# An Efficient Algorithm for the Exact Analysis of Multiclass Queueing Networks with Large Population Sizes

Giuliano Casale
Neptuny R&D
via Durando 10-G, I-20158 Milan, Italy
and Politecnico di Milano - DEI
Via Ponzio 34/5, I-20133 Milan,Italy
giuliano.casale@polimi.it

## ABSTRACT

We introduce an efficient algorithm for the exact analysis of closed multiclass product-form queueing network models with large population sizes. We adopt a novel approach, based on linear systems of equations, which significantly reduces the cost of computing normalizing constants. With the proposed algorithm, the analysis of a model with $N$ circulating jobs of multiple classes requires essentially the solution of $N$ linear systems with order independent of population sizes.

A distinguishing feature of our approach is that we can immediately apply theorems, solution techniques, and decompositions for linear systems to queueing network analysis. Following this idea, we propose a block triangular form of the linear system that further reduces the requirements, in terms of both time and storage, of an exact analysis. An example illustrates the efficiency of the resulting algorithm in presence of large populations.

## Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques

## General Terms

Algorithms, Performance, Theory

## Keywords

Product-form queueing networks, computational algorithms, exact analysis, normalizing constant, multiclass models

## 1. INTRODUCTION

Analytical performance modeling of computer and communication systems is often carried out using queueing network models. For historical reasons, closed product-form networks [2] have been the focus of the field for over twenty years [28]. Using these models, classic performance problems as the capacity planning of computer systems can be easily tackled, typically with much less computational effort than simulation (see, e.g., [17] for an overview of applications). Moreover, computational algorithms for models with product-form solution have largely inspired further research. For instance, queueing networks with extended features that are successfully applied in several areas including software performance evaluation, parallel system analysis, and modeling of networks with priorities or blocking (e.g., [1, 4, 14, 35, 37]), can be solved by means of modified versions of the algorithms proposed for product-form networks.

The distinguishing feature of product-form models is that simple closed-form expressions are known for the equilibrium distribution of network state probabilities. The challenge is to efficiently compute the *normalizing constant* that assures that state probabilities sum to one. Excellent developments in the field [3, 5, 9, 10, 12, 13, 16, 23, 24, 33, 34] have shown that exact solution methods are computationally feasible in several cases of interest. However, the computational requirements usually become prohibitive when a large multiclass population is present in the network. This has led to the well-known conclusion that typically multiclass queueing networks are intractable for large population sizes. This is a severe difficulty of the theory that has been addressed for practical purposes only by approximation techniques as local iterative methods (e.g., [8,15,36]), bounds and asymptotic expansions (e.g., [32, 39]), or with the truncation of Euler summations when numerically inverting the generating function of the normalizing constant [10].

The results presented in this paper indicate that efficient exact algorithms for multiclass models with large populations are possible. In the following sections, we introduce a novel approach for computing normalizing constants that is efficient even in this difficult case. The algorithm is based on the solution of systems of linear equations involving normalizing constants, and follows from a simple consideration. Several recurrence equations for normalizing constants have been proposed in previous works [5,9,13,23,32,33]. Clearly, each of them gives an alternative description of the structure of the normalizing constant. We propose to use the simultaneous information of a set of recurrence equations to reduce the cost of computing normalizing constants.

In particular, the proposed algorithm has the following structure. We consider a set of networks derived from the model to be solved, and defined in such a way that all normalizing constants can be related by a system of linear equations. Then, we increase in a linear fashion, and simultaneously for all networks, the total population from 1 to $N$,


*SIGMetrics/Performance'06,* June 26–30, 2006, Saint Malo, France.

being $N$ the sum of class populations in the original model. After each increase, we compute normalizing constants using the linear system. Finally, we determine performance measures of interest for the original model. In this way, an exact analysis requires essentially the solution of $N$ linear systems. Moreover, despite the composition of the set of networks, and hence linear system structure, may change while increasing populations, linear system order is always independent of population sizes. As a consequence, we show that the computational cost is mostly determined by the number of queues and classes, rather than by the total population, which is usually the largest of the three parameters.

There are benefits and drawbacks connected to the introduction of linear systems of equations into queueing network analysis. Besides the significant reduction of computational complexity, that is anyway a fundamental motivation, there are other significant advantages. For instance, the new approach promotes the application of linear algebra techniques to the analysis of product-form networks. We show an application of this idea by defining a block triangular form of the linear system coefficient matrix that significantly reduces time and storage requirements. Moreover, the diffusion of specialized software for linear algebra, like computer algebra systems (e.g., [38]), may reduce implementation efforts.

We found instead two drawbacks regarding uniqueness and numerical accuracy of solutions achievable with standard (inexact) linear system solvers. Concerning the latter, we believe that the new algorithm, besides the related theoretical results, is also of practical interest due to its efficiency. Thus, granting result correctness is a fundamental objective of the present analysis. A simple and definitive solution to numerical accuracy problems follows by adopting an *exact* linear system solver in implementations (e.g., [6]). This has a limited impact on the asymptotic complexity of the algorithm that remains, also in the worst case, the exact technique of choice for multiclass models with large population sizes. Further, an exact solution let us avoid floating-point range exceptions that frequently arise in normalizing constant algorithms. Concerning uniqueness, we discuss some cases where the linear system does not have a unique solution due to a singular coefficient matrix. We propose strategies to address the problem.

The paper is organized as follows. In Section 2 we give preliminary definitions. Related work is critically analyzed in Section 3, where we also introduce the solution approach based on linear systems. Section 4 defines the algorithm that is further examined in Section 5 using linear algebra techniques. Computational requirements are discussed in Section 6. A numerical example is presented in Section 7. Finally, Section 8 gives conclusions and outlines future work. Theorem proofs are reported in the final appendix.

# 2. PRELIMINARIES AND NOTATION

## 2.1 Multiclass Closed Queueing Networks

Let us consider a closed product-form model composed by $M$ load-independent queues, $R$ job classes and an arbitrary number of delays. The population size for class $s$ is $N_s$, and the population vector is $\mathbf{N} \equiv (N_1, \ldots, N_s, \ldots, N_R)$. We denote by $N = \sum_{s=1}^{R} N_s$ the total population. The loading $\rho_{ks}$ (also called in the literature service demand) is the product between the mean service time per visit and the mean number of visits of class $s$ at queue $k$. We denote by $\rho_{0s}$ the sum of delays for class $s$.

Exact algorithms considered in the following sections perform recursions on the number of jobs and queues in the model. Without loss of generality, we indicate the population currently processed by the recursion with the $R$-dimensional vector $\mathbf{n} \equiv (n_1, \ldots, n_s, \ldots, n_r, 0, \ldots, 0)$, where $n_s > 0$, $1 \leq s \leq r$. Hence $r$, $r \leq R$, is the number of non-empty classes of the current population. We denote by $n = \sum_{s=1}^{r} n_s$ the total population of $\mathbf{n}$. Let $\mathbf{0}$ be the zero vector, and let $\mathbf{e}_s$ be a vector of all zeros except for the $s$-th element that is one. In several algorithms, e.g., Convolution [5,33], LBANC [9], and MVA [34], $\mathbf{n}$ spans the population set

$$\mathbf{prod}(\mathbf{N}) = \{\mathbf{n} \mid \mathbf{0} \leq \mathbf{n} \leq \mathbf{N}\}, \tag{1}$$

with cardinality $card(\mathbf{prod}(\mathbf{N})) = \prod_{s=1}^{R}(N_s + 1)$. As we show later in the paper, a class recursion algorithm as RECAL [13] may be instead reformulated as a recursion on

$$\mathbf{lin}(\mathbf{N}) = \{\mathbf{0}, \mathbf{e}_1, 2\mathbf{e}_1, \ldots, N_1\mathbf{e}_1, N_1\mathbf{e}_1 + \mathbf{e}_2, N_1\mathbf{e}_1 + 2\mathbf{e}_2, \ldots, N_1\mathbf{e}_1 + N_2\mathbf{e}_2, \ldots, \mathbf{N}\}, \tag{2}$$

where $card(\mathbf{lin}(\mathbf{N})) = N + 1$. A graphical comparison of the two sets is given in Figure 1. Note that for large multiclass populations $card(\mathbf{lin}(\mathbf{N})) << card(\mathbf{prod}(\mathbf{N}))$.

While recurring, several *intermediate models* are solved for each population $\mathbf{n}$. We denote by $\mathbf{j} \equiv \mathbf{j}(\mathbf{n})$ the population of an intermediate model solved when the current population is $\mathbf{n}$. The related network structure is specified by a *multiplicity vector* $\mathbf{m} \equiv (m_1, \ldots, m_k, \ldots, m_q)$ such that, for each queue type $k$, $1 \leq k \leq q$, the intermediate model contains $m_k \geq 1$ queues of that type [3,10,23,24]. Here $q$, $q \leq M$, is the number of queue types for the current population. The total number of queues is $m = \sum_{k=1}^{q} m_k$. All $m_k$ queues of type $k$ have identical loadings for the $r$ non-empty classes. With a slight abuse of notation, we denote by $\rho_{ks}$ the loading of class $s$ jobs at queues of type $k$. If not otherwise stated, we assume that for two queue types $k$ and $k'$ such that $k \neq k'$ there exists at least one class $s$, $1 \leq s \leq r$, such that $\rho_{ks} \neq \rho_{k's}$. Hence, each queue type defines a balanced subnetwork of maximum size. Note that by this definition $q \equiv q(r)$, and $q$ is non-decreasing with $r$.

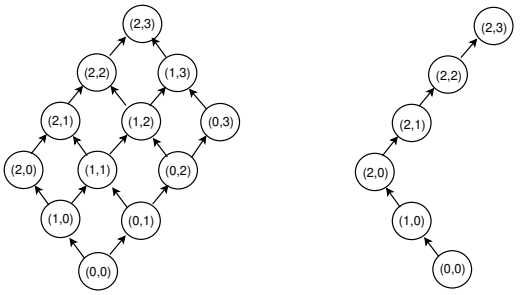
We call *input network* a network that has the same $M$ queues of the model that we want to solve recursively, but with population not necessarily equal to $\mathbf{N}$. The structure of the input network for the current population $\mathbf{n}$ is denoted by $\mathbf{M} \equiv (M_1, \ldots, M_k, \ldots, M_q)$, where $\sum_{k=1}^{q} M_k = M$. Very often, intermediate model structure will be specified using $\mathbf{M}$. For instance, an intermediate model with multiplicity vector $\mathbf{m} = \mathbf{M} + \mathbf{e}_q$ can be obtained by adding to the input network with population $\mathbf{n}$ a queue of type $q$.

For the rest of the paper, if not otherwise stated, $k$, $s$, $c$, $1 \leq k \leq q$, $1 \leq s \leq r$, $1 \leq c \leq r-1$, will index queue types and non-empty classes.

### *Summarizing Example*

As a summarizing example, let us assume that we want to solve recursively a model with $M = 3$, $R = 2$, $\mathbf{N} = (8, 13)$, delays $\rho_{01} = 3$, $\rho_{02} = 7$, and loadings $\rho_{11} = 10$, $\rho_{12} = 5$, $\rho_{21} = 10$, $\rho_{22} = 9$, $\rho_{31} = 10$, $\rho_{32} = 9$. For illustration purposes, we recursively process, for each $\mathbf{n} \in \mathbf{lin}(\mathbf{N})$, a set of $2(q+1)$ intermediate models $(\mathbf{m}, \mathbf{j})$ with multiplicity

$$\mathbf{m} \in \{\mathbf{M}, \mathbf{M} + \mathbf{e}_1, \ldots, \mathbf{M} + \mathbf{e}_q\}, \tag{3}$$




a) **prod**(**N**) b) **lin**(**N**)

**Figure 1: Population sets for $\mathbf{N} = (2, 3)$**

and population

$$\mathbf{j} \in \{\mathbf{n}, \mathbf{n} - \mathbf{e}_r\}. \tag{4}$$

By the definitions, we have the following cases:

- if $1 \leq n_1 \leq 8$ and $n_2 = 0$ then $r = 1$. In this case all queues have identical loadings, thus $q = 1$, $\rho_{01} = 3$, $\rho_{11} = 10$, $\mathbf{M} = 3\mathbf{e}_1$, $(\mathbf{m}, \mathbf{j}) \in \{(3\mathbf{e}_1, n_1\mathbf{e}_1), (3\mathbf{e}_1, n_1\mathbf{e}_1 - \mathbf{e}_1), (4\mathbf{e}_1, n_1\mathbf{e}_1), (4\mathbf{e}_1, n_1\mathbf{e}_1 - \mathbf{e}_1)\}$.
- If $1 \leq n_1 \leq 8$ and $1 \leq n_2 \leq 13$ then $r = 2$, $q = 2$, $\rho_{01} = 3$, $\rho_{02} = 7$, $\rho_{11} = 10$, $\rho_{12} = 5$, $\rho_{21} = 10$, $\rho_{22} = 9$, $\mathbf{M} = \mathbf{e}_1 + 2\mathbf{e}_2$, $(\mathbf{m}, \mathbf{j}) \in \{(\mathbf{e}_1 + 2\mathbf{e}_2, n_1\mathbf{e}_1 + n_2\mathbf{e}_2), (\mathbf{e}_1 + 2\mathbf{e}_2, n_1\mathbf{e}_1 + n_2\mathbf{e}_2 - \mathbf{e}_2), (2\mathbf{e}_1 + 2\mathbf{e}_2, n_1\mathbf{e}_1 + n_2\mathbf{e}_2), (2\mathbf{e}_1 + 2\mathbf{e}_2, n_1\mathbf{e}_1 + n_2\mathbf{e}_2 - \mathbf{e}_2), (\mathbf{e}_1 + 3\mathbf{e}_2, n_1\mathbf{e}_1 + n_2\mathbf{e}_2), (\mathbf{e}_1 + 3\mathbf{e}_2, n_1\mathbf{e}_1 + n_2\mathbf{e}_2 - \mathbf{e}_2)\}$.

Models for $\mathbf{n} = \mathbf{0}$ are solved using termination conditions specified later. Thus, we do not define a specific notation.

## 2.2 Normalizing Constant Equations

We denote by $G = G(\mathbf{m}, \mathbf{j})$ the normalizing constant of a model with multiplicity $\mathbf{m}$ and $\mathbf{j} = (j_1, \ldots, j_s, \ldots, j_r, 0, \ldots, 0)$ jobs. From the BCMP theorem [2], it follows that

$$G = \sum \prod_{i=0}^{m} f_i(\mathbf{j}_i), \tag{5}$$

where the summation is taken on the state-space

$$\left\{ (\mathbf{j}_0, \ldots, \mathbf{j}_i, \ldots, \mathbf{j}_m) \mid \sum_{i=0}^{m} j_{is} = j_s, j_{is} \geq 0, 1 \leq s \leq r \right\},$$

where $\mathbf{j}_i = (j_{i1}, \ldots, j_{is}, \ldots, j_{ir})$, is the population at station $i$, and the product-form factors $f_i$ for queues and delays (the latter is associated to the index $i = 0$) are defined as

$$f_i(\mathbf{j}_i) = \begin{cases} \prod_s \rho_{0s}^{j_{0s}} / j_{0s}!, & \text{if } i = 0, \\ (\sum_s j_{is})! \prod_s \rho_{is}^{j_{is}} / j_{is}!, & \text{if } 1 \leq i \leq m. \end{cases} \tag{6}$$

Note that the number of operations required by (5) is of the order of $\prod_{s=1}^{r} \binom{j_s+m-1}{j_s}$ that is infeasible in most cases.

However, $G$ can be computed using recurrence equations involving intermediate models. For compactness, if $G \equiv G(\mathbf{m}, \mathbf{j})$, we denote by $G_s^{+k}$ the normalizing constant of a model with population $\mathbf{j} - \mathbf{e}_s$ and multiplicity $\mathbf{m} + \mathbf{e}_k$. Similarly, $G_s^{-k}$ refers to a model with $\mathbf{j} - \mathbf{e}_s$ and $\mathbf{m} - \mathbf{e}_k$, $G_s$ refers to $\mathbf{j} - \mathbf{e}_s$ and $\mathbf{m}$, $G^{+k}$ refers to $\mathbf{j}$ and $\mathbf{m} + \mathbf{e}_k$, and $G^{-k}$ refers to $\mathbf{j}$ and $\mathbf{m} - \mathbf{e}_k$. The removal of multiple queues or jobs is denoted, e.g., by $G_{s,c}^{-k,i}$ referring to $\mathbf{j} - \mathbf{e}_s - \mathbf{e}_c$ and $\mathbf{m} - \mathbf{e}_k - \mathbf{e}_i$.

Throughout the paper, we consider the following normalizing constant recurrence equations. The *convolution expressions* [5, 33] for $G$, i.e.,

$$G = G^{-k} + \sum_{s=1}^{r} \rho_{ks} G_s, \qquad 1 \leq k \leq q, \tag{7}$$

and for $G^{+k}$, i.e.,

$$G^{+k} = G + \sum_{s=1}^{r} \rho_{ks} G_s^{+k}, \qquad 1 \leq k \leq q. \tag{8}$$

Note that, without loss of generality, we consider only $q$ convolution expressions among the $m$ possible, referring to the removal of any of the identical queues of type $k$.

We also introduce the network *population constraints* for the models with $\mathbf{j} = \mathbf{n}$, i.e.,

$$n_s G = \rho_{0s} G_s + \sum_{k=1}^{q} m_k \rho_{ks} G_s^{+k}, \qquad 1 \leq s \leq r. \tag{9}$$

which are an application of Little's Law [30], and generalize to the case $m_k > 1$ a similar set of equations presented in [32]. A straightforward proof of (9) is given at the end of this section. Termination conditions for recursions based on (7)-(9) are $G = f_0(\mathbf{j})$, for normalizing constants of models with $\mathbf{m} = \mathbf{0}$, and $G = 1$, for normalizing constants of models with $\mathbf{j} = \mathbf{0}$. The conditions are consistent assuming $0^0 = 1$.

Performance measures can be computed using the following formulas [33] for the mean throughput of class $s$

$$X_s(\mathbf{m}, \mathbf{j}) = G_s / G, \tag{10}$$

and for the mean class $s$ queue length for queues of type $k$

$$Q_{ks}(\mathbf{m}, \mathbf{j}) = \rho_{ks} G_s^{+k} / G. \tag{11}$$

Related measures, as utilizations or response times, can be easily obtained from the above measures (e.g., [29]). Note that a straightforward proof of (9) follows by inserting (10) and (11) for $\mathbf{j} = \mathbf{n}$ into the population constraint for a closed model [25], given by $n_s = \rho_{0s} X_s(\mathbf{m}, \mathbf{n}) + \sum_k m_k Q_{ks}(\mathbf{m}, \mathbf{n})$. This motivates the choice of the name for (9).

# 3. ANALYSIS OF RELATED WORK

This section introduces the new solution approach based on linear systems of normalizing constant recurrence equations. In order to prepare background for the new algorithm, we reformulate popular computational algorithms for closed networks as linear systems of normalizing constant recurrence equations.

## 3.1 Recursive Solution Using Linear Systems

Most exact algorithms for product-form networks compute normalizing constants using the linear recurrence equations (7)-(9). In general, a solution can be obtained by different techniques. We propose a novel approach where, for each population $\mathbf{n}$ considered by the recursion, a vector $\mathbf{g} \equiv \mathbf{g}(\mathbf{n})$ of normalizing constants of intermediate models is computed using a linear system of equations (7)-(9). The elements of $\mathbf{g}$ change with the considered algorithm, and will be described later. Due to the linearity of (7)-(9), the technique takes the form of a sequence of linear systems

$$A\mathbf{g} = \sum_{s=1}^{r} B_s \mathbf{g}_s, \tag{12}$$

where the matrix $A \equiv A(\mathbf{n})$ describes the linear dependencies between the variables in $\mathbf{g}$, the $\mathbf{g}_s \equiv \mathbf{g}(\mathbf{n} - \mathbf{e}_s)$ vectors are computed recursively, and the $B_s \equiv B_s(\mathbf{n})$ matrices determine the linear system known terms from the $\mathbf{g}_s$ vectors. In general, if $\mathbf{n} \in \mathbf{lin}(\mathbf{N})$, then $B_s$ is the zero matrix for $1 \leq s \leq r-1$.

Therefore, from (12) it follows that the recursive solution of queueing network models using a sequence of linear systems requires to select:

1. the set of processed populations $\mathbf{n}$, typically either $\mathbf{lin}(\mathbf{N})$ or $\mathbf{prod}(\mathbf{N})$,
2. the elements of the $\mathbf{g}$ vector for all processed $\mathbf{n}$,
3. the mix of equations (7)-(9) such that $A$ is an invertible matrix and hence (12) has a unique solution.

In the next sections, we show how to reformulate popular computational algorithms in the form (12).

## 3.2 Analysis of the Convolution Algorithm

We begin with an analysis of the Convolution algorithm. In the convolution approach, (7) is considered for an arbitrary queue $k$, and recursively applied to the constants $G^{-k}$ and $G_s$ in the right hand side until the network contains queues or jobs. Hence, a linear system reformulation may consider, for all $\mathbf{n} \in \mathbf{prod}(\mathbf{N})$, a vector $\mathbf{g}$ composed by the normalizing constants of intermediate models $(\mathbf{m}, \mathbf{j})$ with population $\mathbf{j} = \mathbf{n}$ and multiplicity

$$\mathbf{m} \in \{\mathbf{M}, \mathbf{M} - \mathbf{e}_q, \mathbf{M} - 2\mathbf{e}_q, \ldots, \mathbf{M} - M_q\mathbf{e}_q, \\ \mathbf{M} - M_q\mathbf{e}_q - \mathbf{e}_{q-1}, \mathbf{M} - M_q\mathbf{e}_q - 2\mathbf{e}_{q-1}, \ldots, \mathbf{e}_1, \mathbf{0}\}.$$

The number of intermediate models considered for each $\mathbf{n}$ is thus $card(\mathbf{g}) = M + 1$. For each $\mathbf{n} \in \mathbf{prod}(\mathbf{N})$, the mix of equations for (12) is

$$\begin{cases} G - G^{-k} = \sum_{s=1}^{r} \rho_{ks} G_s, \end{cases} \tag{13}$$

that is the set of all possible equations (7) such that

1. the unknown normalizing constants in the left hand side are all elements of the $\mathbf{g}$ vector,
2. the normalizing constants in the right hand side are known terms belonging to the $\mathbf{g}_s$ vectors, which are recursively computed.

Thus, looking at (12), the $A$ matrix is defined by the coefficients of the normalizing constants in the left hand sides, while the $B_s$ matrices depend on the right hand sides only. This notation will be used throughout the rest of the paper.

As an example, a model with $M = 3$ queues, $q = 3$ queue types, and no delays can be solved by means of the following linear system

$$\begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} G \\ G^{-3} \\ G^{-2,3} \\ G^{-1,2,3} \end{bmatrix} = \sum_{s=1}^{r} \begin{bmatrix} \rho_{3s} G_s \\ \rho_{2s} G_s^{-3} \\ \rho_{1s} G_s^{-2,3} \\ 0 \end{bmatrix},$$

where we used the termination condition for $\mathbf{m} = \mathbf{0}$ in the last equation. It is clear from the example that the Convolution algorithm has a coefficient matrix independent of both population and loadings, and that is always invertible.

### 3.2.1 Application Example

We now show a first application of linear systems of normalizing constant recurrence equations. We modify the Convolution algorithm by introducing new equations. We consequently modify the linear system and the recursive structure of the algorithm to account for the new unknowns.

In general, it is quite difficult to integrate, at least in an efficient manner, (8) and (9) within (13), since they introduce several new $G^{+k}$ unknowns. Instead, it may be helpful to include additional convolution expressions (7) for different choices of $k$.

We consider a system of equations (7) for all $1 \leq k \leq q$. In this case, the normalizing constants in $\mathbf{g}$ are of intermediate models $(\mathbf{m}, \mathbf{j})$ with population $\mathbf{j} = \mathbf{n}$ and multiplicity

$$\{\mathbf{m} \mid \mathbf{0} \leq \mathbf{m} \leq \mathbf{M}\}. \tag{14}$$

Thus $card(\mathbf{g}) = \prod_{k=1}^{q}(m_k + 1)$. Considering the same example seen before, and omitting $G^{-1,2,3} = 0$, we have now

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} G \\ G^{-1} \\ G^{-2} \\ G^{-3} \\ G^{-1,2} \\ G^{-1,3} \\ G^{-2,3} \end{bmatrix} = \sum_{s=1}^{r} \begin{bmatrix} \rho_{1s} G_s \\ \rho_{2s} G_s \\ \rho_{3s} G_s \\ \rho_{2s} G_s^{-1} \\ \rho_{3s} G_s^{-1} \\ \rho_{1s} G_s^{-2} \\ \rho_{3s} G_s^{-2} \\ \rho_{1s} G_s^{-3} \\ \rho_{2s} G_s^{-3} \\ \rho_{3s} G_s^{-1,2} \\ \rho_{2s} G_s^{-1,3} \\ \rho_{1s} G_s^{-2,3} \end{bmatrix},$$

which is an overdetermined linear system with 12 equations and just 7 unknowns. Therefore, using the additional equations, and provided that the resulting matrix is invertible, we may avoid to recursively evaluate some terms of the right hand side by moving them into the vector of unknowns.

For instance, let us assume $r = 2$ and observe that the constants $G^{-1,2}$, $G^{-1,3}$ and $G^{-2,3}$ are the normalizing constants of networks with a single queue and no delays, thus equal to the product-form factors $f_3$, $f_2$ and $f_1$, respectively. Considering as new unknowns the normalizing constants $G_1$ and $G_1^{-k}$, $1 \leq k \leq 2$, and reducing the matrix to square by dropping unnecessary variables and equations, we get

$$\begin{bmatrix} 1 & -\rho_{11} & -1 & 0 & 0 & 0 \\ 1 & -\rho_{21} & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & -\rho_{21} & 0 & 0 \\ 0 & 0 & 1 & -\rho_{31} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -\rho_{11} \\ 0 & 0 & 0 & 0 & 1 & -\rho_{31} \end{bmatrix} \begin{bmatrix} G \\ G_1 \\ G^{-1} \\ G_1^{-1} \\ G^{-2} \\ G_1^{-2} \end{bmatrix} = \begin{bmatrix} \rho_{12} G_2 \\ \rho_{22} G_2 \\ f_3 + \rho_{22} G_2^{-1} \\ f_2 + \rho_{32} G_2^{-1} \\ f_3 + \rho_{12} G_2^{-2} \\ f_1 + \rho_{32} G_2^{-2} \end{bmatrix},$$

which has a unique solution if all loadings of class 1 are distinct. Thus, with the inclusion of additional convolution expressions we found a way to avoid the recursion on class 1 when $r = 2$, so to perform a recursion on $\mathbf{lin}(\mathbf{N})$ instead than on $\mathbf{prod}(\mathbf{N})$. In this way, the number of recursively evaluated populations drops from the $(N_1 + 1)(N_2 + 1)$ of the original algorithm to the just $N_1 + N_2 + 1$ of the new formulation. This clearly leads to significant computational savings for models with large population sizes. Nevertheless, new overheads are introduced for linear system solution (e.g., the cost of LU factorization [20]). However, the main problem is that, even if the result generalizes easily to models with a larger number of queues, for $r > 2$ the increased number of unknowns typically makes impossible to process a $\mathbf{lin}(\mathbf{N})$ population set, and this is intrinsically related to the use of (7) in the linear system. Since the scope of the present paper is to develop a general $\mathbf{lin}(\mathbf{N})$ recursion, we shall not consider the proposed variant of the Convolution algorithm in the following sections, as well as the inclusion of (7) in linear systems.

### 3.3 Analysis of the RECAL Algorithm

In this section, we discuss the relations between RECAL [13] and a $\mathbf{lin}(\mathbf{N})$ recursion based on (9) for $s = r$. We begin by first reviewing RECAL. In this algorithm, each queue is univocally associated to a special class, called self-looping class, composed by jobs looping indefinitely through the station. Queues have to be changed into load-dependent stations in order to serve self-looping jobs. The advantage is that the normalizing constant can then be computed using the following efficient class recursion. Let

$$D_k^n = \{(d_1, \ldots, d_n) \mid \sum_t d_t = k,\ d_t \geq 0,\ 1 \leq t \leq n\} \quad (15)$$

be the set of combinations with repetition of $k$ elements among $n$, which has cardinality $card(D_k^n) = \binom{n+k-1}{k}$. Then, the normalizing constant $h(\mathbf{v}, \mathbf{j})$ of a network with $M$ queues, a delay server, population vector $\mathbf{v}$ for the $M$ self-looping classes, and current population $\mathbf{j} = \mathbf{n}$ for the remaining $r$ classes, satisfies the following recurrence equation

$$h(\mathbf{v}, \mathbf{n}) = \sum_{\mathbf{d} \in D_{n_r}^{M+1}} c(\mathbf{v}, \mathbf{d}) h(\mathbf{v} + \mathbf{d}, \mathbf{n} - n_r \mathbf{e}_r), \quad (16)$$

where $\mathbf{d} \equiv (d_0, d_1, \ldots, d_k, \ldots, d_M)$,

$$c(\mathbf{v}, \mathbf{d}) = \frac{\rho_{0r}^{d_0}}{d_0!} \prod_{k=1}^{M} \binom{v_k + d_k}{v_k} \rho_{kr}^{d_k}, \quad (17)$$

and with termination condition $h(\mathbf{v}, \mathbf{0}) = 1$ for all $\mathbf{v}$. Note that the normalizing constant of the input network with population $\mathbf{N}$ and no self-looping classes is immediately obtained by computing $h(\mathbf{0}, \mathbf{N})$ with (16). Furthermore, if $\rho_{0r} = 0$, then the summation is on $(d_1, \ldots, d_M) \in D_{n_r}^M$.

We remark that we did not mention the $\mathbf{m}$ vector since RECAL assumes that the number of queues in a model is always equal to $M$, and that intermediate models differ only for the number of self-looping jobs and non-empty classes. Hence, RECAL has no queue recursions. However, we show that a different interpretation of (16) is possible, as proved by the following theorem.

THEOREM 1. *Let $q \equiv M$ for all populations, and let each type $k$, $1 \leq k \leq M$, be composed by $m_k$ identical queues with loadings equal to those of queue $k$, $1 \leq k \leq M$, of the input network. Then, the recursive insertion $n_r - 1$ times of (9) for class $r$ into all its right hand side normalizing constants gives (16) by setting $v_k = m_k - 1$, $1 \leq k \leq M$.*

Informally, Theorem 1 states that (16) is equivalent to the class-$r$ population constraint, provided that the $M$ queues in the input network are never aggregated into the same queue type. Furthermore, the theorem has the important consequence of showing that, from the relation $v_k = m_k - 1$, the addition of self-looping jobs at queue $k$ is equivalent to the insertion of new queues into the balanced subnetwork $k$. Therefore, two very different physical interpretations of RECAL may be considered.

Moreover, Theorem 1 shows that the definition of queue types implicitly assumed in RECAL differs from that adopted in Section 2.1. But since our queue type definition, which aggregates maximal balanced subnetworks into a single type, is indeed the most efficient for models with a large number of identical queues, we conclude that RECAL can be largely improved in this case, as recently shown by [24]. A possible approach to address the problem consists in recursively solving (9) for class $r$ using the queue type definition that aggregates maximal balanced subnetworks. Note that the artifice of considering single-job classes in order to reduce computational costs may also be applied to (9).

From the above observations, we have the following linear system reformulation of RECAL

$$\left\{ n_r G = \rho_{0r} G_r + \sum_{k=1}^{q} m_k \rho_{kr} G_r^{+k}, \right. \quad (18)$$

for a set of populations $\mathbf{n} \in \mathbf{lin}(\mathbf{N})$, and where the left hand side unknowns refer to intermediate models $(\mathbf{m}, \mathbf{j})$ having population $\mathbf{j} = \mathbf{n}$ and multiplicity

$$\mathbf{m} \in \left\{ \mathbf{M} + (d_1, \ldots, d_q) \,|\, (d_0, d_1, \ldots, d_q) \in D_{N-n}^{q+1} \right\},$$

with $card(\mathbf{g}) = \binom{q+N-n}{N-n}$. Thus, the cardinality of $\mathbf{g}$ increases while recurring on populations. The extension of (18) by including (8) into the linear system leads us to consider the LBANC algorithm.

### 3.4 Analysis of the LBANC Algorithm

The LBANC algorithm [9] has been shown in [27] to be the unnormalized version of the MVA algorithm [34]. The computational requirements of the two methods are very similar, and the main difference is that LBANC uses normalizing constants, while MVA works directly with mean performance indices.

Concerning linear systems of normalizing constant recurrence equations, LBANC illustrates a simultaneous use of (8) and (9). The population recursion is $\mathbf{n} \in \mathbf{prod}(\mathbf{N})$, and the formulation of LBANC discussed in [27] can be reformulated as the following linear system

$$\begin{cases} G^{+k} - G = \sum_s \rho_{ks} G_s^{+k}, & 1 \leq k \leq q, \\ n_r G = \rho_{0r} G_r + \sum_k m_k \rho_{kr} G_r^{+k}, \end{cases} \quad (19)$$

where the left hand side unknowns have population $\mathbf{j} = \mathbf{n}$ and multiplicity

$$\mathbf{m} \in \{\mathbf{M}, \mathbf{M} + \mathbf{e}_1, \ldots, \mathbf{M} + \mathbf{e}_q\}. \quad (20)$$

Therefore, LBANC computes $card(\mathbf{g}) = q + 1$ constants at each population, and it is substantially more efficient than convolution when $q << M$, an advantage that has not been previously pointed out in the literature. Another advantage of LBANC is the direct availability of the terms $G_s^{+k}$ for (11), which are instead computed in the Convolution algorithm with additional recursions. Nevertheless, the main source of complexity in both LBANC and Convolution remains the $\mathbf{prod}(\mathbf{N})$ population recursion. The inclusion of new equations into the linear system (19) in order to reduce computational costs is addressed in the next section.

## 4. ALGORITHM DESCRIPTION

### 4.1 Preliminaries

We initially assume that $\rho_{0s} = 0$, $1 \leq s \leq r-1$. The definition of the new algorithm starts by observing that *the inclusion of the $r-1$ population constraints for the non-empty classes $s \neq r$ into (19) does not introduce new unknowns and may increase the number of independent equations in the linear system.* Thus, these additional equations have the potential to significantly reduce the cost of computing normalizing constants. In particular, as we prove later, if a certain set of equations (8)-(9) is considered, and provided that the resulting linear system is non-singular, then we can solve models using an efficient $\mathbf{lin}(\mathbf{N})$ recursion with very different computational tradeoffs with respect to existing algorithms.

We can show that the above observations hold true also for the general case where we drop the $\rho_{0s} = 0$ conditions. Despite now the additional population constraints introduce new unknowns into the linear system, i.e., the $G_c$ constants for $1 \leq c \leq r-1$, we can always exploit the fact that these constants refer to models with $j_r = n_r$ jobs of class $r$. This allows us to include into the system also the class $r$ population constraints for the models with $\mathbf{j} = \mathbf{n} - \mathbf{e}_c$, i.e.,

$$n_r G_c = \rho_{0r} G_{c,r} + \sum_{k=1}^{q} m_k \rho_{kr} G_{c,r}^{+k}, \qquad 1 \leq c \leq r-1, \quad (21)$$

which allow to compute the new unknowns directly from $\mathbf{g}_r$. Hence, the inclusion of the new $G_c$ unknowns is compensated by the availability of additional class $r$ population constraints.

Finally, we point out that in general the considered linear systems may not have necessarily full rank, but for the sake of simplicity in presenting, we initially take this assumption. Singularity conditions will be discussed in Section 5.2.2, as well as strategies to address the problem.

#### 4.1.1 Preliminary Analysis

We now illustrate some difficulties arising when including (8) and (9) into a linear system with the aim of defining a $\mathbf{lin}(\mathbf{N})$ recursion. We consider a linear system relating normalizing constants of models with multiplicity $\mathbf{m} \in \{\mathbf{M}, \mathbf{M}+\mathbf{e}_1, \mathbf{M}+\mathbf{e}_2\}$ and population $\mathbf{j} \in \{\mathbf{n}, \mathbf{n}-\mathbf{e}_1\}$. Assume that the current population $\mathbf{n}$ is such that $r = 2$ and $q = 2$. We have that the linear system defined by (8), $1 \leq k \leq q$, and (9), $1 \leq s \leq r$, for a $\mathbf{lin}(\mathbf{N})$ recursion is

$$A\mathbf{g} = B_2 \mathbf{g}_2, \quad (22)$$

where

$$A\mathbf{g} \equiv \begin{bmatrix} 1 & -\rho_{11} & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & -\rho_{21} & -1 & 0 \\ 0 & -M_1\rho_{11} & 0 & -M_2\rho_{21} & n_1 & -\rho_{01} \\ 0 & 0 & 0 & 0 & n_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & n_2 \end{bmatrix} \begin{bmatrix} G^{+1} \\ G_1^{+1} \\ G^{+2} \\ G_1^{+2} \\ G \\ G_1 \end{bmatrix},$$

and

$$B_2\mathbf{g}_2 \equiv \begin{bmatrix} \rho_{12} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \rho_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ M_1\rho_{12} & 0 & M_2\rho_{22} & 0 & \rho_{02} & 0 \\ 0 & M_1\rho_{12} & 0 & M_2\rho_{22} & 0 & \rho_{02} \end{bmatrix} \begin{bmatrix} G_2^{+1} \\ G_{1,2}^{+1} \\ G_2^{+2} \\ G_{1,2}^{+2} \\ G_2 \\ G_{1,2} \end{bmatrix},$$

Note that (22) is underdetermined, having more unknowns than equations. This indicates that the considered system of equations (8)-(9) may be insufficient for a population recursion in $\mathbf{lin}(\mathbf{N})$. Thus, a different approach is required to take full advantage of the additional population constraints.

### 4.2 The Linear System

We propose to solve closed multiclass product-form networks using a population recursion in $\mathbf{lin}(\mathbf{N})$. For each recursively processed $\mathbf{n} \in \mathbf{lin}(\mathbf{N})$, we compute the following vector of unknowns

$$\mathbf{g} = \begin{bmatrix} \mathbf{G}^+ \\ \mathbf{G} \end{bmatrix}, \quad (23)$$

where $\mathbf{G}$ is a vector of normalizing constants of intermediate models $(\mathbf{m}, \mathbf{j})$ having population $\mathbf{j} \in \{\mathbf{n}, \mathbf{n}-\mathbf{e}_1, \ldots, \mathbf{n}-\mathbf{e}_{r-1}\}$ and multiplicity

$$\mathbf{m} \in \{\mathbf{M} + (d_1, \ldots, d_q) \,|\, (d_1, \ldots, d_q) \in D_{r-1}^q\}. \quad (24)$$

Instead, $\mathbf{G}^+$ includes all normalizing constants that can be obtained by adding a queue $k$, $1 \leq k \leq q$, to the normalizing constants in $\mathbf{G}$, i.e.:

$$\mathbf{G}^+ \equiv \{G^{+k}, G_1^{+k}, \ldots, G_{r-1}^{+k} \mid 1 \leq k \leq q, \{G, G_1, \ldots, G_{r-1}\} \subseteq \mathbf{G}\}. \quad (25)$$

Thus, $\mathbf{G}^+$ includes constants of intermediate models $(\mathbf{m}^+, \mathbf{j})$ with population $\mathbf{j} \in \{\mathbf{n}, \mathbf{n}-\mathbf{e}_1, \ldots, \mathbf{n}-\mathbf{e}_{r-1}\}$ and multiplicity

$$\mathbf{m}^+ \in \{\mathbf{M} + (d_1, \ldots, d_q) \,|\, (d_1, \ldots, d_q) \in D_r^q\}. \quad (26)$$

We denote by $\mathbf{G}_r^+$ and $\mathbf{G}_r$ the components of $\mathbf{g}_r$. Note that by the definitions, $\mathbf{g}$ has cardinality

$$\begin{aligned} card(\mathbf{g}) &= card(\mathbf{G}^+) + card(\mathbf{G}) \\ &= \binom{q+r-1}{r} r + \binom{q+r-2}{r-1} r, \end{aligned} \quad (27)$$

which is *independent* of population sizes. Thus, for fixed $r$ and $q$, $\mathbf{g}$ and $\mathbf{g}_r$ have identical cardinality, and the linear system order does not increase with $n$.

For the current population $\mathbf{n}$, $\mathbf{g}$ is computed from $\mathbf{g}_r$ by the following linear system:

$$\begin{cases} G^{+k} - G - \sum_{c=1}^{r-1} \rho_{kc} G_c^{+k} = \rho_{kr} G_r^{+k}, & 1 \leq k \leq q \\ n_c G - \rho_{0c} G_c - \sum_{k=1}^{q} m_k \rho_{kc} G_c^{+k} = 0, & 1 \leq c \leq r-1 \\ n_r G = \rho_{0r} G_r + \sum_{k=1}^{q} m_k \rho_{kr} G_r^{+k}, & \\ n_r G_c = \rho_{0r} G_{c,r} + \sum_{k=1}^{q} m_k \rho_{kr} G_{c,r}^{+k}, & 1 \leq c \leq r-1 \end{cases} \quad (28)$$

where $\{G^{+k}, G_c^{+k}\} \subseteq \mathbf{G}^+$, $\{G, G_c\} \subseteq \mathbf{G}$, $\{G_r^{+k}, G_{c,r}^{+k}\} \subseteq \mathbf{G}_r^+$, and $\{G_r, G_{c,r}\} \subseteq \mathbf{G}_r$. We denote the four groups of equations, from the top, by (28a), (28b), (28c) and (28d). We motivate the linear system by the following two observations:

- for each $\mathbf{n}$, we can directly compute all normalizing constants in $\mathbf{G}$ from $\mathbf{g}_r$ by (28c)-(28d).
- Once that $\mathbf{G}$ has been computed, we can determine $\mathbf{G}^+$ by solving the subsystem (28a)-(28b).

Thus, the solution of (28) is always decomposable into two simpler problems, as we discuss in Section 5. Furthermore, we justify (24)-(26) by the following theorem.

THEOREM 2. *The linear system (28) has always a square matrix of coefficients.*

As a consequence of this result, if the coefficient matrix $A$ is non-singular, the system is never underdetermined and (28) has a unique solution. It is important to point out that (24)-(26) define the smallest cardinality vectors for which the above theorem holds, but other definitions of $\mathbf{G}$ and $\mathbf{G}^{+k}$ may also be considered. Numerical conditioning, singularity, and block triangular form of the coefficient matrix of (28) are discussed in Section 5.

#### 4.2.1 Illustrative Example

We now show a small example of (28). We consider the same model of Section 4.1.1. We have $q = 2$ and $r = 2$, and so $\mathbf{g}$ depends on $D^q_{r-1} = D^2_1 = \{(1,0),(0,1)\}$ and $D^q_r = D^2_2 = \{(2,0),(1,1),(0,2)\}$. In particular, we have $\mathbf{g} = \left[\mathbf{G}^+ \ \mathbf{G}\right]^T$ with

$$\mathbf{G}^+ = \left[G^{+1,1} \ G_1^{+1,1} \ G^{+1,2} \ G_1^{+1,2} \ G^{+2,2} \ G_1^{+2,2}\right]^T, \quad (29)$$

$$\mathbf{G} = \left[G^{+1} \ G_1^{+1} \ G^{+2} \ G_1^{+2}\right]^T, \quad (30)$$

and similarly $\mathbf{g}_r = \mathbf{g}_2 = \left[\mathbf{G}_2^+ \ \mathbf{G}_2\right]^T$ where

$$\mathbf{G}_2^+ = \left[G_2^{+1,1} \ G_{1,2}^{+1,1} \ G_2^{+1,2} \ G_{1,2}^{+1,2} \ G_2^{+2,2} \ G_{1,2}^{+2,2}\right]^T, \quad (31)$$

$$\mathbf{G}_2 = \left[G_2^{+1} \ G_{1,2}^{+1} \ G_2^{+2} \ G_{1,2}^{+2}\right]^T. \quad (32)$$

Let

$$t_{zks} \equiv (M_k + z)\rho_{ks},$$

then (28) is

$$A\mathbf{g} = B_2\mathbf{g}_2, \quad (33)$$

where

$$A\mathbf{g} \equiv \begin{bmatrix} 1 & -\rho_{11} & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -\rho_{11} & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & -\rho_{21} & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -\rho_{21} & 0 & 0 & -1 & 0 \\ 0 & -t_{111} & 0 & -t_{021} & 0 & 0 & n_1 & -\rho_{01} & 0 & 0 \\ 0 & 0 & 0 & -t_{011} & 0 & -t_{121} & 0 & 0 & n_1 & -\rho_{01} \\ 0 & 0 & 0 & 0 & 0 & 0 & n_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_2 \end{bmatrix} \begin{bmatrix} G^{+1,1} \\ G_1^{+1,1} \\ G^{+1,2} \\ G_1^{+1,2} \\ G^{+2,2} \\ G_1^{+2,2} \\ G^{+1} \\ G_1^{+1} \\ G^{+2} \\ G_1^{+2} \end{bmatrix},$$

and

$$B_2\mathbf{g}_2 \equiv \begin{bmatrix} \rho_{12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \rho_{12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \rho_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \rho_{22} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ t_{112} & 0 & t_{022} & 0 & 0 & 0 & \rho_{02} & 0 & 0 & 0 \\ 0 & t_{112} & 0 & t_{022} & 0 & 0 & 0 & \rho_{02} & 0 & 0 \\ 0 & 0 & t_{012} & 0 & t_{122} & 0 & 0 & 0 & \rho_{02} & 0 \\ 0 & 0 & 0 & t_{012} & 0 & t_{122} & 0 & 0 & 0 & \rho_{02} \end{bmatrix} \begin{bmatrix} G_2^{+1,1} \\ G_{1,2}^{+1,1} \\ G_2^{+1,2} \\ G_{1,2}^{+1,2} \\ G_2^{+2,2} \\ G_{1,2}^{+2,2} \\ G_2^{+1} \\ G_{1,2}^{+1} \\ G_2^{+2} \\ G_{1,2}^{+2} \end{bmatrix}.$$

Compared to the linear system of Section 4.1.1, the coefficient matrix $A$ is now square, and provided that it is non-singular, we have a unique solution to the linear system. Hence, a $\mathbf{lin}(\mathbf{N})$ recursion is now possible.

### 4.3 The Computational Algorithm

The new algorithm has the following structure:

```
1: Compute g(N_1, 0, ..., 0)
2: for r = 2 to R do
3:     Initialize g(N_1, ..., N_{r-1}, 0, ..., 0)
4:     for n_r = 1 to N_r do
5:         Setup and solve (28) for n = (N_1, ..., N_{r-1}, n_r, 0, ..., 0)
6:     end for
7: end for
8: Compute performance measures
```

**Computation of $\mathbf{g}(N_1, 0, \ldots, 0)$.** The recursion on the first class can be performed using single-class algorithms proposed in previous works (e.g., [5, 9]). Furthermore, under certain assumptions, single class normalizing constants can be computed using closed-form formulas (see, e.g., [3, 22]), but exact arithmetic may be required to avoid numerical instabilities.

**Initialization of $\mathbf{g}(N_1, \ldots, N_{r-1}, 0, \ldots, 0)$.** At each change of the value of $r$, also (23)-(28) change. In particular, the order of the coefficient matrix grows with $r$, and this has the following implications:

1. the solution of the linear system becomes more expensive as $r$ increases. Hence, it is better to process the classes with the largest populations first, since they require the largest number of iterations.
2. At the first population processed after a $r$ change, the vector $\mathbf{G}^+(N_1, \ldots, N_{r-1}, 0, \ldots, 0)$ is not available. For instance, when $r = 3$ the first linear system solved requires $\mathbf{G}^+(N_1, N_2, 0, \ldots, 0)$ with multiplicity depending on the set $D^q_3$, but during recursion for $r = 2$ only intermediate models defined on $D^q_2$ and $D^q_1$ were solved.

The second issue can be addressed as follows. Let $r$ be the new class to be processed, and assume $n_r = 1$. Note that the constants in $\mathbf{G}(\mathbf{n} - \mathbf{e}_r) = \mathbf{G}(N_1, \ldots, N_{r-1}, 0, \ldots, 0)$ are known from the recursion on class $r - 1$. We compute the required $\mathbf{G}^+(\mathbf{n} - \mathbf{e}_r) = \mathbf{G}^+(N_1, \ldots, N_{r-1}, 0, \ldots, 0)$ from $\mathbf{G}(\mathbf{n} - \mathbf{e}_r)$ using the following linear system:

$$\begin{cases} G^{+k} - \sum_{c=1}^{r-1} \rho_{kc} G_c^{+k} = G, & 1 \le k \le q, \\ -\sum_{k=1}^{q} m_k \rho_{kc} G_c^{+k} = -n_c G + \rho_{0c} G_c, & 1 \le c \le r-1 \end{cases} \quad (34)$$

where $\{G, G_c\} \in \mathbf{G}(\mathbf{n} - \mathbf{e}_r)$, $\{G^{+k}, G_c^{+k}\} \in \mathbf{G}^+(\mathbf{n} - \mathbf{e}_r)$. We point out that the matrix of coefficient of (34) is the coefficient matrix $C$ for class $r$ that we will introduce in the next section.

**Performance measures**. In order to compute performance measures by (10) and (11), we need to compute $G$, $G_s$, $1 \le s \le R$, and $G_s^{+k}$, $1 \le k \le q$, $1 \le s \le R$, for the input network with population $\mathbf{N}$. Note that the intermediate models solved by the linear system for $\mathbf{n} = \mathbf{N}$ have multiplicity $\mathbf{m} \ge \mathbf{M}$, thus computing performance measures requires to remove the queues added by (24)-(26). This task has to be performed in different ways depending on network structure. We have the following two cases: all delays $\rho_{0s}$, $1 \le s \le R$, are strictly positive; or there exist one or more classes with zero delay. In the first case, we can recursively remove the queues added by (24)-(26) from the linear system solution for $\mathbf{n} = \mathbf{N}$ using

$$\begin{cases} G = G^{+k} - \sum_{s=1}^{R} \rho_{ks} G_s^{+k}, & 1 \le k \le q, \\ G_s - \dfrac{N_s}{\rho_{0s}} G = -\sum_{k=1}^{q} m_k \dfrac{\rho_{ks}}{\rho_{0s}} G_s^{+k}, & 1 \le s \le R. \end{cases} \quad (35)$$

Instead, in the second case we permute class indices so to process last all classes with $\rho_{0s} = 0$. Then if $N_R \ge R$, the required normalizing constants can be computed from $\mathbf{g}(N_1, \ldots, N_{R-1}, N_R - R)$ by recursive application of (9) for class $R$. This also let us skip the solution of the last $R$ linear systems (28). Otherwise, the recursive solution of (9) comes to a point where the problem is recursively reduced to removing queues in the normalizing constants computed for class $R - 1$. Thus, a recursive procedure can be used to compute performance measures in this case.

# 5. LINEAR ALGEBRA RESULTS

## 5.1 Block Triangular Form

We now apply linear algebra techniques to simplify (28). We begin by expressing the linear system in the form (12)

$$A(n_r)\mathbf{g} = B_r\mathbf{g}_r, \tag{36}$$

where $A(n_r)$ indicates that only the $n_r$ coefficients change in $A$ while recurring on class $r$ populations. The main limitation of this dependence is that we cannot reuse the same LU factorization for linear systems depending on the same $r$. Nevertheless, we show that (23) implicitly defines $A(n_r)$ into an upper block triangular form that suggests a straightforward solution to the problem.

Note that in (28) the population $n_r$ multiplies all and only the constants in $\mathbf{G}$. Therefore, (36) can be expressed using only population-independent matrices as

$$\begin{bmatrix} C & A_{12} \\ 0 & n_r I \end{bmatrix} \begin{bmatrix} \mathbf{G}^+ \\ \mathbf{G} \end{bmatrix} = \begin{bmatrix} B_{1r} \\ B_{2r} \end{bmatrix} \mathbf{g}_r, \tag{37}$$

where $I$ is the identity matrix of order $\binom{q+r-2}{r-1}r$, and $C \equiv C(q,r)$ is a square diagonal block of order $\binom{q+r-1}{r}r$. Using block Gaussian elimination we get

$$\mathbf{G} = n_r^{-1} B_{2r}\mathbf{g}_r, \tag{38}$$

and we only need to solve the linear system

$$C\mathbf{G}^+ = B_{1r}\mathbf{g}_r - A_{12}\mathbf{G}. \tag{39}$$

The last two formulas prove that we can recursively solve (28) using matrices that in no way depend on $n_r$. In particular, only $C$ requires a LU factorization, which can anyway be reused for all class $r$ populations.

Furthermore, by inserting (38) into (39) we get

$$C\mathbf{G}^+ = \left(B_{1r} - n_r^{-1} A_{12} B_{2r}\right) \mathbf{g}_r, \tag{40}$$

and defining

$$F_{1r} \equiv \begin{bmatrix} C^{-1}B_{1r} \\ 0 \end{bmatrix}, \quad F_{2r} \equiv \begin{bmatrix} -C^{-1}A_{12}B_{2r} \\ B_{2r} \end{bmatrix},$$

where the zero matrix in $F_{1r}$ has size $card(\mathbf{G})$ x $card(\mathbf{g}_r)$, we may compactly reformulate (28) as

$$\mathbf{g} = \left(F_{1r} + n_r^{-1} F_{2r}\right) \mathbf{g}_r, \qquad n_r = 1, \ldots, N_r. \tag{41}$$

which is a set of recurrence equations in the $n_r$ variable. Despite that a block Gaussian elimination is typically more efficient than a recursive solution of (41), because $F_{1r}$ and $F_{2r}$ may not preserve the sparsity of $C$, this compact formulation may be of interest for theoretical reasons due to its simplicity.

We also point out that an advantage deriving from the above results is that, if loadings and delays are all strictly positive, then the number and the placement of the non-zero coefficients in linear system matrices are fixed for all models with same $q$ and $r$. Thus, it is possible to optimize the related data structures and algorithms by precomputation. For instance, the position of the non-zeros in $C$ may be easily precomputed. Then, the matrix can be instantiated in an array of minimum size that can be accessed efficiently using hashing. We also remark that linear system solvers perform the same operations and memory accesses for fixed $q$ and $r$, and this may also be used to optimize implementations.

### 5.1.1 Fine-Grain Decomposition

Direct solution methods for linear systems, e.g., Gaussian elimination, require a computational effort that is typically cubic with the order of the coefficient matrix [20]. Starting from this fact, we propose a decomposition of the coefficient matrix $C$ into block triangular form that allows significant computational savings by reducing the order of linear systems. Despite this is not required to grant the correctness of the results, it is useful to limit growth of time and storage requirements.

The decomposition depends on a partitioning of the set $D_r^q$, which immediately implies by (24) a partitioning of $\mathbf{G}^+$ and thus a decomposition of $C$. We partition $D_r^q$ so that combinations $(d_1, \ldots, d_q)$ with same position of non-zero elements are included into the same partition. We denote by $P_{ht}$ the $t$-th partition composed by combinations with $h$ non-zeros in the same position. For example, $D_3^3$ can be partitioned into the sets $P_{11} = \{(3,0,0)\}$, $P_{12} = \{(0,3,0)\}$, $P_{13} = \{(0,0,3)\}$, $P_{21} = \{(1,2,0),(2,1,0)\}$, $P_{22} = \{(1,0,2),(2,0,1)\}$, $P_{23} = \{(0,2,1),(0,1,2)\}$, $P_{31} = \{(1,1,1)\}$. So $\mathbf{G}^+$ becomes

$$\mathbf{G}^+ = \mathbf{G}^+(P_{11}) \cup \mathbf{G}^+(P_{12}) \cup \ldots \cup \mathbf{G}^+(P_{31}). \tag{42}$$

where each subset $\mathbf{G}^+(P_{ht})$ contains normalizing constants of intermediate models $(\mathbf{m}_{ht}, \mathbf{j})$ with population $\mathbf{j} \in \{\mathbf{n}, \mathbf{n} - \mathbf{e}_1, \ldots, \mathbf{n} - \mathbf{e}_{r-1}\}$ and multiplicity

$$\mathbf{m}_{ht} \in \{\mathbf{M} + (d_1, \ldots, d_q) \,|\, (d_1, \ldots, d_q) \in P_{ht}\}. \tag{43}$$

Note that, by definition of $D_r^q$, the number of non-zeros elements $h$ must range in $1 \le h \le H$ where

$$H \equiv \min\{q, r\}. \tag{44}$$

Further, note that the number of possible choices for the positions of the $h$ non-zero elements is $\binom{q}{h}$. The remaining $r - h$ elements are free to be assigned to any of the $h$ positions, and this can be modeled by a set of combination with repetitions $D_{r-h}^h$. From this considerations, we have immediately

$$card\,(P_{ht}) = \binom{h + (r-h) - 1}{r-h} = \binom{r-1}{r-h}, \tag{45}$$

which gives

$$card\left(\mathbf{G}^+\right) = \sum_{h=1}^{H} \binom{q}{h} \binom{r-1}{r-h} r. \tag{46}$$

and the last expression implies

$$\binom{q+r-1}{r} r = \sum_{h=1}^{H} \binom{q}{h} \binom{r-1}{r-h} r, \tag{47}$$

which is an application of Vandermonde's convolution [11], and summarizes the effects of the partitioning.

We also remark that in (28) two normalizing constants in $\mathbf{G}^+$ appear in the same equation only if they refer to models differing by at most one queue assigned to different types. This means that non-zero coefficients in $C$ can relate either normalizing constants of the same $\mathbf{G}^+(P_{ht})$, or constants depending on $P_{ht}$ and $P_{h't}$ with $|h - h'| = 1$. Thus, permuting the columns of $C$ to reflect the partitioning of $\mathbf{G}^+$, and in particular sorting columns according to the number

of non-zeros, we get the block triangular form

$$C \equiv \begin{bmatrix} C_1 & C_{12} & 0 & \dots & 0 & 0 \\ 0 & C_2 & C_{23} & \dots & 0 & 0 \\ 0 & 0 & C_3 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & C_{H-1} & C_{H-1,H} \\ 0 & 0 & 0 & \dots & 0 & C_H \end{bmatrix}, \quad (48)$$

where

$$C_h \equiv diag(c_{h1}, c_{h2}, \dots, c_{ht}, \dots), \quad 1 \le h \le H, \quad (49)$$

is a diagonal matrix of $\binom{q}{h}$ blocks $c_{ht}$ that contains the coefficients for the normalizing constants in $\mathbf{G}^+(P_{ht})$. We now prove the following theorem.

THEOREM 3. *The $\binom{q}{h}$ diagonal blocks $c_{ht} \subseteq C_h$ are square matrices of order $\binom{r-1}{r-h}r$, where $\binom{r-1}{r-h}h$ rows are convolution expressions (28a), and $\binom{r-1}{r-h}(r-h)$ rows are population constraints (28b).*

Therefore, this form greatly simplifies the solution of (39), since LU factorization is now required only for the diagonal blocks that have order $\binom{r-1}{r-h}r << \binom{q+r-1}{r}r$.

## 5.2 Numerical Properties

### 5.2.1 Error Propagation

Preliminary implementations of the proposed algorithm suggest that roundoff errors tend to accumulate while iterating on populations [19]. Even on models where $C$ is numerically well conditioned (i.e., has a low condition number [20]), the computation typically fails if the network contains more than few tens of jobs.

In order to address the problem in a simple and effective way, we propose to *exactly* solve (e.g., [6]) all linear systems (28). This means that exact arithmetic has to be adopted throughout all computations. If this is implemented using rational multiprecision arithmetic libraries (e.g., [18, 21]) all digits of operands are kept, and no roundoff is performed. Despite the additional digits introduce some overheads, exact rational arithmetic does not suffer any kind of numerical instability. So we also avoid underflow or overflow problems that typically affect normalizing constant evaluations. Moreover, nowadays multiprecision libraries (e.g., GMP [21]) offer very good performance for operands with thousands of digits, as we expect in normalizing constants of large models. Other techniques for exact linear algebra may also be adopted, e.g. modular arithmetic [31] that also provides a natural way for parallelizing linear system solution. We point to Section 6 for a discussion on the cost of adopting exact multiprecision arithmetic.

### 5.2.2 Singular Linear Systems

We now consider conditions that produce a singular $C$ matrix, and discuss possible solutions. Noting that the loadings for class $R$ are never included in $C$, we identify the following critical cases:

1. there exist one or more queue types $k$ with $\rho_{kc} = 0$ for some $c \neq R$. This condition yields columns of all zeros in $C$. Thus, some unknowns of $\mathbf{g}$ are not included into any equation.

2. There exist linear relations between loadings that let some $C_h$ blocks become singular.

The first case can be easily addressed by removing the unknowns in $\mathbf{G}^+$ associated with the zero columns. The resulting system is overdetermined and still admits a unique solution.

The second case is more difficult, because dependencies between loadings may result in a singular $C$ matrix. For instance, there exist singular diagonal blocks if two queue types $k_1$ and $k_2$ have $\rho_{k_1 c} = \rho_{k_2 c}$ for some $1 \le c \le r-1$. Singularity conditions can be enumerated by a symbolic computation of the determinants of the $C_h$ blocks. For instance, the $C_3$ diagonal block for $q = 3$, $r = 3$ is

$$C_3 = \begin{bmatrix} 1 & -\rho_{11} & -\rho_{12} \\ 1 & -\rho_{21} & -\rho_{22} \\ 1 & -\rho_{31} & -\rho_{32} \end{bmatrix}, \quad (50)$$

which has determinant

$$\det(C_3) = \rho_{21}\rho_{32} + \rho_{12}\rho_{31} + \rho_{22}\rho_{11} - \rho_{22}\rho_{31} - \rho_{11}\rho_{32} - \rho_{12}\rho_{21}.$$

Considering $\rho_{11}$, we see that for $\rho_{22} \neq \rho_{32}$ the value

$$\rho_{11} = \frac{\rho_{22}\rho_{31} - \rho_{21}\rho_{32} + \rho_{12}\rho_{21} - \rho_{12}\rho_{31}}{\rho_{22} - \rho_{32}} \quad (51)$$

gives a zero determinant. Note that the complexity of singularity conditions grows quickly with block order (e.g., the determinant of the $C_4$ block for $q = 4$ and $r = 4$ includes over 60 product terms). So, a numerical diagnosis of singularity is recommended in implementations.

We propose to address singularity by moving back into the vector of known terms the unknowns responsible for the singularity of (28). This means that an hybrid population recursion has to be performed, where some normalizing constants are computed from linear systems, while the remaining ones are recursively evaluated. For instance, if the normalizing constants $G_c^{+k}$ and $G_c^{+k}$ of the last $t$ classes are moved back to the right hand side of (28), then we can solve the model for all populations in $\mathbf{prod}(N_{R-t+1}, \dots, N_R)$ using $\mathbf{lin}(N_1, \dots, N_{R-t})$ recursions. In this way, the new normalizing constants in the right hand side are directly available from previous recursions. Finally, the recursion for the last population $(N_{R-t+1}, \dots, N_R)$ solves the input network with population $\mathbf{N}$. The computational cost in presence of a singular coefficient matrix is thus an intermediate case between that for non-singular systems and that of existing $\mathbf{prod}(\mathbf{N})$ recursions.

# 6. COMPUTATIONAL REQUIREMENTS

Finally, we discuss the computational requirements of the proposed algorithm. In particular, we consider the cost of solving the sequence of systems (39). This task is typically several orders of magnitude slower than all other operations, e.g., (38) and the computation of performance measures. Throughout this section, we assume to use LU factorization, forward and back substitution [20] to solve (39).

Since the additional complexity due to numerical stabilization is not a theoretical limit to the performance of the proposed algorithm, but rather characterizes the best currently available stable implementation, we first quantify the cost of solving the sequence of linear system regardless of accuracy of solutions. Then, we discuss the asymptotic cost of adopting an exact linear system solver in implementations.

### 6.1 Solving the Sequence of Linear Systems

Let $LU_t(\theta) \cong (2/3)\theta^3$ and $LU_s(\theta) \cong \theta^2$ be respectively the number of operations and the storage required by a LU factorization of a full-rank square matrix of order $\theta$. Similarly, let $BS_t(\theta) \cong 2\theta^2$ be the total number of operations of forward and back substitution. Assume that $C$ is in the form (48), and denote by $\theta_{rh} \equiv \binom{r-1}{r-h} r$ the order of the diagonal blocks $c_{ht}$ when there are $r$ non-empty classes. Then, from the consideration in Section 5.1.1, the time requirement is of the order of

$$\begin{aligned}
&\sum_{r=1}^{R}\sum_{h=1}^{H}\binom{q}{h}\left[LU_t(\theta_{rh}) + \sum_{n_r=0}^{N_r} BS_t(\theta_{rh})\right] \\
&\cong 2\sum_{r=1}^{R}\sum_{h=1}^{H}\binom{q}{h}\left[\binom{r-1}{r-h}r\right]^2\left[\frac{1}{3}\binom{r-1}{r-h}r + N_r + 1\right] \\
&< 2\sum_{r=1}^{R}\sum_{h=1}^{H}\frac{q^h}{h!}\left[\frac{r^h}{(h-1)!}\right]^2\left[\frac{r^h}{(h-1)!} + N_r + 1\right], \quad (52)
\end{aligned}$$

where the case $n_r = 0$ accounts for the initialization of $\mathbf{g}$ after a $r$ change, and $H \equiv \min\{q, r\}$. Note that the formula is typically an upper bound, because it does not account for the computational savings due to the sparsity of $C$. Moreover, assuming to keep in memory only the LU factorizations for the current value of $r$, storage requirement is maximum for $r = R$, and it is of the order of

$$\begin{aligned}
\sum_{h=1}^{H}\binom{q}{h}LS_s(\theta_{Rh}) &\cong \sum_{h=1}^{H}\binom{q}{h}\left[\binom{R-1}{R-h}R\right]^2 \\
&< \sum_{h=1}^{H}\frac{q^h}{h!}\left[\frac{R^h}{(h-1)!}\right]^2, \quad (53)
\end{aligned}$$

which is independent of population sizes.

### 6.2 Asymptotic Cost of Stabilization

If we adopt in implementations an exact solver based on rational multiprecision arithmetic, then we have to account also for the increasing costs of arithmetic operations. Thus, we have in general that the time and storage requirements have to be specified as $LU_t(\theta) \equiv LU_t(\theta, \mathbf{n})$, $LU_s(\theta) \equiv LU_s(\theta, \mathbf{n})$, and $BS_t \equiv BS_t(\theta, \mathbf{n})$. The exact number of operations required by multiprecision arithmetic strongly depends with implementations, thus we limit to study asymptotic behaviors. Because the number of digits of the coefficients in $C$ is constant, operations required to solve (39) have computational requirements that grow with the number of digits of the unknowns in $\mathbf{g}$ and $\mathbf{g}_r$. Let us assume all loadings to be scaled to integers [26], and let us assume to keep with small overheads a common denominator between the variables, e.g., $n_1! \cdots n_r!$ for the normalizing constants in $\mathbf{g}$, and $n_1! \cdots n_{r-1}!(n_r - 1)!$ for those in $\mathbf{g_r}$. Then, the time cost of rational arithmetic operations grows linearly with the number of digits of the numerator, while storage depends by both the size of numerators and of the common denominator. Taking the logarithm of (5) in the worst-case of a model with balanced $m'$ queues, integer loadings all equal to $\overline{\rho} \equiv \max_{k,s}\{\rho_{ks}\}$, $0 \le k \le q$, and population $\mathbf{n}$, we get the following expression

$$\log G = \log \frac{n!\binom{m'+n-1}{n}\overline{\rho}^n}{n_1! \cdots n_r!} < \log \frac{(m'+n-1)^n\overline{\rho}^n}{n_1! \cdots n_r!}, \quad (54)$$

where both numerators and denominators of the logarithm argument are integers. Thus, we see that the growth of the number of digits of the numerator of $G$ is $O(n \log(m' + n))$. Thus, for the constants in $\mathbf{g}$, which have $m' \le M + r + 1$ accounting also the product-form factor of the delay, the time and storage overhead of exact arithmetic grows at most as $O(n \log(M + r + n))$, which may be regarded as a scale factor for the computational cost in the case without stabilization.

### 6.3 Models with Large Population Sizes

Time and storage complexity formulas prove that the presented algorithm is very efficient for large population sizes, even accounting for stabilization. In particular, assuming constant $q$, $M$ and $R$, and considering $N_s = \kappa$ for all classes, we have that the time requirements grow for $\kappa \to +\infty$ as $O(\kappa)$, or $O(\kappa^2 \log \kappa)$ accounting also for numerical stabilization, while under the same conditions the Convolution, LBANC, and the MVA algorithms are approximately $O(\kappa^R)$. Similarly, RECAL and other efficient class recursions (e.g, [16, 23, 24]) require for large population sizes approximately $O(\kappa^M)$ or $O(\kappa^q)$ time. Similarly, accounting for stabilization, the storage requirement of the algorithm, grows as $O(\kappa \log \kappa)$, that is typically several orders of magnitude smaller than existing algorithms. Therefore, the proposed approach may provide substantial computational savings for models with large populations.

## 7. NUMERICAL EXAMPLE

Finally, we compare the efficiency of the proposed algorithm with respect to existing techniques by a numerical example. We solve a model with $N = 6000$ jobs, $M = 25$ queues, $q = 3$ queue types for all populations, and $R = 6$ classes. The loadings $\rho_{ks}$ for the considered model are given in Table 1. In the example, the multiplicities $M_k$ never change during recursions. Since complexity formulas for RECAL are available only for the case with no delays [13], for the purpose of comparison we set $\rho_{0s} = 0$ for all classes.

We used a preliminary implementation of the algorithm written in C, and based on the exact rational arithmetic routines of the GMP library [21] version 4.1.4. The algorithm was run on a AMD Athlon64 3000+ processor using the 32-bit mode. The solution of the linear system in block triangular form was performed by reusing the same LU factorization for all populations of class $r$. The computed normalizing constant was $G(\mathbf{N}) = 1.21173 \times 10^{11328}$. Since we used exact arithmetic, all digits of $G$ were available at the end of the computation. We also determined throughputs and queue-lengths for all classes and queues. For instance, the throughput of class 2 is $X_2(\mathbf{N}) = 1.45435 \times 10^{-2}$ jobs per second, while the average class 1 queue-length at queues of type 1 is $Q_{11}(\mathbf{N}) = 2.45426$ jobs.

Accounting multiprecision overheads and all operations not modeled by (52), the solution was obtained in $173s$. On the considered architecture this is approximately $10^{11}$ operations. During execution, the amount of allocated memory was always less than 20 MBytes, also thanks to the use of specialized data structures for sparse matrices. For the same model, the theoretical number of operations of LBANC is $\approxeq 10^{19}$, $\approxeq 10^{20}$ for Convolution, and $\approxeq 10^{71}$ for RECAL. Therefore, compared to the other methods, the proposed algorithm is the only computationally feasible on the considered example.

## 8. CONCLUSIONS

We presented a new efficient exact algorithm for computing normalizing constants of closed multiclass product-form

**Table 1: Example multiclass model with $N = 6000$ jobs belonging to $R = 6$ classes.**

| Queue | | Class of requests | | | | | |
|---|---|---|---|---|---|---|---|
| Type $k$ | $M_k$ | $\rho_{k1}$ | $\rho_{k2}$ | $\rho_{k3}$ | $\rho_{k4}$ | $\rho_{k5}$ | $\rho_{k6}$ |
| *1* | 12 | 17 | 4 | 15 | 6 | 20 | 2 |
| *2* | 5 | 16 | 7 | 16 | 17 | 5 | 17 |
| *3* | 8 | 2 | 18 | 19 | 8 | 18 | 12 |
| Population | $N_s$ | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 |

networks. The method is based on the new powerful concept of linear systems of normalizing constant recurrence equations, which may indicate further research developments in the area.

An example presented in the paper shows that the new algorithm can be several orders of magnitude faster than existing techniques on models with large populations. A comparison on non-asymptotic populations, as well as on models with large number of queue types and classes, is left as future work.

It is currently not clear whether the results presented in the paper may be extended to the load-dependent case. Lack of results concerning the generalization of (8)-(9) to the load-dependent case makes it difficult to establish whether such extension is possible. In particular, the lack of a suitable generalization of (11) to the load-dependent case seems to be the main issue.

## 9. ACKNOWLEDGEMENTS

Additional material of interest for this work can be found in [7]. I wish to thank my advisor, prof. G. Serazzi, for his help and continuous support of my research. I thank J. Anselmi and the anonymous SIGMetrics reviewers for suggestions that substantially helped in increasing the quality of the paper. I am also in debt with prof. P. Cremonesi for useful discussions. I thank D. Ardagna, M. Roveri, C. Spelta and S. Zanero for constructive comments.

## 10. REFERENCES

[1] S. Balsamo, V. De Nitto Personé, and R. Onvural. Analysis of queueing networks with blocking, 2001.

[2] F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J.ACM*, 22(2):248–260, 1975.

[3] A. Bertozzi and J. McKenna. Multidimensional residues, generating functions, and their application to queueing networks. *SIAM Rev.*, 35(2):239–268, 1993.

[4] R.M. Bryant, A.E.Krzesinski, M.S.Lakshmi, and K.M. Chandy. The MVA priority approximation. *ACM Trans. Comp. Sys.*, 2(4):335–359, 1984.

[5] J.P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Comm. ACM*, 16(9):527–531, 1973.

[6] S. Cabay. Exact solution of linear equations. In *Proc. of the 2nd ACM SYMSAC Symposium*, pages 392–398. ACM Press, 1971.

[7] G. Casale. *The Throughput Analysis of Product-Form Queueing Networks*. PhD thesis, Politecnico di Milano, Milan, Italy, 2006.

[8] K.M. Chandy and D. Neuse. Linearizer: A heuristic algorithm for queuing network models of computing systems. *Comm. ACM*, 25(2):126–134, 1982.

[9] K.M. Chandy and C.H. Sauer. Computational algorithms for product-form queueing networks models of computing systems. *Comm. ACM*, 23(10):573–583, 1980.

[10] G.L. Choudhury, K.K. Leung, and W. Whitt. Calculating normalization constants of closed queuing networks by numerically inverting their generating functions. *J.ACM*, 42(5):935–970, 1995.

[11] D.J.A. Cohen. *Basic Techniques of Combinatorial Theory*. John Wiley and Sons, 1978.

[12] A.E. Conway, E. de Sousa e Silva, and S.S. Lavenberg. Mean value analysis by chain of product form queueing networks. *IEEE Trans. Comp.*, 38(3):432–442, 1989.

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

[14] P. Cremonesi and C. Gennaro. Integrated performance models for SPMD applications and MIMD architectures. *IEEE Trans. Par. Distr. Sys.*, 13(12):1320–1332, 2002.

[15] P. Cremonesi, P.J. Schweitzer, and G. Serazzi. A unifying framework for the approximate solution of closed multiclass queuing networks. *IEEE Trans. Comp.*, 51:1423–1434, 2002.

[16] E. de Sousa e Silva and S.S. Lavenberg. Calculating joint queue-length distributions in product-form queueing networks. *J.ACM*, 36(1):194–207, 1989.

[17] E. de Sousa e Silva and R.R. Muntz. Queueing networks: Solutions and applications. Technical Report CSD-890052, CS Dep. - UCLA, Sep 1989.

[18] J.G. Dumas et al. Linbox: A generic library for exact linear algebra. In *Proc. ICMS'02*, pages 40–50, World Scientific, Singapore, 2002.

[19] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comp. Surv.*, 23(1):5–48, 1991.

[20] G. H. Golub and C. F. Van Loan. *Matrix computations*. Johns Hopkins Univ. Press, 1996.

[21] T. Granlund. *GNU MP: The GNU Multiple Precision Arithmetic Library*, Aug 2000. `http://www.swox.se/`.

[22] P.G. Harrison. On normalizing constants in queueing networks. *Operations research*, 33:464–468, 1985.

[23] P.G. Harrison and S. Coury. On the asymptotic behaviour of closed multiclass queueing networks. *Perf. Eval.*, 47(2):131–138, 2002.

[24] P.G. Harrison and T.T.Lee. A new recursive algorithm for computing generating functions in closed queueing networks. In *Proc. of the 2004 IEEE Int'l MASCOTS Symposium*, pages 223–230. IEEE Press, 2004.

[25] J. Kriz. Throughput bounds for closed queueing networks. *Perf. Eval.*, 4(1):1–10, 1984.

[26] S. Lam. Dynamic scaling and growth behavior of queueing network normalization constants. *J.ACM*, 29(2):492–513, 1982.

[27] S. Lam. A simple derivation of the MVA and LBANC algorithms from the convolution algorithm. *IEEE Trans. Comp.*, 32:1062–1064, 1983.

[28] S.S. Lavenberg. A perspective on queueing models of computer performance. *Perf. Eval.*, 10(1):53–76, 1989.
[29] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik. *Quantitative System Performance.* Prentice-Hall, 1984.
[30] J. D. C. Little. A proof of the queueing formula $L = \lambda W$. *Operations Research*, pages 9:383–387, 1961.
[31] Michael T. McClellan. The exact solution of systems of linear equations with polynomial coefficients. *J.ACM*, 20(4):563–588, 1973.
[32] J. McKenna and D. Mitra. Asymptotic expansions and integral representations of moments of queue lengths in closed markovian networks. *J.ACM*, 31(2):346–360, Apr 1984.
[33] M. Reiser and H. Kobayashi. Queueing networks with multiple closed chains: Theory and computational algorithms. *IBM J. Res. Dev.*, 19(3):283–294, 1975.
[34] M. Reiser and S.S. Lavenberg. Mean-value analysis of closed multichain queueing networks. *J.ACM*, 27(2):312–322, 1980.
[35] J.A. Rolia and K.C. Sevcik. The method of layers. *IEEE Trans. Soft. Eng.*, 21(8):689–700, 1995.
[36] P.J. Schweitzer. Approximate analysis of multiclass closed networks of queues. In *Int'l Conf. on Stoch. Control and Optim., Amsterdam*, pages 25–29, 1979.
[37] M. Sereno. Mean value analysis of product form solution queueing networks with repetitive service blocking. *Perf. Eval.*, 36–37:19–33, 1999.
[38] The PARI Group, Bordeaux. *PARI/GP, version* `2.1.5`, 2004. `http://pari.math.u-bordeaux.fr/`.
[39] J. Zahorjan, K.C. Sevcik, D.L. Eager, and B. Galler. Balanced job bound analysis of queueing networks. *Comm. ACM*, 25(2):134–141, 1982.

# APPENDIX

## Proof of Theorem 1

We show that the recursive unfolding of (9) in the form

$$G(\mathbf{m},\mathbf{n}) = \frac{\rho_{0r}}{n_r} G(\mathbf{m},\mathbf{n}-\mathbf{e}_r) + \sum_{k=1}^{M} m_k \frac{\rho_{kr}}{n_r} G(\mathbf{m}+\mathbf{e}_k,\mathbf{n}-\mathbf{e}_r),$$

gives (16) by setting $v_k \equiv m_k - 1$. Telescoping on class $r$ populations we get

$$G(\mathbf{m},\mathbf{n}) = \sum_{\mathbf{d}\in D_{n_r}^{M+1}} \frac{a(\mathbf{d})}{n_r!} \rho_{0r}^{d_0} \prod_{k=1}^{M} m_k^{(d_k)} \rho_{kr}^{d_k} G(\mathbf{m}+\mathbf{d},\mathbf{n}-n_r\mathbf{e}_r),$$

where $G(\mathbf{m}+\mathbf{d},\mathbf{n}-n_r\mathbf{e}_r)$ refers to a model with class $r$ less, $\mathbf{d} \equiv (d_0, d_1, \ldots, d_M)$, the term $a(\mathbf{d}) = (d_0 + \ldots + d_M)!/(d_0! \cdots d_M!)$ is the number of recursions that terminate with $G(\mathbf{m}+\mathbf{d},\mathbf{n}-n_r\mathbf{e}_r)$ in the right hand side, and $m_k^{(d_k)} = m_k(m_k+1)\cdots(m_k-1+d_k)$ is the rising factorial. We get (16) noting that $(d_0 + \ldots + d_M)! = n_r!$, and setting $v_k = m_k - 1$ in $m_k^{(d_k)}$, i.e.,

$$\frac{a(\mathbf{d})}{n_r!} \prod_{k=1}^{M} m_k^{(d_k)} \rho_{kr}^{d_k} = \frac{1}{d_0!} \prod_{k=1}^{M} \frac{m_k^{(d_k)}}{d_k!} \rho_{kr}^{d_k}$$
$$= \frac{1}{d_0!} \prod_{k=1}^{M} \frac{(m_k-1+d_k)!}{(m_k-1)!d_k!} \rho_{kr}^{d_k} = \frac{1}{d_0!} \prod_{k=1}^{M} \binom{v_k+d_k}{v_k} \rho_{kr}^{d_k}.$$

## Proof of Theorem 2

Note that we can compute from $\mathbf{g}_r$ each constant in $\mathbf{G}$ using a class $r$ population constraint of (28). Therefore, we only need to prove that the number of equations (28a) and (28b) is equal to $card(\mathbf{G}^+) = \binom{q+r-1}{r} r$. Note that for each of the $\binom{q+r-2}{r-1}$ constants $G(\mathbf{m},\mathbf{n}) \in \mathbf{G}$ we have $r-1$ population constraints in (28), so the number of equations (28b) is $\binom{q+r-2}{r-1}(r-1)$. Further, note that we have $q$ possible ways of adding a new queue to $\mathbf{m}$, hence the number of equations (28a) is exactly $\binom{q+r-2}{r-1} q$. By the property $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, the total number of equations (28a) and (28b) is then $\binom{q+r-2}{r-1}(q+r-1) = \binom{q+r-1}{r} r = card(\mathbf{G}^+)$.

## Proof of Theorem 3

By definition, each block $c_{ht} \subseteq C_h$ has coefficients of constants in $\mathbf{G}^+(P_{ht})$. Thus, the number of columns of $c_{ht}$ is $\binom{r-1}{r-h} r$. Note that we can remove at most $h$ stations from $\mathbf{m} \in \{\mathbf{M} + (d_1,\ldots,d_q) \,|\, (d_1,\ldots,d_q) \in P_{ht}\}$ so that the resulting model is in (24). Hence, the number of convolution expressions is $card(P_{ht})h = \binom{r-1}{r-h} h$. The number of population constraints with coefficients in $C_h \cup C_{h(h+1)}$ is given by the $r-1$ constraints (28b) multiplied by the number of systems in (24) with associated combination $(d_1,\ldots,d_q)$ having $h$ non-zeros, i.e., $\binom{r-2}{(r-1)-h}(r-1)$. Now the theorem follows by the relation $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$ that implies that the number of population constraints is $\binom{r-1}{r-h}(r-h)$, and so also the number of rows is $\binom{r-1}{r-h} h + \binom{r-1}{r-h}(r-h) = \binom{r-1}{r-h} r$.