Calculating Normalization Constants of Closed Queuing Networks by Numerically Inverting Their Generating Functions

GAGAN L. CHOUDHURY AND KIN K. LEUNG

AT&T Bell Laboratories, Holmdel, New Jersey

AND

WARD WHITT

AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. A new algorithm is developed for calculating normalization constants (partition functions) and moments of product-form steady-state distributions of closed queuing networks and related models. The essential idea is to numerically invert the generating function of the normalization constant and related generating functions appearing in expressions for the moments. It is known that the generating function of the normalization constant often has a remarkably simple form, but numerical inversion evidently has not been considered before. For p-dimensional transforms, as occur with queuing networks having p closed chains, the algorithm recursively performs p one-dimensional inversions. The required computation grows exponentially in the dimension, but the dimension can often be reduced by exploiting conditional decomposition based on special structure. For large populations, the inversion algorithm is made more efficient by computing large sums using Euler summation. The inversion algorithm also has a very low storage requirement. A key ingredient in the inversion algorithm is scaling. An effective static scaling is developed for multichain closed queuing networks with only single-server and (optionally) infinite-server queues. An important feature of the inversion algorithm is a self-contained accuracy check, which allows the results to be verified in the absence of alternative algorithms.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: *modeling techniques*; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems—*computation of transforms*; G.1.4 [**Numerical Analysis**]: Quadrature and Numerical Differentiation—*multiple quadrature*; G.1.m [**Numerical Analysis**]: *miscellaneous*

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Closed queuing networks, dimension reduction, Euler summation, generating function, normalization constant, numerical transform inversion, partition function, performance analysis, production-form model, scaling.

Authors' addresses: G. L. Choudhury, AT&T Bell Laboratories, Room 1L-238, Holmdel, NJ 07733-3030, e-mail: gagan@buckaroo.att.com; K. K. Leung, AT&T Bell Laboratories, Room 1L-215, Holmdel, NJ 07733-3030, e-mail: kin@buckaroo.att.com; W. Whitt, AT&T Bell Laboratories, Room 2C-178, Murray Hill, NJ 07974-0636, e-mail: wow@research.att.com.

1. *Introduction*

For over twenty years, closed queuing networks and related product–form models have played a major role in the performance analysis of computer systems, communication networks and other complex systems.[1] The success of these models has largely been due to the excellent algorithms for computing the steady-state performance measures that have been developed, such as the convolution algorithm [Buzen 1973; Reiser and Kobayashi 1975], the mean value analysis (MVA) algorithm [Reiser and Lavenberg 1980], the tree convolution algorithm [Lam and Lien 1983], the recursion by chain algorithm (RECAL) [Conway and Georganas 1986a, 1986b], the mean value analysis by chain (MVAC) algorithm [Conway et al. 1989], and the distribution analysis by chain (DAC) algorithm [de Souza e Silva and Lavenberg 1989]. See Bruel and Balbo [1980], Conway and Georganas [1989], and Lavenberg [1983] for an overview.

While these algorithms for closed queuing networks have significant differences, they can all be related by their common recursive approach [Conway and Georganas 1989]. These algorithms have been very successful, but they do encounter difficulties when the model becomes large in one way or another, for example, have many chains, many queues or large populations. Thus, special approaches for analyzing large closed networks also have been developed, such as the algorithm PANACEA based on asymptotic expansions of integral representations [McKenna and Mitra 1982; 1984; McKenna et al. 1981; Ramakrishnan and Mitra 1982]. Other asymptotic methods for large models have also been developed [Birman et al. 1992; Knessl and Tier 1992; Kogan 1992; Kogan and Birman 1992].

In this article, we propose a radically different algorithm for calculating the performance measures of closed queuing networks and related product–form models, which we believe usefully complements existing algorithms, because it applies to both large and small models. In contrast to the recursive approach of the nonasymptotic algorithms above, we directly calculate the difficult normalization constant or partition function at a desired argument (total population vector) by numerically inverting the generating function of the normalization constant. Moreover, we directly calculate mean queue lengths by performing only two inversions. One consequence of this direct approach is a very low storage requirement.

To show that our algorithm can usefully complement existing algorithms, we solve some challenging examples. For instance, Example 8.4 has 1000 queues, 4000 jobs, and 11 closed chains with the product of the chain populations being more than 10^{26}. This example was solved in less than a minute by our algorithm on a SUN SPARC-2 workstation. Some models of this size can also be handled nicely by the asymptotic approaches in Knessl and Tier [1992] and McKenna and Mitra [1982], but we do not need to have any infinite-server queues or be in the normal usage regime as required by McKenna and Mitra [1982], and we do not need to have all chain populations be large as required by Knessl and Tier [1992]. Moreover, we do not need to revert back to one of the other algorithms when the model is small.

We now describe the general class of probability distributions that we consider. Here we do this abstractly; in Section 4, we will consider a special

[1] See, for example, Bruel and Balbo [1980], Conway and Georganas [1989], Kelly [1979], Lavenberg [1983], and Whittle [1986].

class of closed queuing networks. Now let the state variable be a *job vector* $\mathbf{n} = (n_1, \ldots, n_L)$; n_l is the number of jobs of *type l*; n_l might be the number of customers of a particular class at a particular queue. Let there be a specified *population vector* $\mathbf{K} = (K_1, \ldots, K_p)$; K_j is the population of *chain j*, a fixed quantity specified as part of the model data. The *state space* is the set of allowable job vectors, which depends on $\mathbf{K}$ and is denoted by $S(\mathbf{K})$. In this setting, the probability distributions that we consider have the form

$$p(\mathbf{n}) = g(\mathbf{K})^{-1} f(\mathbf{n}), \tag{1.1}$$

where

$$g(\mathbf{K}) = \sum_{n \in S(\mathbf{K})} f(\mathbf{n}) \tag{1.2}$$

and f is a (known) nonnegative real-valued function on the L-fold product of the nonnegative integers. (For example, we might have $f(\mathbf{n}) = \prod_{l=1}^{L} f_l(n_l)$ with $f_l(n_l) = \rho_l^{n_l}$.) The term $g(\mathbf{K})$ in (1.1) and (1.2) is called the *normalization constant* or the *partition function*. For the closed queuing network models (and many other models), we will consider, the state space has the special form

$$S(\mathbf{K}) = \left\{ \mathbf{n} | n_l \geq 0, \sum_{l \in C_j} n_l = K_j, 1 \leq j \leq p \right\} \tag{1.3}$$

for special sets C_j, $1 \leq j \leq p$.

Given a probability distribution as in (1.1), where the function f is relatively tractable, the major complication is determining the normalization constant $g(\mathbf{K})$. In this setting, the convolution algorithm calculates $g(\mathbf{K})$ by expressing it in terms of values $g(\mathbf{K}')$ where $\mathbf{K}' < \mathbf{K}$ (i.e., $K_l' \leq K_l$ for all l and $K_l' < K_l$ for at least one l) [Bruel and Balbo 1980; Conway and Georganas 1989; Lavenberg 1983]. Other existing nonasymptotic algorithms proceed in a similar recursive manner. See Conway and Georganas [1989] for a unified view.

In contrast, we calculate $g(\mathbf{K})$ by numerically inverting its generating function

$$G(\mathbf{z}) \equiv \sum_{K_1=0}^{\infty} \cdots \sum_{K_p=0}^{\infty} g(\mathbf{K}) \prod_{j=1}^{p} z_j^{K_j} \tag{1.4}$$

where $\mathbf{z} \equiv (z_1, \ldots, z_p)$ is a vector of complex variables. To quickly see the potential advantage of this approach, note that we can calculate $g(\mathbf{K})$ for one vector $\mathbf{K}$ without calculating $g(\mathbf{K}')$ for all the $\prod_{j=1}^{p} K_j$ nonnegative vectors $\mathbf{K}'$ less than or equal to $\mathbf{K}$.

There are two obvious requirements for carrying out our program. First, we need to be able to compute the generating function values in (1.4) and, second, we need to be able to perform the numerical inversion. The first requirement often turns out to be surprisingly easy, because the generating function of a normalization constant often has a remarkably simple form. This has long been known in statistical mechanics. In that context, the normalization constant is usually referred to as the partition function and its generating functions is referred to as the *grand partition function*; for example, see Reif [1965, pp. 213 and 347]. For the special case of multi-chain closed queuing networks, with only single-server and infinite-server queues, the generating function is displayed quite compactly in (4.4) below. Generating functions of other queuing network models (e.g., involving load-dependent servers) are given in Bertozzi and McKenna [1993]. They are not quite so nice as the subclass we consider here, but they tend to be tractable. See Choudhury et al. [1995] for generating functions of other product–form models.

Generating functions of normalization constants have not been used much to study closed queuing networks, but they have been used. Indeed, Reiser and Kobayashi [1975] used generating functions to derive their convolution algorithm for the normalization constants in multichain networks. Another early use of generating functions is by Williams and Bhandiwad [1976]; Kelly [1979] also briefly discusses generating functions. More recently, in the tradition of the statistical mechanics, generating functions have been used to do asymptotics by Birman et al. [1992], Kogan [1992], and Kogan and Birman [to appear]. Gordon [1990] and Bertozzi and McKenna [1993] have also recently used the generating functions to derive closed-form expressions for the normalization constant by applying Cauchy's theorem and the theory of residues. This extends the classical closed-form expression for the normalization constant in a single-chain network with only distinct single-server queues due to Koenigsberg [1958; 1986] and Harrison [1985]. In addition to deriving important new expressions for the single-chain case, Bertozzi and McKenna [1993] also consider the relatively complicated multidimensional case (with restrictions). They also provide a nice overview of the generating function structure of closed queuing networks.

To numerically invert the generating function of the normalization constant, we apply a Fourier-series method. In particular, we recursively apply the lattice-Poisson numerical inversion algorithm for one-dimensional generating functions in Abate and Whitt [1992a; 1992b] and Choudhury et al. [1994a] p times to treat a p-dimensional generating function. (For a closed queuing network, the dimension p is equal to the number of closed chains.) As noted in Choudhury et al. [1994a], the one-dimensional algorithm applies to complex-valued functions as well as real-valued functions, so such a recursive procedure is possible.

Unfortunately, however, the numerical inversion here is not routine. The previous numerical inversion algorithms in Abate and Whitt [1992a; 1992b] and Choudhury et al. [1994a] focused on calculating probability distributions. Since probabilities are bounded between 0 and 1, it is much easier to develop an effective algorithm (control the numerical errors) to compute probabilities. In sharp contrast, the normalization constants $g(\mathbf{K})$ depend on the vector argument $\mathbf{K}$ in a relatively complicated way and may be very small or very large. We address this problem here by introducing appropriate scaling, see (2.10). The scaling here parallels the previous use of scaling in the algorithm of Choudhury and Lucantoni [1995] to compute high-order moments of probability distributions and in the convolution algorithm, see Lam [1982] and Lavenberg [1983, p. 132]. However, unlike Choudhury and Lucantoni [1995] and Lam [1982], the scaling here is static rather than dynamic, that is, determined prior to the main computation. As a result, the implementation of our approach can be simpler than that of dynamic scaling. In addition, computation of the scale parameters is insignificant compared to the remaining computation. Another difference with Lam [1982] is that the scaling there is used to keep the intermediate and final partition functions within the range of floating point computation. In contrast, our main goal in scaling is to control the aliasing error in the inversion. (See (2.4) and (2.12).) Of course, we also take steps to ensure that no difficulty arises due to numbers getting outside the range of floating point computation. Typically, small problems can be solved without scaling, but scaling becomes necessary as the model becomes large.

Our general approach is applicable to a large class of product-form models, but in this paper we only develop a detailed scaling algorithm for a subclass of closed queueing networks (i.e., ones with only single-server and (optionally) infinite-server queues; see Sections 4 and 5). The procedure here applies directly to this subclass of models and illustrates what can be done by similar methods for other classes of models. Indeed, in Choudhury et al. [1995], we report on a different scaling algorithm to calculate normalization constants by numerical inversion in resource-sharing models as in Kaufman [1981] and circuit-switched communication network models as in Kelly [1986].

The computational requirement of our algorithm grows exponentially in the dimension of the generating function. Since the dimension of the generating function is the number of closed chains, the computational requirement grows exponentially in the number of closed chains, just as for the convolution algorithm. (See Section 2.5 for a discussion of computational complexity.) However, we show that this difficulty can often be circumvented by appropriate dimension reduction based on model structure. Indeed, a critical step in solving the 11-dimensional example mentioned at the outset is reducing the dimension from 11 to 2. Given this dimension reduction to 2, the original dimension could just as well be 100 or ever 1000 instead of 11. Our experience with large models indicates that they often have such special structures, that dramatic dimension reduction may be the rule rather than the exception in applications.

It turns out that, conceptually, our dimension reduction scheme parallels the dimension reduction achieved with the convolution algorithm using the tree convolution algorithm of Lam and Lien [1983] even though the actual algorithms are quite different. The dimension reduction is perhaps easier to understand with generating functions, because convolution of normalization constants corresponds simply to multiplication of their generating functions, and multiplication is inherently a more elementary operation than convolution.

Another important way that we gain computational efficiency is by using Euler summation to compute large sums, just as is done for Laplace transforms in Abate and Whitt [1992a] and Choudhury et al. [1994a]. (Other acceleration techniques could be used as well [Wimp 1981].) Unlike the inversion formula for Laplace transforms [Abate and Whitt 1992a; Choudhury et al. 1994a], the inversion formula for a generating function is a finite sum [Abate and Whitt 1992a, 1992b]. Hence, here we do not face the problem of calculating an infinite series. However, the size of the finite series is directly proportional to the desired function argument (here a chain population). When this is large, it may be possible to greatly reduce the computation by employing a summation acceleration method. For this purpose, we use Euler summation. For example, with acceleration the computation for a population of 100,000 or more may require a sum of fewer than 100 terms. Since saving is possible for each dimension (chain), the total saving grows geometrically with the number of chains. In this way, the inversion algorithm is able to exploit the special structure of large populations, much as the asymptotic methods do. (See Section 2.4 for more details.) Without using acceleration techniques, the computational complexity of our inversion algorithm is essentially the same as for the tree convolution algorithm [Lam and Lien 1983], but the acceleration techniques enable us to do better with large chain populations. There also is a much smaller storage requirement with the inversion algorithm. (See Section 2.5.)

In this paper, we propose both a general algorithm for a large class of problems (Sections 2 and 3) and a specific algorithm for an important subclass of problems, the multichain closed queuing networks with only single-server and (optionally) infinite-server queues (Sections 4–6). The specific algorithm should usually perform as a black box (give proper answers for any specified model parameters without any adjustments or "tuning" of algorithm parameters), whereas the general algorithm requires tuning in the choice of algorithm parameters, for example, scaling and the use of Euler summation.

A major part of the inversion algorithm involves error control. There are always two sources of error: aliasing error and roundoff error. In addition, if Euler summation is used to compute large sums, a third source of error, truncation error, is also introduced (note that we do not do pure truncation). Sections 2.1, 2.2, and 2.4 explain these errors and the techniques for controlling them. We do not have a simple expression for the final error but we do provide a good practical procedure for determining it. This is done by performing two different computations based on different parameters. Since the inversion involves entirely different contours in the complex plane for different algorithm parameters, the accuracy can usually be seen from the number of places where these computations agree. We also provide means to improve the accuracy if it turns out to be inadequate.

Here is how the rest of this paper is organized. In Section 2, we describe the basic numerical inversion algorithm for computing the normalization constant $g(\mathbf{K})$ in (1.2) given the generating function $G(\mathbf{z})$ in (1.4). In Section 3, we discuss dimension reduction. In Section 4, we consider the special class of closed queuing network models with only single-server and (optionally) infinite-server queues. In Section 5, we develop our scaling algorithm for this class of models. In Section 6, we indicate how to calculate mean queue lengths and higher moments directly by performing only two numerical inversions. We give a concise summary of the algorithm in Section 7. Then, we present illustrative numerical examples in Section 8. Finally, we draw conclusions in Section 9. For an overview of other recent applications of numerical inversion to performance analysis models, see Choudhury et al. [1994b].

2. *The Basic Inversion Algorithm*

Let $g(\mathbf{K})$ be the p-dimensional partition function in (1.2) and let $G(\mathbf{z})$ be its p-dimensional generating function in (1.4). We can invert $G(\mathbf{z})$ by a direct p-dimensional inversion as in Choudhury et al. [1994a] or by recursively performing p one-dimensional inversions. (In general, a p-dimensional inversion can be done in such a recursive manner [Choudhury et al. 1994a].) We use the recursive approach because it seems easier to handle the scaling and dimension reduction with it.

2.1. THE RECURSIVE ALGORITHM. To represent the recursive inversion, we define *partial generating functions* by

$$g^{(j)}(\mathbf{z}_j, \mathbf{K}_{j+1}) = \sum_{K_1=0}^{\infty} \cdots \sum_{K_j=0}^{\infty} g(\mathbf{K}) \prod_{i=1}^{j} z_i^{K_i} \qquad \text{for} \quad 0 \le j \le p, \qquad (2.1)$$

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 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

$$g^{(j-1)}(\mathbf{z}_{j-1}, \mathbf{K}_j) = I_j\left[g^{(j)}(\mathbf{z}_j, \mathbf{K}_{j+1})\right], \qquad 1 \le j \le p, \tag{2.2}$$

starting with $j = p$ and decreasing j by 1 each step. In the actual program implementation, we attempt the inversion shown in (2.2) for $j = 1$. In order to compute the right-hand side, we need another inversion with $j = 2$. This process goes on until at step p the function on the right-hand side becomes the p-dimensional generating function and is explicitly computable. By simply relabeling the p transform variables, we see that the scheme above can be applied to the p variables in any order.

In each step, we use the lattice-Poisson inversion algorithm in Abate and Whitt [1992a; 1992b] with modifications to improve precision and allow for complex inverse functions as in Choudhury et al. [1994a]. We show below the inversion formula at the *jth* step. For simplicity, we suppress those arguments that 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 is

$$\begin{aligned} g_j(K_j) = \frac{1}{2 l_j K_j r_j^{K_j}} \sum_{k_1=0}^{l_j-1} \exp\left(-\frac{\pi i k_1}{l_j}\right) \\ \times \sum_{k=-K_j}^{K_j-1} (-1)^k G_j\left(r_j \exp\left(\frac{\pi i (k_1 + l_j k)}{l_j K_j}\right)\right) - e_j, \end{aligned} \tag{2.3}$$

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

$$e_j = \sum_{n=1}^{\infty} g_j(K_j + 2 n l_j K_j) r_j^{2 n l_j K_j}. \tag{2.4}$$

Note that, for $j = 1$, $g_1(K_1) = g(\mathbf{K})$ is real, so that $G_1(\bar{z}_1) = \overline{G_1(z_1)}$. This enables us to cut the computation in (2.3) by about one half. For $j = 1$, we replace (2.3) by

$$\begin{aligned} g_1(K_1) = \frac{1}{2 l_1 K_1 r_1^{K_1}} \Bigg[G_1(r_1) - (-1)^{K_1} G_1(-r_1) + 2 \sum_{k_1=1}^{l_1} \exp\left(\frac{-\pi i k_1}{l_1}\right) \\ \times \sum_{k=0}^{K_1-1} G_1\left(r_1 \exp\left(\frac{\pi i (k_1 + l_1 k)}{l_1 k_1}\right)\right) \Bigg] - e_1. \end{aligned} \tag{2.5}$$

Equations (2.3) and (2.4) can be obtained by applying the discrete Poisson summation formula [Abate and Whitt 1992a; 1992b]. The idea is to consider the sequence $h(K_j) \equiv g_j(K_j) r_j^{K_j}$ for $K_j \ge 0$ and $h(K_j) \equiv 0$ otherwise, and then construct a periodic sequence by adding translations of the original sequence (with successive shifts of multiples of $2 l_j K_j$), that is, by *aliasing*. The resulting periodic sequence is $(g_j(K_j) + e_j) r_j^{K_j}$ for e_j in (2.4). The discrete Fourier series representation of this periodic sequence is the right-hand side of (2.3) (without the term $-e_j$) multiplied by $r_j^{K_j}$. Next, (2.3) is obtained by multiplying both sides by $r_j^{-K_j}$ and moving e_j to the right side.

An alternative way of obtaining (2.3) without the error term is simply to apply the trapezoidal rule form of numerical integration to the Cauchy contour integral

$$g_j(K_j) = \frac{1}{2\pi i} \int_{C_{r_j}} G_j(z_j) z_j^{-(K_j+1)}\, dz, \tag{2.6}$$

where C_{r_j} is a circle of radius r_j which does not contain any of the singularities of $G_j(z_j)$. In this context, the error expression e_j in (2.4) may also be regarded as the discretization error of the trapezoidal rule. Note that even though, in general, the trapezoidal rule is a rather elementary form of numerical integration, in the current context it really outperforms more sophisticated numerical integration procedures. This is due to the oscillating nature of the integrand, and is verified by the tight and easily controllable error expression given by (2.4); that is, the actual error in (2.4) is usually much less than the customary error bound associated with the trapezoidal rule; see Abate and Whitt [1992a] for more discussion and references.

To control the aliasing error (2.4), we choose

$$r_j = 10^{-\gamma_j/2l_jK_j}. \tag{2.7}$$

Inserting (2.7) into (2.4), we get

$$e_j = \sum_{n=1}^{\infty} g_j(K_j + 2nl_jK_j)10^{-\gamma_j n}. \tag{2.8}$$

This choice of r_j enables us to more easily control the aliasing error e_j using the parameter γ_j. For instance, if g_j were bounded above by 1, as is the case with probabilities, then the aliasing error would be bounded above by $10^{-\gamma_j}/(1 - 10^{-\gamma_j}) \approx 10^{-\gamma_j}$.

As is clear from (2.8), a bigger γ_j decreases the aliasing error. However, since $r_j^{-K_j} = 10^{\gamma_j/2j}$, the factor $r_j^{-K_j}$ in (2.3) increases sharply with γ_j and thus can cause roundoff error problems. Since the parameter l_j does not appear in the aliasing error term (2.8), it can be used to control the growth of $r_j^{-K_j}$ without altering the aliasing error. Bigger values of l_j yield less roundoff error, but more computation because the number of terms in (2.3) is proportional to l_j. (See Choudhury et al. [1994a] for more discussion.)

An inner loop of the inversion requires more accuracy than an outer loop since the inverted values in an inner loop are used as transform values in an outer loop. With a goal of about eight significant digit accuracy, the following sets of l_j and γ_j typically are adequate:

(i) $l_1 = 1, \quad \gamma_1 = 11$,
(ii) $l_2 = l_3 = 2, \quad \gamma_2 = \gamma_3 = 13$,
(iii) $l_4 = l_5 = l_6 = 3, \quad \gamma_4 = \gamma_5 = \gamma_6 = 15$,

assuming that computations are done using double-precision arithmetic. It is usually not a good idea to use the same l_j for all j, because then more computation is done to achieve the same accuracy.

In Abate and Whitt [1992a; 1992b] and Choudhury et al. [1994a], the inverse function was mainly assumed to be a probability, so that the aliasing error e_j in (2.8) could be easily bounded. In contrast, here the normalization constants may be arbitrarily large and therefore the aliasing error e_j in (2.8) may also be arbitrarily large. Thus, in order to control errors, we scale the generating function in each step by defining a *scaled generating function* as

$$\overline{G}_j(z_j) = \alpha_{0j} G_j(\alpha_j z_j), \tag{2.9}$$

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. (See Section 2.2 below.)

Let $\bar{g}_j(K_j)$ represent the inverse function of $\overline{G}_j(z_j)$. The desired inverse function $g_j(K_j)$ may then be recovered from $\bar{g}_j(K_j)$ by

$$g_j(K_j) = \alpha_{0j}^{-1} \alpha_j^{-K_j} \bar{g}_j(K_j). \tag{2.10}$$

2.2. Choosing Scaling Parameters. Note that the inversion procedure in (2.2) is a nested procedure, so that scaling done at one step will also modify the functions in subsequent steps. By (2.8), the aliasing error term for computing $\bar{g}_j(K_j)$ from the scaled generating function $\overline{G}_j(z_j)$ in (2.9) is

$$\bar{e}_j = \sum_{n=1}^{\infty} \bar{g}_j(K_j + 2nl_jK_j) 10^{-\gamma_j n}. \tag{2.11}$$

Since the quantities needed in product-form models typically involve ratios of normalization constants, we really care about the relative error and not the absolute error. The relative error is given by

$$e_j' \equiv \frac{\bar{e}_j}{\bar{g}_j(K_j)} = \sum_{n=1}^{\infty} \frac{\bar{g}_j(K_j + 2nl_jK_j)}{\bar{g}_j(K_j)} 10^{-\gamma_j n}, \tag{2.12}$$

so that it can be bounded via

$$|e_j'| \le \sum_{n=1}^{\infty} \left| \frac{\bar{g}_j(K_j + 2nl_jK_j)}{\bar{g}_j(K_j)} \right| 10^{-\gamma_j n}. \tag{2.13}$$

Let

$$C_j = \operatorname*{Max}_{n} \left\{ \left| \frac{\bar{g}_j(K_j + 2nl_jK_j)}{\bar{g}_j(K_j)} \right| \right\}^{1/n} \tag{2.14}$$

Then

$$|e_j'| \le \sum_{n=1}^{\infty} C_j^n 10^{-\gamma_j n} \le \frac{C_j 10^{-\gamma_j}}{1 - C_j 10^{-\gamma_j}} \approx C_j 10^{-\gamma_j}. \tag{2.15}$$

Hence, to limit the aliasing error, we want C_j in (2.14) to be not too large, that is,

$$C_j \ll 10^{\gamma_j}. \tag{2.16}$$

Our main purpose in scaling is to satisfy (2.16), which in turn controls the relative aliasing error. However, note that only the scale parameter α_j is useful for this purpose since C_j is independent of the other scale parameters α_{0j}. We can use the other scale parameter α_{0j} to make it more likely that $\bar{g}_j(K_j)$ does not exceed the range of floating point computation, but we also use another measure to avoid floating point range exception. We compute and store only the logarithms of large quantities or the ratios of large quantities. In that context we also treat sums $(a_1 + \cdots + a_n)$ where a_1 is the largest term as products of the form $a_1\,(1 + a_2/a_1 + \cdots + a_n/a_1)$.

An obvious problem with condition (2.16) is that we do not know $g_j(\mathrm{K}_j)$ explicitly. However, since 10^{γ_j} is very large, it should be possible to satisfy (2.16) by roughly controlling the growth rate of $g_j(K_j)$ as K_j gets large based on the structure of the generating function. Specifically, in many cases it is possible to express $g_j(\mathrm{K}_j)$ as

$$g_j(K_j) = \sum_{i=1}^{m} A_i B_i(K_j), \tag{2.17}$$

where the A_i's are usually unknown constants, but the B_i's are known functions of K_j. Indeed, we show that this structure holds for the closed queuing networks in Section 5. Then our strategy is to identify the fastest growing function $B_i(K_j)$, and introduce a scaled version

$$\overline{B}_i(K_j) = \alpha_{0j}\alpha_j^{K_j} B_i(K_j) \tag{2.18}$$

as in (2.10), so that

$$\operatorname*{Max}_{n} \left\{ \left| \frac{\overline{B}_i(K_j + 2nl_jK_j)}{\overline{B}_i(K_j)} \right| \right\}^{l/n} \le \beta, \tag{2.19}$$

where β is of the order of 1. Indeed, we identify the fastest growing function by requiring that (2.19) hold for *all* i. For the scaling of closed queuing networks in Section 5, we find that it suffices to consider only the case $n = 1$ in (2.19). Given (2.19), we use the scaling in (2.18) in (2.10).

We have just described a general scaling strategy. We present a specific scaling algorithm for a class of closed queuing networks that follows this strategy in Section 5. With the aid of (2.19), we are able to treat multiplicities in the factors of the generating function. However, to treat near multiplicities, we must go beyond (2.19) as we do in Section 5.5. Hence, (2.19) should be regarded only as a reasonable starting point. Difficult cases will require refinement.

2.3. Verification of the Accuracy of Computation. The inversion algorithm given by (2.2), (2.3), and (2.5) with scaling (2.10) is really a family of algorithms, one for each vector $\mathbf{l} \equiv (l_1, \ldots, l_p)$. Our experience is that there exists a minimum vector $\mathbf{l}_{\min} \equiv (l_{1,\min}, \ldots, l_{p,\min})$ such that the algorithm will be sufficiently accurate whenever $\mathbf{l} \ge \mathbf{l}_{\min}$. However, the required computation increases as $\mathbf{l}$ increases, so we do not want $\mathbf{l}$ larger than necessary. Typically $l_{1,\min} = 1$, $l_{2,\min} = l_{3,\min} = 2$, $l_{4,\min} = l_{5,\min} = l_{6,\min} = 3$, etc. Each $\mathbf{l}$ corresponds to a unique set of contours in the p-dimensional complex plane.

In addition to being a means to achieve greater accuracy, we use the vectors $\mathbf{l}$ to verify the accuracy. If we run the algorithm with $\mathbf{l}_A$ and $\mathbf{l}_B$, where $\mathbf{l}_A, \mathbf{l}_B \ge \mathbf{l}_{\min}$, $\mathbf{l}_A \ne \mathbf{l}_B$ and the answers agree up to t significant places (with large t, say 6 or higher) then we regard both answers as accurate up to t significant places. (This agreement could happen by chance only with very small probability.) We have tested this property for many cases where the results can be computed easily by other methods.

2.4. EULER SUMMATION AND TRUNCATION. In Abate and Whitt [1991a] and Choudhury et al. [1994a], Euler summation is used to efficiently calculate infinite series arising in the inversion of Laplace transforms. We also use Euler summation here to speed up the calculation of large finite series in (2.3). Euler summation is especially appropriate for alternating series. Note that we have chosen our parameters so that the inner sums of the inversion formula (2.3) will tend to be nearly an alternating series.

Consider a real series of the form

$$S = \sum_{k=0}^{\infty} (-1)^k a_k, \tag{2.20}$$

and let the partial sums be

$$S_j = \sum_{k=0}^{j} (-1)^k a_k. \tag{2.21}$$

Here $a_k = 0$ for $k > K$, but in general this need not hold. Euler summation approximates S in (2.20) by

$$E(m,n) = S_n + (-1)^{n+1} \sum_{k=0}^{m-1} (-1)^k 2^{-(k+1)} \Delta^k a_{n+1} = \sum_{k=0}^{m} \binom{m}{2} 2^{-k} S_{n+k}, \tag{2.22}$$

where Δ is the finite-difference operator; that is, $\Delta a_n = a_{n+1} - a_n$ and $\Delta^k a_n = \Delta(\Delta^{k-1} a_n)$.

We suggest using the Euler sum on the inner sums in (2.3) and (2.5) whenever K_j is large. (Note that for the inner sum in (2.3) the Euler sum should be applied twice, once for positive k and once for negative k). Clearly, using the Euler sum (2.22) makes sense only if $K_j > (m + n)$. We typically use $n = 11$ and $m = 20$ or 40.

In general, it is difficult to show that Euler summation will necessarily be effective. Hence, we just apply it and see if it works, using the error estimate $|E(m,n) - E(m, n+1)|$ and the accuracy verification in Section 2.3. Euler summation has worked for all the closed queuing network examples we have considered, including the examples here in Section 8. In some cases, Euler summation is effective because the terms in the series S very rapidly approach 0. Then Euler summation is tantamount to simple truncation, but this is not the only case.

We mention that for the loss networks considered in Choudhury et al. [1995] Euler summation is less effective, but significant computational savings are still possible via appropriate truncation, after identifying where the primary contribution to the sum comes from. There also are many alternative acceleration procedures; see Wimp [1981]. In other words, the inversion algorithm offers the possibility of further computational savings through further analysis.

2.5. COMPUTATIONAL COMPLEXITY AND STORAGE REQUIREMENTS. To understand the computational complexity, look at the basic inversion formula in (2.3) and (2.5). The computation complexity is determined by how many times we have to compute the transform $G(\mathbf{z})$. Other work (such as finding the scaling factors) is relatively small. From (2.2), (2.3) and (2.5), we see that the total number of times $G(\mathbf{z})$ has to be computed is about

$$2^{p-1} \prod_{j=1}^{p} l_j \prod_{j=1}^{p} K_j, \tag{2.23}$$

where p is the dimension of the generating function (number of closed chains), K_j is the population of the jth chain and l_j is the jth roundoff-error-control parameter in (2.3).

To say more, we need to know more about the structure of $G(\mathbf{z})$. For the special case of the closed queuing networks considered in Section 4, $G(z)$ is given in (4.5). For that generating function, each computation of $G(z)$ requires about $pq' + p + q' \approx pq'$ complex multiplications, where q' is the number of distinct single-server queues (excluding multiplicities). Hence, the total number of complex multiplications is of order

$$pq'2^{p-1} \prod_{j=1}^{p} l_j \prod_{j=1}^{p} K_j. \tag{2.24}$$

An analysis of the computational complexity of previous algorithms has been done by Conway and Georganas [1989]. From Conway and Georganas [1989, p. 177] we see that the number of multiplications and additions in the convolution algorithm for the case of all single-server queues is

$$2p(N-1) \prod_{j=1}^{p} (K_j + 1), \tag{2.25}$$

where N is the total number of queues.

From (2.24) and (2.25), we see that the dominant term is $\prod_{j=1}^{p} K_j$ in both cases. Based on this initial analysis, our algorithm is worse than the convolution algorithm, because we have the additional factor $2^{p-1}\prod_{j=1}^{p} l_j$ and because we need complex multiplication. As an example, for $p = 4$ we typically have $l_1 = 1$, $l_2 = l_3 = 2$, and $l_3 = 3$; this gives $2^{p-1}\prod_{j=1}^{p} l_j = 96$.

However, there are several other considerations strongly favoring our algorithm. First, q' may be significantly less than N, as is the case in our numerical examples. Second, if the class populations K_j are large, then we can exploit Euler summation as in Section 2.4. If Euler summation is indeed used with $m + n = 31$, then (2.24) is replaced by

$$pq'2^{p-1} \prod_{j=1}^{p} l_j \prod_{j=1}^{p} \min\{31, K_j\}. \tag{2.26}$$

For example, if $K_j = 3100$ for all j, then (2.26) is a reduction of (2.24) by a factor of 100^p.

Conway and Georganas [1989, p. 177] point out that the term $\prod_{j=1}^{p}(K_j + 1)$ in (2.25) needs to be replaced by $\prod_{j=1}^{p}(K_j + 1)(K_j + 2)/2$ if all queues have load-dependent service rates, causing a dramatic increase in computation. In contrast, whenever the generating function can be expressed in a simple closed form, the computational burden with our algorithm will not increase significantly.

Another dramatic reduction of (2.24) and (2.26) with our algorithm occurs if we can exploit dimension reduction, which we discuss in the next section. This effectively reduces p.

A final consideration is the storage requirement. We have virtually no storage requirement since we compute $g(\mathbf{K})$ directly without storing intermediate values. In contrast, the storage requirement with the convolution algorithm is about $2\prod_{j=1}^{p}(K_j + 1)$.

3. *Dimension Reduction by Decomposition*

In general, the inversion of a p-dimensional generating function $G(\mathbf{z})$ represents a p-dimensional inversion whether it is done directly or by our proposed recursive technique. Fortunately, however, it is often possible to reduce the dimension significantly by exploiting special structure. To see the key idea, note that if $G(\mathbf{z})$ can be written as product of factors, where no two factors have common variables, then the inversion of $G(\mathbf{z})$ can be carried out by inverting the factors separately and the dimension of the inversion is thus reduced. For example, if $G(z_1, z_2, z_3) = G_1(z_1)G_2(z_2, z_3)$, then G can be treated as one two-dimensional problem plus one one-dimensional problem, which is essentially a two-dimensional problem, instead of one three-dimensional problem. The factors can be treated separately because factors not involving the variable of integration pass through the sum in (2.3). We call this an *ideal decomposition.* It obviously provides reduction of computational complexity, but we do not really expect to be able to exploit it, because it essentially amounts to having two or more completely separate models, which we would not have with proper model construction. (We would treat them separately to begin with.)

Even though ideal decomposition will virtually *never* occur, key model elements (e.g., closed chains) are often only *weakly coupled,* so that we can still exploit a certain degree of decomposition to reduce the inversion dimensionality, often dramatically. The idea is to look for *conditional decomposition.* The possibility of conditional decomposition stems from the fact that when we perform the $(j - 1)$st inversion in (2.2), the outer variables $z_1, \ldots, z_{j-1}$ are fixed. Hence, for the $(j - 1)$st inversion it suffices to look for decomposition in the generating functions regarded as a function of the remaining $p - j + 1$ variables. For example, if $G(z_1, z_2, z_3) = G_1(z_1, z_2)G_2(z_1, z_3)$, then for each fixed z_1, the transform G as a function of (z_2, z_3) factors into the product of two functions of a single variable. Hence, G can be treated as two two-dimensional problems instead of one three-dimensional problems.

More generally, we select d variables that we are committed to invert. We then look at the generating function with these d variables fixed and see if the remaining function of $p - d$ variables can be factored. Indeed, we write the function of the remaining $p - d$ variables as a product of factors, where no two factors have any variables in common. The maximum dimension of the additional inversion beyond the designated d variables is equal to the maximum number of the $p - d$ remaining variables appearing in one of the factors, say m. The overall inversion can then be regarded as being of dimension

$d + m$. The idea, then, is to select an appropriate d variables, so that the resulting dimension $d + m$ is small.

This dimension reduction can be done whenever a multidimensional transform can be written as a product of factors. As we will see in Section 4, this structure always occurs with closed queuing networks. For closed queuing networks there is a factor for each queue in the network, and the variable z_j appears in the factor for queue i if and only if chain j visits queue i. Thus, conditional decomposition tends to occur when chains tend to visit relatively few queues. This property is called sparseness of routing chains in Lam and Lien [1983]. In that paper, it is noted that this sparseness property is likely to be present in large communication networks and distributed systems.

To carry out this dimension reduction, we exploit the representation of the generating function $G(\mathbf{z})$ as a product of separate factors, that is:

$$G(\mathbf{z}) = \prod_{i=1}^{m} \hat{G}_i(\hat{\mathbf{z}}_i) \tag{3.1}$$

where $m \geq 2$ and $\hat{\mathbf{z}}_i$ is a subset of $\{z_1, z_2, \ldots, z_p\}$. We assume that each $\hat{G}_i(\hat{\mathbf{z}}_i)$ cannot be further factorized into multiple factors, unless at least one of the latter is a function of all variables in the set $\hat{\mathbf{z}}_i$.

We now represent the conditional decomposition problem as a graph problem. We construct a graph, called an *interdependence graph*, to represent the interdependence of the variables z_k in the factors. We let each variable z_k be represented by a node in the graph. For each factor $\hat{G}_i(\hat{\mathbf{z}}_i)$ in (3.1), form a fully connected subgraph Γ_i by connecting all nodes (variables) in the set $\hat{\mathbf{z}}_i$. Then let $\Gamma = \bigcup_{i=1}^{m} \Gamma_i$.

Now for any subset D of Γ, we identify the *maximal connected subsets* $S_i(D)$ of $\Gamma - D$; that is, $S_i(D)$ is connected for each i, $S_i(D) \cap S_j(D) = \emptyset$ when $i \neq j$ and $\bigcup_i S_i(D) = \Gamma - D$. Let $|A|$ be the cardinality of the set A. Then, the dimension of the inversion resulting from the selected subset D is

$$\text{inversion dimension} = |D| + \max_i\{|S_i(D)|\}. \tag{3.2}$$

It is natural to consider the problem of minimizing the overall dimension of the inversion. This is achieved by finding the subset D to achieve the following minimum:

$$\text{minimal inversion dimension} = \min_{D \subseteq \Gamma}\Big\{|D| + \max_i\{|S_i(D)|\}. \tag{3.3}$$

In general, it seems difficult to develop an effective algorithm to solve this graph optimization problem, and we suggest it as an interesting research problem. However, for the small-to-moderate number of variables that we typically encounter, we can solve (3.3) by inspection or by enumeration of the subsets of Γ in increasing order of cardinality. Since our overall algorithm is likely to have difficulty if the reduced dimension is not relatively small (e.g., ≤ 10), it is not necessary to consider large sets D in (3.3). This dimension reduction is illustrated in Section 5.4 and Examples 8.3 and 8.4 in Section 8.

As we mentioned in the introduction, it turns out that our approach to dimension reduction is essentially equivalent to the tree algorithm of Lam and Lien [1983] used with the convolution algorithm, even though the actual algorithms are different. The connection is easily seen by noting that convolution of normalization constants corresponds to multiplication of the generating functions. It appears that the dimension reduction is easier to understand and implement with generating functions, because multiplication is a more elementary operation than convolution.

4. *Closed Queuing Networks*

In this section, we consider multichain closed queuing networks with only single-server queues (service centers with load-independent service rates) and (optionally) infinite-server queues. This section closely follows Sections 2.1 and 2.2 of Bertozzi and McKenna [1993], which, in turn, closely follow Bruel and Balbo [1980]. However, we do not consider the most general models in Bertozzi and McKenna [1993] and Bruel and Balbo [1980]. We use the following notation:

—p = number of closed chains

—M = number of job classes ($M \geq p$).

—N = number of queues (service centers). Queues $1, \dots, q$ are assumed to be of the single-server type and queues $q + 1, \dots, N$ are assumed to be of the infinite-server (IS) type. As usual, for the single-server queues, the service discipline may be first-come first-served (FCFS), last-come first-served preemptive-resume (LCFSPR) or processor sharing (PS). In the case of FCFS, the service times of all job classes at a queue are assumed to be exponential with the same mean.

—$R_{ri,sj}$ = routing matrix entry, probability that a class r job completing service at queue i will next proceed to queue j as a class s job for $1 \leq i, j \leq N$, $1 \leq r, s \leq M$ (i.e., class hopping is allowed). The pair (r, i) is referred to as a stage in the network.

—class vs. chain. Two classes, r and s, communicate with each other if some i and j, stage (s, j) can be reached from stage (r, i) in a finite number of steps and vice versa. With respect to the relation of communication, all the classes can be divided into mutually disjoint equivalence classes called (closed) chains (ergodic sets in Markov chain theory). All classes within a chain communicate. No two classes belonging to different chains communicate. Since we are considering the steady-state distribution of a model with only closed chains, we do not need to consider any transient stages, that is, stages (r, i) that will not be reached infinitely often.

—K_j = number of jobs in the jth closed chain, $1 \leq j \leq p$, which is fixed.

—$\mathbf{K} = (K_1, \dots, K_p)$, the population vector, specified as part of the model data.

—n_{ri} = number of jobs of class r in queue i, $1 \leq r \leq M$, $1 \leq i \leq N$.

—n_i = number of jobs in queue i, that is, $n_i = \Sigma_{r=1}^{M} n_{ri}$, $1 \leq i \leq N$.

—$\mathbf{n} = (n_{ri})$, $1 \leq r \leq M$, $1 \leq i \leq N$, the job vector, the queue lengths, the state variable.

—C_j = set of stages in the jth closed chain. Clearly, $\Sigma_{(r,i) \in C_j} n_{ri} = K_j$, $1 \leq j \leq p$.

—$q_{ji} = \Sigma_{r: (r,i) \in C_j} n_{ri}$, number of jobs from chain j at queue i.

—$s(\mathbf{K})$ = state space of allowable job vectors or queue lengths (including those in service), that is,

$$S(\mathbf{K}) = \left\{ \mathbf{n} \colon n_{ri} \in \mathbf{Z}^+ \text{ and } \sum_{(r,i) \in C_j} n_{ri} = K_j, 1 \leq j \leq p \right\}, \tag{4.1}$$

where $\mathbf{Z}^+$ is the set of nonnegative integers.

—e_{ri} = visit ratio, that is, solution of the traffic rate equation

$$\sum_{(r,i)\in C_k} e_{ri}R_{ri,sj} = e_{sj} \qquad \text{for all} \quad (s,j) \in C_k \text{ and } 1 \le k \le p. \tag{4.2}$$

For each chain there is one degree of freedom in (4.2). Hence, for each chain j, the visit ratios $\{e_{ri}\colon (r,i) \in C_j\}$ are specified up to a constant multiplier.

—t_{ri} = the mean service time for class r at queue i.

—$\rho'_{ri} = t_{ri}e_{ri}$, $1 \le r \le M$, $1 \le i \le N$, the relative traffice intensities.

—The steady-state distribution is given by (1.1) and the partition function by (1.2), where

$$f(\mathbf{n}) = \left[\prod_{i=1}^{q} n_i! \prod_{r=1}^{M} \frac{\rho'^{\,n_{ri}}_{ri}}{n_{ri}!}\right]\left[\prod_{i=q+1}^{N} \prod_{r=1}^{M} \frac{\rho'^{\,n_{ri}}_{ri}}{n_{ri}!}\right]. \tag{4.3}$$

—$\rho_{j0} = \sum_{i=q+1}^{N}\sum_{(r,i)\in C_j}\rho'_{ri}$ and $\rho_{ji} = \sum_{(r,i)\in C_j}\rho'_{ri}$ for $i = 1, 2, \dots, q$, the aggregate relative traffic intensities.

—The generating function $G(\mathbf{z})$ is given by (1.4), using (1.2) and (4.3). As shown in (2.25) of Bertozzi and McKenna [1993], $G(\mathbf{z})$ can be expressed simply as

$$G(\mathbf{z}) = \frac{\exp\left(\sum_{j=1}^{p} \rho_{j0} z_j\right)}{\prod_{i=1}^{q}\left(1 - \sum_{j=1}^{p} \rho_{ji} z_j\right)}. \tag{4.4}$$

In general, there may be multiplicity in the denominator factors of (4.4) if two or more queues are identical with respect to visits by customers of all classes. In such a situation (4.4) becomes

$$G(\mathbf{z}) = \frac{\exp\left(\sum_{j=1}^{p} \rho_{j0} z_j\right)}{\prod_{i=1}^{q'}\left(1 - \sum_{j=1}^{p} \rho_{ji} z_j\right)^{m_i}}, \tag{4.5}$$

where

$$\sum_{i=1}^{q'} m_i = q. \tag{4.6}$$

For us, (4.5) is the relevant form, not (4.4); that is, the key parameters are p and q'. Our algorithm simplifies by having different queues with identical single-server parameters. (Evidently, the reverse is true for the theory of residues [Bertozzi and McKenna 1993].)

Given the normalization constant $g(\mathbf{K})$ in (1.2) and (4.3), we can directly compute the steady-state probability mass function $p(\mathbf{n})$ in (1.1). Moreover, several important performance measures can be computed directly from ratios of normalization constants. For example, the throughput of class r jobs at queue i is

$$\theta_{ri} = e_{ri}\frac{g(\mathbf{K} - \mathbf{1}_j)}{g(\mathbf{K})} \qquad \text{for} \quad (r,i) \in C_j, \tag{4.7}$$

where $\mathbf{1}_j$ is the vector with a 1 in the jth place and 0's elsewhere; for example see Lavenberg [1983, p. 147]. The means $E[n_{ri}]$ and $E[q_{ji}]$ and higher moments $E[n_{ri}^k]$ and $E[q_{ji}^k]$ can also be computed directly from the normalization constants, but the standard formulas involve more than two normalization constant values. We develop an improved algorithm for means and higher moments via generating functions in Section 6 below.

From (4.7) we see that we will often need to compute normalization constants $g(\mathbf{K})$ for several closely related arguments $\mathbf{K}$. When the population vector $\mathbf{K}$ is large, it is possible to calculate such closely related normalization constants efficiently by exploiting shared computation. This technique is described in Choudhury et al. [1995, Sect. 5.2].

5. *Scaling for Closed Queuing Networks with Single-Server and Infinite-Server Queues*

In this section, we indicate how to choose the scale parameters α_{0j} and α_j in (2.10) in order to invert the generating function $G(\mathbf{z})$ in (4.5) for the class of closed queuing networks considered here. Our final scaling algorithm is given in Section 5.5 beginning with (5.41). We develop it by starting with more elementary cases and work our way up to the full generality of (4.5). The general strategy has already been described in Section 2.

5.1. SCALING FOR A SINGLE CHAIN WITH ONLY SINGLE-SERVER QUEUES. In this case, we have (4.5) with $p = 1$ without the term in the numerator corresponding to infinite-server queues. Using (2.10), we see that the scaled generating function is given by

$$\bar{G}(z_1) = \frac{\alpha_{01}}{\prod_{i=1}^{q'}(1 - \rho_{1i}\alpha_1 z_1)^{m_i}}. \tag{5.1}$$

When there are no multiplicities (i.e., $m_i = 1$ for all i) and one aggregate relative traffic intensity ρ_{1i} is dominant (substantially larger than all others), it is easy to see that we should have $\alpha_1 = 1/\max\{\rho_{1i}\}$. We now give a more careful analysis to account for the multiplicities. See Section 5.5 below for a discussion of near multiplicities.

Carrying out a partial fraction expansion with (5.1), we get

$$\bar{G}(z_1) = \alpha_{01}\sum_{i=1}^{q'}\sum_{j=1}^{m_i}\frac{A_{ij}}{(1 - \rho_{1i}\alpha_1 z_1)^j}, \tag{5.2}$$

where

$$A_{ij} = \frac{(-1)^{m_i - j}}{\alpha_{01}(m_i - j)!(\rho_{1i}\alpha_1)^{m_i - j}}\left[\frac{d^{m_i - j}}{dz_1^{m_i - j}}\bar{G}(z_1)\right]\Bigg|_{z_1 = 1/\rho_{1i}\alpha_1}. \tag{5.3}$$

In general, the constants A_{ij} in (5.3) are difficult to compute (except when $m_i = 1$ for all i), so we will treat them as unknown constants. Through a term-by-term binomial expansion and by collecting the coefficient of $z_1^{K_1}$, we get

$$\bar{g}(K_1) = \sum_{i=1}^{q'}\sum_{j=1}^{m_i} A_{ij}\bar{B}_{ij}(K_1), \tag{5.4}$$

where

$$\bar{B}_{ij}(K_1) = \alpha_{01}\binom{j + K_1 - 1}{K_1}\rho_{1i}^{K_1}\alpha_1^{K_1}. \tag{5.5}$$

Note that (5.4) is of the same form as (2.17). Hence, we can get the scale parameters by focusing on the fastest growing function $B_{ij}(K_1)$, and imposing condition (2.19). Note that, for any given i, the term corresponding to $j = m_i$ is the fastest growing; therefore we concentrate only on that term. It can be shown that the most restrictive condition in (2.19) comes from $n = 1$ and that (2.19) will be satisfied with $\beta = 1$ by setting

$$\alpha_{01} = 1 \qquad \text{and} \qquad \alpha_1 = \operatorname*{Min}_i \left\{ \frac{a_i}{\rho_{1i}} \right\}, \tag{5.6}$$

where

$$a_i = \begin{cases} 1 & \text{for} \quad m_i = 1 \\ \left[\displaystyle\prod_{j=1}^{m_i - 1} \frac{K_1 + j}{K_1 + 2l_1K_1 + j} \right]^{1/2l_1K_1} & \text{for} \quad m_i > 1. \end{cases} \tag{5.7}$$

From (5.6) and (5.7), we see that we do indeed have the obvious scaling with $a_i = 1$ when $m_i = 1$ for all i. Moreover, we have this same scaling when K_1 becomes very large, because $a_i \to 1$ for all i as $K_1 \to \infty$ when $m_i > 1$.

5.2. Scaling for a Single Chain with Single-Server and Infinite-Service Queues. In this case, the term in the numerator of (4.5) corresponding to the infinite-server queues has to be included. Instead of (5.2), from the partial-fraction expansion we get

$$\overline{G}(z_1) = \alpha_{01} \sum_{i=1}^{q'} \sum_{j=1}^{m_i} \frac{A_{ij} \exp(\rho_{10}\alpha_1 z_1)}{(1 - \rho_{1i}\alpha_1 z_1)^j}. \tag{5.8}$$

At first assume $m_i = 1$ for all i. For ease of notation, also let $A_{i1} \equiv A_i$. Then (5.8) becomes

$$\overline{G}(z_1) = \alpha_{01} \sum_{i=1}^{q'} \frac{A_i \exp(\rho_{10}\alpha_1 z_1)}{(1 - \rho_{1i}\alpha_1 z_1)}. \tag{5.9}$$

Collecting the coefficient of $z_1^{K_1}$ on the right side of (5.9), and after some manipulation, we get

$$\bar{g}(K_1) = \sum_{i=1}^{q'} A_i \overline{B}_i(K_1), \tag{5.10}$$

where

$$\overline{B}_i(K_1) = \alpha_{01}(\rho_{1i}\alpha_1)^{K_1} \sum_{j=0}^{K_1} \frac{(\rho_{10}/\rho_{1i})^j}{j!}. \tag{5.11}$$

Again (5.10) is of the same form as (2.17). The fastest growing term in this case is the one corresponding to the largest ρ_{1i}. Let

$$\rho_{1,\max} = \operatorname*{Max}_i \{\rho_{1i}\}. \tag{5.12}$$

To proceed further with (5.11), we consider two cases: In Case 1, $\rho_{10}/\rho_{1,\max} \leq K_1$, while in Case 2, $\rho_{10}/\rho_{1,\max} > K_1$. In Case 1, by noticing the connection to the Poisson distribution, it can be shown that

$$\frac{1}{2}\exp\left(\frac{\rho_{10}}{\rho_{1,\max}}\right) \le \sum_{j=0}^{l} \frac{(\rho_{10}/\rho_{1,\max})^j}{j!} \le \exp\left(\frac{\rho_{10}}{\rho_{1,\max}}\right) \qquad \text{for} \quad l \ge K_1. \tag{5.13}$$

Using (5.13), we see that condition (2.19) is satisfied (with $\beta = 2$) by making the following choices of scale parameters

$$\alpha_{01} = \exp\left(-\left(\frac{\rho_{10}}{\rho_{1,\max}}\right)\right) \qquad \text{and} \qquad \alpha_1 = \frac{1}{\rho_{1,\max}}. \tag{5.14}$$

In Case 2, relation (5.13) does not hold. In this case we control the growth rate of the fastest growing term in (5.11). Let $B_i'(K_1)$ represent this term, which is given by

$$B_i'(K_1) = \alpha_{01}\frac{(\alpha_1\rho_{10})^{K_1}}{K_1!} \approx \frac{\alpha_{01}(\alpha_1\rho_{10})^{K_1}}{\sqrt{2\pi K_1}\,K_1^{K_1}\exp(-K_1)}. \tag{5.15}$$

The approximate expression in (5.15) is obtained by Stirling's approximation to the factorial. Note that $B_i'(K_1)$ is independent of i (since Case 2 ensures that $\rho_{10}/\rho_{1i} > K_1$ for all i). Therefore, we can control the growth rate of $B_i'(K_1)$ for all i by choosing the scale parameters to cancel out the two dominant terms in the expression for the factorial. This is done with the choice

$$\alpha_{01} = \exp(-K_1) \qquad \text{and} \qquad \alpha_1 = \frac{K_1}{\rho_{10}}. \tag{5.16}$$

The scaling for Cases 1 and 2 in (5.14) and (5.16) can be combined as follows:

$$\alpha_{01} = \exp(-\alpha_1\rho_{10}) \qquad \text{and} \qquad \alpha_1 = \operatorname*{Min}_i\left\{\frac{K_1}{\rho_{10}}, \frac{1}{\rho_{1i}}\right\}. \tag{5.17}$$

Finally, based on (5.6) and (5.17), we extend the scaling in the case $m_i > 1$ as follows:

$$\alpha_{01} = \exp(-\alpha_1\rho_{10}) \qquad \text{and} \qquad \alpha_1 = \operatorname*{Min}_i\left\{\frac{K_1}{\rho_{10}}, \frac{a_i}{\rho_{1i}}\right\}, \tag{5.18}$$

where a_i is as in (5.7).

5.3. Scaling for Multiple Chains with Single-Server and Infinite-Server Queues. For the general case, the generating function is as in (4.5). In order to carry out the innermost level of inversion (i.e., the pth or last step of inversion) with respect to z_p we may assume that z_i for $i = 1, 2, \ldots, p-1$ is constant. We rewrite (4.5) as

$$G(\mathbf{z}) = \left[\frac{\exp\left(\sum_{j=1}^{p-1}\rho_{j0}z_j\right)}{\prod_{i=1}^{q'}\left(1 - \sum_{j=1}^{p-1}\rho_{ji}z_j\right)^{m_i}}\right] \times\left[\exp(\rho_{p0}z_p)\Big/\prod_{i=1}^{q'}\left(1 - \frac{\rho_{pi}z_p}{1 - \sum_{j=1}^{p-1}\rho_{ji}z_i}\right)^{m_i}\right]. \tag{5.19}$$

The first factor on the right side of (5.19) is a constant, while the second factor is the same as in the one-dimensional inversion considered in Section 5.2 with ρ_{10} replaced by ρ_{p0} and ρ_{1i} replaced by $\rho_{pi}/[1-\Sigma_{j=1}^{p-1}\rho_{ji}z_j]$. The second parameter is complex, so we replace it by its modulus. Using (5.18), we get the scaling as

$$\alpha_{0p} = \exp(-\alpha_p \rho_{p0}) \qquad \text{and} \qquad \alpha_p = \operatorname*{Min}_{i}\left\{\frac{K_p}{\rho_{p0}}, \left(\frac{a_{ip}}{\rho_{pi}}\right)\left|1-\sum_{j=1}^{p-1}\rho_{ji}z_j\right|\right\}, \tag{5.20}$$

where

$$a_{ip} = \begin{cases} 1 & \text{for} \quad m_i = 1 \\ \left[\prod_{j=1}^{m_i-1}\dfrac{K_p+j}{K_p+2l_pK_p+j}\right]^{1/2l_pK_p} & \text{for} \quad m_i > 1. \end{cases} \tag{5.21}$$

Note that it is possible to have $\rho_{pi}=0$ for some i, but not all i. If $\rho_{pi}=0$, $a_{ip}/\rho_{pi}=\infty$, so that the ith term does not yield the minimum in (5.20).

Next we consider the inversion with respect to z_{p-1}. The generating function in this step is the inverse function in the pth step. Using (5.19), we can write it as

$$\begin{aligned} g^{(p-1)}(\mathbf{z}_{p-1},\mathbf{K}_p) &= I_p[G(\mathbf{z})] \\ &= \frac{\exp\left[\Sigma_{j=1}^{p-1}\rho_{j0}z_j\right]}{\Pi_{i=1}^{q'}\left[1-\Sigma_{j=1}^{p-1}\rho_{ji}z_j\right]^{m_i}} \\ &\quad \times \sum_{i=1}^{q'}\sum_{j=1}^{m_i}\sum_{k=0}^{K_p} A_{ij}\frac{(\rho_{p0})^k}{k!}\binom{j+K_p-k-1}{K_p-k} \\ &\quad \times \frac{\rho_{pi}^{K_p-k}}{\left[1-\Sigma_{j=1}^{p-1}\rho_{ji}z_j\right]^{K_p-k}} \end{aligned} \tag{5.22}$$

where A_{ij} has an expression similar to (5.3). The dominant term in (5.22) is of the form

$$\frac{\exp\left(\Sigma_{j=1}^{p-1}\rho_{j0}z_j\right)}{\Pi_{i=1}^{q'}\left(1-\Sigma_{j=1}^{p-1}\rho_{ji}z_j\right)^{K_p+m_i}}. \tag{5.23}$$

Note that in (5.22) we have implicitly assumed that $\rho_{pi}\neq 0$. If instead $\rho_{pi}=0$ for some i, then corresponding to that i the term

$$\frac{\rho_{pi}^{K_p-k}}{\left[1-\Sigma_{j=1}^{p-1}\rho_{ji}z_j\right]^{K_p-k}}$$

would be missing. Hence, instead of (5.23), in general the dominant term in (5.22) is

$$\frac{\exp\left(\sum_{j=1}^{p-1}\rho_{j0}z_j\right)}{\prod_{i=1}^{q'}\left[1-\sum_{j=1}^{p-1}\rho_{ji}z_j\right]^{K_p\eta_{pi}+m_i}}, \tag{5.24}$$

where $\eta_{pi} = 1$ if $\rho_{pi} \neq 0$ and $\eta_{pi} = 0$, otherwise.

Note that (5.24) is similar to $G(\mathbf{z})$ with p replaced by $p-1$ and m_i replaced by $K_p\eta_{pi} + m_i$. Therefore, from (5.20), we get the scaling in the $(p-1)$st step as $\alpha_{0,p-1} = \exp(-\alpha_{p-1}\rho_{p-1},0)$ and

$$\alpha_{p-1} = \operatorname*{Min}_{i}\left\{\frac{K_{p-1}}{\rho_{p-1,0}}, \left(\frac{a_{i,p-1}}{\rho_{p-1,i}}\right)\left|1-\sum_{j=1}^{p-2}\rho_{ji}z_j\right|\right\}, \tag{5.25}$$

where

$$a_{i,p-1} = \left[\sum_{j=1}^{N_{i,p-1}}\frac{K_{p-1}+j}{K_{p-1}+2l_pK_{p-1}+j}\right]^{1/2l_{p-1}K_{p-1}}, \tag{5.26}$$

where

$$N_{i,p-1} = K_p\eta_{pi} + m_i - 1$$

with η_{pi} defined as above.

Proceeding as above, we get the scaling in step j, $1 \le j \le p$, as

$$\alpha_{0j} = \exp(-\alpha_j\rho_{j0}) \qquad \text{and} \qquad \alpha_j = \operatorname*{Min}_{i}\left\{\frac{K_j}{\rho_{j0}}, \left(\frac{a_{ij}}{\rho_{ji}}\right)\left|1-\sum_{k=1}^{j-1}\rho_{ki}z_k\right|\right\}, \tag{5.27}$$

where

$$a_{ij} = \left[\sum_{l=1}^{N_{ij}}\frac{K_j+l}{K_j+2l_jK_j+1}\right]^{1/2l_jK_j} \tag{5.28}$$

and

$$N_{ij} = m_i - 1 + \sum_{k=j+1}^{p}K_k\eta_{ki}, \tag{5.29}$$

with $\eta_{ki} = 1$ if $\rho_{ki} \neq 0$ and $\eta_{ki} = 0$, otherwise. In (5.29) if an upper sum index is smaller than the lower sum index, then that should be interpreted as a vacuous sum.

Note that the scale parameters above are not constant. The scaling at the jth step depends on z_k for $1 \le k \le j-1$. Since the z_k's change during the course of inversion, so do the scale parameters. From (2.3), it is clear that the z_k values are always on a circle and therefore the modulus $|z_k|$ is constant. Furthermore, since the parameters ρ_{ki} are nonnegative the most restrictive scaling (smallest values of α_j) comes when $z_k = |z_k|$ (i.e., the point z_k is on the positive real line) for $1 \le k \le j-1$. From (2.3), it is clear that this *restrictive scaling* is indeed done once during the course of the inversion algorithm. If we use the restrictive scaling for all cases, then the scale parameter at the jth step becomes constant. The advantage of this scaling is that we then need to compute each scale parameter only once. Secondly, we need not recover the original inverse function from the inverse of the scaled generating function in each step using (2.10). Instead, the recovery may be done only once at the end of all inversions and all intermediate computation may be done using only scaled functions. The use of this restrictive scaling makes all the computations related to scaling insignificant compared to the overall computations. Through numerical investigations we observed that the restrictive scaling produces about the same accuracy as the scaling in (5.27), so that we recommend using it. It is as given below:

$$\alpha_{0j} = \exp(-\alpha_j \rho_{j0}) \qquad \text{and} \qquad \alpha_j = \operatorname*{Min}_{i} \left\{ \frac{K_j}{\rho_{j0}}, \left(\frac{a_{ij}}{\rho_{ji}} \right) \left(1 - \sum_{k=1}^{j-1} \rho_{ki} |z_k| \right) \right\}, \tag{5.30}$$

where a_{ij} is as given in (5.28).

Our numerical experience indicates that the scaling procedure just described works in almost all cases. However, for some very large examples (e.g., Example 8.4) the scale parameter α_j may need to be modified slightly. This is done by multiplying α_j by a factor β_j in the range of $0.8 \le \beta_j \le 1.2$. We can determine if further tuning is necessary by using the accuracy checked based on different sets of the l_j parameters.

5.4. SCALING WITH DIMENSION REDUCTION. In many situations, there will be dimension reduction. As indicated in Section 3, the possibility of dimension reduction can be checked using the interdependence graph approach. The scaling can be done independently of the dimension reduction, just as described in Section 5.3, except that the dimension reduction determines the order of the variables to be inverted. The d variables requiring full inversion become variables $z_1, \ldots, z_d$ and the remaining $p - d$ variables become $z_{d+1}, \ldots, z_p$; that is, the remaining variables appear in the inner loops.

It is also possible to directly determine scaling after the dimension reduction is done. We illustrate that in this subsection with an example. We also show how it is sometimes possible to replace a numerical inversion by an explicit formula.

Consider a closed queuing network with p closed chains, one infinite-server queue and $p - 1$ groups of single-server queues where each group i has m_i identical queues. The ith chain $(2 \le i \le p)$ goes through each single-server queue in group i at least once and the infinite-server queue any number of times. (For notational simplicity, we assume that the group index i ranges from 2 to p, so that there are no groups with index 1.) As for the first chain, it goes through the infinite-server queue and *all* single-server queues in the model at least once. Note that we have not fully specified the routing matrix here, which is not necessary for computing $g(\mathbf{K})$.

For this model, $q' = p - 1$ and $\rho_{ji} = 0$ unless $j = 1$ or $j = i$ or $i = 0$, that is, the generating function becomes

$$G(\mathbf{z}) = \frac{\exp\left[\sum_{j=1}^{p} \rho_{j0} z_j\right]}{\prod_{i=2}^{p} (1 - \rho_{ii} z_i - \rho_{1i} z_1)^{m_i}}. \tag{5.31}$$

To construct the interdependence graph for $G(\mathbf{z})$, we form a subgraph with two nodes for each factor in the denominator, z_1 and z_i for $2 \le i \le p$. Since

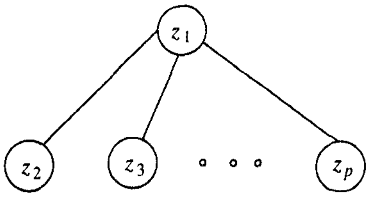

FIG. 1. Interdependence graph Γ for (5.31).

the numerator can be viewed as a product of p factors, each with one z variable, each factor in the numerator is represented by a one-node subgraph. Taking the union of all these subgraphs, we obtain the interdependence graph Γ for (5.31) is depicted in Figure 1.

If we delete the subset $D = \{z_1\}$, then Γ becomes disconnected into subsets $S_i(D) = \{z_i\}$ for $2 \leq i \leq p$. According to (3.2), the dimension of the inversion resulting from the subset D is 2 in this case. That is, the inversion dimension can be reduced from p to 2.

To show the inversion order more precisely, eq. (5.31) can be rewritten as

$$G(\mathbf{z}) = \frac{\exp(\rho_{10} z_1)}{\prod_{i=2}^{p}(1 - \rho_{1i} z_1)^{m_i}} \prod_{i=2}^{p} \frac{\exp(\rho_{i0} z_i)}{(1 - (\rho_{ii} z_i / 1 - \rho_{1i} z_1))^{m_i}}. \tag{5.32}$$

If we keep z_1 constant, then the first factor on the right side of (5.32) becomes constant and the second factor becomes a product of $p - 1$ terms, each of which is a function of a single z variable. Each such factor may be inverted independently with respect to the corresponding z variable, and once we take a product of each inverse function, we get a function of z_1 and $K_2, K_3, \ldots, K_p$. A final inversion with respect to z_1 yields the desired normalization constant $g(\mathbf{K})$. So the p-dimensional problem is reduced to a two-dimensional one. Next comes the question of scaling.

From the scaling in Section 5.2, we see that the scaling required to invert the ith factor $(2 \leq i \leq p)$ in (5.32) is given by

$$\alpha_{0i} = \exp(-\alpha_i \rho_{i0}) \qquad \text{and} \qquad \alpha_i = \operatorname{Min}\left\{ \frac{K_i}{\rho_{i0}}, \left(\frac{a_i}{\rho_{ii}} \right)(1 - \rho_{1i}|z_1|) \right\}, \tag{5.33}$$

where

$$a_i = \begin{cases} 1 & \text{for} \quad m_i = 1 \\ \left[\displaystyle\prod_{j=1}^{m_i - 1} \frac{K_i + j}{K_i + 2 l_i K_i + j} \right]^{1/2 l_i K_i} & \text{for} \quad m_i > 1. \end{cases} \tag{5.34}$$

It is also possible to explicitly invert the ith product factor $(2 \leq i \leq p)$ in (5.32) and, when we do, we get

$$g^{(1)}(\mathbf{z}_1, \mathbf{K}_2) = \frac{\exp(\rho_{10} z_1)}{\prod_{i=2}^{p}(1 - \rho_{1i} z_1)^{m_i}} \times \prod_{i=2}^{p} \sum_{k=0}^{K_i} \frac{(\rho_{i0})^k}{k!} \binom{m_i + K_i - k - 1}{K_i - k} \frac{\rho_{ii}^{K_i - k}}{(1 - \rho_{1i} z_1)^{K_i - k}}. \tag{5.35}$$

The dominant term in (5.35) is of the form

$$\frac{\exp(\rho_{10} z_1)}{\prod_{i=2}^{p}(1-\rho_{1i} z_1)^{K_i+m_i}}.$$

Therefore, from (5.18), we get the scaling in order to invert with respect to z_1 as

$$\alpha_{0,1} = \exp(-\alpha_1 \rho_{10}) \qquad \text{and} \qquad \alpha_1 = \operatorname*{Min}_{i}\left\{\frac{K_1}{\rho_{10}}, \frac{a_{i1}}{\rho_{1i}}\right\}, \tag{5.36}$$

where

$$a_{i1} = \left[\sum_{j=1}^{N_{i1}} \frac{K_1 + j}{K_1 + 2l_1 K_1 + j}\right]^{1/2l_1K_1} \tag{5.37}$$

and

$$N_{i1} = m_i - 1 + K_i. \tag{5.38}$$

It is to be noted that all scaling done in this subsection could also be derived from the final scaling equations in Section 5.3 ((5.28)–(5.30)); in this section $\eta_{ji} = 1$ for $j = 1$ and $j = i$ and $\eta_{ji} = 0$, otherwise.

5.5. Near Multiplicities. We have indicated that much of the difficulty in the scaling is due to the possibility of multiple factors in the denominator of (4.5). It should be intuitively clear that these same difficulties can arise with near multiplicities, and that so far our scaling algorithm does not account for near multiplicities. Moreover, in an application we might not recognize exact multiplicities.

In this subsection, we introduce a heuristic scheme to account for near multiplicities. Consider the single-chain setting with only single-server queues in Section 5.1, and assume, without loss of generality, that $\rho_{1i} \ge \rho_{1,i+1}$ for all i. (This involves a sorting, which is of low complexity.) Without multiplicities or near multiplicities, the scaling should be $a_1 = 1/\rho_{11}$, but with multiplicities or near multiplicities perhaps other terms should play a role in the minimum in (5.6). To capture the effect of near multiplicities, for the scaling only, we replace ρ_{1i}, in (5.1) and (5.6) by the average of the i largest relative traffic intensities, that is

$$\hat{\rho}_{1i} = \frac{1}{i}\sum_{k=1}^{i} \rho_{1k}. \tag{5.39}$$

Moreover, in (5.7) we act as if the multiplicity associated with the ith group of queues is

$$\hat{m}_i = \sum_{k=1}^{i} m_k; \tag{5.40}$$

that is, we replace a_i in (5.7) with $\hat{a}_i$ based on $\hat{m}_i$ in (5.40). Note that $\hat{a}_1 = a_1$ and $\hat{\rho}_{11} = \rho_{11}$, but that $\hat{a}_i \le a_i$ and $\rho_{1i} \ge \rho_{1i}$, so that the first ratio a_1/ρ_{11} is unchanged, but the other ratios in (5.6) are decreased, which may reduce α_1 in (5.6). The extra reduction will tend to occur when there are near multiplicities.

Based on this idea, we propose replacing the restrictive scaling in (5.30) by an even more restrictive scaling. We let

$$\alpha_{0j} = \exp(-\alpha_j \rho_{j0}) \qquad \text{and} \qquad \alpha_j = \operatorname*{Min}_i \left\{ \frac{K_j}{\rho_{j0}}, \frac{a_{ij}}{\bar{\rho}_{ji}} \right\}, \tag{5.41}$$

where

$$a_{ij} = \left[\sum_{l=1}^{N_{ij}} \frac{K_j + l}{K_j + 2l_j K_j + l} \right]^{1/2l_j K_j}, \tag{5.42}$$

$$N_{ij} = \overline{m}_i - 1 + \sum_{k=j+1}^{p} K_k \eta_{ki} \tag{5.43}$$

$$\bar{\rho}_{ji} = \begin{cases} 0 & \tilde{\rho}_{ji} = 0 \\ \dfrac{1}{i} \displaystyle\sum_{k=1}^{i} \tilde{\rho}_{jk}, & \tilde{\rho}_{ji} > 0, \end{cases} \tag{5.44}$$

$$\overline{m}_i = \sum_{k=1}^{i} \overline{m}_k, \tag{5.45}$$

$$\eta_{ki} = \begin{cases} 1 & \text{if} \quad \rho_{ki} \neq 0 \\ 0 & \text{if} \quad \rho_{ki} = 0 \end{cases} \tag{5.46}$$

with $\{ \tilde{\rho}_{ji}\colon 1 \le i \le q' \}$ being the sorted version (in decreasing order of magnitude) of $\{ \rho_{ji}/(1 - \Sigma_{k=1}^{j-1} \rho_{ki}|z_k|)\colon 1 \le i \le q' \}$ and $\{ \tilde{m}_i\colon 1 \le i \le q' \}$ is the rearranged version of $\{ m_i\colon 1 \le i \le q \}$ associated with $\{ \hat{\rho}_{ji} \}$.

6. *Moments via Generating Functions*

Given the steady-state probability mass function, we can calculate moments. Without loss of generality, let $(r, i) \in C_1$. We start with a standard expression for the probability mass function of q_{1i}, the number of chain 1 customers at queue i, namely.

$$P(q_{1i} = k) = \frac{\rho_{1i}^k \bigl(g(\mathbf{K} - k\mathbf{1}_1) - \rho_{1i} g(\mathbf{K} - (k+1)\mathbf{1}_1)\bigr)}{g(\mathbf{K})}, \tag{6.1}$$

see (3.257) on p. 147 of Lavenberg [1983]. (A similar expression holds for the mass function of n_{ri} [Lavenberg 1983]. It involves ρ'_{ri} instead of ρ_{1i}.)

From the telescoping property of (6.1), we can write the tail probabilities as

$$P(q_{1i} \ge k) = \frac{\rho_{1i}^k g(\mathbf{K} - k\mathbf{1}_1)}{g(\mathbf{K})}. \tag{6.2}$$

From (6.2) we obtain the standard formula for the mean,

$$E[q_{1i}] = \sum_{k=1}^{\infty} P(q_{1i} \ge k) = \sum_{k=1}^{K_1} \rho_{1i}^k \frac{g(\mathbf{K} - k\mathbf{1}_1)}{g(\mathbf{K})}; \tag{6.3}$$

for example, see (3.258) on p. 147 of Lavenberg [1983]. Unfortunately, formula (6.3) is not too convenient for us, because it requires $K_1 + 1$ normalization function calculations and thus $K_1 + 1$ numerical inversions. We now show how this mean can be calculated by *two* inversions.

For this purpose, we rewrite (6.3) as

$$E[q_{1_t}] = \frac{\rho_{1_t}^{K_1} h(\mathbf{K})}{g(\mathbf{K})} - 1, \tag{6.4}$$

where

$$h(\mathbf{K}) = \sum_{k=0}^{K_1} \rho_{1_t}^{-k} g(k, \mathbf{K}_2). \tag{6.5}$$

Let $H(z_1)$ be the generating function of $h(\mathbf{K})$ with respect to K_1. Then

$$\begin{aligned} H(z_1) &= \sum_{m=0}^{\infty} z_1^m h(m, \mathbf{K}_2) \\ &= \sum_{m=0}^{\infty} z_1^m \sum_{k=0}^{m} \rho_{1_t}^{-k} g(k, \mathbf{K}_2) \\ &= \sum_{k=0}^{\infty} \rho_{1_t}^{-k} g(k, \mathbf{K}_2) \sum_{m=k}^{\infty} z_1^m \\ &= \frac{g^{(1)}(z_1/\rho_{1_t}, \mathbf{K}_2)}{1 - z_1}, \end{aligned} \tag{6.6}$$

where $g^{(1)}(\mathbf{z}_1, \mathbf{K}_2)$ is the partial generating function in (2.1). Now, if $H(\mathbf{z})$ represents the full generating function of $h(\mathbf{K})$, then from (6.6) it is clear that

$$H(\mathbf{z}) = \frac{G(z_1/\rho_{1_t}, z_2, \ldots, z_p)}{1 - z_1}. \tag{6.7}$$

Since $H(\mathbf{z})$ is of the same form as $G(\mathbf{z})$ it may be inverted by the established inversion procedure. Hence, we can obtain the mean $E[q_{1_t}]$ using two inversions from (6.4). We invert $G(\mathbf{z})$ and $H(\mathbf{z})$, respectively, to obtain $g(\mathbf{K})$ and $h(\mathbf{K})$.

By the same approach, we can also calculate higher moments. For example,

$$E[q_{1_t}(q_{1_t} - 1)] = 2 \sum_{k=0}^{\infty} k P(q_{1_t} \geq k) = \frac{2\rho_{1_t}^{K_1} h_1(\mathbf{K})}{g(\mathbf{K})}, \tag{6.8}$$

where

$$h_1(\mathbf{K}) = \sum_{k=0}^{K_1} k \rho_{1_t}^{-k} g(k, \mathbf{K}_2). \tag{6.9}$$

Let $H_1(z_1)$ be the generating function of $h_1(\mathbf{K})$ with respect to K_1. Then

$$\begin{aligned} H_1(z_1) &= \sum_{m=0}^{\infty} z_1^m h_1(m,\mathbf{K}_2) \\ &= \sum_{m=0}^{\infty} z_1^m \sum_{k=0}^{K_1} k\rho_{1i}^{-k} g(k,\mathbf{K}_2) \\ &= \sum_{k=0}^{\infty} k\rho_{1i}^{-k} g(k,\mathbf{K}_2) \sum_{m=k}^{\infty} z_1^m \\ &= \left(\frac{z_1}{1-z_1}\right) \frac{\partial}{\partial z_1} g^{(1)}\left(\frac{z_1}{\rho_{1i}}, \mathbf{K}_2\right). \end{aligned} \tag{6.10}$$

Moreover, the full generating function of $h_1(K)$ is

$$H_1(\mathbf{z}) = \frac{z_1}{1-z_1} \frac{\partial}{\partial z_1} G\left(\frac{z_1}{\rho_{1i}}, z_2, \dots, z_p\right). \tag{6.11}$$

Finally, note from (4.5) that

$$\frac{\partial G(\mathbf{z})}{\partial z_1} = \left(\rho_{10} + \sum_{i=1}^{q'} \frac{m_i \rho_{1i}}{\left(1 - \sum_{j=1}^{p} \rho_{ji} z_j\right)}\right) G(\mathbf{z}), \tag{6.12}$$

so that the new generating function is a sum of generating functions of the same form as $G(\mathbf{z})$.

For higher factorial moments, we proceed in the same way using (6.2) and

$$E[q_{1i}(q_{1i}-1)\cdots(q_{1i}-l+1)] = l\sum_{k=0}^{K_1} k(k-1)\cdots(k-l+2)P(q_{1i} \geq k). \tag{6.13}$$

For more on moments, see McKenna [1989], McKenna and Mitra [1984], and references there.

7. *A Summary of the Algorithm*

For general p-dimensional transforms, the overall inversion algorithm is obtained by combining (2.2), (2.3), (2.5), (2.9), and (2.10), which requires specifying the parameters γ_j, l_j, α_{0j} and α_j, $1 \leq j \leq p$. The function $g(\mathbf{K})$ is thus calculated by p nested inversions of $\overline{G}_j(z_j)$, $1 \leq j \leq p$, in (2.9). We first determine the order of the variables to be inverted by applying the dimension reduction scheme in Section 3. Given the order of the variables to be inverted, we use the special structure of the closed queuing networks to determine the scale parameters α_{0j} and α_j. In particular, we use the scale parameters α_{0j} and α_j in (5.41) in (2.9). We then apply (2.2)–(2.5) to calculate $\bar{g}_1(K_1) = \bar{g}(\mathbf{K})$ and (2.10) to calculate $g(\mathbf{K})$. In particular, from (2.10) we obtain

$$g(\mathbf{K}) = \prod_{j=1}^{p} \alpha_{0j}^{-1} \prod_{j=1}^{p} \alpha_j^{-K_j} \bar{g}(\mathbf{K}). \tag{7.1}$$

Finally, to do (2.2)–(2.5), we need to specify the inversion parameters γ_j and l_j.

As indicated in Section 2, as standard values we use $l_1 = 1, \gamma_1 = 11; l_2 = l_3 = 2$, $\gamma_2 = \gamma_3 = 13; l_4 = l_5 = l_6 = 3, \gamma_4 = \gamma_5 = \gamma_6 = 15$. If necessary, we increase γ_j to reduce the aliasing error, and we increase l_j to reduce the roundoff error, with the understanding that this will increase the computation time. The general idea is to control errors more carefully on the inner loops (for large j).

We have just given a full algorithm with specific values for the parameters γ_j, l_j, α_{0j} and α_j, $1 \leq j \leq p$. These parameters can be used for tuning if necessary. We repeat the calculation with different sets of parameters l_j to obtain an accuracy check.

For large models, we can calculate using logarithms, as indicated in Section 2.2, and we can exploit shared computation of closely related normalization constants as indicated at the end of Section 4.

8. *Numerical Examples*

All computations reported in this section were done on a SUN SPARC-2 workstation using double-precision arithmetic.

As a preliminary sanity check, we verified the accuracy of our algorithm by testing it against simple examples for which the exact value is readily available. In particular, we tested our algorithm against the classical Koenigsberg [1958] formula for single-chain models with only distinct single-server queues, [Bertozzi and McKenna 1993, (3.9)], and the generalization allowing IS queues [Bertozzi and McKenna 1993, (3.22)]. In all cases, there was agreement to 8–12 significant digits. Furthermore, whenever our results based on two different contours agreed up to n significant places, they also agreed with the alternate formula to that many places. This observation indicates that we can rely on our internal accuracy check.

In this section, we give numerical results for our algorithm applied to four more challenging closed queuing network examples. For each example, we calculate the normalization constant $g(\mathbf{K})$ in (1.2) and (4.3) for specified population vectors $\mathbf{K}$ from the generating function $G(\mathbf{z})$ in (4.5). Thus, the parameters are the number of chains, p, the number of distinct single-server queues, q', the multiplicities m_i, the aggregate relative traffic intensities ρ_{ji}, $1 \leq j \leq p$, $0 \leq i \leq q'$, and the desired population vector $\mathbf{K}$.

Note that the normalization constant $g(\mathbf{K})$ only depends on these parameters p, q', m_i, ρ_{ji} and $\mathbf{K}$. Hence, we do not fully specify the models below. In particular, we do not give the routing probabilities $R_{ri,sj}$ or the mean service times t_{ri}. Thus, there are many detailed models consistent with our partial model specifications. One possible routing matrix consistent with the data that we provide is a cyclic routing matrix, all of whose entries are 0's and 1's, which yields visit ratios $e_{ri} = 1$ for all stages (r, i) from (4.2). If we consider this case, then $t_{ri} = \rho'_{ri}$ and the throughputs θ_{ri} in (4.7) coincide with the normalization constant ratios $g(\mathbf{K} - \mathbf{1}_j)/g(\mathbf{K})$. We also display these ratios along with the values of $g(\mathbf{K})$ in our numerical results below. We note that the throughputs for any more detailed model can be found by solving (4.2) for the visit ratios e_{ri} and then applying (4.7).

For the first three examples, the total number of single-server queues is the same, namely, $q = 50$. However, in each example we consider ten distinct queues, each with multiplicity five. Thus, $q' = 10$ in each example. This reduces the computational complexity for our algorithm, but not for the others. We also allow for IS queues, with almost no additional computational complexity. (Note that any number is IS queues can be aggregated trivially into a single IS queue.) Multiplicities and the presence is IS queues evidently complicate the theory of residues [Bertozzi and McKenna 1993].

What is different in our examples is the number of closed chains. The first example has only one chain, and is thus the easiest example. The second example has four chains, while the third and fourth examples have eleven chains. However, the dimension can be reduced from eleven to two in the last two examples, whereas the dimension cannot be reduced below four in the second example, so that for our algorithm the second example is the most difficult. The last case of the second example took about half an hour (on the SUN SPARC-2 workstation).

The numerical results below are shown to seven significant figures, which is more than adequate for most applications. However, the realized accuracy was found to be 8–12 significant places. This was determined either by comparison with the convolution algorithm or by our internal accuracy check using two sets of l_j values for each case. (The parameters l_j appears in (2.3); see Section 2.3.) Each number below is expressed as aek, which means $a \times 10^k$.

For all the examples, we used Euler summation whenever K_j exceeds C, requiring the computation of C terms. Euler summation works in all these examples, providing tremendous computational savings whenever a particular K_j is large. For all examples, $C = 100$ was adequate and for most examples $C = 31$ was adequate.

Example 8.1. In the first example, there are an arbitrary number of infinite-server queues and a total of 50 single-server queues, which consist of 10 distinct queues, each with a multiplicity of 5. The model has one closed chain going through all infinite-server queues and each single-server queue at least once.

The classical closed-form expression for the normalization constant $g(K)$ due to Koenigsberg [1958; 1986] and Harrison [1985] holds in a single-chain model when there are no multiplicities and no load-dependent servers; see Bertozzi and McKenna [1993, (3.9)]. They derived corresponding closed-form expressions for $g(K)$ when there are multiplicities but no load-dependent servers [Bertozzi and McKenna 1993, (3.12)] and when there is an IS queue but no multiplicities [Bertozzi and McKenna 1993, (3.22)], but evidently no closed-form expression has yet been derived for the case considered here in which there are both multiplicities and an IS queue. Moreover, with either multiplicities or an IS queue, our algorithm may be competitive with computation from the closed-form formulas. However, without multiplicities or load-dependent servers, the simple classical formula [Bertozzi and McKenna 1993, (3.9)] seems clearly best.

Here $p = 1$, $q' = 10$ and $m_i = 5$, $1 \le i \le 10$. The relative traffic intensities are $\rho_{10} = 5$ and

$$\rho_{1i} = 0.1i \qquad \text{for} \quad i = 1, 2, \dots, 10,$$

so that ρ_{1i} ranges from 0.1 to 1.0. We consider eight cases for the total population: $K_1 = 2 \times 10^k$ for $k = 0, 1, \dots, 7$. We give the numerical below in Table I.

TABLE I. NUMERICAL RESULTS FOR EXAMPLE 8.1.

population K_1	normalization constant $g(K_1)$	ratio $g(K_1 - 1)/g(K_1)$
2	5.377500e2	6.043701e-2
20	1.906584e13	4.461782e-1
200	1.381312e26	9.659115e-1
2000	1.284918e31	9.978987e-1
20,000	1.541538e35	9.997990e-1
200,000	1.569301e39	9.999800e-1
2,000,000	1.572100e43	9.999980e-1
20,000,000	1.572380e47	9.999998e-1

For all but the last case, the numbers agree with the convolution algorithm (the agreement is more than the displayed seven digits; often up to 12 digits). A straightforward implementation of the convolution algorithm seemed to have a numerical under/overflow problem, but we corrected this by working with logarithms and ratios of very large or very small quantities. In the last case, the convolution algorithm runs out of storage space, so that we verify the accuracy only by our internal accuracy check using two different values of l_1 as explained in Section 2.3.

In terms of computation time, our inversion algorithm is virtually instantaneous (less than one half-second) for all these cases, because Euler summation with 31 terms works. In contrast, the convolution algorithm took 2 minutes for $K_1 = 200$ and 20 minutes for $K_1 = 2000$. Thus, we would project that the convolution algorithm would take over three hours for the last case with $K_1 = 20{,}000$.

To provide a further computational check, we also considered this same example with the multiplicity of each queue changed from 5 to 1. Then we can also apply formula (3.22) of Bertozzi and McKenna [1993]. Moreover, from that formula we can calculate the limiting value of the normalization constant as $K_1 \to \infty$, which is finite in this case. In particular, in this case

$$\lim_{K_1 \to \infty} g(K_1) = \frac{\exp(\rho_{1,0}/\rho_{1,10})}{\prod_{i=1}^{9}(\rho_{1,10} - \rho_{1,i})} = \frac{e^5 10^9}{9!} = 408986.88024.$$

Here are some of our results: For $K_1 = 2$, $g(K_1) = 57.050000000$; for $K_1 = 200$. $g(K_1) = 408986.87849$; and for $K_1 \geq 2000$, $g(K_1)$ agrees with the asymptotic formula to all exhibited digits. For the first two cases, our results agree with Bertozzi and McKenna [1993, (3.22)] to all exhibited digits.

Example 8.2. This example is identical to Example 8.1, except that now $p = 4$ and each of the four chains goes through each of the 50 single-server queues at least once and the first chain does not go through any infinite-server queue. No dimension reduction is possible in this case, because ρ_{ji} is positive for all i and j. Since we have 10 distinct queues, each with multiplicity 5, this model does not satisfy the assumptions for the residue results in Bertozzi and McKenna [1993]. However, as we have noted before, the multiplicities reduce the computational complexity here. The relative traffic intensities are:

$$\rho_{10} = 0, \qquad \rho_{20} = 5, \qquad \rho_{30} = 10, \qquad \rho_{40} = 20$$

TABLE II. NUMERICAL RESULTS FOR EXAMPLE 8.2.

chain populations					
K_1	K_2	K_3	K_4	normalization constant $g(\mathbf{K})$	ratio $g(\mathbf{K}-\mathbf{1}_1)/g(\mathbf{K})$
2	2	2	2	3.915239e14	6.347791e-2
20	2	2	2	1.066795e25	4.429595e-1
100	2	2	2	2.254450e36	8.677053e-1
500	2	2	2	3.288626e44	9.793468e-1
5000	2	2	2	4.527130e54	9.979961e-1
20	20	2	2	3.859354e45	3.499500e-1
100	100	2	2	2.719940e129	5.030302e-1
500	500	2	2	1.653278e497	5.221141e-1
500	500	5	5	1.026054e515	5.100207e-1

and

$$\rho_{ji} = 0.1i + j - 1 \qquad \text{for} \quad i = 1, \ldots, 10 \text{ and } j = 1, \ldots, 4$$

so that ρ_{ji} ranges from 0.1 to 4.0. (Again there are both multiplicities and near multiplicities.) We give numerical results for nine cases in Table II.

Here again, we verified accuracy in all but the last case using the convolution algorithm. The agreement, again, is more than the displayed seven digits. The convolution algorithm runs out of storage space for the last case, so that we verify accuracy in that case only by our internal accuracy check by, for which we used the following sets of l_j parameters:

Set 1: $l_1 = 1$, $l_2 = 2$, $l_3 = 2$, $l_4 = 3$
Set 2: $l_1 = 1$, $l_2 = 2$, $l_3 = 3$, $l_4 = 3$

The computation time for our algorithm is half an hour for the last case using Euler summation with 31 terms, which effectively reduces both K_1 and K_2 from 500 to 31. If the convolution algorithm did not have a storage problem, it would have taken over three and a half hours in the last case. It is to be noted that even if we multiply K_1 and K_2 by a factor of 10 or 100, our computation time would remain about the same, but the convolution algorithm would have its run-time and storage requirement multiplied by 10^2 or 100^2.

Example 8.3. Again $q = 50$, $q' = 10$, and $m_i = 5$, $1 \leq i \leq 10$, but now $p = 11$. However, here it is possible to reduce the dimension to 2, so that this example is actually easier than Example 8.2 for our algorithm. (This would also be the case for the tree convolution algorithm [Lam and Lien 1983.]) The relative traffic intensities are:

$$\begin{aligned} \rho_{j0} &= 5j - 10 & &\text{for} \quad j = 2, 3, \ldots, 11, \text{ and } \rho_{10} = 50, \\ \rho_{jj} &= 0.1(j-1) & &\text{for} \quad j = 2, \ldots, 11, \\ \rho_{1j} &= 1 + 0.1(j-1) & &\text{for} \quad j = 2, \ldots, 11, \end{aligned}$$

with $\rho_{ji} = 0$ for all other (j, i).

This example is precisely of the form considered in Section 5.4. Indeed, the generating function is given in (5.31) and the interdependence graph is given in Figure 1 there. Thus, the dimension can be reduced from 11 to 2. The numerical results for eight cases are given in Table III.

TABLE III. NUMERICAL RESULTS FOR EXAMPLE 8.3.

chain populations			
K_j for $2 \le j \le 11$,	K_1	normalization constant $g(\mathbf{K})$	ratio $g(\mathbf{K}-\mathbf{1}_1)/g(\mathbf{K})$
2	2	1.235628e25	1.492001e-2
2	20	7.503087e45	1.296652e-1
2	200	5.970503e129	4.477497e-1
2	2000	1.937826e683	4.982502e-1
$5(j-1)$	2	3.004462e107	8.039024e-3
$5(j-1)$	20	1.677866e133	6.803960e-2
$5(j-1)$	200	8.032122e260	2.858885e-1
$5(j-1)$	2000	1.617153e926	4.746674e-1

TABLE IV. NUMERICAL RESULTS FOR CASE 1 OF EXAMPLE 8.4.

chain populations			
K_j for $2 \le j \le 11$	K_1	normalization constant $g(\mathbf{K})$	ratio $g(\mathbf{K}-\mathbf{1}_1)/g(\mathbf{K})$
200	20	1.232036e278	4.582983e-3
200	200	2.941740e579	4.281094e-2
200	2000	3.399948e2037	2.585489e-1
200	20,000	9.07177e8575	4.846930e-1

In each case, the inversion for variables $z_2, z_3, \ldots, z_{11}$ are done explicitly, using (5.35), and hence no l_j is involved for $2 \le j \le 11$. Hence, only a one-dimensional algorithm is required for this example. The accuracy is checked by doing the calculation with $l_1 = 1$ and $l_1 = 2$.

Example 8.4. This is our most challenging problem. It is the same as Example 8.3 except that we change the multiplicities m_i and the chain populations K_j. We increase m_i from 5 to 100, $1 \le i \le 10$. Hence, now there are $q = 1000$ single-server queues, but still only $q' = 10$ distinct queues. We consider three cases. First, we increase the chain populations to $K_j = 200$ for $2 \le j \le 11$. We obtain three subcases by considering three different values for K_1. Our numerical results are given below in Table IV.

As in Example 8.3 the accuracy was checked by performing the calculations twice, once with $l_1 = 1$ and once with $l_1 = 2$. Again, the inversions for variables $z_2, z_3, \ldots, z_{11}$ are done explicitly by (5.35), so that the numerical inversion was essentially one-dimensional.

The results in Table IV were obtained in less than 1 minute by exploiting Euler summation with 51 terms. This example would seem to be out of the range of existing algorithms, with the possible exception of the tree-convolution algorithm and the ones based on asymptotics [Knessl and Tier 1992; McKenna and Mitra 1982]. For example, convolution would require a storage of size $200^{10} \times 2 \times 10^4 = 2.5 \times 10^{27}$ for the last case of Table IV. The last case would appear to be difficult even for the tree algorithm.

From McKenna and Mitra [1982; 1984], McKenna et al. [1981]; and Ramakrishnan and Mitra [1982], we see that the PANACEA algorithm based on asymptotics requires that there be an IS queue and that each chain visit this queue. Unlike McKenna and Mitra [1982; 1984], McKenna et al. [1981]; and Ramakrishnan and Mitra [1982], the asymptotic approximation in Knessl and Tier [1992] does not require any IS queue, but it requires that *all* chain populations be large. To show that our algorithm does not have such limitations, we consider two modifications of Case 1. Case 2 has classes 1 and 2 with

TABLE V. NUMERICAL RESULTS FOR CASE 2 OF EXAMPLE 8.4.

chain populations K_j for $2 \le j \le 11$	K_1	normalization constant $g(\mathbf{K})$	ratio $g(\mathbf{K}-\mathbf{1}_1)/g(\mathbf{K})$
in all cases:	2	3.842031e407	5.128582e-4
$K_2 = 2$,	20	1.484823e454	5.087018e-3
$K_3 = 5$ and	200	6.003231e747	4.706783e-2
$K_j = 200,\ 4 \le j \le 11$	2000	5.442693e2154	2.705391e-1
	20,000	2.494765e8617	4.852817e-1

TABLE VI. NUMERICAL RESULTS FOR CASE 3 OF EXAMPLE 8.4.

chain populations K_j for $2 \le j \le 11$	K_1	normalizations constant $g(\mathbf{K})$	ratio $g(\mathbf{K}-\mathbf{1}_1)/g(\mathbf{K})$
in all cases:	2	9.959619e313	4.762073e-4
$K_2 = 2$,	20	1.447107e361	4.728591e-3
$K_3 = 5$ and	200	1.222889e660	4.417444e-2
$K_j = 200,\ 4 \le j \le 11$	2000	2.948943e2096	2.645993e-1
	20,000	4.210541e8588	4.851015e-1

small populations, while the other class populations remain large. In particular, we let $K_2 = 2$ and $K_3 = 5$. Numerical results for Case 2 appear below in Table V.

Case 3 is a modification of Case 2 in which we remove all the IS nodes, that is, we set $\rho_{j0} = 0$ for all j. Numerical results for Case 3 appear below in Table VI. As for Case 1, Cases 2 and 3 required about a minute on the SUN SPARC-2 workstation.

Before closing this section, we point out that if the model is such that we cannot take advantage of any of our speed-up techniques (namely, dimension reduction, fast summation of large sums and large queue multiplicities), then our algorithm will be slower than the convolution algorithm, as already indicated in Section 2.5. Similarly, if dimension reduction is possible but none of the other speed-ups, then our algorithm will be slower than the tree convolution algorithm, which also does dimension reduction in the time domain.

To illustrate, Lam and Lien [1983] analyze an example with 64 queues and 32 chains, requiring about 3×10^7 operations. We analyzed this model and observed that the effective dimension can be reduced from 32 to 9. However, all chain populations are between 1 and 5 and so our speed-up technique based on Euler summation does not apply. Also there are no multiplicities. We estimated that our operation count for this example would be about 10^{12}, so that our algorithm is considerably slower than the tree-convolution algorithm, even though our algorithm is faster than the pure convolution algorithm, which has an operation count of about 10^{23}. It appears that our algorithm nicely complements the tree algorithm, because the tree algorithm will be faster if the effective dimension-after-dimension reduction remains large but all chain populations are small. In contrast, our algorithm will be faster if the effective dimension-after-dimension reduction is small (typically 5 or less) but some of the chain populations are large.

9. *Conclusions*

We have presented a general algorithm for calculating normalization constants in product-form models by numerically inverting their generating functions (Section 2). An advantage of this algorithm over recursive algorithms is the very small storage requirement. Another advantage is the fast computation of large sums using acceleration methods, in particular Euler summation. We have shown how the dimension can often be dramatically reduced by exploiting conditional decomposition (Section 3). We have indicated that this dimension reduction scheme is closely related to the tree convolution algorithm of Lam and Lien [1983].

We have considered in detail the special case of closed queuing networks with only single-server and IS queues, where the IS queues are optional (Section 4), and developed an explicit scaling algorithm for this class of models (Section 5). We have shown how to calculate mean queue lengths and higher-order moments directly by performing only two inversions of the same form as for the normalization constant (Section 6). We have summarized the inversion algorithm (Section 7). Finally, we have presented a few numerical examples illustrating the algorithm and showing that it can solve some challenging problems (Section 8).

In subsequent work (e.g., [Choudhury et al. 1995]), we have developed detailed scaling algorithms for other subclasses of product-form models, extending Section 5. In other, more recent work, we have developed an effective automatic, dynamic scaling algorithm that requires only limited knowledge of special generating function structure, in the spirit of Lam [1982]. This scaling allows us to analyze models with general state-dependent service rates just as efficiently as the models considered in this paper.

In conclusion, we think that the numerical inversion algorithm here usefully complements existing algorithms for closed queuing networks and related models, and that there is significant potential for further progress with this approach.

ACKNOWLEDGMENT. We thank Yaakov Kogan for discussion that helped motivate this work. We thank Yonatah Levy and the referees for useful comments.

REFERENCES

ABATE, J., AND WHITT, W. 1992a. The Fourier-series method for inverting transforms of probability distributions. *Que. Sys. 10*, 5–88.

ABATE, J., AND WHITT, W. 1992b. Numerical inversion of probability generating functions. *Op. Res. Letters 12*, 245–251.

BERTOZZI, A., AND MCKENNA, J. 1993. Multidimensional residues, generating functions, and their application to queueing networks *SIAM Rev. 35*, 239–268.

BIRMAN, A., FERGUSON, D., AND KOGAN, Y. 1992. Asymptotic solutions for a model of large multiprogramming systems with multiple workloads. IBM T. J. Watson Research Center, Yorktown Heights, N.Y.

BRUEL, S. C., AND BALBO, G. 1980. *Computational Algorithms for Closed Queueing Networks.* Elsevier, New York.

BUZEN, J. P. 1973. Computational algorithms for the closed queueing networks with exponential servers. *Commun. ACM 16*, 9 (Sept.), 527–531.

CHOUDHURY, G. L., LEUNG, K. K., AND WHITT, W. 1995. An algorithm to compute blocking probabilities in multi-rate multi-class multi-resource loss models. *Adv. Appl. Prob.*, to appear.

CHOUDHURY, G. L., AND LUCANTONI, D. M. 1995. Numerical computation of the moments of a probability distribution from its transforms. *Op. Res.*, to appear.

Choudhury, G. L., Lucantoni, D. M., and Whitt, W. 1994a. Multidimensional transform inversion with applications to the transient M/G/1 queue. *Ann. Appl. Prob. 4*, 719–740.

Choudhury, G. L., Lucantoni, D. M., and Whitt, W. 1994b. Numerical transform inversion to analyze teletraffic models. *The Fundamental Role of Teletraffic in the Evolution of Telecommunications Networks, Proceedings of the 14*th *International Teletraffic Congress*, J. Labetoulle and J. W. Roberts, eds. Elsevier, Amsterdam, the Netherlands, pp. 1043–1052.

Choudhury, G. L., Lucantoni, D. M., and Whitt, W. 1995. Numerical solution of $M_t/G_t/1$ queues. *Op. Res.*, to appear.

Conway, A. E., and Georganas, N. D. 1986a. RECAL—A new efficient algorithm for the exact analysis of multiple-chain closed queuing networks. *J. ACM 33*, 4 (Oct.), 768–791.

Conway, A. E., and Georganas, N. D. 1986b. Decomposition and aggregation by class in closed queueing networks. *IEEE Trans. Softw. Eng. 12*, 1025–1040.

Conway, A. E., and Georganas, N. D. 1989. *Queueing Networks—Exact Computational Algorithms: A Unified Theory Based on Decomposition and Aggregation.* MIT Press, Cambridge, Mass.

Conway, A. E., de Souza e Silva, E., and Lavenberg, S. S. 1989. Mean value analysis by chain of product form queueing networks. *IEEE Trans. Comput. 38*, 432–442.

de Souza e Silva, E., and Lavenberg, S. S. 1989. Calculating joint queue-length distributions in product-form queuing networks. *J. ACM 36*, 1 (Jan.), 194–207.

Gordon, J. J. 1990. The evaluation of normalizing constants in closed queueing networks. *Op. Res. 38*, 863–869.

Harrison, P. G. 1985. On normalizing constants in queueing networks. *Op. Res. 33*, 464–468.

Kaufman, J. S. 1981. Blocking in a shared resource environment. *IEEE Trans. Commun. COM*-29, 1474–1481.

Kelly, F. P. 1979. *Reversibility and Stochastic Networks.* Wiley, New York.

Kelly, F. P. 1986. Blocking probabilities in large circuit-switched networks. *Adv. Appl. Prob. 18*, 473–505.

Knessl, C., and Tier, C. 1992. Asymptotic expansions for large closed queueing networks with multiple job classes. *IEEE Trans. Comput. 41*, 480–488.

Koenigsberg, E. 1958. Cyclic queues. *Op. Res. Quart. 9*, 22–35.

Koenigsberg, E. 1986. Comments on "On normalization constants in queueing networks," by P. G. Harrison. *Op. Res. 34*, 330.

Kogan, Y. 1992. Another approach to asymptotic expansions for large closed queueing networks. *Op. Res. Lett. 11*, 317–321.

Kogan, Y., and Birman, A. 1992. Asymptotic analysis of closed queueing networks with bottlenecks. In *IFIP Transaction C-5, Performance of Distributed Systems and Integrated Communication Networks*, T. Hasegawa, H. Takagi, and Y. Takahashi, eds. North-Holland, Amsterdam, The Netherlands, pp. 265–280.

Lam, S. S. 1982. Dynamic scaling and growth behavior of queuing network normalization constants. *J. ACM 29*, 2 (Apr.), 492–513.

Lam, S. S., and Lien, Y. L. 1983. A tree convolution algorithm for the solution of queueing networks. *Commun. ACM 26*, 3 (Mar.), 203–215.

Lavenberg, S. S., Ed. 1983. *Computer Performance Modeling Handbook.* Academic Press, Orlando, Fla.

McKenna, J. 1989. A generalization of Little's law to moments of queue lengths and waiting times in closed product-form queuing networks. *J. Appl. Prob. 26*, 121–133.

McKenna, J., and Mitra, D. 1982. Integral representations and asymptotic expansions for closed Markovian queueing networks: normal usage. *Bell Syst. Tech. J. 61*, 661–683.

McKenna, J., and Mitra, D. 1984. Asymptotic expansions and integral representations of moments of queue lengths in closed Markovian networks. *J. ACM 31*, 2 (Apr.), 346–360.

McKenna, J., Mitra, D., and Ramakrishnan, K. G. 1981. A class of closed Markovian queuing networks: Integral representations, asymptotic expansions, generalizations. *Bell Sys. Tech. J. 60*, 599–641.

Ramakrishnan, K. G., and Mitra, D. 1982. An overview of PANACEA, a software package for analyzing Markovian queueing networks. *Bell Sys. Tech. J. 61*, 2849–2872.

Reif, F. 1965. *Fundamentals of Statistical and Thermal Physics.* McGraw-Hill, New York.

Reiser M., and Kobayashi, H. 1975. Queueing networks with multiple closed chains: theory and computational algorithms. *IBM J. Res. Dev. 19*, 283–294.

REISER, M., AND LAVENBERG, S. S. 1980. Mean value analysis of closed multichain queuing networks. *J. ACM 27*, 2 (Apr.), 313–322.

WHITTE, P. 1986. *Systems in Stochastic Equilibrium*, Wiley, New York.

WILLIAMS, A. C., AND BHANDIWAD, R. A. 1976. A generating function approach to queueing network analysis of multiprogrammed computers. *Networks 6*, 1–22.

WIMP, J. 1981. *Sequence Transformations and Their Applications*. Academic Press, Orlando, Fla.

RECEIVED NOVEMBER 1993; REVISED APRIL 1995; ACCEPTED FEBRUARY 1995