AN ALGORITHM FOR PRODUCT-FORM LOSS NETWORKS BASED ON NUMERICAL INVERSION OF GENERATING FUNCTIONS

Gagan L. Choudhury, [1] Kin K. Leung [2] and Ward Whitt [3]

[1] AT&T Bell Laboratories, Room 1L-238, Holmdel, NJ 07733-3030
[2] AT&T Bell Laboratories, Room 1K-211, Holmdel, NJ 07733-3030
[3] AT&T Bell Laboratories, Room 2C-178, Murray Hill, NJ 07974-0636

Abstract

We consider a family of product-form loss networks with multiple classes of calls, each of which requires multiple trunks. The calls can use multiple circuits on each trunk (the multi-rate case). There can be upper-limit and guaranteed-minimum sharing policies as well as the standard complete-sharing policy. If all the requirements of a call cannot be met upon arrival, then the call is blocked. We develop an algorithm for computing the (exact) steady-state blocking probability of each traffic class. The algorithm is based on the numerical inversion of generating functions of the normalization constants and a scaling approach for error control. The computational complexity can often be reduced dramatically by exploiting conditional decomposition based on special structure and by appropriately truncating large finite sums. We show that the proposed algorithm is effective by several numerical examples.

1. Introduction

We describe a new algorithm for calculating the (exact) blocking probabilities in a large class of product-form loss networks. In these networks, the blocking probabilities have simple expressions in terms of normalization constants (or partition functions). To calculate the blocking probabilities, we calculate the normalization constants by numerically inverting their generating functions (which we derive).

Following the approach in our previous paper [2], our algorithm here uses the numerical inversion algorithm developed by Abate and Whitt [1] and enhanced by Choudhury, Lucantoni and Whitt [4], which is based on the Fourier-series method. In [2], we developed a full algorithm for a class of closed queueing networks. Now we develop a full version of the numerical inversion algorithm for a class of loss networks.

Our first contribution of this paper is to derive the generating functions of the normalization constants for the loss networks. Our second contribution is to develop an effective static scaling algorithm to control the discretization error. As with the closed queueing networks, the scaling is an essential step since the normalization constant can be very large or very small. The general principles for developing effective scaling apply as before, but we cannot just apply the old scaling algorithm. Here we develop a new scaling algorithm especially tailored to the loss networks. We give a brief description here; a more extensive treatment is given in [3].

In Section 2 we review the model and its product-form steady-state distribution. In Section 3 we present the generating functions and in Section 4 we describe the algorithm. We give numerical examples in Sections 5–6. There we compare our algorithm to previous algorithms, where applicable.

2. The Model and its Product-Form Solution

Loss networks involve trunks, circuits and classes of calls; see Kelly [9] for a review. Consider a loss network with p *trunks* and r *classes of calls*. Let the trunks be indexed by i and the class of call by j. Let trunk i have K_i *circuits*, $1 \le i \le p$, and let $\mathbf{K} \equiv (K_1, \ldots, K_p)$ be the *capacity vector*. Each class-j call requires a_{ij} circuits on trunk i, where a_{ij} is a nonnegative integer. Let $\mathbf{A}$ be the $p \times r$ *requirements matrix* with elements a_{ij}. Let the calls come from independent Poisson processes, with class-j calls having *arrival rate* λ_j. (In forthcoming papers, we treat state-dependent arrival rates and batch arrivals.) Let the call holding times be mutually independent and independent of the arrival processes. Let the *mean holding time* of a type j call be t_j. Thus the *offered load* of class j is $\rho_j \equiv \lambda_j t_j$. Each call is accepted if all desired circuits can be provided; otherwise, the call is blocked and lost. All circuits used by a call are released at the end of the call holding time.

Let the system *state vector* be $\mathbf{n} \equiv (n_1, \ldots, n_r)$, where n_j is the number of class j calls currently in process. Let $S_P(\mathbf{K})$ be the set of allowable states, which depends on the capacity vector $\mathbf{K}$ and the *sharing policy* P. The traditional loss network model assumes *complete sharing* (CS) of the circuits on a trunk among all competing traffic classes. However, it is often desirable to consider other sharing policies to provide different grades of service or to protect one traffic class from another. Thus, we consider two other candidate trunk sharing policies: The first is the *upper limit* (UL) policy, which provides upper limits on the numbers of circuits that can be used by each class on each trunk. The second is the *guaranteed minimum* (GM) policy, which reserves specified numbers of circuits on each trunk for each traffic class.

By the product-form model theory [8], the steady-state probability mass function has the simple product form

$$\pi(\mathbf{n}) = g(\mathbf{K})^{-1} \prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j!}, \qquad (1)$$

with the *normalization constant*

0-7803-1820-X/94 $4.00 © 1994 IEEE

$$g(\mathbf{K}) \equiv g_P(\mathbf{K}) = \sum_{\mathbf{n}\in S_P(\mathbf{K})} \prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j!} . \quad (2)$$

We calculate the normalization constants by numerically inverting the generating function

$$G(\mathbf{z}) \equiv \sum_{K_1=0}^{\infty} \sum_{K_2=0}^{\infty} \cdots \sum_{K_p=0}^{\infty} g(\mathbf{K}) z_1^{K_1} z_2^{K_2} \cdots z_p^{K_p} , \quad (3)$$

where $\mathbf{z}\equiv(z_1,z_2,\ldots,z_p)$ is a vector of complex variables. (For UL and GM policies, there are extra variables in g and thus G.)

A generic formula for the blocking probability for class j is

$$B_j = 1 - \left[\sum_{n\in S_p'(K)} \prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j!} \right] / \left[\sum_{n\in S_p(K)} \prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j!} \right], \quad (4)$$

where $S_p(\mathbf{K})$ is the set of allowable states and $S_p'(\mathbf{K})$ is the subset of $S_p(\mathbf{K})$ in which another class j request is allowed.

Algorithms for computing the normalization constants, and thus the blocking probabilities, have previously been developed only for special cases. For the case of a *single trunk* and the CS policy, Kaufman [7] and Roberts [14] developed an effective recursion. Extensions of this recursion to networks were derived by Dziong and Roberts [6], and Pinsky and Conway [13], but they do not seem so effective. For a class of two-hop tree networks, Mitra [11] performed an asymptotic analysis and developed bounds, while Kogan [10] developed an algorithm for that model by relating it to a closed queueing network. Other algorithms for tree networks have been developed by Tsang and Ross [16] and Ross and Tsang [15]. Our algorithm applies to these tree networks as well and more; we discuss tree network examples in Section 6. Chuah [5] recently developed a recursive algorithm for a single trunk with the UL policy, which evidently extends to two trunks.

3. Closed-Form Expressions for the Generating Functions

With the CS sharing policy, the set of allowable states is $S_{CS}(\mathbf{K}) = \{\mathbf{n}\in \mathbf{Z}_+^r: \mathbf{An}' \le \mathbf{K}\}$, where $\mathbf{Z}_+$ is the set of nonnegative integers and $\mathbf{Z}_+^r$ is its r-fold cartesian product. By (4), the stationary blocking probability of an arbitrary type j request is given by

$$B_j = 1 - g(\mathbf{K}-\mathbf{Ae}_j')/g(\mathbf{K}), \quad (5)$$

where $\mathbf{e}_j$ is an r-dimensional *row vector* with a 1 in the j^{th} place and 0's elsewhere, and $\mathbf{e}_j'$ is its transpose.

We calculate B_j in (5) by calculating $g(\mathbf{K}-\mathbf{Ae}_j')$ and $g(\mathbf{K})$. By (2) and (3), and changing the order of summation, we obtain

$$G(\mathbf{z}) = \sum_{n_1=0}^{\infty} \cdots \sum_{n_r=0}^{\infty} \sum_{K_1=\sum_{j=1}^{r} a_{1j}n_j}^{\infty} \cdots \sum_{K_p=\sum_{j=1}^{r} a_{pj}n_j}^{\infty} \prod_{j=1}^{r} \frac{\rho_j^{n_j}}{n_j!} z_1^{K_1} \cdots z_p^{K_p}$$

$$= \exp\left[\sum_{j=1}^{r} \rho_j \prod_{i=1}^{p} z_i^{a_{ij}} \right] / \prod_{i=1}^{p} (1 - z_i) . \quad (6)$$

We now assume that, in addition to the overall capacity constraint provided by $\mathbf{K}$, an upper limit L_{ij} is imposed on class-j calls on trunk i for each (i,j) pair. Call trunk i the *type-j limiting trunk* if $\left\lfloor L_{ij}/a_{ij} \right\rfloor \le \left\lfloor L_{i'j}/a_{i'j} \right\rfloor$ for all $1 \le i' \le p$, and let $M_j = L_{ij}/a_{ij}$. Note that $S_{UL}(\mathbf{K},\mathbf{M}) = \{\mathbf{n}\in \mathbf{Z}_+^r : \mathbf{An}' \le \mathbf{K}\ ,\ \mathbf{n}' \le \mathbf{M}\}$. Let $g(\mathbf{K},\mathbf{M})$ be the normalization constant as a function of the pair $(\mathbf{K},\mathbf{M})$. Paralleling (5), the blocking probability with UL is

$$B_j = 1 - g(\mathbf{K}-\mathbf{Ae}_j',\mathbf{M}-\mathbf{e}_j')/g(\mathbf{K},\mathbf{M}) . \quad (7)$$

Since $(\mathbf{K},\mathbf{M})$ is of dimension $p + r$, so will be the generating function of $g(\mathbf{K},\mathbf{M})$. Let $\mathbf{y} \equiv (y_1,\ldots,y_r)$ be a vector of complex variables which we will use for the last r dimensions. Then, similar to the CS policy, we obtain the generating function of $g(\mathbf{K},\mathbf{M})$ in a compact form:

$$G(\mathbf{z},\mathbf{y}) = \left[\prod_{i=1}^{p} \frac{1}{1-z_i} \right] \left[\prod_{j=1}^{r} \frac{1}{1-y_j} \right] \exp\left[\sum_{j=1}^{r} \rho_j y_j \prod_{i=1}^{p} z_i^{a_{ij}} \right] \quad (8)$$

With the GM policy, a certain number of circuits on each trunk are reserved for each type of call. To obtain a tractable expression in this case, we assume that there is a vector $\mathbf{b} \equiv (b_1,\ldots,b_r)$ such that a_{ij} equals either b_j or 0 for all i and j. Moreover, we assume that the number of circuits guaranteed for each type of call is the same for all trunks used by that call class. Let M_j be the capacity guaranteed to class j in each of its required trunks. We continue to use $\mathbf{M}$ to denote the vector $(M_1,\ldots,M_r)$. Given $\mathbf{K}$ and $\mathbf{M}$, let $S_{GM}(\mathbf{K},\mathbf{M})$ be the set of all allowable system states under the GM policy. Then

$$S_{GM}(\mathbf{K},\mathbf{M}) = \{\mathbf{n}\in \mathbf{Z}_+^r : \sum_{j=1}^{r} \max(a_{ij}n_j, \delta_{ij}M_j) \le K_i \text{ for } i = 1 \text{ to } p\}.$$

Let $g(\mathbf{K},\mathbf{M})$ be the normalization constant. Paralleling (5) and (7), the blocking probability now, by (4), is

$$B_j = 1 - g(\mathbf{K}-\mathbf{Ae}_j',\mathbf{M}-\mathbf{Be}_j')/g(\mathbf{K},\mathbf{M}) . \quad (9)$$

Similar to the CS and UL policies, we obtain the generating function of the normalization constant for the GM policy as

$$G(\mathbf{z},\mathbf{y}) = \prod_{i=1}^{p} \frac{1}{1-z_i} \prod_{j=1}^{r} \{ \frac{\exp(\rho_j \prod_{i=1}^{p} z_i^{a_{ij}}) - y_j \exp(\rho_j y_j^{b_j} \prod_{i=1}^{p} z_i^{a_{ij}})}{1-y_j}$$
$$+ \frac{y_j \prod_{i=1}^{p} z_i^{\delta_{ij}} \exp(\rho_j y_j^{b_j} \prod_{i=1}^{p} z_i^{\delta_{ij}b_j})}{1 - y_j \prod_{i=1}^{p} z_i^{\delta_{ij}}} \}. \quad (10)$$

In [3], we also treat the case of mixed sharing policies. Furthermore, it is possible to have UL and GM in the same model (see a forthcoming paper).

4. The Numerical Inversion Algorithm

In this section, we briefly review the numerical inversion algorithm; for background see [1,2,4] and for more details see [3]. Throughout we use the notation in (3); thus our goal is to calculate $g(\mathbf{K})$ given $G(\mathbf{z})$ where $\mathbf{K}$ and $\mathbf{z}$ are p-dimensional.

Dimension Reduction

Our approach to inverting p-dimensional generating functions is to recursively perform p one-dimensional

inversions. Generally, the computational complexity is exponential in p, but a dramatic reduction often occurs due to special structure if we perform the one-dimensional inversions in a good order. Details on the dimension reduction by conditional decomposition are given in [2,3].

For example, the generating function for the UL policy in (8) appears to require a $(p+r)$-dimensional inversion, but exploiting special structure, the problem can be reduced to a $(p+1)$-dimensional or even an p-dimensional inversion. Specifically, if we invert all the z_i variables first, then the inversions for the different y_j variables clearly can be done separately. This reduces the dimension to $p + 1$.

The Basic Algorithm

Given that the order of inversion has been specified, we perform (up to) p one-dimensional inversions recursively. To represent the recursive inversion, we define *partial generating functions* by

$$g^{(j)}(\mathbf{z}_j,\mathbf{K}_{j+1}) = \sum_{K_1=0}^{\infty} \cdots \sum_{K_j=0}^{\infty} g(\mathbf{K}) \prod_{i=1}^{j} z_i^{K_i} \text{ for } 1\le j\le p, \quad (11)$$

where $\mathbf{z}_j=(z_1,z_2,\ldots,z_j)$ and $\mathbf{K}_j=(K_j,K_{j+1},\ldots,K_p)$ for $1\le j\le p$. Let $\mathbf{z}_0$ and $\mathbf{K}_{p+1}$ be null vectors. Clearly, $\mathbf{K}=\mathbf{K}_1$, $\mathbf{z}=\mathbf{z}_p$, $g^{(p)}(\mathbf{z}_p,\mathbf{K}_{p+1})=G(\mathbf{z})$ and $g^{(0)}(\mathbf{z}_0,\mathbf{K}_1)=g(\mathbf{K})$.

Let I_j represent inversion with respect to z_j. Then the step-by-step nested inversion approach is

$$g^{(j-1)}(\mathbf{z}_{j-1},\mathbf{K}_j) = I_j[g^{(j)}(\mathbf{z}_j,\mathbf{K}_{j+1})], \quad 1 \le j \le p, \quad (12)$$

starting with $j=p$ and decreasing j by 1 each step. Our program implements this nested inversion in a recursive manner.

In each step we use the LATTICE-POISSON inversion algorithm in [1] with modifications to improve precision and allow for complex inverse functions as in [4]. We show below the inversion formula at the j^{th} step. For simplicity, we suppress those arguments which remain constant during this inversion, letting $g_j(K_j)=g^{(j-1)}(\mathbf{z}_{j-1},\mathbf{K}_j)$ and $G_j(z_j)=g^{(j)}(\mathbf{z}_j,\mathbf{K}_{j+1})$. With this notation, the inversion formula (12) is

$$g_j(K_j) = \frac{1}{2l_jK_jr_j^{K_j}} \sum_{k=-l_jK_j}^{l_jK_j-1} G_j(r_je^{\pi ik/l_jK_j})e^{-\pi ik/l_j} - e_j, \quad (13)$$

where $i=\sqrt{-1}$, l_j is a positive integer and e_j represents the *aliasing error*, which is given by

$$e_j = \sum_{n=1}^{\infty} g_j(K_j + 2nl_jK_j)r_j^{2nl_jK_j}. \quad (14)$$

To control the aliasing error in (14), we choose $r_j = 10^{-a_j}$ for $a_j = \gamma_j/(2l_jK_j)$. From (14), we see that a bigger γ_j decreases the aliasing error. Also, as explained in [4], the parameter l_j controls roundoff error, with bigger values causing less roundoff error.

Scaling

When the inverse function is a probability, the aliasing error e_j can easily be bounded. In contrast, here the normalization constants may be arbitrarily large and therefore the aliasing error e_j may also be arbitrarily large. Thus, we scale the generating function in each step by defining a *scaled generating function* and associated scaled normalization constants by

$$\bar{G}_j(z_j)=\alpha_{0j}G_j(\alpha_jz_j) \text{ and } \bar{g}_j(K_j)=\alpha_{0j}\alpha_j^{K_j}g_j(K_j) \quad (15)$$

where α_{0j} and α_j are positive real numbers. We invert this scaled generating function after choosing α_{0j} and α_j so that the errors are suitably controlled.

We choose the parameters α_{0j} and α_j in (15) to control the aliasing error

$$\bar{e}_j = \sum_{n=1}^{\infty} \bar{g}_j(K_j + 2nl_jK_j)10^{-\gamma_jn}. \quad (16)$$

Since we are interested in ratios of normalization constants, we focus on *relative errors* $e_j' = \bar{e}_j / \bar{g}_j(K_j)$, which is bounded by

$$|e_j'| \le \sum_{n=1}^{\infty} \left|\frac{\bar{g}_j(K_j + 2nl_jK_j)}{\bar{g}_j(K_j)}\right| 10^{-\gamma_jn}. \quad (17)$$

$$\text{Let} \quad C_j = \max_n \left\{ \left| \bar{g}_j(K_j + 2nl_jK_j) / \bar{g}_j(K_j) \right| \right\}^{1/n}$$

$$= \alpha_j^{2l_jK_j} \max_n \left\{ \left| g_j(K_j+2nl_jK_j) / g_j(K_j) \right| \right\}^{1/n}. \quad (18)$$

$$\text{Then} \quad |e_j'| \le \sum_{n=1}^{\infty} C_j^n 10^{-\gamma_jn} \le \frac{C_j10^{-\gamma_j}}{1-C_j10^{-\gamma_j}} \approx C_j10^{-\gamma_j}. \quad (19)$$

Note that C_j in (18) is independent of α_{0j}. We use the second parameter α_{0j} mainly to keep $\bar{g}_j(K_j)$ in (15) close to 1, so as to avoid numerical underflow or overflow.

Hence, our main goal is to choose α_j so that $C_j \ll 10^{\gamma_j}$. Of course, in general we do not know $g_j(K_j)$ and thus we do not know C_j. However, we aim to achieve $C_j \ll 10^{\gamma_j}$ by roughly controlling the growth rate of $\bar{g}_j(K_j)$, or its fastest growing term, exploiting the structure of the generating function.

For the CS policy we choose scaling parameters α_i, $1 \le i \le p$, so that they satisfy the unequalities $0 < \alpha_i \le 1$ and

$$\sum_{j=1}^{r} \rho_j \prod_{k=1}^{p} \alpha_k^{a_{kj}} \prod_{k=1}^{i-1} r_k^{a_{kj}} a_{ij} \le K_i, \quad 1 \le i \le p. \quad (20)$$

Once the scaling variables α_i have been obtained, we obtain α_{0i} recursively starting with $i = p$ by

$$\prod_{k=i}^{p} \alpha_{0k} = \exp\left[-\sum_{j=1}^{r} \rho_j \prod_{k=1}^{p} \alpha_k^{a_{kj}} \prod_{k=1}^{i-1} r_k^{a_{kj}} \right]. \quad (21)$$

To find a maximal vector $(\alpha_1,\ldots,\alpha_p)$ satisfying (20), we start with $i = p$ and successively decrease i. We use the fact that the left side of (20) is monotone in α_i. When $i = l$, the values of α_i for $i \ge l + 1$ are known. We approximate by acting as if $\alpha_i = 1$ for $i \le l-1$ and find α_l satisfying the constraint for $i = l$ in (20). When we are done we obtain a maximal vector; i.e., if any α_i is less than 1, then at least one constraint in (20) is necessarily satisfied as an equality. Hence, we cannot increase the vector without violating a constraint. However, in general there may be many maximal vectors. We could obtain alternative, perhaps more balanced, scaling vectors $(\alpha_1,\ldots,\alpha_p)$ by iterating, letting α_i for $i \le l-1$ be the old

value instead of 1, but we have not found this refinement to be necessary. We discuss the (largely heuristic) derivation of (20) and (21) in [3].

For the UL policy the generating function in (8) has a form similar to the CS generating function in (6) if we treat the y_j variables like the z_i variables. Thus the scaling for UL is straightforward extension of CS scaling in (20) and (21). The generating function for the GM policy in (10) is more complicated. Hence, for scaling only, we treat GM by acting as if it were UL. For each class j on each resource i, we use the upper limit obtained by subtracting the guaranteed minima of all other classes from the capacity. We then use the UL scaling with these upper limits. This procedure is exact for two classes, but a heuristic approximation for more than two classes.

Other Algorithm Issues

As can be seen from (13), the inversion formula in each dimension is a sum of $2l_iK_i$ terms. If K_i is large, then it is natural to look for ways to accelerate convergence of the finite sum. For this purpose, we find that *truncation* is effective. The inversion formula in each step is a weighted sum of generating function values evaluated over equidistant points along the circumference of a circle. The weights are complex numbers, but they have constant amplitude. As the capacities K_i grow, the amplitude of the generating function typically becomes unevenly distributed along the circumference of the circle. There are several local maximum points and the amplitude drops sharply away from these points. If we can identify all the relative maximum points, and then consider only those points around them that have non-negligible relative amplitude, we can obtain a significant reduction in computation. A general procedure is developed in [3].

In order to compute the blocking probability for each of the r classes, the computational complexity is $O(r^2)$, because $r + 1$ normalization constant values have to be calculated and the computation required for each is $O(r)$. However, for large capacity vectors $\mathbf{K}$ it is possible to compute the $r + 1$ normalization constants simultaneously with the *bulk of the computations shared,* so that the required computation for all normalization constants is only slightly more than for one. This reduces the overall complexity from $O(r^2)$ to $O(r)$.

We now roughly analyze the *computational complexity* of the inversion algorithm. For simplicity, assume that the capacity of each resource is K. Let C_P represent the computational complexity for sharing policy P, where P may be CS, UL or GM. The main computational burden is carrying out the p-fold nested inversion in (13). Other work, such as finding the scale parameters is insignificant compared to that. A straightforward application of our algorithm in the CS case to compute one normalization constant would require $O(K^p)$ evaluations of the generating function, each of which would involve $O(r)$ work. In order to compute the blocking probability for each class, we need to compute $r + 1$ normalization constants, but we have just shown that all this work can be done in time $O(1)$ by sharing the bulk of the computation (requiring storage only of $O(r)$). Without further enhancements, this yields $C_{CS} = O(rK^p)$.

However, we can use truncation to reduce K to $\bar{K}$ and, with special structure, we can reduce p to $\bar{p} << p$. So, finally, we get

$$C_{CS} = O(r\bar{K}^{\bar{p}}) \tag{22}$$

where $\bar{K} \leq K$ and $\bar{K} << K$ for large K

$$\bar{p} \leq p \text{ and } \bar{p} << p \text{ with special structure .} \tag{23}$$

In contrast, the algorithm in [6] and [13] has $C_{CS} = O(rK^p)$. Similarly, for the other two policies (exploiting conditional decomposition as described in Section 3.1), we get

$$C_{UL} = C_{GM} = O(r\bar{K}^{\bar{p}+1}) \tag{24}$$

for $\bar{K}$ and $\bar{p}$ in (23).

In all cases the storage requirement is $O(r)$, which is relatively low.

5. Examples of a Single Trunk

We now give several examples. All computations are done on a SUN SPARC-2 workstation.

Our first numerical example is the classical resource-sharing model involving a single trunk and the CS sharing policy. We present a brief summary of results here; details are reported in [3]. As a basis for comparison, we also implemented the recursive algorithm of Kaufman [7] and Roberts [14] and the uniform asymptotic approximation (UAA) of Mitra and Morrison [12]. For the specific set of model parameters, when the number of circuits in the trunk (denoted by K) can be handled by the recursion algorithms, results of the inversion and recursion algorithms agreed well beyond eight significant digits. However, when K exceeds a certain value, the recursions either had numerical underflows or overflows, or took too long to run. On the other hand, our inversion results agreed closely with UAA, and, as expected, the agreement improves as K increases. In this example, we also checked the accuracy of the inversion algorithm by running it twice, with $l_1 = 1$ and $l_1 = 2$. Our algorithm took less than half a second for this example.

We continue to consider a single trunk, but now with the UL and GM sharing policies as well as the CS policy. The generating functions for the UL and GM policies are given in (8) and (10), and the blocking probabilities are given in (7) and (9).

We first consider a smaller example, for which we can apply a direct algorithm (developed in Section 6 of [3]) as well as the inversion algorithm. We let the capacity be $K = 150$ and consider 5 classes. The 5 classes require 1,2,3,4 and 5 circuits per call, respectively, and the corresponding offered loads are 20,15,12,10 and 9. For UL, the class limits are 20,30,50,60 and 70. For GM, the corresponding guaranteed minima are 5,18,25,36 and 40.

The blocking probability for each class and each sharing policy is shown in Table 1. The direct and inversion algorithms agreed to at least 12 digits in each case. This example thus serves to validate the generating functions and both algorithms. For this smaller example, the inversion algorithm runs in less than a second while the direct algorithm runs in minutes.

sharing policy	class				
	1	2	3	4	5
CS	0.060513	0.118849	0.174972	0.228857	0.280487
UL	0.1760450	0.214567	0.162187	0.195747	0.239105
GM	0.148226	0.254277	0.285198	0.216798	0.244159

Table 1. Blocking probabilities for $K = 150$.

Next we consider a larger example with capacity $K = 600$ and 10 classes, for which the inversion algorithm runs in seconds, while the direct algorithm can no longer run in reasonable time. Class j requires j units, $1 \le j \le 10$. The vector of offered loads for the 10 classes is (30,25,20,18,16,14,13,12,11,10). The vector class limits with the UL policy is (30,50,60,80,90,100,110,120, 130,140), while the associated vector of guaranteed minima for the GM policy is (5,10,20,30,40, 50,60,70,80,100).

The blocking probabilities for classes 1, 2 and 10 for each sharing policy are given in Table 2. For the CS policy, the inversion algorithm was validated by applying the Kaufman [7] and Roberts [14] recursion. For the UL and GM policies, the inversion was validated by performing two separate inversions with the parameters $l_1 = 1$ and $l_1 = 2$. There was agreement to at least 11 digits in each case.

sharing policy	class		
	1	2	10
CS	0.0427417	0.0838918	0.3613882
UL	0.1455476	0.1706272	0.3199203
GM	0.0973615	0.1861317	0.1865667

Table 2. Blocking probabilities for $K = 600$.

6. Examples of Networks with Special Structure

Many networks have special structure allowing drastic dimension reduction. We consider here two such structures, to be referred to as *structure A* and *B*, but clearly others are also possible.

In structure A, commonly referred to as a tree network, class j calls require a_{ij} circuits on trunk i, where a_{ij} is allowed to be non-zero only for at most two values of i, one of which has to be p. If both the non-zero values of a_{ij} are 1 then that corresponds to a single-rate tree network, considered by Mitra [11] and Kogan [10]. Tsang and Ross [16] considered the multi-rate case in which the two non-zero values of a_{ij} are the same but may be bigger than 1. Ross and Tsang [15] allow the two possible non-zero values of a_{ij} to be either the same or one of them to be zero. We consider a further generalization in that we allow the two values of a_{ij} to be different, without necessarily requiring one of them to be zero. Furthermore, all the earlier work was restricted to the CS policy, whereas we allow the UL and GM sharing policies.

In structure B, evidently not considered earlier, for each class j calls, a_{ij} is allowed to be non-zero for at most 3 values of i, one of which has to be q and the other two have to be k and $q + k$, for $1 \le k \le q-1$.

In structure A, divide the traffic classes according to the trunks they use by letting $1 \equiv r_0 < r_1 < r_2 < \ldots < r_{p-1} \equiv r$; then class j uses trunks k and p if $r_{k-1}+1 \le j \le r_k$. Similarly, in structure B, let $1 \equiv r_0 < r_1 < r_2 \ldots < r_{q-1} \equiv r$. traffic class j uses trunks k, q and $q+k$ if $r_{k-1}+1 \le j \le r_k$. By (6), the generating functions for structures A and B with Poisson arrivals and the CS policy are, respectively,

$$G(z) = \frac{1}{1-z_p} \prod_{k=1}^{p-1} \frac{\exp\left(\sum_{j=r_{k-1}}^{r_k} \rho_j z_k^{a_{kj}} z_p^{a_{pj}}\right)}{1-z_k} \qquad (25)$$

and

$$G(z) = \frac{1}{1-z_q} \prod_{k=1}^{q-1} \frac{\exp\left(\sum_{j=r_{k-1}+1}^{r_k} \rho_j z_k^{a_{kj}} z_q^{a_{qj}} z_{q+k}^{a_{q+k,j}}\right)}{(1-z_k)(1-z_{q+k})}. \qquad (26)$$

In (25), if we fix z_p, then each term within the product becomes independent of others and requires one-dimensional inversion. The overall inversion thereby is two-dimensional. In (26) if we fix z_q each term within the product is independent of the others and requires two-dimensional inversion. The overall inversion thereby is three-dimensional.

We first provide numerical examples for structure A. We start with the single-rate tree network example on p. 235 of Mitra [11] using the CS policy. Here $a_{ij} = 1$ whenever it is non-zero, $r_k = k$, $r = 7$ and $p = 8$. The capacity vector is $K = (30,30,20,20,15,15,15,134)$. By inverting z_1 to z_7 analytically in (25) with $p = 8$, the blocking probabilities can be obtained by one-dimensional inversion (i.e., inverting z_8) in a fraction of a second. The results are displayed in Table 3. We see that our results are between the lower and upper bounds (denoted by LB and UB) by Mitra's method [11] in each case. Interestingly the results are very close to the mean of the two bounds, which Mitra suggested as an estimate. The last column of Table 3 also shows results of a more challenging example where the values of ρ_j's and K_i's are increased by 100 times.

class		Mitra example			larger example
j	ρ_j	LB	inversion	UB	inversion
1	35	0.2201	0.220749	0.2214	0.144901
2	30	0.1333	0.134067	0.1347	0.029743
3	25	0.2801	0.280729	0.2813	0.201814
4	17	0.0880	0.088875	0.0896	0.028701
5	20	0.3302	0.330783	0.3313	0.251658
6	15	0.1805	0.182201	0.1828	0.033965
7	9	0.0257	0.026632	0.0274	0.028701

Table 3. Blocking probs. in the tree network with CS policy.

We now consider more general tree networks with non-CS policies and multiple rates, for which existing methods do not apply. We allow more than one class to use a non-common trunk and the requirements of each class for the two trunks not to be identical.

This example has 6 trunks, with the sixth trunk being the common trunk. The capacity vector is $\mathbf{K}=(15,25,25,30,$

20,90). Our example has 15 classes, with class j using trunks $\lceil j/3 \rceil$ and 6 (possibly at a 0 level), where $\lceil x \rceil$ is the least integer greater than or equal to x (i.e., classes 1,2 and 3 use trunks 1 and 6, classes 4,5 and 6 use trunks 2 and 6, etc.).

The specific offered loads and requirements for each class are given in Table 4. We consider both the CS and UL sharing policies. The limits for the UL policy are also given in Table 4. (These are not used with the CS policy.) The blocking probability for each class is given in Table 4.

model parameters						blocking probabilities	
j	ρ_j	$a_{\lceil j/3 \rceil, j}$	$L_{\lceil j/3 \rceil, j}$	$a_{6,j}$	$L_{6,j}$	CS	UL
1	10	1	10	0	0	0.328927	0.333529
2	10	0	0	1	15	0.004393	0.036827
3	10	1	10	1	15	0.331579	0.333777
4	5	1	10	1	10	0.109389	0.104160
5	5	2	15	2	15	0.219572	0.225724
6	5	2	15	1	10	0.216223	0.225384
7	4	1	8	1	8	0.112727	0.114086
8	4	3	16	3	16	0.326724	0.330792
9	4	3	16	1	10	0.320927	0.330258
10	3	1	7	1	7	0.061176	0.025887
11	3	4	15	4	15	0.236906	0.347366
12	3	4	15	1	8	0.226213	0.347187
13	2	1	6	1	6	0.075576	0.046674
14	2	5	13	5	13	0.433225	0.465534
15	2	5	13	1	5	0.422868	0.465049

Table 4. Blocking probability in a multi-rate tree network with the CS and UL sharing policies.

We obtain the generating function for CS from (6). It is

$$G(\mathbf{z}) = \exp\left[\sum_{j=1}^{15} \rho_j z_{\lceil j/3 \rceil}^{a_{\lceil j/3 \rceil, j}} z_6^{a_{6j}}\right] \Big/ \prod_{i=1}^{6} (1 - z_i) \ . \qquad (27)$$

We employ conditional decomposition to reduce the dimension from 6 to 2; i.e., for any fixed value of z_6, the generating function can be represented as a product of 5 factors with z_i appearing only in the i^{th} factor. We obtain the generating function for the UL policy from (8):

$$G(\mathbf{z},\mathbf{y}) = \exp\left(\sum_{j=1}^{15} \rho_j y_j^{b_j} z_{\lceil j/3 \rceil}^{a_{\lceil j/3 \rceil, j}} z_6^{a_{6,j}}\right) \Big/ \prod_{i=1}^{6} (1 - z_i) \prod_{j=1}^{15} (1 - y_j). \quad (28)$$

Conditional decomposition reduces the dimension from 21 to 3. As with the CS policy, we first invert z_6, then we invert the other z_i variable in each factor. For any fixed values of the two z variables, the generating function can be represented as a product of 15 factors with y_j appearing only in the j^{th} factor.

The computation of all blocking probabilities took several seconds. We can increase the numbers of classes, trunks and capacities each by factors of ten and still carry out the computations in several minutes using truncation.

References

[1] ABATE, J. and WHITT, W. 1992. The Fourier-series method for inverting transforms of probability distributions. *Queueing Systems* **10,** 5-88.

[2] CHOUDHURY, G. L., LEUNG, K. K. and WHITT, W. 1993. Calculating Normalization Constants of Closed Queueing Networks by Numerically Inverting Their Generating Functions. *J. ACM* to appear.

[3] CHOUDHURY, G. L., LEUNG, K. K. and WHITT, W. 1994. An Algorithm to Compute Blocking Probabilities in Multi-Rate Multi-Class Multi-Resource Loss Models. *Adv. Applied Prob.* to appear.

[4] CHOUDHURY, G. L., LUCANTONI, D. M. and WHITT, W. 1994. Multidimensional Transform Inversion With Applications to the Transient M/G/1 Queue. *Ann. Appl. Prob.* to appear.

[5] CHUAH, M. C. 1993. General Pricing Framework for Multiple Service, Multiple Resource Systems. AT&T Bell Laboratories, Holmdel, New Jersey.

[6] DZIONG, Z., and ROBERTS, J. W. 1987. Congestion Probabilities in a Circuit-Switched Integrated Services Network. *Perf. Eval.* **7,** 267-284.

[7] KAUFMAN, J. S. 1981. Blocking in a Shared Resource Environment. *IEEE Trans. Commun.* **COM-29,** 1474-1481.

[8] KELLY, F. P. 1979. *Reversibility and Stochastic Networks,* Wiley, New York.

[9] KELLY, F. P. 1991. Loss Networks. *Ann. Appl. Prob.* **1,** 319-378.

[10] KOGAN, Y. 1989. Exact Analysis for a Class of Simple, Circuit-Switched Networks with Blocking. *Adv. Appl. Prob.* **21,** 952-955.

[11] MITRA, D. 1987. Asymptotic Analysis and Computational Methods for a Class of Simple, Circuit-Switched Networks with Blocking. *Adv. Appl. Prob.* **19,** 291-239.

[12] MITRA, D., and MORRISON, J. A. 1993. Erlang Capacity and Uniform Approximations for Shared Unbuffered Resources. AT&T Bell Laboratories, Murray Hill, New Jersey.

[13] PINSKY, E., and CONWAY, A. E. 1992. Computational Algorithms for Blocking Probabilities in Circuit-Switched Networks. *Ann. Opns. Res.* 35, 31-41.

[14] ROBERTS, J. W. 1981. A Service System with Heterogeneous User Requirements. *Perf. of Data Commun. Systems and their Applications,* G. Pujolle (Ed.), North-Holland Publishing, 423-431.

[15] ROSS, K. W., and TSANG, D. 1990. Teletraffic Engineering for Product-Form Circuit-Switched Networks. *Adv. Appl. Prob.* **22,** 657-675.

[16] TSANG, D., and ROSS, K. W. 1990. Algorithms to Determine Exact Blocking Probabilities for Multirate Tree Networks. *IEEE Trans. Commun.* **38,** 1266-1271.