Approximate Linear Programming for Network Control: Column Generation and Subproblems

Michael H. Veatch and Nathan Walker
Department of Mathematics, Gordon College, Wenham, MA 01984
mike.veatch@gordon.edu
nathan.walker@gordon.edu

February 26, 2008

Subject classifications: Dynamic programming/optimal control: approximations/large-scale problems. Queues: control of queueing networks. Production/scheduling: dynamic scheduling.

Abstract

Approximate linear programming (ALP) has been shown to provide useful bounds on optimal average cost for multiclass queueing network control problems. Approximation architectures can be chosen so that a manageable number of variables gives moderate accuracy; however, the number of constraints grows exponentially in the number of queues. We use column generation to more efficiently solve the dual of this LP. Approximation architectures that are additive across part types are shown to simplify the column selection step of the algorithm. Efficient methods are developed for quadratic approximation architectures. A very efficient algorithm is given when the quadratic approximation architecture is also required to be additive across sets of servers. Examples with up to 40 queues are approximated.

1 Introduction

The natural modeling framework for many systems, such as supply chains and communication networks, is a multiclass queueing network (MQNET). Efficient operation of these systems can be addressed as a dynamic control problem; however, the utility of these models is limited by the inability to solve all but the smallest network optimization problems. Even under the simplest assumptions of exponentially distributed service and interarrival times, linear holding costs, and closed networks, MQNET control problems are NP-hard so that we cannot hope to solve large problems exactly [12]. Standard dynamic programming (DP) algorithms are too computationally intensive to use even on moderate size problems, particularly in heavy traffic.

This paper combines approximate linear programming (ALP) and column generation to compute average cost bounds for larger MQNETs than prior LP based bounds–much larger than can be solved using exact DP methods. ALP employs value function approximation and works directly with the LP form of the optimality equations. Value function approximation is also used in simulation-based approximate dynamic programming methods, including neuro-dynamic programming [2] and the gradient-based method of [13]. These approaches have been successful on a number of large DPs. Recently, the direct LP method described

here has shown promise. To make the number of variables tractable, the differential cost is approximated by a linear combination of functions. One key to using approximate dynamic programming is selecting a compact but accurate class of approximating functions. For MQNETs, quadratic approximating functions have been used, partly because of their simplicity. Another motivation is that the "cost to drain" of the associated fluid model is either quadratic or piece-wise quadratic and captures the quadratic growth of the differential cost in each direction in the state space.

The second key issue when using the ALP method is reducing the number of constraints. The ALP contains one constraint for every state-action pair, which is impractical for the problems of interest. Constraint reduction and approximation methods have been used in [10], [11], and [17] to reduce the constraints to a finite, but exponentially large set. In [7], constraint sampling is used for discounted problems to select a manageable number of constraints. However, their probabilistic accuracy guarantees assume that the optimal stationary distribution known, so their numerical results are only suggestive of what will occur in practice.

Adelman and Mersereau [1] propose a column generation algorithm for discounted DPs and demonstrate its computational efficiency on multiarm bandit-type problems. We adapt the column generation method to our average cost ALP and use it to compute lower bounds on the optimal average cost for MQNETs. There is a delicate connection between the network topology and approximation architecture used and the tractability of column generation on the resulting ALP. First, inspired by their "weakly coupled dynamic program" approach, we consider approximation architectures that decompose the network into sets of queues, or subproblems, but allow any value of the differential cost in each state of the subproblems. This approximation does not scale to larger subproblems; it uses a lookup table in each subproblem. However, we show that the column selection step of the algorithm can be made efficient for certain decompositions.

Next, we consider quadratic approximation architectures. This approximation is compact. Although the corresponding ALP can be reduced to be much smaller than the DP without function approximation, the size of the ALP is still exponential in the number of queues. Column generation dramatically reduces the size of the LP to be solved. Numerical tests demonstrate that column generation can be used on problems with up to 20 queues. When the approximation architecture is further simplified to be quadratic and decomposed into subproblems, somewhat larger networks can be handled. Finally, when the decomposition is chosen so that servers are nested within subproblems, the column selection step is more efficient and the bound is computed on problems with up to 40 queues. No theoretical results are available on the accuracy of the average cost lower bounds obtained. Their accuracy has been studied numerically on small problems [17]; the average cost bound is generally within 20% of optimal, but can be even looser at high traffic intensities. We compare the quadratic and quadratic with subproblems ALPs numerically. The use of subproblems often has a relatively small impact on the bound, ranging from 1% to 5% for our series line examples and from 4% to 20% for our two part-type series line examples.

The ALP approach was originally proposed by [14]. It is applied to discounted network problems in [6] using quadratic approximation architectures. Bounds have also been obtained using the achievable region method [4], [3]; however, it is shown in [10] that these bounds are weaker than the ALP bound with quadratic functions. The neuro-dynamic programming approach is applied to average cost network problems in [5]. Their work differs from ours in that the approximating functions are ridge functions based on asymptotic heavy traffic analysis and the focus is on policies, not bounds. Also, they consider a crossbar switch, where activities require multiple resources.

The rest of this paper is organized as follows. After defining the MQNET sequencing problem in Section 2, Section 3 presents the general ALP column generation and decomposition approach. Section 4 specializes the algorithm to the case of quadratic approximations. Numerical tests for quadratic approximations are reported in Section 5 and the relationship between speed of the algorithm and problem structure is discussed further in Section 6. Section 7 summarizes the results.

2 MQNET sequencing control

In the standard MQNET model there are n job classes and m servers, 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 server $\sigma(i)$. These assignments define the $m \times n$ constituency matrix with entries $C_{ji} = 1$ if server j serves class i and $C_{ji} = 0$ otherwise. Routing is deterministic. After processing at class i, jobs move to successor class $s(i)$ or leave the system, in which case $s(i) = 0$. There is no merging: class i has at most one predecessor class $p(i)$ such that $s(p(i)) = i$. If class i has no predecessor, set $p(i) = 0$. Each route through the network is called a part type. The algorithm described below could be extended to probabilistic routing but would not be as fast.

Exogenous arrivals occur at one or more classes according to independent Poisson processes with rate λ_i in class i. Processing times are assumed to be independently exponentially distributed with mean $1/\mu_i$ in class i. It will also be convenient to define $\mu_0 = 0$. Stability requires that $\rho < 1$, where ρ_j is the traffic intensity at server j.

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. Let $\mathcal{A}$ be the set of all actions (satisfying the first constraint) and $\mathcal{A}(x)$ be the set of actions in state x (satisfying both constraints).

The objective is to minimize long-run average cost

$$J(x,u) = \limsup_{T\to\infty} \frac{1}{T} E_{x,u} \int_0^T c^T x(t) dt.$$

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 $\sum_{i=1}^n (\lambda_i + \mu_i) = 1$. Define T_u, the one-step operator for policy u, by

$$(T_u h)(x) = \sum_{i=1}^n [c_i x_i + \lambda_i h(x + e_i) + \mu_i h(x + u_i(x)(-e_i + e_{s(i)}))].$$

Here e_i is a unit vector with ith component equal to one and we use the convention $e_0 = 0$ for jobs that

leave the network.

Under the condition $\rho < 1$, average cost is constant, $J_* = \min_u J(x,u)$ for all x, and solves the average cost optimality equation

$$J + h(x) = \min_{u \in \mathcal{A}(x)} (T_u h)(x). \tag{1}$$

Under the additional condition that, for some $L > 0$ and $M > 0$,

$$-L \le h(x) \le M(1 + |x|^2) \tag{2}$$

there is a unique solution J_* and h_* to (1) satisfying $h_*(0) = 0$. Furthermore, J_* is the optimal average cost, any policy

$$u_*(x) = \arg\min_u (T_u h_*)(x)$$

is optimal, and h_* is the differential cost of this policy,

$$h_*(x) = \limsup_{T \to \infty} E_{x,u_*} \int_0^T c^T x(t) dt - E_{0,u_*} \int_0^T c^T x(t) dt. \tag{3}$$

See [8, Theorem 2.1 and Section 7], [9, Theorem 7] and, for general MDPs, [15, Theorems 7.2.3 and 7.5.6] for proofs of these properties.

It is well known that, for finite state spaces, an inequality relaxation of Bellman's equation gives an equivalent LP in the same variables. For our countable state problem (1), the LP form is

$$\begin{aligned} \text{(LP)} \quad & \max \ J \\ & \text{s.t.} \ \ J + h(x) \le (T_u h)(x) \ \ \text{for all } x \in \mathbf{Z}_+^n, \ u(x) \in \mathcal{A}(x) \\ & \qquad h(0) = 0. \end{aligned}$$

The equivalence with (1) holds under condition (2).

3 Decomposition and column generation

3.1 The approximate linear program and subproblems

To create a tractable LP, the differential cost is approximated by some functional form. In this section we develop a column generation algorithm for the approximate linear program (ALP) assuming that the differential cost is additive across subproblems. However, the number of constraints must be reduced before solving. In Sections 4 and 5 a quadratic approximation is used within each subproblem, allowing the constraints to be reduced to a manageable set. For other differential cost approximations, constraint sampling [7] or other approximate methods [17] could be used.

Partition the classes into K subsets, or subproblems, numbered $1, \ldots, K$ and let $\tau(i)$ be the subproblem to which class i belongs. Also define $\tau(0) = 0$. Several ways of defining subproblems will be considered, including part types (for which τ is mnemonic). Require the differential cost approximation to be additive across subproblems,

$$h(x) = \sum_{k=1}^{K} h^k(x) \tag{4}$$

where $h^k(x)$ is constant over x_i if $\tau(i) \neq k$. Note that h^k can be written as a function of just those x_i with $\tau(i) = k$ in subproblem k; we use the argument $x \in Z_+^n$ for notational convenience. The resulting ALP is

$$\begin{aligned}
\text{(ALP)} \quad & \max \; J \\
& \text{s.t.} \;\; J - \sum_{k=1}^{K} \left(\sum_{i:\tau(i)=k} \lambda_i [h^k(x+e_i) - h^k(x)] + \sum_{i:\tau(i)=k} u_i \mu_i [h^k(x - e_i + e_{s(i)}) - h^k(x)] \right) \\
& \quad - \sum_{i:\tau(i)\neq\tau(s(i))\neq 0} u_i \mu_i [h^{\tau(s(i))}(x+e_{s(i)}) - h^{\tau(s(i))}(x)] \leq c^T x \;\; \text{for all } x \in \mathbf{Z}_+^n, \; u \in \mathcal{A}(x) \\
& \quad h^k(0) = 0 \;\; \text{for } k = 1, \ldots, K
\end{aligned} \tag{5}$$

Once again, (2) is required. Alternatively, a finite state space can be used. The last summation reflects the fact that, if a job moves from subproblem $\tau(i)$ to a different subproblem $\tau(s(i))$, both $h^{\tau(i)}$ and $h^{\tau(s(i))}$ are changed. If such transitions occur, then the transition probabilities of these subproblems are not independent. If different subproblems use the same server, they are also coupled by the constraints $Cu \leq 1$ on the actions.

Since (ALP) is equivalent to the exact LP with the constraints (4) added, the exact LP is a relaxation. Hence, (ALP) gives a lower bound on J_*. (ALP) has one variable for each subproblem state (we could eliminate $h^k(0)$) and one constraint for each state-action pair. The usual approach to solving (1) is to truncate the state space to $\mathbf{X} = \{x \in \mathbf{Z}_+^n : x_i < N\}$. We will refer to the corresponding state space for subproblem k as $\mathbf{X}^k$. Then (ALP) has $\sum_{k=1}^{K} N^{n_k}$ variables and roughly $N^n \prod_{k=1}^{K} n_k$ constraints, where n_k is the number of classes in subproblem k. In contrast, (LP) has N^n variables. Because there are many more constraints than variables, it is convenient to work with the dual. Returning to the unbounded state space

and letting $y(x,u)$ denote the dual variables, the dual of (ALP) is

$$
\begin{aligned}
\text{(DALP)}\quad \min\ & \sum_{x\in Z_+^n}\sum_{u\in\mathcal{A}(x)} (c^T x) y(x,u) \\
\text{s.t.}\ & \sum_{x\in Z_+^n}\sum_{u\in\mathcal{A}(x)} y(x,u) = 1 && J \\
& - \sum_{i:\tau(i)=k} \lambda_i \sum_{\substack{x\in Z_+^n: x_i\neq 0 \text{ and} \\ x_j=x_j^k,\tau(j)=k}} \sum_{u\in\mathcal{A}(x-e_i)} y(x-e_i,u) \\
& - \sum_{\substack{i:\tau(i)=k \\ \text{or } \tau(s(i))=k}} \mu_i \sum_{\substack{x\in Z_+^n: x_{s(i)}\neq 0 \text{ and} \\ x_j=x_j^k,\tau(j)=k}} \sum_{\substack{u\in\mathcal{A}(x+e_i-e_{s(i)}): \\ u_i=1}} y(x+e_i-e_{s(i)},u) \\
& + \sum_{\substack{x\in Z_+^n: \\ x_j=x_j^k,\tau(j)=k}} \sum_{u\in\mathcal{A}(x)} \left(\sum_{i:\tau(i)=k} \lambda_i + \sum_{\substack{i:u_i=1 \text{ and} \\ \tau(i)=k \text{ or } \tau(s(i))=k}} \mu_i \right) y(x,u) = 0 && h^k(x) \\
& \qquad \text{for all } x^k \in \mathbf{Z}_+^{n_k} \text{ for } k=1,\ldots,K \\
& y(x,u) \geq 0.
\end{aligned}
$$

The primal variables are listed beside each constraint. Here jobs moving from one subproblem to another create additional terms in the sums over i: a μ_i term is included if jobs in class i move into subproblem k, i.e., $\tau(s(i)) = k \neq \tau(i)$.

3.2 Column generation

The large number of constraints in (ALP) suggests that a method is needed to select constraints, rather than attempting to solve the problem using a standard LP algorithm. Column generation [cite ??] involves forming and solving an incomplete LP, then searching for additional variables to improve the solution. We then add this column and repeat the procedure, continuing until no column can be identified to improve the solution–this gives an optimal solution to the original LP. Applying column generation to (DALP) amounts to selecting constraints of (ALP) and adding them to the incomplete LP until all the binding constraints have been found. This section describes a column generation algorithm for (DALP), but using the finite state space $\mathbf{X}$. Suitable modifications to the constraints must be made to enforce the upper bounds on x. In Section 4 the algorithm is adapted to a quadratic approximation architecture, which does not require state space truncation.

At each iteration the algorithm uses some subset $\mathcal{C}$ of the variables in (DALP) and solves DALP(C), by which we denote (DALP) with all variables except those in $\mathcal{C}$ deleted. Let $R(x,u)$ be the reduced cost of the variable $y(x,u) \notin \mathcal{C}$ in DALP(C); equivalently, $R(x,u)$ is the slack in the corresponding constraint in (ALP).

Setting J and $h^k(\cdot)$ to the optimal solution of the corresponding ALP(C),

$$\begin{aligned} R(x,u) &= c^T x - J + \sum_{k=1}^{K} \sum_{i:\tau(i)=k} [\lambda_i (h^k(x+e_i) - h^k(x)) + u_i \mu_i (h^k(x - e_i + e_{s(i)}) - h^k(x))] \\ &+ \sum_{i:\tau(i)\neq\tau(s(i))\neq 0} u_i \mu_i [h^{\tau(s(i))}(x + e_{s(i)}) - h^{\tau(s(i))}(x)]. \end{aligned} \tag{6}$$

If all reduced costs are nonnegative, the optimal solution to DALP(C) is also optimal for (DALP) and the algorithm terminates. Otherwise, any variable with negative reduced cost can be added to $\mathcal{C}$. Finding the most negative reduced cost requires solving

$$\begin{aligned} \text{IP(C)} \quad \min_{u,x\in\mathbf{X}} \quad & R(x,u) \\ \text{s.t.} \quad & Cu \leq 1 \\ & u \leq x \\ & u_i = 0,1 \quad \text{for } i = 1,\ldots,n \end{aligned}$$

This IP is linear in u, so the integer restriction on u can be relaxed. Solving IP(C) by exhaustive search would be analogous to the revised simplex method, which checks every nonbasic column. In this case the time required by column generation would be comparable to the revised simplex method.

Another advantage of column generation is that a lower bound can be computed at each iteration [cite??]. Let $\underline{J}$ be the optimal objective function value for (DALP). If DALP(C) and IP(C) are solved, giving the objective function values $J_{\mathcal{C}}$ and $R^{\mathcal{C}}$, then

$$J_{\mathcal{C}} + R^{\mathcal{C}} \leq \underline{J} \leq J_{\mathcal{C}}. \tag{7}$$

3.3 Multiple part types and decompositions

Column generation has typically been used for problems in which the column selection problem is easily solved. We now present a case where the structure of IP(C) allows it to be more easily solved. Consider a network with multiple part types and define the subproblems as the part types. Then $\tau(i)$ is the part type to which class i belongs. Because there are no transitions between subproblems, i.e., $\tau(i) = \tau(s(i))$ unless i is an exit class, the final summation in (ALP) and (6) is eliminated. Thus, $R(x,u)$ decomposes over subproblems:

$$\begin{aligned} R(x,u) &= -J + \sum_{k=1}^{K} g_k(x,u), \text{ where} \\ g_k(x,u) &= \sum_{i:\tau(i)=k} [c_i x_i + \lambda_i (h^k(x+e_i) - h^k(x)) + u_i \mu_i (h^k(x - e_i + e_{s(i)}) - h^k(x))]. \end{aligned}$$

Again, although g_k only depends on x_i and u_i for $i : \tau(i) = k$, we write it as a function of x and u for notational convenience. The constraints in IP(C) are still coupled by servers that are shared between part types. If there are few such shared servers, one could consider every allocation of servers to part types. Let

$\sigma_j^k = 1$ if server j is allocated to part type k and 0 otherwise. Also denote by S the set of possible allocations σ of servers to part types. Given the allocation σ, the part type k IP is

$$\begin{aligned} g_k^\sigma \quad = \quad & \min g_k(x,u) \\ & \text{s.t.} \sum_{i:\tau(i)=k \text{ and } \sigma(i)=j} u_i \le \sigma_j^k, \quad j = 1,\ldots,m \\ & \qquad\qquad\qquad\qquad u_i \le x_i, \text{ for } i : \tau(i) = k. \end{aligned}$$

The first constraint only allows part types to use the servers allocated to them under σ. The form of IP(C) for part type subproblems, then, is

$$\min_{\sigma \in S} -J + \sum_{k=1}^{K} g_k^\sigma.$$

4 Quadratic approximation

Consider the quadratic differential cost approximation

$$h(x) = \frac{1}{2} x^T Q x + px \tag{8}$$

where $Q = [q_{ij}]$ and $q_{ij} = 0$ for $\tau(i) \neq \tau(j)$. In this section, for ease of exposition we do not assume Q is symmetric. The smaller, but not as easily written, LPs for the usual symmetric form of Q are given in the Appendix. A quadratic approximation is motivated by fluid analysis and by its tractability; see [16], [17], and [10]. The block-diagonal form of Q is appealing because it can greatly reduce the size of the LPs; it essentially assumes there is no interaction between subproblems. Using (8) allows two simplifications of (ALP). First, the constraints for each u are linear in x. Hence, the constraints at each $x \ge u$ are equivalent to one constraint at $x = u$ plus n "limiting" constraints, one for each component of x that can increase. Second, the limiting constraints for the x_i direction depend only on subproblem $\tau(i)$ and their predecessor classes, eliminating many of these constraints. Let

$$\mathcal{A}^k = \{u \in \mathcal{A} : u_i = 0 \text{ if } \tau(i) \neq k \text{ and } i \neq p(j) \text{ for all classes } j \text{ s.t. } \tau(j) = k\}.$$

Note that $\mathcal{A}^k$ contains the allowable actions for subproblem k and its predecessor classes. To simplify notation, its elements are n-vectors but in other classes we set $u_i = 0$. Also define $u_0 = 0$. (ALP) with the

block-diagonal quadratic approximation (8) is

$$
\begin{aligned}
\text{(ALPQ)}\ \max\ & J \\
\text{s.t.}\ \ & J - \sum_{i=1}^{n}[u_i(\lambda_i - \frac{1}{2}u_i + u_{p(i)}\mu_{p(i)}) + \frac{1}{2}u_{p(i)}\mu_{p(i)} + \frac{1}{2}\lambda_i]q_{ii} \\
& - \sum_{i=1}^{n}\sum_{\substack{j:j\neq i,\\ \tau(j)=\tau(i)}} u_i(\lambda_j - u_j\mu_j + 1_{\{j\neq s(i)\}}u_{p(j)}\mu_{p(j)})q_{ij} \\
& - \sum_{i=1}^{n}(\lambda_i - u_i\mu_i + u_{p(i)}\mu_{p(i)})p_i \le \sum_{i=1}^{n} u_i c_i \ \text{ for } u \in \mathcal{A} && y(u) \\
& - \sum_{j:\tau(j)=\tau(i)} (\lambda_j - u_j\mu_j + u_{p(j)}\mu_{p(j)})q_{ij} \le c_i && w(i,u) \\
& \qquad\qquad \text{for all } i \text{ s.t. } \tau(i) = k \text{ and } u \in \mathcal{A}^k \text{ for each } k.
\end{aligned}
$$

The dual variables are listed beside each constraint. (ALPQ) is derived in the Appendix. Its dual is

$$
\begin{aligned}
\text{(DALPQ)}\ \ \min\ & \sum_{u\in\mathcal{A}}(c^T u)y(u) + \sum_{i=1}^{n} c_i \sum_{u\in\mathcal{A}^{\tau(i)}} w(i,u) \\
\text{s.t.}\ \ & \sum_{u\in\mathcal{A}} y(u) = 1 && J \\
& - \sum_{u\in\mathcal{A}}[u_i(\lambda_i - \frac{1}{2}u_i + u_{p(i)}\mu_{p(i)}) + \frac{1}{2}u_{p(i)}\mu_{p(i)} + \frac{1}{2}\lambda_i]y(u) \\
& \quad - \sum_{u\in\mathcal{A}^{\tau(i)}} (\lambda_i - u_i\mu_i + u_{p(i)}\mu_{p(i)})w(i,u) = 0 \ \text{ for all } i && q_{ii} \\
& - \sum_{u\in\mathcal{A}} u_i(\lambda_j - u_j\mu_j + 1_{\{j\neq s(i)\}}u_{p(j)}\mu_{p(j)})y(u) \\
& \quad - \sum_{u\in\mathcal{A}^{\tau(i)}} (\lambda_j - u_j\mu_j + u_{p(j)}\mu_{p(j)})w(i,u) = 0 \ \text{ for all } i,j : \tau(i) = \tau(j),\ i \neq j && q_{ij} \\
& - \sum_{u\in\mathcal{A}}(\lambda_i - u_i\mu_i + u_{p(i)}\mu_{p(i)})y(u) = 0 \ \text{ for all } i && p_i \\
& y(x) \ge 0, \quad w(i,u) \ge 0.
\end{aligned}
$$

The primal variables are listed beside each constraint.

If we use the "symmetric Q" form of (ALPQ) from the Appendix, it has roughly $n + \sum_{k=1}^{K} n_k^2/2$ variables, $|\mathcal{A}|$ constraints associated with the $y(u)$ variables, and $\sum_{k=1}^{K} n_k|\mathcal{A}^k|$ variables associated with $w(i,u)$. There will tend to be many more "y" constraints than "w" constraints. Although the use of subproblems has not eliminated the dependence on $|\mathcal{A}|$ in the size of the LP, it has made the LP smaller and sparser. If the subproblems are part types (see Section 3.3) then there are no predecessor classes outside of a subproblem and $|\mathcal{A}^k|$ is somewhat smaller.

To apply the column generation method described in Section 3.2, let DALPQ($\mathcal{C}$) denote (DALP) with all variables except those in $\mathcal{C}$ deleted. Consider first the problem of selecting a "y" column. Setting J, Q,

and p to the optimal solution of the corresponding ALPQ($\mathcal{C}$), the reduced cost of $y(u) \notin \mathcal{C}$ in DALPQ($\mathcal{C}$) is

$$\begin{aligned} R_y(u) &= c^T u + \sum_{i=1}^{n} [u_i(\lambda_i - \frac{1}{2}u_i + u_{p(i)}\mu_{p(i)}) + \frac{1}{2}u_{p(i)}\mu_{p(i)} + \frac{1}{2}\lambda_i]q_{ii} \\ &+ \sum_{i=1}^{n} \sum_{\substack{j:j\neq i,\\ \tau(j)=\tau(i)}} u_i(\lambda_j - u_j\mu_j + 1_{\{j\neq s(i)\}}u_{p(j)}\mu_{p(j)})q_{ij} \\ &+ \sum_{i=1}^{n} (\lambda_i - u_i\mu_i + u_{p(i)}\mu_{p(i)})p_i. \end{aligned} \tag{9}$$

Finding the most negative reduced cost requires solving

$$\begin{aligned} \min_u \ & R_y(u) \\ \text{s.t.} \quad & Cu \leq 1 \\ & u_i = 0, 1 \ \text{ for } i = 1, \ldots, n. \end{aligned} \tag{10}$$

This binary integer program is non-trivial because of the quadratic terms $u_i u_j$ in its objective function.

We will select "w" columns separately for each class i. The reduced cost of $w(i, u) \notin \mathcal{C}$ in DALPQ($\mathcal{C}$) is

$$R_w(i, u) = c_i + \sum_{j:\tau(j)=\tau(i)} (\lambda_j - u_j\mu_j + u_{p(j)}\mu_{p(j)})q_{ij}. \tag{11}$$

Finding the most negative reduced cost for class i requires solving

$$\min_{u \in \mathcal{A}^{\tau(i)}} R_w(i, u). \tag{12}$$

Recall that $\mathcal{A}^k$ only allows u_j to vary for classes j that are in subproblem k or are predecessor classes to subproblem k, limiting the number of variables in (12). More importantly, $R_w(i, u)$ is linear in u, so (12) is equivalent to its trivial LP relaxation: each server serves the class j with the largest summand in (11).

The other ingredient of a column generation procedure is the selection of an initial set of columns that will guarantee DALPQ($\mathcal{C}$) is feasible. As it is not obvious how to choose such a set, we introduce an artificial variable for each row using the "Big M" method. If any of these artificial variables have non-zero values in the optimal solution when the algorithm terminates, the original problem is infeasible.

We end this section with a summary of the column generation algorithm for (DALPQ). Solving the LP from the previous optimal solution when a column is added is not time-consuming and is done after each column is added.

1. Initialize $\mathcal{C}$ as the set of artificial variables, one for each row in (DALPQ), and initialize the current solution using these variables.

2. Solve DALPQ($\mathcal{C}$) using the current solution as the initial solution. Update the current solution.

3. Using the shadow prices J, Q and p from DALPQ($\mathcal{C}$), solve (10). If the optimal objective function value is negative, add the variable $y(u^*)$ to $\mathcal{C}$ and its column to DALPQ($\mathcal{C}$) and perform step 2.

Number of classes	λ	μ	c
n	$0.9, 0, \ldots, 0$	$1 + \frac{n-1}{0.1}, 1 + \frac{n-2}{0.1}, \ldots, 1$	$1, 1 + \frac{1}{n-1}, 1 + \frac{2}{n-1}, \ldots, 2$

Table 1: Parameter values for the series line example

4. For each class i: Solve (12). If the optimal objective function value is negative, add the variable $w(i, u^*)$ to $\mathcal{C}$ and its column to DALPQ($\mathcal{C}$) and perform step 2.

5. If columns were added during steps 3 and 4, go to step 3. Otherwise stop.

5 Series line examples

The size and speed of the column generation algorithm for the block-diagonal quadratic approximation were tested. All tests were done using the smaller, symmetric form (DALPS) given in the Appendix. For comparison, the simplex algorithm was used to solve (DALPS) when possible. CLPEX 9.0 to solve the LPs and IPs on a machine with 8 gigabytes of RAM. A series line and a series line with two part types were used as examples so that the number of classes could be easily varied. The parameter values for the series line are given in Table 1. The traffic intensity is 0.9 and the downstream server (class n) is the bottleneck. Holding costs are higher downstream, so that idling the upstream servers $1, \ldots, n-1$ is optimal in some states.

First, consider the effect of subproblems on the size of (DALPQ) and the accuracy of the average cost. Figure 1 compares the number of columns in (DALPQ) for the series line without subproblems and when divided into two and four subproblems of equal size. Using two subproblems, i.e., two diagonal blocks in the quadratic approximation, reduces size about an order of magnitude from the full quadratic approximation. Further partitioning has little effect on size. Figure 2 compares the average cost, found by solving (DALPQ), for the same problems. Recall that these are lower bounds on the optimal average cost, which cannot be computed for problems of this size. Using two subproblems degrades the bound by 1 to 5%. Interestingly, there is less degradation for longer lines.

∞

The convergence of the column generation algorithm and its lower bound (7) are illustrated in Figure 3 for the series line with 12 classes and no subproblems. The problem is solved after adding 441 of the 53,428 columns of the full LP. The column generation algorithm is compared with CPLEX's dual simplex algorithm for (DALPQ) in Figures 4 and 5. As the number of classes increases, the number of columns in (DALPQ) grows rapidly, but the number of those columns used by the column generation algorithm grows much more slowly, reaching 1,084 columns for 16 classes. The run time also grows more slowly using column generation, though the difference is not as great as in problem size. With 16 classes, the column generation algorithm took 16.3 seconds. Using two subproblems reduces the run time to 2.6 seconds. Most of this time is devoted to solving the column selection IP (10). When columns are added to the LP, it is re-optimized in just a few iterations of the simplex algorithm.

The algorithm was also tested on a two part-type series line. The service rates and holding costs for the first part type are as shown in Table 1; for the second part type the service rates are three times as large and the holding costs twice as large as the first part type. Both part types have $\lambda_1 = (3/4)0.9$, giving a traffic intensity of 0.9. As shown in Figure 6, a problem with 10 servers (20 classes) solves in 500 seconds

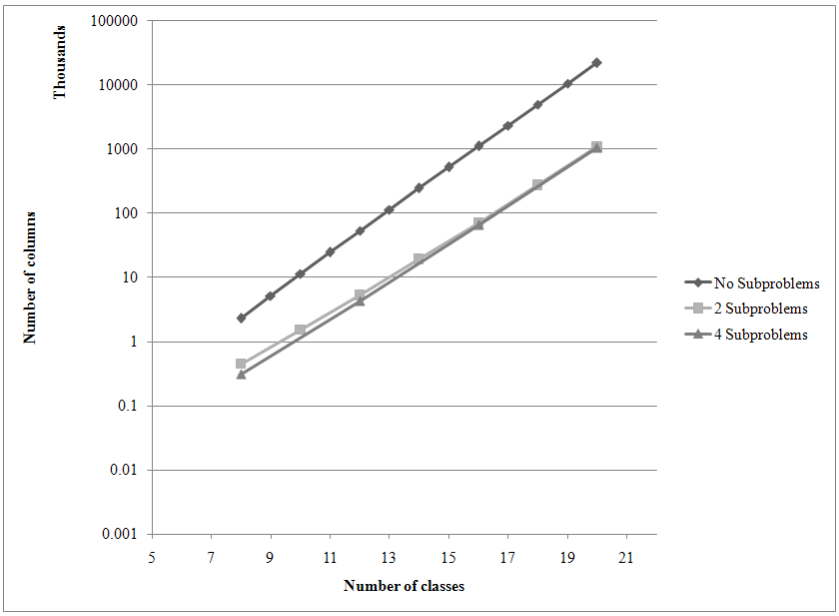

Figure 1: Size of full LP with and without subproblems, series line.

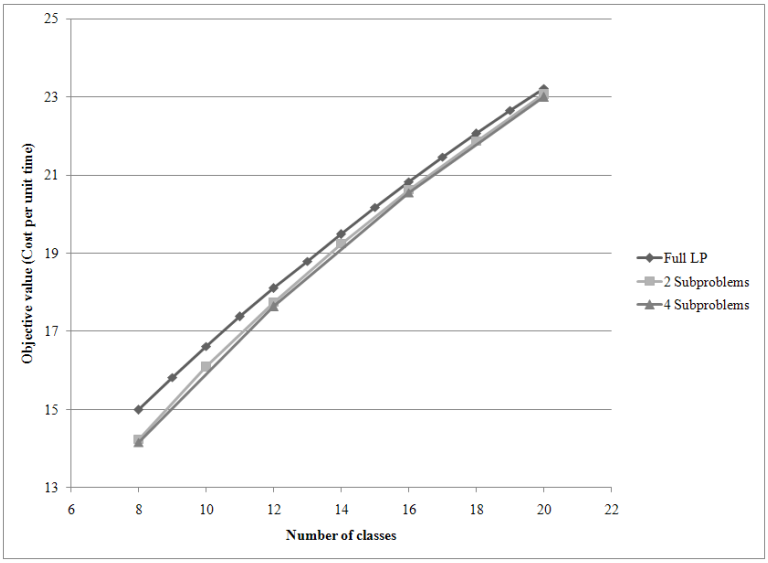

Figure 2: Average cost lower bound with and without subproblems, series line.

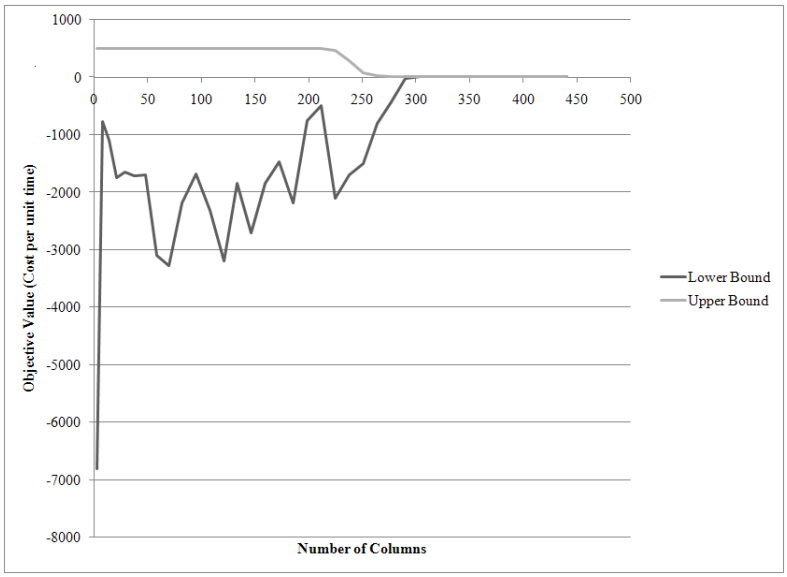

Figure 3: Convergence of column generation algorithm, 12-class sereis line.

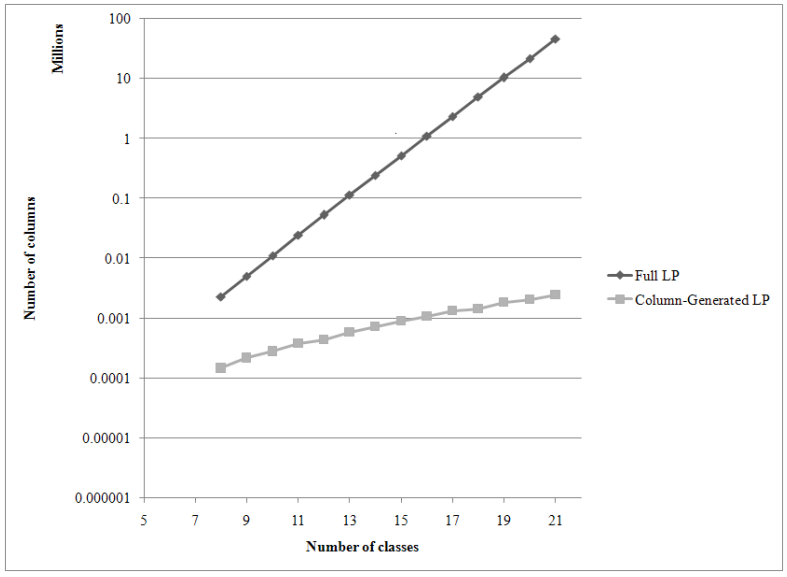

Figure 4: Problem size: full LP vs. column-generated LP, series line.

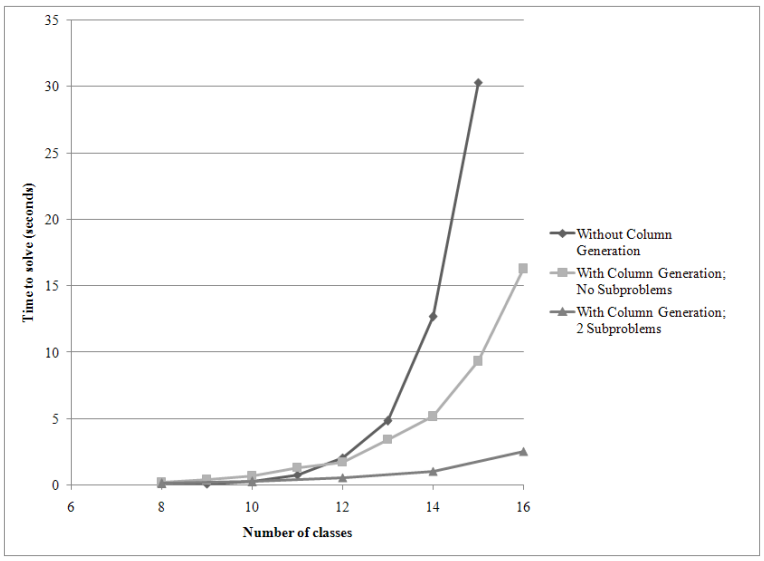

Figure 5: Run time of column generation and standard CPLEX algorithm, series line.

using the full quadratic approximation and 14 seconds when upstream and downstream servers are divided into two subproblems of the same size.

6 Structure of the column selection IP

The limiting step in computing the average cost bound using a quadratic approximation is solving the column selection binary integer program (10), which contains quadratic terms in the objective function. In this section we discuss a choice of subproblems for which faster solutions are possible. We will require servers to be nested within subproblems. Let $\tau_\sigma(j)$ be the subproblem containing (all classes served by) server j. In this case we say subproblems are "by server." The constraints in (10) decompose by subproblem. Unfortunately, the objective function is not separable by subproblem; however, the number of linking variables is limited. Consider the term $u_i u_{p(j)}$ in (9) and suppose $\tau(i) \neq \tau(p(j))$. Then, since $\tau(i) = \tau(j)$, $p(j)$ is a class that transitions into another subproblem. Let $\mathcal{T}^k = \{i : \tau(s(i)) = k \text{ and } \tau(i) \neq k\}$ be the set of classes that transition into subproblem k. Once u_i is fixed for all i in $\mathcal{T}^k$ for all subproblems k, (9) is separable by subproblem. To decompose the objective function, let

$$\begin{aligned} g_k(u) \quad = \quad & \sum_{i:\tau(i)=k} c_i u_i + \sum_{i:\tau(i)=k} [u_i(\lambda_i - \frac{1}{2}u_i + u_{p(i)}\mu_{p(i)}) + \frac{1}{2}u_{p(i)}\mu_{p(i)} + \frac{1}{2}\lambda_i]q_{ii} \\ & + \sum_{i:\tau(i)=k} \sum_{\substack{j:j\neq i,\\ \tau(j)=\tau(i)}} u_i(\lambda_j - u_j\mu_j + 1_{\{j\neq s(i)\}}u_{p(j)}\mu_{p(j)})q_{ij} \\ & + \sum_{i:\tau(i)=k} (\lambda_i - u_i\mu_i + u_{p(i)}\mu_{p(i)})p_i. \end{aligned}$$

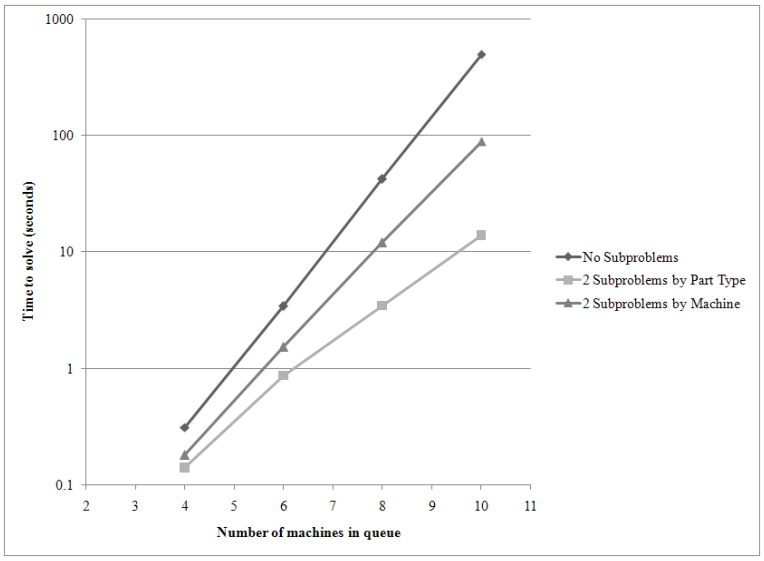

Figure 6: Run time using column generation with and without subproblems, two part type series line.

Note that g_k depends only on variables in subproblem k and in $\mathcal{T}^k$. Given $u_i = \overline{u}_i$, $i \in \mathcal{T}^k$, the IP for subproblem k is

$$\begin{aligned} g_k^{\overline{u}} \quad = \quad & \min g_k(u) \\ & \text{s.t.} \sum_{i:\ \sigma(i)=j} u_i \leq 1 \quad \text{for } j : \tau_\sigma(j) = k \\ & \qquad u_i = \overline{u}_i \quad \text{for } i \in \mathcal{T}^k \\ & \qquad u_i = 0, 1 \quad \text{for } i : \tau(i) = k. \end{aligned} \tag{13}$$

Then (10) can be solved by solving the subproblem IPs (13) for each feasible value of the variables in $\mathcal{T}^k$. The form of (10) when subproblems are by server is

$$\min \sum_{k=1}^{K} g_k^{\overline{u}} \tag{14}$$

$$\text{s.t.} \sum_{i:\ \sigma(i)=j \text{ and } i \in \mathcal{T}} \overline{u}_i \leq 1 \quad \text{for } j = 1, \ldots, m$$

$$\overline{u}_i = 0, 1 \quad \text{for } i \in \mathcal{T}^k. \tag{15}$$

The first constraint requires that the $\overline{u}_i$ are feasible for each server; constraints for servers not represented in $\mathcal{T}^k$ can be eliminated.

To illustrate the advantage of (14) over (10), consider again the two part-type series line of Section 5 and divide the m servers into two subproblems, the downstream half and the upstream half. Then $\mathcal{T}^1$ is empty and $\mathcal{T}^2$ contains just the two classes served by the last server in the upstream subproblem. For the upstream subproblem, (13) contains m variables and must be solved once. For the downstream subproblem,

(13) contains $m+2$ variables and must be solved three times. In comparison, (10) contains $n = 2m$ variables.

This algorithm solved the two part-type series line example of Section 5 with 10 servers (20 classes) and two subproblems in 9 seconds, compared to 19 seconds using the algorithm in Section 5. For larger problems the speedup is much larger. With 20 servers (40 classes) and two subproblems, this algorithm required 70 minutes, while the Section 5 algorithm did not finish in 36 hours. Dividing it into four subproblems cut the run time to 18 minutes.

7 Summary

We have demonstrated that approximate linear programming (ALP) and column generation make it feasible to compute lower bounds on optimal average cost for fairly large multiclass networks. Using quadratic approximation architectures, the size of the ALP–while much smaller than the dynamic program–is still exponential in the number of queues. Column generation dramatically reduces the size of the LP to be solved. Numerical tests demonstrate that column generation can be used on problems with up to 20 queues. When the approximation architecture is further simplified to be quadratic and decomposed into sets of queues or subproblems, with servers nested within subproblems, up to 40 queues can be handled. These examples are much larger than those for which LP-based bounds have been reported previously. Compared with a quadratic approximation, the further approximation of using two subproblems has a relatively small impact on the bound, ranging from 1% to 5% for series line examples and from 4% to 20% for two part-type series line examples.

Decomposition can also be used when the differential cost is allowed to take on any value in each state of the subproblems. Although this approximation is not practical for large subproblems, it exploits another structural property: the column selection step of the algorithm can be made efficient if subproblems are separate part types.

Two major questions for further investigation are the accuracy of the bounds obtained and whether useful policies can be obtained from the ALP. The accuracy of a quadratic approximation has been studied numerically on small problems [17]; the average cost bound is generally within 20% of optimal, but can be even looser at high traffic intensities. Expanding the approximation architecture has the potential to improve accuracy, but it is not clear how to do so while preserving the tractability of the ALP. Policies obtained from approximate dynamic programs have been effective for many problems, including the networks studied in [5]. However, no theoretical bounds are available for the average cost ALP method used here. Our preliminary testing has shown that policies based on quadratic approximations can contain reasonable structure but also can be unstable.

Acknowledgements

This work was supported in part by NSF grant CMMI-0620787.

References

[1] D. Adelman and A. Mersereau. Relaxations of weakly coupled stochastic dynamic programs. To appear, Oper. Res., 2004.

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

[3] D. Bertsimas, D. Gamarnik, and J.N. Tsitsiklis. Performance of multiclass Markovian queueing networks via piecewise linear lyapunov functions. *Ann. Appl. Probab.*, 11:1384–1428, 2001.

[4] D. Bertsimas, I.Ch. Paschaladis, and J.N. Tsitsiklis. Optimization of multiclass queueing networks: Polyhedral and nonlinear characterizations of achievable performance. *Ann. Appl. Probab.*, 4:43–75, 1994.

[5] S. Kumar C.C. Moallemi and B. Van Roy. Approximate and data-driven dynamic programming for queueing networks. Working paper, Graduate School of Business, Columbia University, 2006.

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

[7] D.P. de Farias and B. Van Roy. On constraint sampling for the linear programming approach to approximate dynamic programming. *Math. Oper. Res.*, 29(3):462–478, 2004.

[8] S.P. Meyn. The policy iteration algorithm for Markov decision processes with general state space. *IEEE Trans. Automat. Control*, AC-42:191–197, 1997.

[9] S.P. Meyn. Sequencing and routing in multiclass queueing networks. Part I: Feedback regulation. *SIAM J. Control Optim.*, 40:741–776, 2001.

[10] J.R. Morrison and P.R. Kumar. New linear program performance bounds for queueing networks. *J. Optim. Theory Appl.*, 100(3):575–597, 1999.

[11] J.R. Morrison and P.R. Kumar. Linear programming performance bounds for Markov chains with polyhedrally translation invariant transition probabilities and applications to unreliable manufacturing systems and enhanced wafer fab models. In *Proceedings of IMECE2002*, New Orleans, LA, 2002. Full-length version available at http://black.csl. uiuc.edu/∼prkumar.

[12] C.H. Papadimitriou and J.N. Tsitsiklis. The complexity of optimal queuing network control. *Math. Oper. Res.*, 24:293–305, 1999.

[13] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* Wiley, 2007.

[14] P. Schweitzer and A. Seidmann. Generalized polynomial approximations in Markovian decision processes. *J. of Mathematical Analysis and Applications*, 110:568–582, 1985.

[15] L.I. Sennott. *Stochastic Dynamic Programming and the Control of Queueing Systems.* Wiley, New York, 1999.

[16] M.H. Veatch. Using fluid solutions in dynamic scheduling. In S. B. Gershwin, Y. Dallery, C. T. Papadopoulos, and J. M. Smith, editors, *Analysis and Modeling of Manufacturing Systems*, pages 399–426, New York, 2002. Kluwer.

[17] M.H. Veatch. Approximate dynamic programming for networks: Fluid models and constraint reduction. Working paper, Gordon College, Dept. of Math. Available at faculty.gordon.edu/ns/mc/Mike-Veatch/index.cfm, May 2005.

Appendix Derivations of the quadratic ALPs

This appendix derives (ALPQ), which uses the quadratic differential cost approximation, and the smaller version of (ALPQ) and its dual assuming that the matrix Q for the quadratic function (8) is symmetric. A similar derivation of (ALPQ) is given in [17] for a quadratic approximation without subproblems. It is convenient to let $x = z + u$, so that a control u is feasible for all $z \in Z_+^n$. Note that the assumption $q_{ij} = 0$ for $\tau(i) \neq \tau(j)$ makes (8) additive across subproblems as in (ALP). Substitution into (5) in (ALP) yields

$$J \leq d^u + (c^u)^T z \tag{16}$$

for all $z \in Z_+^n$ and all $u \in \mathcal{A}$, where

$$\begin{aligned} c_i^u &= c_i + \sum_{j:\tau(j)=\tau(i)} (\lambda_i + u_{p(j)}\mu_{p(j)} - u_j\mu_j)q_{ij} \\ d^u &= \sum_{i=1}^{n} [u_i(c_i^u + \mu_i(\frac{1}{2}q_{ii} + \frac{1}{2}q_{s(i),s(i)} - 1_{\{\tau(s(i))=\tau(i)\}}q_{i,s(i)} + p_{s(i)} - p_i)) + \lambda_i(\frac{1}{2}q_{ii} + p_i)] \end{aligned}$$

and $c^u = [c_i^u]$. As noted in [10] and [17], the linearity of the constraints (16) in x allows them to be reduced to the equivalent constraints

$$J \leq d^u \tag{17}$$

$$c_i^u \geq 0 \tag{18}$$

for all i and u. Now, constraint (18) for a given i depends only on actions in subproblem $\tau(i)$ and their predecessor classes, eliminating many of these constraints. Thus, using the definition of $\mathcal{A}^k$ in Section 4, (ALP) with the block-diagonal quadratic approximation (8) reduces to

$$\begin{aligned} &\max\ J \\ &\text{s.t.}\quad J - d^u \leq 0 \ \text{ for } u \in \mathcal{A} \\ &\qquad\quad - c_i^u \leq 0 \ \text{ for all } i \text{ s.t. } \tau(i) = k \text{ and } u \in \mathcal{A}^k \text{ for each } k \end{aligned}$$

which is equivalent to (ALPQ) in Section 4.

Now add the requirement that Q is symmetric, $q_{ij} = q_{ji}$, and eliminate the lower triangular variables

q_{ij}, $i > j$ from (ALPQ) by replacing them with q_{ji}. Collecting the two q_{ij} terms, the double summation in (ALPQ) becomes

$$-\sum_{i=1}^{n} \sum_{\substack{j:j>i,\\ \tau(j)=\tau(i)}} [u_i(\lambda_j - u_j\mu_j + 1_{\{j\neq s(i)\}}u_{p(j)}\mu_{p(j)}) + u_j(\lambda_i - u_i\mu_i + 1_{\{i\neq s(j)\}}u_{p(i)}\mu_{p(i)})]q_{ij}$$

and the last summation can be written

$$-(\lambda_i - u_i\mu_i + u_{p(i)}\mu_{p(i)})q_{ii} - \sum_{\substack{j:j>i,\\ \tau(j)=\tau(i)}} (\lambda_i - u_i\mu_i + u_{p(i)}\mu_{p(i)} + \lambda_j - u_j\mu_j + u_{p(j)}\mu_{p(j)})q_{ij}.$$

Call this symmetric form (ALPS). Its dual is

$$\begin{aligned}
\text{(DALPS)}\quad \min\ & \sum_{u\in\mathcal{A}} (c^T u)y(u) + \sum_{i=1}^{n} c_i \sum_{u\in\mathcal{A}^{\tau(i)}} w(i,u) \\
\text{s.t.}\ & \sum_{u\in\mathcal{A}} y(u) = 1 && J \\
& -\sum_{u\in\mathcal{A}} [u_i(\lambda_i - \frac{1}{2}u_i + u_{p(i)}\mu_{p(i)}) + \frac{1}{2}u_{p(i)}\mu_{p(i)} + \frac{1}{2}\lambda_i]y(u) \\
& \quad - \sum_{u\in\mathcal{A}^{\tau(i)}} (\lambda_i - u_i\mu_i + u_{p(i)}\mu_{p(i)})w(i,u) = 0 \ \text{ for all } i && q_{ii} \\
& -\sum_{u\in\mathcal{A}} [u_i(\lambda_j - u_j\mu_j + 1_{\{j\neq s(i)\}}u_{p(j)}\mu_{p(j)}) + u_j(\lambda_i - u_i\mu_i + 1_{\{i\neq s(j)\}}u_{p(i)}\mu_{p(i)})]y(u) \\
& \quad - \sum_{u\in\mathcal{A}^{\tau(i)}} (\lambda_i - u_i\mu_i + u_{p(i)}\mu_{p(i)} + \lambda_j - u_j\mu_j + u_{p(j)}\mu_{p(j)})w(i,u) = 0 && q_{ij} \\
& \qquad\qquad \text{for all } i,j : \tau(i) = \tau(j),\ i < j \\
& -\sum_{u\in\mathcal{A}} (\lambda_i - u_i\mu_i + u_{p(i)}\mu_{p(i)})y(u) = 0 \ \text{ for all } i && p_i \\
& y(x) \geq 0, \quad w(i,u) \geq 0.
\end{aligned}$$

The reduced costs (9) and (11) are replaced by

$$\begin{aligned}
R_y(u) \quad = \quad & c^T u + \sum_{i=1}^{n} [u_i(\lambda_i - \frac{1}{2}u_i + u_{p(i)}\mu_{p(i)}) + \frac{1}{2}u_{p(i)}\mu_{p(i)} + \frac{1}{2}\lambda_i]q_{ii} \\
& + \sum_{i=1}^{n} \sum_{\substack{j:j>i,\\ \tau(j)=\tau(i)}} [u_i(\lambda_j - u_j\mu_j + 1_{\{j\neq s(i)\}}u_{p(j)}\mu_{p(j)}) + u_j(\lambda_i - u_i\mu_i + 1_{\{i\neq s(j)\}}u_{p(i)}\mu_{p(i)})]q_{ij} \\
& + \sum_{i=1}^{n} (\lambda_i - u_i\mu_i + u_{p(i)}\mu_{p(i)})p_i
\end{aligned} \tag{19}$$

and

$$R_w(i,u) = c_i + (\lambda_i - u_i\mu_i + u_{p(i)}\mu_{p(i)})q_{ii} + \sum_{\substack{j:j>i,\\ \tau(j)=\tau(i)}} (\lambda_i - u_i\mu_i + u_{p(i)}\mu_{p(i)} + \lambda_j - u_j\mu_j + u_{p(j)}\mu_{p(j)})q_{ij}. \quad (20)$$