0 \) and \( \xi_1^{(1,0)} = 0, \zeta^{(1,0)} = 0 \),

\[
\begin{align*}
c_1^{(1,0)} &= c_1 - (\mu_1 - \alpha)q_{11} + \mu_1q_{12} - r_2(\mu_1 - \mu_2), \\
d^{(1,0)} &= c_1^{(1,0)} + \frac{1}{2}(\alpha + \mu_1)q_{11} - \mu_1q_{12} + \frac{1}{2}\mu_1q_{22} - (\mu_1 - \alpha)p_1 + \mu_1p_2 \\
&\quad - r_1(\mu_1 - \mu_2) - r_2(\mu_2 - \alpha) + r_3\mu_2.
\end{align*}
\]

Finally, for \( u = x = (0, 0) \), \( \zeta^{(0,0)} = 0 \) and

\[
d^{(0,0)} = \frac{1}{2}q_{11} + \alpha p_1 + \alpha r_2.
\]

Now we reduce the constraints (48). For \( u = (1, 1) \), \( \xi^{(1,1)} = 0 \), so as \( z_1 \to \infty \) we must have

\[
c_1^{(1,1)} \geq 0, \quad i = 1, 2. \tag{49}
\]

Given (49), (48) is tightest at \( z_1 = 0 \) for each \( z_2 \). However, depending on the value of \( \zeta^{(1,1)} \) it could be tightest at any \( z_2 \). Thus, we will approximate (ALP) by including (48) at \( z_1 = 0, \ z_2 = 0, \ldots, N - 1 \) for some \( N \). Now consider \( u = (0, 1) \). For each \( z_2 \), the \( z_1 \) coefficient must be nonnegative,

\[
c_1^{(0,1)} + \beta \zeta_1^{(0,1)} \geq 0.
\]

Because of the monotonicity in \( z_2 \), this is equivalent to

\[
c_1^{(0,1)} + \beta \zeta_1^{(0,1)} \geq 0 \tag{50}
\]

and \( c_1^{(0,1)} \geq 0 \). Letting \( z_2 \to \infty \) in (48) gives another constraint, so we have

\[
c_1^{(0,1)} \geq 0, \quad i = 1, 2. \tag{51}
\]

Given (50) and (51), (48) is tightest at \( z_1 = 0 \) but, depending on \( \zeta^{(0,1)} \) and \( \xi_2^{(0,1)} \), could be tightest at any \( z_2 \), so we include (48) at \( u = (0, 1), \ z_1 = 0, \) and \( z_2 = 0, \ldots, N - 1 \). Next, for \( u = (1, 0) \) we must have \( z_2 = 0 \). For (48) to hold as \( z_1 \to \infty \), we must have

\[
c_1^{(1,0)} \geq 0. \tag{52}
\]
In light of (52), (48) is tightest at $z_1 = 0$, so we include (48) at $u = (1,0)$ and $z = (0,0)$. Finally, we include (48) at $u = z = (0,0)$.

To summarize, the approximate reduced ALP contains the $2N + 8$ constraints (48) at $u = (1,1)$, $z_1 = 0$, $z_2 = 0, \ldots, N-1$; $u = (0,1)$, $z_1 = 0$, $z_2 = 0, \ldots, N-1$; $u = (1,0)$, $z = (0,0)$; and $u = z = 0$; plus (49)-(52).

**Acknowledgements**

The numerical work in this paper was done using software developed by my colleague Jonathan Senning and our students Taylor Carr, Adam Elnagger, and Christopher Pfohl. Also assisting with numerical work were my students Lauren Berger, Lauren Carter, Jane Eisenhauer, Jeff Fraser, Michael Frechette, Melissa LeClair, Lauren Meitzler, Josh Nasman, and Nathan Walker. I would also like to thank Sean Meyn for his many suggestions. This work was supported in part by National Science Foundation grant 0620787.

**References**


