A Generalized Method of Moments for Closed Queueing Networks

Giuliano Casale

Department of Computing
Imperial College London
180 Queen's Gate
London SW7 2AZ, UK.
email: g.casale@imperial.ac.uk

Abstract

We introduce a new solution technique for closed product-form queueing networks that generalizes the Method of Moments (MoM), a recently proposed exact algorithm that is several orders of magnitude faster and memory efficient than the established Mean Value Analysis (MVA) algorithm. Compared to MVA, MoM recursively computes higher-order moments of queue lengths instead of mean values, an approach that remarkably reduces the computational costs of exact solutions, especially on models with large number of jobs.

In this paper, we show that the MoM recursion can be generalized to include multiple recursive branches that evaluate models with different number of queues, a solution approach inspired by the Convolution algorithm. Combining the approaches of MoM and Convolution simplifies the evaluation of normalizing constants and leads to large computational savings with respect to the recursive structure originally proposed for MoM.

Key words: Queueing network models, computational algorithms, method of moments

1 Introduction

Product-form queueing networks [1] are popular stochastic models used in capacity planning of computer systems with the purpose of predicting the effects of resource contention on system scalability under a variety of workload conditions. In many applications, notably in modern multi-tier applications, workloads are best described as multiclass, i.e., requests are assigned to different classes according to the statistical characteristics of the service demand they place on the servers. In spite of their practical importance, multiclass workloads are challenging to analyze exactly in queueing networks even using state-of-the-art solution techniques such as Mean Value Analysis (MVA) [27], Convolution [5,26], RECAL [13], LBANC [10], MVAC [14], or more recent methods based on the generating function approach and Monte Carlo sampling [2, 11, 20, 28]. The main problem is that multiclass models typically involve tens or hundreds of competing requests, possibly belonging to several classes, and the underlying network can be composed by many servers [21]. For models with these characteristics, available exact solution methods require computational costs that are often prohibitive, for example memory requirements can be of the order of terabytes. As a result, large multiclass networks cannot be usually solved with exact techniques and the focus is on approximation methods [9, 15, 29] which yet do not return probabilistic measures because they ignore the normalizing constant of the steady state probabilities. This is a limiting factor because probabilistic measures of queueing networks are becoming increasingly important for detailed performance assessment of computer system behavior.

A recent proposal to address the above issues is the Method of Moments (MoM) [6, 7], a new exact technique for multiclass models that recursively computes higher-order moments of queue lengths instead of mean queue-lengths evaluated in the MVA algorithm. Higher-order moments have the key characteristic that they can be computed recursively via a simple linear system of equations from the solution of a model having a job less of a single class. The structure of the MoM recursion is thus similar to the one used in the single class MVA, but radically differs from the one used in multiclass MVA algorithm which decreases at each recursive step the population of multiple classes simultaneously. Hence, the MoM recursion tree grows linearly with the population size rather than combinatorially as in the multiclass MVA. This significantly decreases computational requirements in both time and space.

Although MoM is more efficient than MVA, it has been observed that exact solutions become inefficient if the number of queues and classes grow simultaneously because the computational costs of MoM are driven by the minimum between these two parameters [6]. This feature of MoM is shared also by its "dual" formulation, the Class-oriented Method of Moments (CoMoM) [7], and makes models with many classes and many queues, at present, very hard or even prohibitive to analyze with available techniques. In order to extend applicability of exact solution methods to such models, we propose a new recursive structure for the MoM approach which yields lower computational requirements than the formulation proposed in [6], while style operating under the same higher-order moment principle. We show both analytically and experimentally that significant improvements can be achieved on models with several queues and classes where MoM is less efficient.

Our main idea consists of integrating within MoM the recursive equations used in the multiclass Convolution algorithm [5, 26]. MoM jointly considers in a linear system of equations the exact recursive formulas used in RECAL [13] and LBANC [10], but not those used in Convolution. This linear system is the main tool used to recursively compute normalizing constants, hence any improvement to its structure can result in significant reductions of computational costs for MoM.

By adding the formula used in Convolution to the MoM linear system we obtain a new computational algorithm which recursively considers also models having less queues than in the original network. This is a marked difference compared to the structure of the original MoM in [6], which does not remove queues from the network. We find that this modification provides substantial computational savings in model solution, thus also showing that adding independent equations to the MoM linear system provides a simple and effective way to reduce computational costs of queueing network analysis.

The remainder of this paper is organized as follows. After giving background and notation in Section 2, we review MoM in Section 3 using a simple multiclass model which also illustrates the principles of the generalization proposed in this paper. The effects of integrating in MoM a recursion on models with different number of queues are discussed in Section 4, where a generalized MoM algorithm is proposed. Computational requirements of the new algorithm are discussed in Section 5. Experiments showing the performance of the generalized MoM in a software implementation are provided in Section 6. Section 7 gives concluding remarks. A numerical example that illustrates a recursive step of the generalized MoM algorithm is given in the final appendix.

2 Background

We consider a closed product-form queueing network with M queues and R workload classes. Jobs are routed probabilistically through the queues where they receive service. After completing service, jobs re-enter the network after a delay of Z_r units of time which depends on the job workload class $r = 1, \dots, R$. The service demand $D_{k,r}$ equals the mean service time multiplied by the mean number of visits [16] of class r jobs at queue k. We assume that the M queues are all distinct, i.e., for each pair of queues there exist at least a workload class $r, 1 \leq r \leq R,$ which places different service demands at the two queues; we refer to a pair of stations that are not distinct as *replicas* of each other. Finally, the number of jobs of class r is the integer N_r, $\vec{N} = (N_1, N_2, \dots, N_R)$ is a population vector, and $N = N_1 + N_2 + \dots + N_R$ is the total number of jobs circulating in the network.

In a closed product-form network, the probability of observing queue k in state $\vec{n}_k = (n_{k,1}, n_{k,2}, \dots, n_{k,R})$, being $n_{k,r}$ the number of class r jobs in the waiting buffer or in the server of queue k, is

$$\Pr(\vec{n}_k) = \frac{F_k(\vec{n}_k) G(\vec{m} - \vec{1}_k, \vec{N} - \vec{n}_k)}{G(\vec{m}, \vec{N})}, \qquad F_k(\vec{n}_k) = n_k! \prod_{r=1}^{R} \frac{D_{k,r}^{n_{k,r}}}{n_{k,r}!}, \qquad (1)$$

where $G(\vec{m}, \vec{N})$ denotes the normalizing constant of the equilibrium state probabilities of the Markov chain underlying the queueing network [19], $\vec{1}_k$ indicates a vector composed of all zeros except for a one in the kth position, and $\vec{m} \equiv (m_1, m_2, \ldots, m_M)$ is a *multiplicity vector* such that the multiplicity m_k is the number of queues in the network with identical mean service demands $D_{k,1}, D_{k,2}, \ldots, D_{k,R}$. For instance, $G(\vec{m} - \vec{1}_k, \vec{N} - \vec{n}_k)$ refers to a model with a replica of queue k less and where the population is specified by the vector $\vec{N} - \vec{n}_k$. Similarly, $G(\vec{m} + \vec{1}_k, \vec{N} - \vec{1}_r)$ represents the normalizing constant of a model including an additional queue with demands $D_{k,1}, D_{k,2}, \ldots, D_{k,R}$, i.e., a replica of queue k, and having a job of class r removed from the network. Note that if $\vec{m} \equiv (1, 1, \ldots, 1)$ then each queue in the network has a distinct set of service demands, i.e., there are no queue replicas. For ease of presentation, we assume in the rest of the paper that $\vec{m} \equiv (1, 1, \ldots, 1)$ and $D_{k,r} > 0$, which provide an upper bound on the costs of solving the queueing network model. Our methodology extends also to models falling outside such assumptions.

The goal of solution algorithms for queueing networks is to evaluate performance indices such as the mean throughput $X_r(\vec{N})$ and the mean response time $R_r(\vec{N}) = N_r / X_r(\vec{N}) - Z_r$ of class r jobs. Additionally, for each queue k and class r, it is often useful to compute the utilization $U_{k,r}(\vec{N}) = D_{k,r} X_r(\vec{N})$, the mean queue length $Q_{k,r}(\vec{N})$, and the mean residence times $R_{k,r}(\vec{N}) = Q_{k,r}(\vec{N}) / X_r(\vec{N})$. These quantities are uniquely determined if one knows how to compute efficiently throughput and mean queue lengths, which are given by the following ratios [24]

$$X_r(\vec{N}) = \frac{G(\vec{m}, \vec{N} - \vec{1}_r)}{G(\vec{m}, \vec{N})}, \qquad Q_{k,r}(\vec{N}) = \frac{D_{k,r} G(\vec{m} + \vec{1}_k, \vec{N} - \vec{1}_r)}{G(\vec{m}, \vec{N})}, \tag{2}$$

where $\vec{m}$ is the multiplicity vector of the model for which performance indexes are computed. Instead, the marginal probabilities $\Pr(\vec{n}_k)$ cannot be computed either by MVA or by local iterative approximations [9, 15, 29] that ignore the normalizing constant. Thus, the normalization constant approach followed in this paper is more applicable than MVA if one is interested in evaluating state probabilities.

Finally, we stress that increasing the number of queue replicas with the operator $\vec{m} + \vec{1}_k$ appearing in (2) and in several derivations throughout the next sections, is fundamental for the MoM approach. A probabilistic interpretation of the replica addition operation follows by first observing that the normalizing constant can be written as

$$G(\vec{m}, \vec{N}) = \sum_{\vec{n}} \sum_{k=1}^{m_1+m_2+\ldots+m_M} F_k(\vec{n}_k), \tag{3}$$

where we have assumed for simplicity that the network has $Z_r = 0$ for all classes, $\vec{n} = (\vec{n}_1, \vec{n}_2, \ldots, \vec{n}_{m_1+m_2+\ldots+m_k})$, $\vec{n}_k = (n_{k,1}, n_{k,2}, \ldots, n_{k,R})$, describes the state of all the queues in the network and the term $m_1 + m_2 + \ldots + m_M$ is the total number of queues in the model, also accounting for queue replicas. Using basic combinatorial analysis [12], (3) can then be rewritten as

$$G(\vec{m},\vec{N}) = \sum_{\vec{n}'} \sum_{k=1}^{M} \binom{m_k + n'_k - 1}{n'_k} F_k(\vec{n}'_k), \tag{4}$$

where $\vec{n}' = (\vec{n}'_1, \vec{n}'_2, \dots, \vec{n}'_M)$ is the network state and $\vec{n}'_k = (n'_{k,1}, n'_{k,2}, \dots, n'_{k,R})$ is a vector describing the local populations within the subnetwork composed only by m_k queues with identical demands $D_{k,1}$, $D_{k,2}$, . . ., $D_{k,R}$. From (4) it is easy to prove that increasing the multiplicities in $\vec{m}$ is equivalent to computing un-normalized joint binomial moments of queue-lengths. The same conclusion is obtained if one considers the general case of a queueing network including delays [6, 8].

2.1 Computational Algorithms

The analysis of queueing networks can be performed efficiently either by approaches that directly evaluate mean queue lengths and throughputs in a recursive fashion, such as the MVA [27], or by computational methods for the normalizing constant [4]. Among the two solution paradigms, the normalizing constant approach is slightly more efficient [3], but it often suffers numerical issues that do not apply to mean value formulas. In particular, the rapid growth of the state space size can lead the summation in (3) to floating-point range overflows or underflows [23]. However, issues of this kind can be easily addressed in modern software at the cost of overheads for representing normalizing constants with programming libraries that support exact multi-precision arithmetic or modulo representations [17].

From a probabilistic perspective, the MVA algorithm and some methods for the normalizing constant, such as the LBANC algorithm [10], can be interpreted as a recursive evaluation of mean queue lengths, un-normalized in the case of LBANC, over models with different population sizes. Recently, [6, 7] have noted that recursively evaluating a set of higher-order moments of queue lengths can be much more efficient computationally than obtaining mean values, while still returning at the end of the execution the correct solution of the model; this solution includes mean performance indexes, such as $X_r(\vec{N})$ and $Q_{k,r}(\vec{N})$. The Method of Moments (MoM) [6] is an algorithm that implements this higher-order moment paradigm and that we generalize for increased efficiency in the next sections. Due to limited space and thanks to wide availability of material on the subject, we point to the literature for an introduction to MVA [27], LBANC [10], RECAL [13], MVAC [14], and Convolution [5, 26], comparative analyses can be found in [3, 6]; the remainder of this section focuses instead on MoM.

2.1.1 Method of Moments (MoM)

MoM computes the normalizing constant by simultaneously considering into a linear system of equations the following exact formulas for normalizing constants: the *convolution expression* (CE) [10, 24]

$$G(\vec{m}+\vec{1}_k,\vec{N}) = G(\vec{m},\vec{N}) + \sum_{r=1}^{R} D_{k,r} G(\vec{m}+\vec{1}_k,\vec{N}-\vec{1}_r) \tag{5}$$

for all $1 \leq k \leq M$, and the *population constraint* (PC) [6, 13]

$$N_r G(\vec{m},\vec{N}) = Z_r G(\vec{m},\vec{N}-\vec{1}_r) + \sum_{k=1}^{M} m_k D_{k,r} G(\vec{m}+\vec{1}_k,\vec{N}-\vec{1}_r) \tag{6}$$

for all $1 \leq r \leq R$, which are also the fundamental recursive equations used in the LBANC and RECAL algorithms. These recursions are subject to the termination conditions $G(\vec{m},\vec{0}) = 1$ and $G(\vec{m},\vec{N}) = 0$ if any entry in $\vec{N}$ is negative. It is important to note that, while (5) is similar to the recursive equation used in the Convolution algorithm [5, 26], (5) fundamentally differs from the latter because it does not allow to remove queues. We illustrate the practical difference between the CE and the formula used in Convolution in Section 3.2.

In classic algorithms, $G(\vec{m},\vec{N})$ is obtained by recursively evaluating either (5) or (6) until termination conditions are met and possibly with a scaling of the normalizing constants into mean values. Following this approach, time and space requirements grow with respect to the population size as $O(N^R)$ if (5) is used (e.g., LBANC, MVA) and as $O(N^M)$ if (6) is used (e.g., RECAL, MVAC). In practice, these costs are often prohibitive since in modeling modern systems it is not rare to have N of the order of hundreds or even thousands, and several queues and classes (see [21] for a recent case study). This makes the storage requirement of hundreds of gigabytes or terabytes regardless of the recursion used.

MoM overcomes these computational problems by observing that, if one tries to compute a certain vector of normalizing constants $V(\vec{m},\vec{N})$, henceforth called *basis* [1], then with a proper definition this vector can be evaluated recursively by using jointly (5) and (6) in a matrix recurrence equation

$$\mathbf{A}(\vec{m},\vec{N})V(\vec{m},\vec{N}) = \mathbf{B}(\vec{m},\vec{N})V(\vec{m},\vec{N}-\vec{1}_R). \tag{7}$$

In the above equation, $V(\vec{m},\vec{0})$ is known from the termination conditions of (5)-(6) and the matrices $\mathbf{A}(\vec{m},\vec{N})$ and $\mathbf{B}(\vec{m},\vec{N})$ are square of identical size and defined

[1] The name "basis" stresses that $V(\vec{m},\vec{N})$ usually carries the minimum information needed to perform a recursion that is linear in the total population of the network. Yet, suboptimal implementations where $V(\vec{m},\vec{N})$ is not minimal are also useful to simplify implementation efforts [8]. To avoid unnecessary complexity, we use the term basis also for such implementations.

by the coefficients of the equations (5)-(6) that relate *all and only* the normalizing constants in $V(\vec{m},\vec{N})$ with those in $V(\vec{m},\vec{N}-\vec{1}_R)$. The fundamental conditions for application of (7) are

- to define a basis that allows to formulate the linear system exactly as in (7);
- to have a non-singular matrix $\mathbf{A}(\vec{m},\vec{N})$.

In particular, if the choice of the normalizing constants is made such that the coefficient matrix of the linear system (7) is square and non-singular, it is easy to compute

$$V(\vec{m},\vec{N}) = \mathbf{A}^{-1}(\vec{m},\vec{N})\mathbf{B}(\vec{m},\vec{N})V(\vec{m},\vec{N}-1_R)$$

and solve recursively the model. Equivalently, for a rectangular over-determined $\mathbf{A}(\vec{m},\vec{N})$ of maximum rank it is

$$V(\vec{m},\vec{N}) = (\mathbf{A}^T(\vec{m},\vec{N})\mathbf{A}(\vec{m},\vec{N}))^{-1}\mathbf{A}^T(\vec{m},\vec{N})\mathbf{B}(\vec{m},\vec{N})V(\vec{m},\vec{N}-1_R). \quad (8)$$

Indeed, matrix inversion can be replaced in implementations by much more efficient linear system solution techniques, such as sparse Gaussian elimination or the Wiedemann algorithm [22,31], which apply effectively to problems where the normalizing constants are formulated using modulo or exact arithmetic representations to address range overflows and underflows problems [2].

MoM leverages on the above linear system approach by defining the basis as

$$\begin{aligned} V(\vec{m},\vec{N}) = \{&G(\vec{m}+\vec{\delta},\vec{N}), G(\vec{m}+\vec{\delta},\vec{N}-\vec{1}_1),\ldots,G(\vec{m}+\vec{\delta},\vec{N}-\vec{1}_{R-1}) \\ &\mid \vec{\delta}=(\delta_1,\delta_2,\ldots,\delta_M),\ R-1 \le \textstyle\sum_{k=1}^{M}\delta_k \le R, 0 \le \delta_k \le R, \delta_k \in \mathbb{N}\}, \end{aligned} \quad (9)$$

which is the set of normalizing constants of models where we have added R or $R-1$ queue replicas, for all possible choices of the multiplicity increase vector $\vec{\delta}$, and where the models are evaluated over populations $\vec{N}$, $\vec{N}-\vec{1}_1$, ..., $\vec{N}-\vec{1}_{R-1}$. Following the probabilistic interpretation of (4), we see that MoM defines $V(\vec{m},\vec{N})$ as the set of all (un-normalized) joint binomial moments of order $R-1$ and R conditioned on different populations. Recalling that conditioning on a population $\vec{N}-\vec{1}_s$ equals evaluating the network state at arrival times of class s jobs at any tagged queue [30], we conclude that the recursive computation of $V(\vec{m},\vec{N})$ in MoM is also a recursive evaluation of higher-order moments of queue lengths, including both the values at equilibrium and the values embedded at job arrival times. It is found that such values are sufficient to uniquely determine performance indexes such as mean throughput and mean queue-lengths [6,8]. Furthermore, we remark that other definitions of the basis exist which enjoy similar properties, but different computational trade-offs, such as the basis proposed in CoMoM [7]. The CoMoM basis provides reduced complexity for models having a large number of classes by

[2] At present, the LinBox open source library (`http://www.linalg.org`) offers a free implementation of the Wiedemann algorithm, exact Gaussian elimination, and other exact methods that can be used to solve the MoM recursion.

applying the $\vec{\delta}$ vector in (9) to the population vector $\vec{N}$ rather than to the multiplicity vector $\vec{m}$. However, due to the substantially different recursive structure of this algorithm compared to MoM we do not longer consider it throughout the next sections.

Finally, we remark that the MoM basis definition in (9) has been shown to meet the requirements for formulating (7), which is solved using a recursion that grows linearly with the total population of jobs N. Therefore, the normalizing constant can be computed in just N steps instead of the $O(N^R)$ or $O(N^M)$ steps required by (5) or (6) when evaluated in isolation. It can be shown that the extra costs required to adopt exact or modulo arithmetic in implementations lead to a complexity for MoM of $O(N^2 \log N)$ time and $O(N \log N)$ space with respect to the total population N.

3 Motivating Example

In the remainder of this paper, we describe a new technique for defining the basis $V(\vec{m}, \vec{N})$ that offers computational advantages with respect to existing approaches. Our approach is first illustrated by the motivating example in this section.

3.1 Structure and Improvement of the MoM Recursion

We begin by illustrating the principles of the MoM recursion (7) on a simple queueing network with $M = 2$ distinct queues and $R = 2$ classes. The job population vector is $\vec{N} = (N_1, N_2)$ and we assume that all queues are distinct. Following definition (9), the basis of normalizing constants for this model has the structure depicted in Figure 1. Informally, the figure represents un-normalized joint binomial moments of order $R-1$ and R: each circle stands for the normalizing constants $G(\vec{m}+\vec{\delta}, \vec{N})$ and $G(\vec{m}+\vec{\delta}, \vec{N}-1_1)$ in $V(\vec{m}, \vec{N})$, we have argued previously that these constants are equivalent to binomial moments. Labels specify the multiplicity increase vector $\vec{\delta}$ in (9), e.g., the circle labeled $+\vec{1}_2$ represents the set $\{G(\vec{m}+\vec{1}_2, \vec{N}), G(\vec{m}+\vec{1}_2, \vec{N}-1_1)\}$. Arrows indicate dependencies between normalizing constants that arise due to CEs and PCs. In order to express compactly such dependencies, we often denote in the rest of this section $d_{z,k,s} \equiv (z+m_k) \cdot D_{k,s}$ and $G^{+a,b}_{c,d} \equiv G(\vec{m}+\vec{1}_a+\vec{1}_b, \vec{N}-\vec{1}_c-\vec{1}_d)$; similar abbreviations, such as $G^{-a} \equiv G(\vec{m}-\vec{1}_a, \vec{N})$, are also used throughout.

Using the new notation, we have that the MoM basis in Figure 1 is the vector

$$V(\vec{m}, \vec{N}) = [G^{+1,1}, G^{+1,1}_1, G^{+1,2}, G^{+1,2}_1, G^{+2,2}, G^{+2,2}_1, G^{+1}, G^{+1}_1, G^{+2}, G^{+2}_1]^T$$

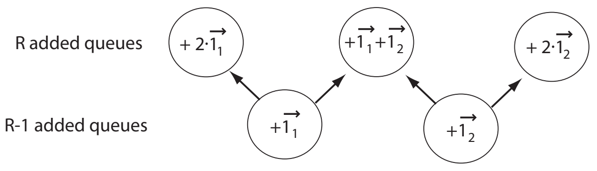

Fig. 1. Basis of normalizing constants $V(\vec{m},\vec{N})$ for $M=2$ queues and $R=2$ classes

and similarly, for $R=2$, it is

$$V(\vec{m},\vec{N}-\vec{1}_R)=[G_2^{+1,1},G_{1,2}^{+1,1},G_2^{+1,2},G_{1,2}^{+1,2},G_2^{+2,2},G_{1,2}^{+2,2},G_2^{+1},G_{1,2}^{+1},G_2^{+2},G_{1,2}^{+2}]^T.$$

Dependencies arising between these two bases due to CEs for $k=1$ are

$$G^{+1,1}=G^{+1}+D_{1,1}G_1^{+1,1}+D_{1,2}G_2^{+1,1} \tag{10}$$
$$G^{+1,2}=G^{+2}+D_{1,1}G_1^{+1,2}+D_{1,2}G_2^{+1,2} \tag{11}$$

and similarly due to CEs for $k=2$ are

$$G^{+1,2}=G^{+1}+D_{2,1}G_1^{+1,2}+D_{2,2}G_2^{+1,2} \tag{12}$$
$$G^{+2,2}=G^{+2}+D_{2,1}G_1^{+2,2}+D_{2,2}G_2^{+2,2}. \tag{13}$$

The PCs for class 1 are instead

$$N_1G^{+1}=Z_1G_1^{+1}+d_{1,1,1}G_1^{+1,1}+d_{0,2,1}G_1^{+1,2} \tag{14}$$
$$N_1G^{+2}=Z_1G_1^{+2}+d_{0,1,1}G_1^{+1,2}+d_{1,2,1}G_1^{+2,2}, \tag{15}$$

while the PCs for class 2 are

$$N_2G^{+1}=Z_2G_2^{+1}+d_{1,1,2}G_2^{+1,1}+d_{0,2,2}G_2^{+1,2} \tag{16}$$
$$N_2G_1^{+1}=Z_2G_{1,2}^{+1}+d_{1,1,2}G_{1,2}^{+1,1}+d_{0,2,2}G_{1,2}^{+1,2} \tag{17}$$
$$N_2G^{+2}=Z_2G_2^{+2}+d_{0,1,2}G_2^{+1,2}+d_{1,2,2}G_2^{+2,2} \tag{18}$$
$$N_2G_1^{+2}=Z_2G_{1,2}^{+2}+d_{0,1,2}G_{1,2}^{+2,1}+d_{1,2,2}G_{1,2}^{+2,2}. \tag{19}$$

Note that all normalizing constants used to formulate the above equations are included in one between $V(\vec{m},\vec{N})$ and $V(\vec{m},\vec{N}-\vec{1}_R)$. Other equations exist that involve some of the constants in $V(\vec{m},\vec{N})$ or $V(\vec{m},\vec{N}-\vec{1}_R)$, however these also involve normalizing constants not included in these two bases and for this reason are not considered in MoM. For example, observe that the PCs for class 1 are less than those for class 2 because removing a job of class 1 from G_1^{+1} would require the constant $G_{1,1}^{+1}$ that is not included in $V(\vec{m},\vec{N})$ or in $V(\vec{m},\vec{N}-\vec{1}_R)$.

Following the above discussion, the MoM solution approach consists in rearranging the above equations into the linear system

$$
\underbrace{\left[\begin{array}{cccccccccc}
1 & -D_{1,1} & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & -D_{1,1} & \cdot & \cdot & \cdot & \cdot & -1 & \cdot \\
\hline
\cdot & \cdot & 1 & -D_{2,1} & \cdot & \cdot & -1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & -D_{2,1} & \cdot & \cdot & -1 & \cdot \\
\hline
\cdot & -d_{1,1,1} & \cdot & -d_{0,2,1} & \cdot & \cdot & N_1 & -Z_1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & -d_{0,1,1} & \cdot & -d_{1,2,1} & \cdot & \cdot & N_1 & -Z_1 \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & N_2 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & N_2 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & N_2 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & N_2
\end{array}\right]}_{\mathbf{A}(\vec{m},\vec{N})}
\underbrace{\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}}_{V(\vec{m},\vec{N})}
=
\underbrace{\left[\begin{array}{cccccccccc}
D_{1,2} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & D_{1,2} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\hline
\cdot & \cdot & D_{2,2} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & D_{2,2} & \cdot & \cdot & \cdot & \cdot & \cdot \\
\hline
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\hline
d_{1,1,2} & \cdot & d_{0,2,2} & \cdot & \cdot & \cdot & Z_2 & \cdot & \cdot & \cdot \\
\cdot & d_{1,1,2} & \cdot & d_{0,2,2} & \cdot & \cdot & \cdot & Z_2 & \cdot & \cdot \\
\cdot & \cdot & d_{0,1,2} & \cdot & d_{1,2,2} & \cdot & \cdot & \cdot & Z_2 & \cdot \\
\cdot & \cdot & \cdot & d_{0,1,2} & \cdot & d_{1,2,2} & \cdot & \cdot & \cdot & Z_2
\end{array}\right]}_{\mathbf{B}(\vec{m},\vec{N})}
\underbrace{\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}}_{V(\vec{m},\vec{N}-\vec{1}_R)}
$$

where $\cdot$ indicates a zero element and the four blocks of the coefficient matrices represent from top: the CEs (10)-(11) formulated for $k = 1$, the CEs (12)-(13) for $k = 2$, the PCs (14)-(15) for $r = 1$, and the PCs (16)-(19) for $r = 2$. The above structure is exactly the one expressed by (7) and provides a way to compute recursively the normalizing constants that compose the MoM basis. This can be achieved provided that the coefficient matrix is non-singular, which may depend on the specific values of the service demands. Note in particular that the CEs involve only constants with $\delta_1 + \delta_2 = 2$, and not those with $\delta_1 + \delta_2 = 1$, because in the latter case one would need also the value of the constant G in the left hand side of (6), which is not included in the basis.

The questions addressed in the rest of the paper is whether the integration in the linear system of the recursive equations used in the Convolution algorithm, which require some constants not included in the MoM basis, could be performed in a computationally efficient manner. In particular, we study whether it is possible to obtain linear systems throughout the recursion that have smaller order than those used in MoM. We found this to be possible in practice and we show later that the same model considered in the previous example can be solved by a linear system having only 6 equations and 6 unknowns instead of the 10 equations and 10 unknowns used in MoM. The main observations leading to this improvement are the following:

(1) we first note that it is possible to add independent equations to the linear system shown in this subsection by taking in consideration a generalization of the convolution expression (5) which corresponds to the equation used in the Convolution algorithm. This generalization provides independent information and makes the linear system over-determined.
(2) Given that the linear system is over-determined, we show that the basis $V(\vec{m}, \vec{N})$ can be defined in a different way that makes the basis size smaller, while still allowing to solve recursively the model. This basis size reduction leads to remarkable computational savings in the linear system solution.

(3) However, as we explain in Section 4, for models of arbitrary size the additional independent information comes at the price of additional recursions over models with different number of queues. We investigate in the rest of the paper if accepting these additional recursions is cost effective in light of the computational savings implied by the basis size reduction.

The previous observations are further illustrated in the next subsection.

3.2 Improved Computation of the Basis of Normalizing Constants

We begin by observing that (5) can be seen as a specialization of the recursive equation used by the Convolution Algorithm [5, 26], which we call throughout this paper the *general convolution expression* (GCE)

$$G(\vec{m},\vec{N}) = G(\vec{m}-\vec{1}_k,\vec{N}) + \sum_{r=1}^{R} D_{k,r} G(\vec{m},\vec{N}-\vec{1}_r), \tag{20}$$

for all $1 \leq k \leq M$. In the GCE, queues are removed through the parameter $\vec{m}-\vec{1}_k$, instead of being added as in the CE, up to reaching the termination condition $G(\vec{0},\vec{N}) = \prod_{r=1}^{R} Z_r/N_r!$. This implies that a recursion involving (20) evaluates models which contain less queues than in the original queueing network, while using only CEs implies the opposite. However, by formulating (20) for a model with multiplicity $\vec{m}+\vec{1}_k$ instead of $\vec{m}$, it is found that the GCE expression becomes identical to the CE expression, thus CEs can be seen as a subset of GCEs formulated for normalizing constants with increased multiplicities [3]. However, if a GCE is formulated on models with fewer queues than in the original network, then the information provided by (20) is linearly independent with respect to the one provided by (5) because the two equations are defined over models with different network structure. Since we are only interested in adding independent information to the MoM linear system, *we henceforth refer to GCEs meaning the subset of equations (20) that are not CEs*.

As an example of application of GCEs, it is useful to add (20) to the linear system of Section 3.1, formulated as

$$G^{+1,1} = G^{+1,1,-2} + D_{2,1} G_1^{+1,1} + D_{2,2} G_2^{+1,1}, \tag{21}$$

because the normalizing constant $G^{+1,1,-2} \equiv G(\vec{m}+2\cdot\vec{1}_1-\vec{1}_2,\vec{N})$ lies outside the basis $V(\vec{m},\vec{N})$, thus (21) does not reduce to a CE and brings independent information into the linear system. Note that $G^{+1,1,-2}$ is not included in $V(\vec{m},\vec{N})$

[3] This property also clarifies the term "general" that we have used in the GCE acronym. Note that GCEs are slightly more general than CEs, for example it is impossible to consider with CEs a recursion that involves only normalizing constants for models with $M \leq 1$.

or $V(\vec{m}, \vec{N} - \vec{1}_R)$, yet this term is easily obtained being the normalizing constant of a model where queue 2 has been completely removed, i.e., $m_2 = 0$. Thus, the normalizing constant can be computed immediately by the closed-form formulas for the normalizing constant of a network composed only of m_1 identical queues with demands $D_{1,1}, D_{1,2}, \dots, D_{1,R}$. For $Z_1 = Z_2 = 0$, such normalizing constant may be obtained by the closed-form formula [25]

$$G((0, \dots, 0, m_k, 0, \dots, 0), \vec{N}) = \binom{m_k + N - 1}{N} F_k(\vec{m}, \vec{N}). \tag{22}$$

Thanks to the last observations, we conclude that the addition of (21) to the linear system of the example model provides additional information and does not increase the number of unknowns in the linear system. This property has been illustrated for the case of a model without delays, but we show in Section 4.2 that extends to general models. The main question investigated in the remainder of this paper is whether this independent information can be used to reduce the size of the basis $V(\vec{m}, \vec{N})$.

To illustrate the applicability of this concept, consider a new basis

$$V'(\vec{m}, \vec{N}) = [G^{+1}, G_1^{+1}, G^{+2}, G_1^{+2}, G, G_1]^T$$

that is composed by less normalizing constants than $V(\vec{m}, \vec{N})$, and consider also the same basis formulated for a model without queue $M = 2$, i.e.

$$V'(\vec{m} - \vec{1}_M, \vec{N}) = [G^{+1-2}, G_1^{+1-2}, G^{-2}, G_1^{-2}]^T.$$

Then, by (5), (6), and (21) we can formulate a linear system

$$\mathbf{A}'(\vec{m}, \vec{N}) V'(\vec{m}, \vec{N}) = \mathbf{B}'(\vec{m}, \vec{N}) V'(\vec{m}, \vec{N} - \vec{1}_R) + \mathbf{C}'(\vec{m}, \vec{N}) V'(\vec{m} - \vec{1}_M, \vec{N}) \tag{23}$$

where in the specific example we are considering it is

$$\underbrace{\left[\begin{array}{cccccc} 1 & -D_{1,1} & \cdot & \cdot & -1 & \cdot \\ \hline \cdot & \cdot & 1 & -D_{2,1} & -1 & \cdot \\ \hline 1 & -D_{2,1} & \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & -d_{0,1,1} & \cdot & -d_{0,2,1} & N_1 & -Z_1 \\ \hline \cdot & \cdot & \cdot & \cdot & N_2 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & N_2 \end{array}\right]}_{\mathbf{A}'(\vec{m},\vec{N})} \underbrace{\begin{bmatrix} G^{+1} \\ G_1^{+1} \\ G^{+2} \\ G_1^{+2} \\ G \\ G_1 \end{bmatrix}}_{V'(\vec{m},\vec{N})} = \underbrace{\left[\begin{array}{cccccc} D_{1,2} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & D_{2,2} & \cdot & \cdot & \cdot \\ \hline D_{2,2} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hline d_{0,1,2} & \cdot & d_{0,2,2} & \cdot & Z_2 & \cdot \\ \cdot & d_{0,1,2} & \cdot & d_{0,2,2} & \cdot & Z_2 \end{array}\right]}_{\mathbf{B}'(\vec{m},\vec{N})} \underbrace{\begin{bmatrix} G_2^{+1} \\ G_{1,2}^{+1} \\ G_2^{+2} \\ G_{1,2}^{+2} \\ G_2 \\ G_{1,2} \end{bmatrix}}_{V'(\vec{m},\vec{N}-\vec{1}_R)}$$

$$+ \underbrace{\left[\begin{array}{cccc} \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot \\ \hline 1 & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{array}\right]}_{\mathbf{C}'(\vec{m},\vec{N})} \underbrace{\begin{bmatrix} G^{+1-2} \\ G_1^{+1-2} \\ G^{-2} \\ G_1^{-2} \end{bmatrix}}_{V'(\vec{m}-\vec{1}_M,\vec{N})} .$$

In the above expression the new vector $V'(\vec{m} - \vec{1}_M, \vec{N})$ includes the known normalizing constant $G^{+1,-2} \equiv G(\vec{m} + \vec{1}_1 - \vec{1}_2, \vec{N})$ and the blocks of the coefficient matrix are from the top: the CE for $k = 1$, the CE for $k = 2$, the GCE (20), the PC for $r = 1$, and the PC for $r = 2$. Note that this linear system has square coefficient matrix, thus if the inverse of $\mathbf{A}'(\vec{m}, \vec{N})$ exists the solution of the linear system (23) provides a new way to recursively compute normalizing constants that is cheaper than in the MoM linear system (7). Specifically, (23) almost halves the order of the coefficient matrix with respect to (7), which is significant since the computational costs of linear system solution are quadratic or cubic, depending on the algorithm used, with respect to the coefficient matrix order. We stress that without (21) the new system (23) would be under-determined, thus GCEs are the key tool behind this new approach.

The fundamental issue connected with the approach outlined in this subsection is that, for queueing networks larger than the one considered in the previous example, the normalizing constants in $V'(\vec{m} - \vec{1}_M, \vec{N})$ may not be available from closed-form expressions. In this case, the computation of $V'(\vec{m} - \vec{1}_M, \vec{N})$ requires a new recursive branch that is orthogonal to the usual recursion over the population vectors $\vec{N}$, $\vec{N} - \vec{1}_R$, . . ., $\vec{0}$; this additional recursion is over models having less queues than in the queueing network under study. The proposed recursive structure is thus more complex than in MoM and requires detailed investigation, presented in the next sections, to clarify its computational properties.

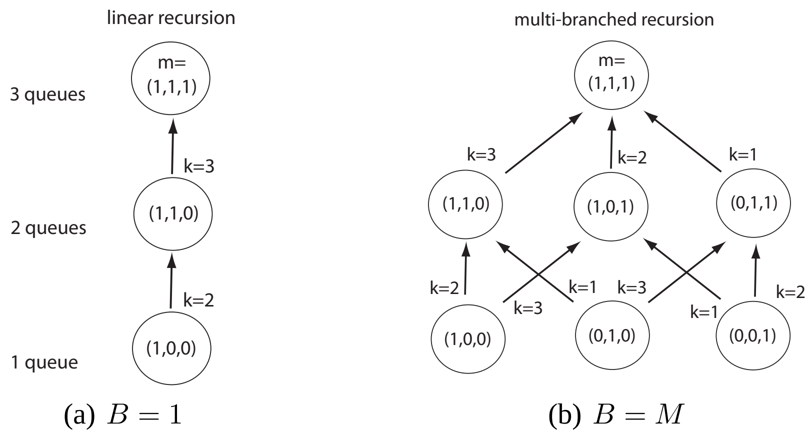

Fig. 2. Recursion after adding a single or multiple GCEs to the MoM linear system for a model with $M = 3$ queues

4 The Generalized Method of Moments

We generalize the observations in Section 3.2 and prove that (20) can help in reducing the computational costs of the MoM recursion in models with arbitrary number of queues, classes, and jobs. For ease of exposition, we initially assume that the model has no delays such that $Z_r = 0$ for all workload classes r. The generalization to the case with arbitrary delay values is simple and discussed in Section 4.2.

We begin with observing that, in a model with M distinct queues, (20) can be formulated and added to the MoM linear system either for a single queue with given index k or jointly for several queues having different indexes k, up to a maximum of M queues. Let us call *branching factor* B, $1 \leq B \leq M$, the number of equations (20) that are added simultaneously to the MoM linear system for a given model with M queues. A comparison of two possible recursion trees arising from the two limiting cases $B = 1$ and $B = M$ is given in Figure 2. In Figure 2(a), a single queue is removed at a time from the model under study, yielding a recursion that evaluates a total of M models, each differing from the previous one only for having a queue less. Each model is studied using a different basis of binomial moments $V(\vec{m}, \vec{N})$, where $\vec{m}$ describes the number of queues considered at the current stage of the recursion. The recursion terminates upon reaching models with a single queue for which the normalizing constant is computed by (22). Conversely, Figure 2(b) shows the recursive branch that derives from removing simultaneously multiple queues at each step of the recursion. Note that at the initial step the model includes $M = 3$ queues and therefore up to three GCEs can be formulated. Instead, at the level immediately below there are only $M - 1 = 2$ queues which reduces the number of GCEs to two; finally the recursion terminates at the bottom level by evaluating trivial models with a single distinct queue. Indeed, the recursive application of this approach leads to a quick growth in the number of models to be considered. However, we find in Section 5 the counter-intuitive result that the case

$B = M$ is the most efficient in practice, and the combinatorial growth of recursions with multiple branches such as in Figure 2(b) is widely compensated by the concurrent decrease in the computational costs of the linear systems solved throughout the recursion. The general structure of the recursion arising for arbitrary B values is investigated below.

4.1 *Solution of Generalized Linear System*

Studying the structure of the MoM linear system when (20) is included requires to distinguish between:

- cases where a *fixed* number of queues is removed in all models considered throughout the recursion and until termination conditions are reached. For instance, in the case shown in Figure 2(a), it is always possible to formulate the requested $B = 1$ GCEs at all steps of the recursion, provided that the specific queue k chosen in (20) is selected among the queues left in the current model. Note that cases where $B > 1$ can be implemented with this paradigm only if terminations conditions have been previously computed for models having $B - 1$ queues or more, otherwise one would not be able to formulate B distinct GCEs for such models since there are only $B - 1$ queues left.
- Cases where the number of queues to be removed is *variable* and thus B can depend on the number of queues left in the model. For instance, the choice $B = M$ leads to first removing M queues from the initial queueing network, then for the resulting models with $M - 1$ queues one could set at most $B = M - 1$, and so forth up to the termination conditions (22) where $B = 1$.

Indeed, there is a difference in the recursive structure of the algorithm depending on B being fixed or variable. Hence, we characterize the MoM linear system solution in the two cases separately.

4.1.1 *Recursion by Fixed Branching Factor*

Understanding the properties of the generalized MoM linear system requires first looking into the block matrix reformulation of (7). Let us define a *basis of level* l, $l \geq 1$, as

$$V_l(\vec{m}, \vec{N}) = \{G(\vec{m} + \vec{\delta}, \vec{N}), G(\vec{m} + \vec{\delta}, \vec{N} - \vec{1}_1), \dots, G(\vec{m} + \vec{\delta}, \vec{N} - \vec{1}_{R-1}) \\ \mid \vec{\delta} = (\delta_1, \dots, \delta_M),\ \textstyle\sum_{k=1}^{M} \delta_k = l, \delta_k \in \{0, 1\}\},$$

which is the set of normalizing constants with l additional queue replicas or, equivalently, a set of un-normalized joint binomial moments of order l. According to this definition, the MoM basis is simply the union

$$V(\vec{m},\vec{N}) = V_R(\vec{m},\vec{N}) \cup V_{R-1}(\vec{m},\vec{N})$$

as we have illustrated in Figure 1. Note that a basis of level l has cardinality

$$card(V_l(\vec{m},\vec{N})) = \binom{M+l-1}{M-1} R.$$

Using the new notation, it is easy to see that the MoM recursion (7) may be rewritten in upper block triangular form as

$$\underbrace{\begin{bmatrix} \mathbf{A}_{l,l} & \mathbf{A}_{l,l-1} \\ \mathbf{0} & N_R\mathbf{I} \end{bmatrix}}_{A(\vec{m},\vec{N})} \underbrace{\begin{bmatrix} V_l(\vec{m},\vec{N}) \\ V_{l-1}(\vec{m},\vec{N}) \end{bmatrix}}_{V(\vec{m},\vec{N})} = \underbrace{\begin{bmatrix} \mathbf{B}_{l,l} & \mathbf{0} \\ \mathbf{B}_{l-1,l} & \mathbf{B}_{l-1,l-1} \end{bmatrix}}_{B(\vec{m},\vec{N})} \underbrace{\begin{bmatrix} V_l(\vec{m},\vec{N}-\vec{1}_R) \\ V_{l-1}(\vec{m},\vec{N}-\vec{1}_R) \end{bmatrix}}_{V(\vec{m},\vec{N}-\vec{1}_R)}, \quad (24)$$

where $l = R$, the notation $\mathbf{0}$ indicates a matrix of all zeros of proper size, $\mathbf{A}_{i,j}$ and $\mathbf{B}_{i,j}$ indicate matrices having $card(V_i(\vec{m},\vec{N}))$ rows and $card(V_j(\vec{m},\vec{N}))$ columns, and $N_R\mathbf{I}$ is diagonal of order $card(V_{l-1}(\vec{m},\vec{N}))$. In (24) the matrices $\mathbf{A}_{l,l}$, $\mathbf{A}_{l,l-1}$, and $\mathbf{B}_{l,l}$ are defined by the coefficients of the PCs for classes $r = 1,\ldots,R-1$ and by the coefficients of all the CEs (5). Conversely, $\mathbf{B}_{l-1,l}$ and $\mathbf{B}_{l-1,l-1}$ are coefficients of the PCs for class R. Specifically $\mathbf{B}_{l-1,l-1}$ includes all and only the Z_R terms, thus it reduces to $\mathbf{B}_{l-1,l-1} = \mathbf{0}$ in the case $Z_R = 0$ studied in this subsection. Note that a solution of (24) can be easily decomposed into two steps

(1) compute $V_{l-1}(\vec{m},\vec{N}) = (\mathbf{B}_{l-1,l}V_l(\vec{m},\vec{N}-\vec{1}_R)+\mathbf{B}_{l-1,l-1}V_{l-1}(\vec{m},\vec{N}-\vec{1}_R))/N_R$
(2) solve $\mathbf{A}_{l,l}V_l(\vec{m},\vec{N}) = -\mathbf{A}_{l,l-1}V_{l-1}(\vec{m},\vec{N}) + \mathbf{B}_{l,l}V_l(\vec{m},\vec{N}-\vec{1}_R)$

thus MoM complexity is dominated by the cost of solving a linear system of order

$$card(V_l(\vec{m},\vec{N})) = card(V_R(\vec{m},\vec{N})) = \binom{M+R-1}{R} R.$$

In the proposed generalization, for a model with M queues and $B = 1$ we consider GCEs involving models obtained by removing only queue M. This creates a new recursive branch such that

$$\mathbf{A}(\vec{m},\vec{N})V(\vec{m},\vec{N}) = \mathbf{B}(\vec{m},\vec{N})V(\vec{m},\vec{N}-\vec{1}_R) + \mathbf{C}(\vec{m},\vec{N})V(\vec{m}-\vec{1}_M,\vec{N}),$$

which has the following block structure

$$\underbrace{\begin{bmatrix} \mathbf{A}_{l,l'} & \mathbf{0} \\ \mathbf{A}_{l,l} & \mathbf{A}_{l,l-1} \\ \mathbf{0} & N_R\mathbf{I} \end{bmatrix}}_{\mathbf{A}(\vec{m},\vec{N})} \underbrace{\begin{bmatrix} V_l(\vec{m},\vec{N}) \\ V_{l-1}(\vec{m},\vec{N}) \end{bmatrix}}_{V(\vec{m},\vec{N})} = \underbrace{\begin{bmatrix} \mathbf{D}_{l,l} & \mathbf{0} \\ \mathbf{B}_{l,l} & \mathbf{0} \\ \mathbf{B}_{l-1,l} & \mathbf{0} \end{bmatrix}}_{\mathbf{B}(\vec{m},\vec{N})} \underbrace{\begin{bmatrix} V_l(\vec{m},\vec{N}-\vec{1}_R) \\ V_{l-1}(\vec{m},\vec{N}-\vec{1}_R) \end{bmatrix}}_{V(\vec{m},\vec{N}-\vec{1}_R)}$$
$$+ \underbrace{\begin{bmatrix} \mathbf{C}_{l,l'} & \mathbf{C}_{l,l'-1} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_{\mathbf{C}(\vec{m},\vec{N})} \underbrace{\begin{bmatrix} V_{l'}(\vec{m}-\vec{1}_M,\vec{N}) \\ V_{l'-1}(\vec{m}-\vec{1}_M,\vec{N}) \end{bmatrix}}_{V(\vec{m}-\vec{1}_M,\vec{N})} \qquad (25)$$

where l' is the maximum basis level set for models having $M-1$ queues, and the matrices $\mathbf{A}_{l,l'}$, $\mathbf{C}_{l,l'}$, $\mathbf{C}_{l,l'-1}$, and $\mathbf{D}_{l,l}$ include all and only coefficients of the GCEs that relate $V_l(\vec{m},\vec{N})$ with $V(\vec{m},\vec{N}-\vec{1}_R)$ and $V(\vec{m}-\vec{1}_M,\vec{N})$. The problem we consider in this section is the determination of the smallest l and l' such that the above linear system can be solved for $V(\vec{m},\vec{N})$ for arbitrary number of classes, queues, and jobs. Note that having the smallest possible basis level provides the largest computational savings in linear system solutions, since l and l' determine the cardinality of the bases being evaluated.

Theorem 1 *The basis level l such that (25) can be solved recursively using a fixed branching factor $B = 1,\ldots,M$, provided that all coefficient matrices are non-singular*[4]*, is given by*

$$l = l' = \max\{1, R-B\},$$

which is strictly lower than in the MoM basis where $l = R$. Thus, $\mathbf{C}_{l,l'-1} \equiv \mathbf{0}$ and the termination conditions for the recursion on the number of queues are known bases of models having $B-1$ queues or more.

Proof 1 The proof is by induction on the number of queues M in the model.
Case $M = B-1$. Straightforward, the bases are known.
Case $M \geq B$. We show the following properties that prove the theorem:

P1) $V(\vec{m}-\vec{1}_M,\vec{N})$ includes all constants of models with $M-1$ queues that are needed to formulate the GCEs;
P2) $V_{l-1}(\vec{m},\vec{N})$ can be obtained recursively from $V(\vec{m},\vec{N}-\vec{1}_R)$;

[4] For a model where $\mathbf{A}(\vec{m},\vec{N})$ is rectangular, the coefficient matrix is defined to be the matrix $\mathbf{A}^T(\vec{m},\vec{N})\mathbf{A}(\vec{m},\vec{N})$ accordingly with (8).

P3) if $l = \max\{1, R-B\}$, then the linear system

$$\begin{bmatrix} \mathbf{A}_{l,l'} \\ \mathbf{A}_{l,l} \end{bmatrix} V_l(\vec{m}, \vec{N}) = \begin{bmatrix} \mathbf{C}_{l,l'} V_{l'}(\vec{m} - \vec{1}_M, \vec{N}) \\ -\mathbf{A}_{l,l-1} V_{l-1}(\vec{m}, \vec{N}) \end{bmatrix} + \begin{bmatrix} \mathbf{D}_{l,l} V_l(\vec{m}, \vec{N} - \vec{1}_R) \\ \mathbf{B}_{l,l} V_l(\vec{m}, \vec{N} - \vec{1}_R) \end{bmatrix} \qquad (26)$$

used to compute $V_l(\vec{m}, \vec{N})$ in (25) has square or over-determined coefficient matrix.

Proof of property P1. We have the inductive hypothesis that $l' = \max\{1, R-B\}$, thus $V(\vec{m} - \vec{1}_M, \vec{N})$ is available from the previous recursive steps on models with $M-1$ queues. For example, suppose that queue M in a model of $V_l(\vec{m}, \vec{N})$ is removed by a GCE, then the corresponding normalizing constant is in $V_l(\vec{m} - \vec{1}_M, \vec{N})$, using the fixed branching approach it is always $l = l'$ thus

$$V_l(\vec{m} - \vec{1}_M, \vec{N}) \equiv V_{l'}(\vec{m} - \vec{1}_M, \vec{N}),$$

which proves the property.

Proof of property P2. Let us observe that, similarly to MoM, we can compute

$$V_{l-1}(\vec{m}, \vec{N}) = (\mathbf{B}_{l-1,l} V_l(\vec{m}, \vec{N} - \vec{1}_R) + \mathbf{B}_{l-1,l-1} V_{l-1}(\vec{m}, \vec{N} - \vec{1}_R))/N_R$$

provided that $N_R > 0$. If $N_R = 0$, the model has $R-1$ non-empty classes and the condition $l = \max\{1, R-B\}$ implies that $V_{l-1}(\vec{m}, \vec{N})$ is simply the union of the basis $V_l(\vec{m}, \vec{N})$ for a model with $R-1$ classes, which is available from previous recursive computations, with a vector of zeros that represent the normalizing constants $G(\vec{m} + \vec{\delta}, \vec{N} - \vec{1}_{R-1})$ not included in the basis for models with $R-1$ classes.

Proof of property P3. Using the expression of the cardinality of a basis of order l, the linear system coefficient matrix is square or over-determined if and only if

$$n_{GCE} + n_{CE} + n_{PC} \geq \binom{M+l-1}{l} R, \qquad (27)$$

where the right hand side is the number of unknowns $card(V_l(\vec{m}, \vec{N}))$, while the left hand side terms describe, respectively, the number of GCEs, CEs, and class r PCs, $r = 1, \ldots, R-1$. Note that the integer parameter l requires $l \geq 1$, otherwise it would not be possible to formulate the basis $V_{l-1}(\vec{m}, \vec{N})$. In what follows, we show that (27) is equivalent to $l \geq \max\{1, R-B\}$.

From [6], it is known that

$$n_{CE} + n_{PC} = \binom{M+l-2}{l-1} M + \binom{M+l-2}{l-1} (R-1) = \binom{M+l-2}{l-1} (M+R-1).$$

Instead, the number of GCEs is derived as follows. Without loss of generality, suppose that only the queues labeled $k = M - B + 1, \dots, M$ can be removed, thus up to a maximum of B GCEs can be formulated simultaneously for a given model. We first observe that we can formulate a GCE relating $V_l(\vec{m}, \vec{N})$ and $V(\vec{m} - \vec{1}_M, \vec{N})$ only if the queue removed among the B tagged queues has still unit multiplicity after addition of the l replicas. Consider a normalizing constant in $V_l(\vec{m}, \vec{N})$, where thus we have added l queue replicas. Define h as the number of distinct queues interested by the replica addition and call $b \leq h$ the number of queues that belong to the tagged set of cardinality B. The number of normalizing constants in $V_l(\vec{m}, \vec{N})$ for a given pair (h, b) is thus

$$\binom{l-1}{l-h}\binom{M-B}{h-b}\binom{B}{b}$$

where $M - B$ is the number of stations that cannot be removed by a GCE and the first binomial coefficient is the number of ways of distributing the $l - h$ residual replicas among the h queues selected. Hence, one concludes that the number of GCEs is

$$n_{GCE} = \sum_{h=1}^{\min(M,l)} \binom{l-1}{l-h} \sum_{b=0}^{\min(B,h)} \binom{M-B}{h-b}\binom{B}{b}(B-b)$$

being $(B - b)$ the number of queues among the B tagged queues that are left with unit multiplicity and thus where GCEs can be formulated. By Vandermonde's convolution formula [12], the above expression remarkably simplifies to

$$n_{GCE} = \binom{M+l-2}{l}B,$$

thus condition (27) becomes

$$\binom{M+l-2}{l-1}(M+R-1) + \binom{M+l-2}{l}B \geq \binom{M+l-1}{l}R,$$

that readily simplifies to

$$l(M+R-1) + (M-1)B \geq (M+l-1)R.$$

However, the last expression holds true exactly for $l \geq \max\{1, R - B\}$. $\square$

The theorem states that, given termination conditions for models having $B - 1$ queues, it is possible to solve the model by a recursion with fixed branching factor. However, it should be noted that such technique appears of practical interest only in the cases $B \leq 2$, where one can compute the basis for a model having $B - 1$ queues from the closed-form expression (22). Unfortunately, as we show in the experimental validation section, small values of B often make the generalized MoM algorithm scale quite similarly to the original MoM as M and R grow. Hence, we focus below on the analysis of the more flexible variable branching factor approach.

4.1.2 Recursion by Variable Branching Factor

In variable branching, the factor B changes while recurring on the number of queues of the model. Assume that the recursion proceeds in a bottom-up fashion, evaluating first all models with 1 queue, then all models with 2 queues, up to the original model with M queues. Denote by $d = 1, \ldots, M$ the number of queues for the currently evaluated model. A variable branching factor allows to express the number of GCEs being as a function $B \equiv B(d)$ ranging in $B(d) = 1, \ldots, d$. Using a branching factor $B(d)$ means that all models having d queues are recursively computed from models with $d-1$ queues through $B(d)$ GCEs. Without loss of generality, we assume that for a model with M queues we remove by GCEs the set of queues indexed by $M - B(M) + 1, \ldots, M-1, M$. This brings a recursive structure of the type

$$\mathbf{A}(\vec{m},\vec{N})V(\vec{m},\vec{N}) = \mathbf{B}(\vec{m},\vec{N})V(\vec{m},\vec{N}-\vec{1}_R) + \sum_{j=M-B(M)+1}^{M} \mathbf{C}(\vec{m},\vec{N})V(\vec{m}-\vec{1}_j,\vec{N}),$$

which generalizes straightforwardly to models with $d < M$ queues. The properties of the recursive structure described above are given by the following theorem.

Theorem 2 *Consider a model that is solved recursively using a variable branching factor $B(d)$, where d is the number of queues for the models evaluated in the current step of the recursion. If all coefficient matrices are non-singular, then the model can be solved recursively provided that the basis for models with d queues has levels*

$$l = \max\{1, R - B(d)\} \tag{28}$$
$$l' = \max\{1, R - B(d-1)\} \tag{29}$$

and $B(d)$ is defined such that for all values of d it is $l = l' - 1$ or $l = l'$. Termination conditions are given by (22) for models with a single queue ($d = 1$).

Proof 2 We have to evaluate for each set of models with d queues the same three properties verified in the proof of Theorem 1. It is easy to see that $P2$ and $P3$ still hold after l is replaced by $l(d)$. Property $P1$ is instead affected by the changes of the branching factor.

Proof of property P1. Consider a step of the recursion where d queues are evaluated, then the GCEs involve models having $d + l$ and $d + l - 1$ queues. This means that the normalizing constants in $V(\vec{m} - \vec{1}_j, \vec{N})$, for some values of j, should include the ones of models having $d + l - 1$ queues. However, it is sufficient that $l' = l$ or $l' = l - 1$ to achieve this result since $V(\vec{m} - \vec{1}_j, \vec{N})$ includes both $V_{l'}(\vec{m}, \vec{N})$ and $V_{l'-1}(\vec{m}, \vec{N})$. $\square$

According to the above theorem, if the basis level for models with a queue less is l or $l - 1$, then it is possible to recursively formulate the linear system of equations.

Note in particular that if $l' = l$ the linear system has $\mathbf{C}_{l,l'-1}(\vec{N}) \equiv \mathbf{0}$, otherwise it is $\mathbf{C}_{l,l'}(\vec{N}) \equiv \mathbf{0}$.

Finally, we remark that variable branching includes fixed branching as a special case, i.e., $B(d) = 1$ for $d \geq 1$.

4.2 Generalized MoM Algorithm

We begin by observing this general property of variable branching.

Corollary 1 *The variable branching factor $B(d) = d$ that adds to the linear system the maximum possible number of GCEs satisfies the assumptions of Theorem 2.*

Proof 3 We have $l = \max\{1, R-d\}$ and $l' = \max\{1, R-d+1\}$, thus we have that l and l' are either equal to 1 or they differ by a unit when $l = R-d+1$ and $l = R-d$, which proves the corollary. $\square$

Based on the last result, we define the generalized MoM algorithm using a variable branching with levels
$$B(d) = \min\{d, B_{max}\},$$
where B_{max} is a user specified maximum branching factor. This structure allows to use in the recursion all available GCEs up to a maximum of B_{max} for each model. Intuitively, this choice of $B(d)$ corresponds to the case where we use all available information of the GCEs within the constraints provided by the user on the growth of the recursion tree. In practice, the choices $B_{max} = 1$ and $B_{max} = M$ are the most relevant, since the former corresponds to minimum overhead due to the growth of the recursion tree, while the latter yields the largest reduction of the linear system size. Note also that $B_{max} = 0$ corresponds to the original MoM recursion, since in this case there are no GCEs and the basis level $l = R$ is sufficient to perform the recursion without branching into models with less queues.

Compared to other definitions of $B(d)$, the definition used in the generalized MoM allows to terminate always by the closed-form formulas (22). The structure of the generalized MoM algorithm is summarized by the following pseudo-code; furthermore, an illustration of a recursive step of the MoM algorithm is provided in the final appendix.

```
ALGORITHM MoM($B_{max}$)
$\mathbf{m}(M) = \{\vec{m}\}$
FOR $d = M-1, M-2, \ldots, 1$
    $\mathbf{m}(d) = \{$ set of models obtained by removing the last $\min\{d, B_{max}\}$ queues
              from each model in $\mathbf{m}(d+1)\}$
END FOR
```

```
FOR $r = 2, \dots, R$
   INITIALIZE new class
   FOR $n_r = 1, \dots, N_r$
   $\vec{n} = (N_1, N_2, \dots, N_{r-1}, n_r, 0, \dots, 0)$
      FOREACH $\vec{m}' \in \mathbf{m}(1)$
         COMPUTE $V(\vec{m}', \vec{N})$ by the closed-form formulas (22)
      END FOREACH
      FOR $d = 2, \dots, M$
         FOREACH $\vec{m}' \in \mathbf{m}(d)$
            SOLVE $\mathbf{A}(\vec{m}', \vec{n})V(\vec{m}', \vec{n}) = \mathbf{B}(\vec{m}', \vec{n})V(\vec{m}', \vec{n} - \vec{1}_R) + \mathbf{C}(\vec{m}', \vec{n})V(\vec{m}' - \vec{1}_M, \vec{n})$
         END FOREACH
      END FOR
   END FOR
END FOR
RETURN normalizing constants in $V(\vec{m}, \vec{N})$
END ALGORITHM
```

The generalized MoM algorithm computes the joint binomial moments in $V(\vec{m}, \vec{N})$ from which performance indexes can be easily derived [6, Sec. 4.3]. The execution of the algorithm is organized as a double recursion on the number of queues and jobs. It should be noted that the lowest computational costs are usually obtained if the iteration on the number of queues is executed as the inner loop, since in the opposite case one should keep in memory a set of normalizing constants for all N populations evaluated in the recursion. Therefore queues are added in a bottom-up fashion through the innermost loop on the variable d, up to reaching the target model with M queues. For each value of d, we consider all models $\vec{m}'$ that are analyzed due to the recursive branching specified by the $B(d)$ function. In the outermost loops on r and n_r, jobs are added progressively one class at a time. An initialization phase is required before processing a class population in order to increase the basis level if the level l increases when moving from a model with r classes to a model with $r+1$ classes. Let l be the basis level for class r and let l_* be the basis level for class $r+1$. Then, if $l_* = l$, the basis $V_l(\vec{N})$ is immediately converted into the basis $V_{l_*}(\vec{N})$ by adding zero entries for the normalizing constants $G(\vec{m} + \vec{\delta}, \vec{n} - \vec{1}_{r+1})$. Conversely, if $l_* = l + 1$ we fall in the case discussed in the proof of property *P2* in Theorem 1, where $V_l(\vec{N})$ provides $V_{l_*-1}(\vec{N})$ and the basis $V_{l_*}(\vec{N})$ is obtained from the reduced linear system

$$\begin{bmatrix} \mathbf{A}_{l_*,l'_*} \\ \mathbf{A}_{l_*,l_*} \end{bmatrix} V_l(\vec{m}, \vec{N}) = \begin{bmatrix} \mathbf{C}_{l_*,l'_*} V_{l'_*}(\vec{m} - \vec{1}_M, \vec{N}) \\ -\mathbf{A}_{l_*,l_*-1} V_{l_*-1}(\vec{m}, \vec{N}) \end{bmatrix}$$

where l'_* is the basis level used for models with $r+1$ classes and a queue less, and which uses the condition $V_l(\vec{m}, \vec{N} - \vec{1}_{r+1}) = \vec{0}$ since there are no jobs of class $r+1$.

Finally, we observe that the application of the generalized MoM approach to models with arbitrary delays Z_r requires the following simple modification. Upon considering a model with a single queue, the corresponding normalizing constant cannot be evaluated directly by (22), since this does not include the delays Z_r. In practice, such delays cannot be incorporated in the termination formula directly and an algorithmic evaluation of the normalizing constant becomes necessary. However, it is immediately found that the recursive computation of the normalizing constants with a single queue follows efficiently from (24). In particular, the order of the linear system coefficient matrix for a model with a single queue is given by

$$\binom{1+R-1}{R} R = R,$$

which is inexpensive for all models of practical interest.

4.3 Handling Singular Models

In the previous sections, we have assumed that the models being evaluated lead to a non-singular coefficient matrix in all recursive steps of the generalized MoM algorithm. Indeed, such condition is a basic requirement for the application of systems of linear equations to the solution of closed queueing networks. Indeed, as observed also in our earlier works [6–8], there exist a number of situations where it is known in advanced that the linear system will be singular. These are as follows:

- *Sparsity of service demands*. This case arises when one or more service demands are equal to zero. This leads the coefficient matrices to include column of all zeros. This situation can be corrected by removing such columns from the evaluation, an approach that leads to considering an over-determined (and thus still solvable) system of linear equations.
- *Identical queues*. If two queues are replicas of each others, it is mandatory that this is accounted for in the multiplicity vector. In fact, if two identical queues are considered explicitely assuming that for each of them the multiplicity is unitary, then the system of linear equations would include identical rows.
- *Identical demands for some workload classes*. A problem that is harder to address arises when there exist a subset of classes r such that $D_{k,r} = D_{k',r}$ for two distinct queues k and k'. This situation leads again to a singular coefficient matrix, but unfortunately does not appear possible to correct this problem within the MoM framework. A solution proposed in our earlier work, that applies immediately also to the generalized MoM is to perform the computation of the normalizing constants for a subset of classes under an hybrid algorithm that jointly uses the MoM linear system and the LBANC or MVA algorithm, see the appendix in [7] for a technical discussion on the structure of such recursion. It can be proved similarly to see [7] that such hybrid algorithm would still outperform MVA or LBANC, although its effectiveness would be reduced

compared to the application of MoM on a non-singular model with the same number of queues, jobs, and classes.

- *Other degeneracies*. Finally, there exist rather infrequent cases where the specific values of demands lead to a singular coefficient matrix. These cases can be addressed again using the hybrid algorithm approach. Alternatively, if the service demand values are obtained from measurement, a less elegant solution involves increasing the number of digits used to specify initially the service demands. This approach would clearly leads to numerical instabilities in standard linear algebra solver, however this is no longer true if exact algebra is used, as we have assumed for MoM, since the condition number does not affect the correctness of the final solution.

5 Computational Requirements

We derive analytical formulas to estimate the computational requirements of the generalized MoM algorithm. In general, time requirements may vary in real implementation due to the specific choice of the programming language, of the linear system solver, and of the technique for the exact representation of the normalizing constants. Below, we propose an analysis assuming that the linear system solver uses Gaussian elimination for dense matrices based on LU decomposition and back-substitution [18]. This case enjoys simple formulas for the number of operations used for LU operations and generally provides an upper bound to the costs of an optimized implementation using more advanced linear system solution techniques that leverage on sparsity of the matrices. To cover also the latter case, we report in Section 6 experiments with a prototype implementation based on the Wiedemann algorithm [31].

5.1 Time Requirements

The components of the time requirements of the generalized MoM algorithm of Section 4.2 are as follows:

(1) N outer loop iterations on r and n_r
(2) M inner iterations on d
(3) for each set $\mathbf{m}(d)$, the cost of evaluating $card(\mathbf{m}(d))$ models
(4) each evaluation solves a linear system of order $card(V_l(\vec{m}', \vec{N}))$
(5) overheads due to the exact representation of the normalizing constants

We now derive formulas for the last three components of the above list.

Component (3). The cardinality of $\mathbf{m}(d)$ is given by the following expression

$$card(\mathbf{m}(d)) = \binom{B(d+1)+M-d-1}{M-d},$$

which can be easily proved by induction on the recursion step $M-d$, starting from the initial condition $card(\mathbf{m}(M)) = card(\{\vec{m}\}) = 1$. The formula states that, at the step of the recursion where there are d queues, we have a number of models to evaluate that is determined by the branching level $B(d+1)$ of the step above and by the number of queues $M-d$ that cannot be removed by GCEs. For example, when $d = M-1$ we have

$$card(\mathbf{m}(d)) = \binom{\min\{M, B_{max}\}}{1} = B_{max},$$

which is exactly the number of different ways of removing up to B_{max} distinct queues from a group of $M \geq B_{max}$.

Component (4). For each model $\vec{m}' \in \mathbf{m}(d)$, while processing class r we have $l = \max\{1, r-B(d)\} = \max\{1, r-\min\{d, B_{max}\}\}$ which implies

$$card(V_l(\vec{m}', \vec{N})) = \binom{d+l-1}{d-1}r = \binom{d+\max\{1, r-\min\{d, B_{max}\}\}-1}{d-1}r.$$

Recall that for a dense matrix of order n, a linear system can be solved by Gaussian elimination in $(2/3)n^3$ operations for LU decomposition and n^2 operations for LU back-substitution [18]. Noting that the linear systems evaluated in the generalized MoM require LU decomposition only for adding the first job of a new class, while the remaining jobs one can reuse the same decomposition, it follows that evaluating the linear systems for all the population of class r requires approximately $(2/3)n^3$ at the first population for storing computing the decomposition, followed by n^2 operations at each population to solve the linear system, where $n = card(V_l(\vec{m}', \vec{N}))$. Note that this is a pessimistic estimate since it ignores sparsity of matrices.

Component (5). Similarly to [6], it can be shown that for a model with d queues the overhead due to the exact arithmetic representation of the normalizing constants can be approximated as

$$S(r, n_r, d) \approx n_{dgt} \log n_{dgt}, \qquad n_{dgt} = \left(n_r + \sum_{s=1}^{r-1} N_s\right) \log(d+l-1),$$

where $l = \max\{1, r-\min\{d, B_{max}\}\}$ is the basis level and n_{dgt} is an approximation for the number of digits in the normalizing constant. This approximation follows by considering the worst-case number of digits for a normalizing constant of a balanced queueing network where all demands are equal to $D = \max_{k,r}\{D_{k,r}, Z_r\}$, which can be easily analyzed by closed-form formulas [6].

Summary. Based on the above derivations, we conclude that the time requirements of the generalized MoM algorithm solved using exact Gaussian elimination may be approximated by the expression

$$\sum_{r=1}^{R}\sum_{d=1}^{M}\binom{B(d+1)+M-d-1}{M-d}\left(\frac{2}{3}\left(\binom{d+l-1}{d-1}r\right)^3+\sum_{n_r=0}^{N_r}\left(\binom{d+l-1}{d-1}r\right)^2 S(r,n_r,d)\right),$$

for $B_{max} \geq 1$, where the basis level is $l = \max\{1, r - \min\{d, B_{max}\}\}$. We remark that the term $n_r = 0$ accounts for the class initialization phase, which requires solution of the reduced linear system described in Section 4.2. In the special case $B_{max} = 0$, the algorithm reduces to the original MoM and the the inner summation on d does not appear in the time requirements expression. The leading term of the above expression scales as $O(N^2 \log N)$ as the total population grows.

Figure 3 illustrates the trend of the time requirements for a model with same number of queues and classes, i.e., $M = R$. This is the most challenging case for the original MoM [6]. This case also shows the maximum theoretical gap between the performance of MoM(M) and the performance of the original algorithm. The figure illustrates that MoM(0) and MoM(1) are asymptotically very similar as the number of queues and class grows. Conversely, MoM(M) appears to be much faster than the other methods starting from models with 4 or 5 classes. We have observed in the experiments reported in Section 6 that with techniques different from Gaussian elimination, namely with the Wiedemann algorithm, this can become visible from models starting from 5 or 6 classes. This is because such methods have computational requirements that are lower than in Gaussian elimination, thus the effects of a reduced basis size are initially compensated by such increased efficiency. Nevertheless, also in the experiments reported later it is evident that the generalized MoM scales much better than the original MoM.

Finally, figures 5 and 4 illustrate the growth in time requirements when $M = 3R$ or $R = 3M$. The trends confirm that MoM(M) improves over MoM(0) and MoM(1) whenever the number of classes is greater than 4 or 5. It is also found that the relative improvement becomes larger when M is much larger than R. This is a desirable property since, as shown in [7], the MoM algorithm is most effective for models with more queues than classes, while for models in which $R > M$ the CoMoM algorithm proposed in [7] is preferable.

5.2 Space Requirements

Space requirements of the generalized MoM algorithm are determined by

(1) the space requirements for storing the bases,
(2) the space requirements for solving the linear systems.

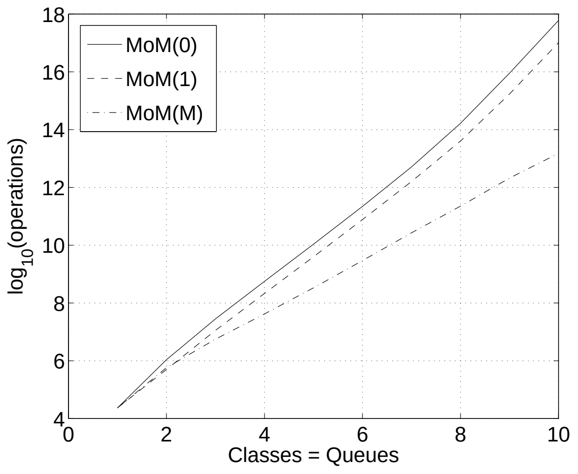

Fig. 3. Approximate time requirements of MoM(B_{max}) for $M = R$ when solved with Gaussian elimination. The total population in the network is set to $N = 100$ and balanced across classes.

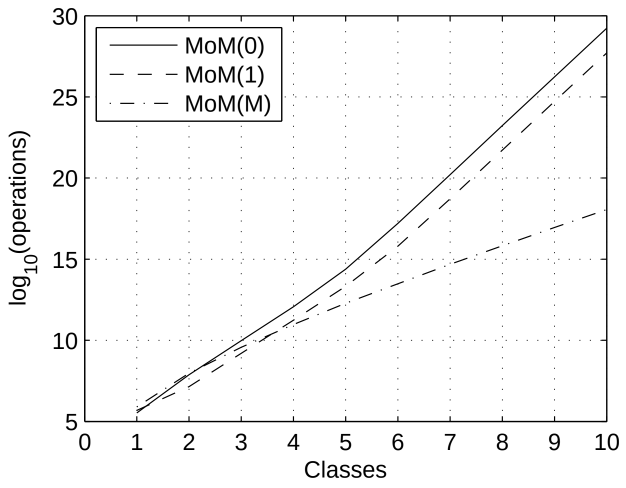

Fig. 4. Approximate time requirements of MoM(B_{max}) for $M = 3R$ when solved with Gaussian elimination. The total population in the network is set to $N = 100$ and balanced across classes.

Both components grow with the number of jobs, classes, and queues. Thus, without loss of generality, we can estimate the worst-case storage occupation of the generalized MoM at the last step of the recursion when the population vector is $\vec{n} = \vec{N}$ and $r = R$.

Component (1). This is given by

$$\sum_{d=1}^{M} 2card(\mathbf{m}(d))S(R, N_R, d)$$

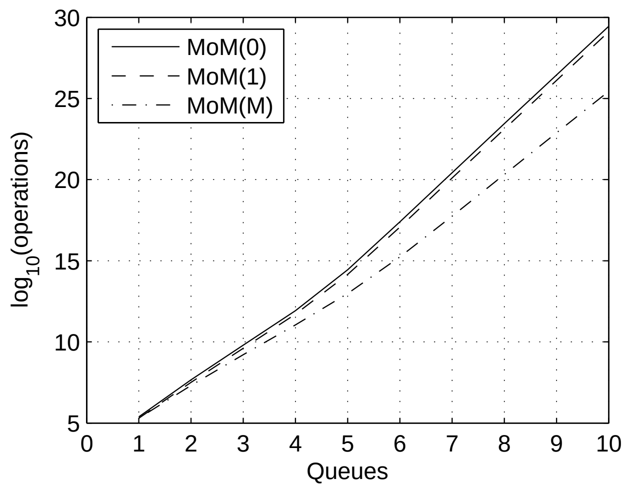

Fig. 5. Approximate time requirements of MoM(B_{max}) for $R = 3M$ when solved with Gaussian elimination. The total population in the network is set to $N = 100$ and balanced across classes.

which expands to

$$\sum_{d=1}^{M} 2\binom{B(d+1)+M-d-1}{M-d}\left(\binom{d+\max\{1,R-\min\{d,B_{max}\}\}-1}{d-1}R\right)S(R,N_R,d).$$

The factor 2 in the above expression follows from the fact that at most we need to store both $V(\vec{m},\vec{N})$ and $V(\vec{m},\vec{N}-1_R)$ in memory to perform the recursion.

Component (2). Recall that, for a dense matrix of order n, the storage requirement of LU decomposition and back-substitution is approximately n^2. Hence the maximum space requirement in the generalized MoM may be estimated

$$\max_d \left(\binom{d+\max\{1,R-\min\{d,B_{max}\}\}-1}{d-1}R\right)^2,$$

which is independent of $S(R,N_R,d)$ since the number of digits in the coefficient matrix $A(\vec{N})$ does not grow beyond range during the recursion.

Summary. The expression for the space requirements of the generalized MoM are obtained immediately by taking the maximum between components (1) and (2). Note that these yield a space complexity of $O(N\log N)$ as the total population N grows.

Figure 6 illustrates space requirements for models with $M = R$ queues and classes, assuming dense representation of the involved matrices. Similarly to Figure 3, we see that MoM(M) is superior to MoM(0) and MoM(1) also with respect to storage requirements. This is essentially due to the quadratic growth of the elements of the linear system coefficient matrix, which is very sensitive to the basis level l. Since l is minimized in MoM(M), this method is clearly more efficient than the other

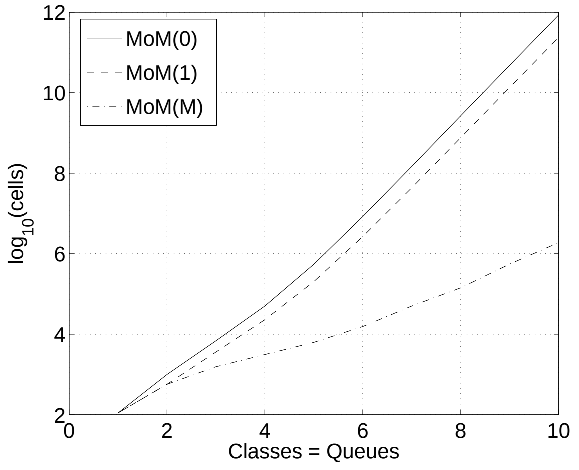

Fig. 6. Approximate space requirements of MoM(B_{max}) for $M = R$ when solved with Gaussian elimination. The total population in the network is set to $N = 100$ and balanced across classes.

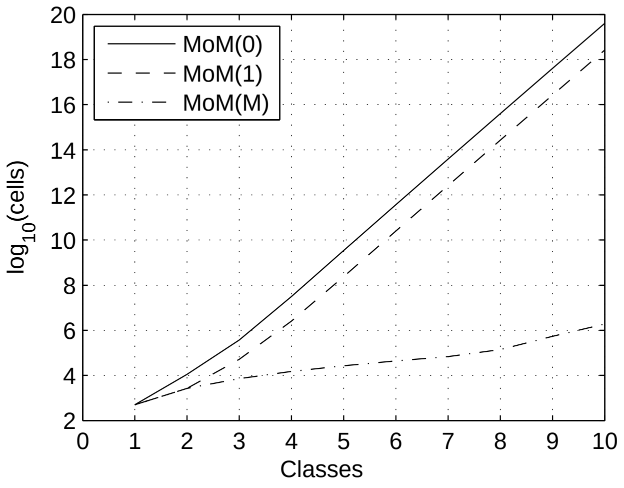

Fig. 7. Approximate space requirements of MoM(B_{max}) for $M = 3R$ when solved with Gaussian elimination. The total population in the network is set to $N = 100$ and balanced across classes.

techniques. Yet it should be observed that all techniques appear to be feasible over a large range of queueing networks, only for difficult models having more than 8 queues and classes or more MoM(M) could help in addressing inefficiencies of MoM(0) and MoM(1). Thus, we conclude that the main advantage of MoM(M) over MoM(0) and MoM(1) is the significant decrease of time requirements for models with several classes and queues.

Finally, figures 8 and 7 confirm the above observations on cases where $R = 3M$ and $M = 3R$. Similarly to the time requirements, it is found that the greatest improvements are obtained when M is much larger than R, whereas the computational costs of the three methods tend to converge when R is larger than M.

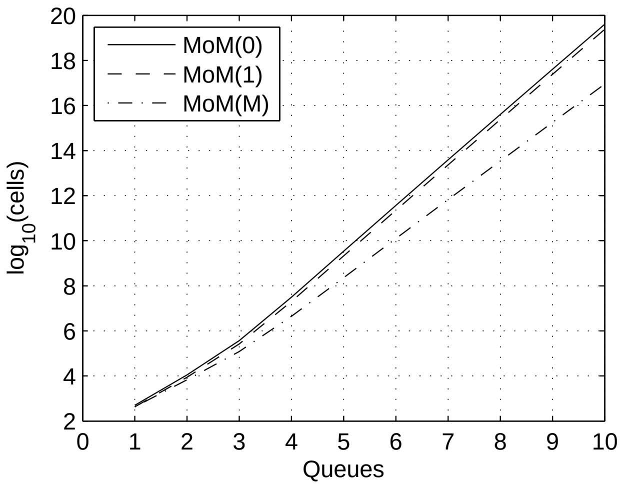

Fig. 8. Approximate space requirements of $\text{MoM}(B_{max})$ for $R = 3M$ when solved with Gaussian elimination. The total population in the network is set to $N = 100$ and balanced across classes.

6 Experimental Results

We have performed an experimental campaign to provide evidence that the generalized MoM algorithm is more efficient than the original MoM on models with several queues and classes. In particular, due to the relative efficiency of existing algorithms on models where $M >> R$ and $R >> M$, we have focused on the most difficult case where $M = R$.

We have defined a prototype implementation of the generalized MoM algorithm based on the Wiedemann algorithm solver [31] of the LinBox open source library (`http://www.linalg.org`). We have then run an experimental campaign on 36 models having different numbers of queues, classes, and jobs to illustrate the growth of computational requirements in the $\text{MoM}(B_{max})$ algorithm introduced in Section 4.2. Experiments have been run on a dual-core Intel CPU with 3GHz frequency and 4GB of RAM. Service demands are random integers in the range $[1, 50]$; think times are set to zero. For instance, let $\mathbf{D} = \{D_{k,r}\}$ be the matrix of service demands such that a column refers to a given class and a row refers to a given queue. The model used for $M = R = 7$ for all population values is as follows:

$$\mathbf{D} = \begin{bmatrix} 9 & 23 & 44 & 9 & 16 & 38 & 14 \\ 27 & 33 & 28 & 2 & 25 & 18 & 49 \\ 32 & 15 & 38 & 1 & 14 & 30 & 10 \\ 2 & 48 & 45 & 30 & 37 & 24 & 16 \\ 24 & 35 & 42 & 31 & 7 & 5 & 21 \\ 44 & 11 & 7 & 46 & 38 & 42 & 13 \\ 7 & 28 & 10 & 37 & 41 & 35 & 11 \end{bmatrix}$$

The maximum branching factor has been chosen as $B_{max} = \{0, 1, M\}$ such that

- MoM(0) is the original MoM algorithm
- MoM(1) is a fixed branching recursion generalizing the one in Figure 2(a)
- MoM(M) is a variable branching recursion generalizing the one in Figure 2(b)

Results of the proposed experimental campaign are illustrated in Table 1. The table reports the cumulative processor time for the solution of the generalized MoM linear systems (cpu) and the respective maximum memory occupation (mem). We have set a time limit for execution of the slowest algorithm of 30 minutes. Under this constraint all techniques complete models with up to 9 queues and classes, with the hardest model being for a population of $N = 100$ jobs divided homogeneously over the classes (fractional population values are rounded up to the closest integer).

Experimental results suggest the following remarks.

For the simplest models where $M = R = 3$, the original MoM is slightly more efficient than MoM(1) and MoM(M). However, computational requirements are very low for all techniques. The relative efficiency of MoM(0) may be explained by the fact that additional recursions on the number of queues are not truly advantageous in this case, since the order n of the linear system coefficient matrix is very small in all algorithms ($n \leq 30$).

Models with $M = R = 4$ illustrate cases where the MoM(1) algorithm is the most efficient. These models are intermediate cases where recurring on the number of queues can be advantageous with respect to MoM(0), provided that the recursion tree is not too large. This is because the size of the MoM(0) coefficient matrices is only $n = 140$, hence a combinatorial recursion tree has size that is comparable in magnitude and may not provide substantial savings.

Models with $M = R = 5$ show the first case where MoM(M) becomes more efficient than MoM(0) and MoM(1). It is interesting to note that MoM(0) requires about 100% more CPU time than MoM(1) and MoM(M), while the latter have

Table 1
Evaluation of the generalized MoM. Fractional class populations are rounded up.

			MoM(0)		MoM(1)		MoM(M)	
N	M	R	cpu [s]	mem [Mb]	cpu [s]	mem [Mb]	cpu [s]	mem [Mb]
10	3	3	0.01	2.4	0.02	2.4	0.04	2.4
10	4	4	0.15	2.4	0.11	2.4	0.31	2.4
10	5	5	2.28	2.7	1.01	2.4	1.01	2.4
10	6	6	28.38	4.1	15.83	3.1	3.67	2.4
10	7	7	740.75	30.8	356.63	27.3	14.15	2.4
10	8	8	LIMIT	N/A	LIMIT	N/A	186.74	2.5
10	9	9	LIMIT	N/A	LIMIT	N/A	LIMIT	N/A
50	3	3	0.31	2.4	0.34	2.4	0.91	2.4
50	4	4	1.79	2.4	1.31	2.4	1.52	2.4
50	5	5	22.02	2.8	10.0	2.7	9.19	2.4
50	6	6	393.02	4.7	166.65	3.5	31.98	2.5
50	7	7	LIMIT	N/A	LIMIT	N/A	164.39	2.5
50	8	8	LIMIT	N/A	LIMIT	N/A	1837.45	3.3
50	9	9	LIMIT	N/A	LIMIT	N/A	LIMIT	N/A
100	3	3	1.06	2.4	1.08	2.4	1.91	2.4
100	4	4	5.48	2.4	4.44	2.4	10.6	2.4
100	5	5	78.97	2.9	37.24	2.8	35.02	2.5
100	6	6	1324.19	5.4	533.02	4.2	111.87	2.5
100	7	7	LIMIT	N/A	LIMIT	N/A	512.10	2.5
100	8	8	LIMIT	N/A	LIMIT	N/A	LIMIT	N/A

very similar computational requirements. Hence, one concludes that $M = R = 5$ may be used as a practical threshold to discriminate whether MoM(1) or MoM(M) should be used for model solution.

The case $M = R = 6$ provides clear evidence of the computational gains of MoM(M) over the other techniques on the most challenging models. Irrespectively of the population size value, MoM(M) requires approximately one order of magnitude less in time requirements than MoM(0). This is because the maximum order of the MoM(0) matrices is $n = 2772$, while for MoM(M) it is just $n = 60$. MoM(1) provides substantial gains over MoM(0) since $n = 1512$, yet these gains are not competitive with the ones of MoM(M), especially for large populations. Interestingly, the ratio of cpu times of MoM(1) and MoM(M) is roughly constant as N increases.

It is easy to see that that these trends are expected to make the gap between MoM(M) and the other methods even bigger for models with more than 6 queues or classes. The case $M = R = 7$ show that MoM(M) is always efficient, whereas MoM(0)

and MoM(1) can solve within the time limits models with just ten jobs. Interestingly, $M = R = 8$ show a case where MoM(M) still scales for networks with up to $N = 50$ jobs. We have been also successfully run MoM(M) in the case $N = 100$, but the execution takes more a few hours, thus falling outside the time limit of 2000s. Finally, the case $M = R = 9$ represents a limit case where none of the technique can successfully obtain a solution. For such models, MoM(0) and MoM(1) are too memory consuming. Conversely, MoM(M) becomes quite expensive due to the combinatorial growth in the number of models to be solved at each step of the recursion, therefore the bottleneck is computational time rather than memory.

Storage requirements are instead low for all algorithms; note that the value 2.4Mb appears frequently in the results since this is a minimum memory allocation performed by the LinBox library for linear system solution. Low memory requirements are a substantial advantage of the higher-order moment approach compared to established techniques such as the multiclass MVA, where often it is a memory bottleneck to limit applicability rather than time requirements. We notice that, among the MoM techniques, the MoM(M) algorithm appears also in this case the best choice, since storage requirements did not change significantly throughout the experimental campaign.

Comparison with Convolution. To illustrate that the generalized MoM provides substantial gains over techniques that do not follow the higher-order moment approach, we have also run the Convolution algorithm on the most challenging models with $N = 100$ jobs. For models of this scale, Convolution has the lowest computational costs among all algorithms that do not follow the higher-order moments approach, for instance, it is usually orders of magnitude faster than RECAL [7]. For the models with the smallest number of classes, cpu times of Convolution are quite similar to the best between the three MoM algorithms, e.g., for $M = R = 5$ Convolution requires 34.98 seconds. The relative efficiency of Convolution is also because the population $N = 100$ is only moderately large, however on larger population sizes the method becomes intractable compared to MoM [8]. However, we have found that the storage requirements of Convolution grow very quickly as the model complexity increases. For a model with $M = R = 4$, Convolution requires 145MB, which grows to 1638MB for $M = R = 5$: this quick growth makes models with larger number of queues and classes infeasible to analyze with Convolution due to a memory bottleneck. Thus, the examples for $M = R = 6$ document a case where the generalized MoM is superior to Convolution. This makes the case that the proposed methodology improves computational analysis techniques for closed queueing network models.

7 Conclusions

In this paper, we have presented a generalization of the Method of Moments (MoM), a recently proposed algorithm for the exact analysis of multiclass queueing network models which are widely used in capacity planning of computer systems and networks [6, 7]. We have integrated in the MoM equations also the recursive formula used in the Convolution Algorithm [5,26], here called the general convolution equation (GCE). We have shown that using the GCE in MoM significantly changes the structure of its recursion leading to the evaluation of models with different number of queues, which can be solved much more efficiently than the larger models considered by MoM. As a result, the computational costs in time and space of the generalized algorithm are much smaller than the original MoM recursion. Future work will focus on a similar extension for the CoMoM algorithm presented in [7].

A MoM(M) Linear System: an Illustrative Example

This appendix illustrates the structure of the MoM(M) recursion on an example. We consider a model with the following demands: $D_{1,1} = 2, D_{1,2} = 4, D_{1,3} = 8, D_{2,1} = 5, D_{2,2} = 6, D_{2,3} = 3, D_{3,1} = 10, D_{3,2} = 3, D_{3,3} = 6$, $Z_1 = Z_2 = Z_3 = 0$, thus the model has $M = 3$ and $R = 3$ classes. We evaluate the final step in the solution of a model with population $\vec{N} = (1, 1, 1)$. The basis levels are $l = l(d) = 1$ for all d, and the intermediate models evaluated are $\mathbf{m}(1) = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$, $\mathbf{m}(2) = \{(1, 1, 0), (1, 0, 1), (0, 1, 1)\}$, and $\mathbf{m}(3) = \{(1, 1, 1)\}$. We do not report the recursion on $d = 1$ since these models have a single queue and are solved by (22). As in the proof of Theorem 1, after computing the bases $V_{l-1}(\vec{m}, \vec{N})$ using the CEs of class $r = R = 3$, the basis $V_l(\vec{m}, \vec{N})$ is obtained by the reduced linear system

$$\begin{bmatrix} \mathbf{A}_{l,l'} \\ \mathbf{A}_{l,l} \end{bmatrix} V_l(\vec{m}, \vec{N}) = \underbrace{\begin{bmatrix} \sum_{i \in \mathbf{m}(d)} \mathbf{C}^i_{l,l'} V_{l'}(\vec{m} - \vec{1}_i, \vec{N}) \\ -\mathbf{A}_{l,l-1} V_{l-1}(\vec{m}, \vec{N}) \end{bmatrix}}_{V^*} + \begin{bmatrix} \mathbf{D}_{l,l} V_l(\vec{m}, \vec{N} - \vec{1}_R) \\ \mathbf{B}_{l,l} V_l(\vec{m}, \vec{N} - \vec{1}_R) \end{bmatrix}$$

Table A.1 illustrates the above linear system for the example model. This example case is interesting for two reasons: first, it involves the bases $V_{l-1}(\vec{m}, (1, 1, 0))$, which include zero elements corresponding to normalizing constants for population $(1, 1, -1)$. Such normalizing constants are used only in the CEs of class $r = R = 3$ for computation of the bases $V_{l-1}(\vec{m}, \vec{N})$. Additionally, this case illustrates a basis level where the coefficient matrix for the model $\vec{m} = (1, 1, 1)$ is rectangular and overdetermined. This is because Theorem 1 and Theorem 2 do not assure that the coefficient matrices used in the generalized MoM recursion are square. Yet the solution of the linear system follows easily by (8).

$\left[\begin{array}{c}\mathbf{A}_{l,l'}\\ \hline \mathbf{A}_{l,l}\end{array}\right]$	$\left[\begin{array}{c}\mathbf{D}_{l,l}\\ \hline \mathbf{B}_{l,l}\end{array}\right]$	V^*	$V_l(\vec{m},\vec{N}-\vec{1}_R)$	$V_l(\vec{m},\vec{N})$
$d=3$, $\vec{m}=(1,1,1)$, $\vec{n}=(1,1,1)$				
$\left[\begin{array}{ccccccccc}1 & -10 & -3 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ 1 & -5 & -6 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1 & -10 & -3 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1 & -2 & -4 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & -5 & -6\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & -2 & -4\\ \hline 1 & -2 & -4 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1 & -5 & -6 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & -10 & -3\\ \cdot & -2 & \cdot & \cdot & -5 & \cdot & \cdot & -10 & \cdot\\ \cdot & \cdot & -4 & \cdot & \cdot & -6 & \cdot & \cdot & -3\end{array}\right]$	$\left[\begin{array}{ccccccccc}6 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ 3 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 6 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 8 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 3 & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 8 & \cdot & \cdot\\ \hline 8 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 3 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 6 & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\end{array}\right]$	$\left[\begin{array}{r}5100\\ 7176\\ 5680\\ 7830\\ 9464\\ 9720\\ 7920\\ 7920\\ 7920\\ -7920\\ -7920\end{array}\right]$	$\left[\begin{array}{c}399\\ 17\\ 0\\ 516\\ 19\\ 0\\ 530\\ 16\\ 0\end{array}\right]$	$\left[\begin{array}{r}14490\\ 525\\ 582\\ 15074\\ 466\\ 546\\ 17956\\ 454\\ 772\end{array}\right]$
$d=2$, $\vec{m}=(1,1,0)$, $\vec{n}=(1,1,1)$				
$\left[\begin{array}{cccccc}1 & -5 & -6 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1 & -2 & -4\\ \hline 1 & -2 & -4 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1 & -5 & -6\\ \cdot & -2 & \cdot & \cdot & -5 & \cdot\\ \cdot & \cdot & -4 & \cdot & \cdot & -6\end{array}\right]$	$\left[\begin{array}{cccccc}3 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 8 & \cdot & \cdot\\ \hline 8 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 3 & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\end{array}\right]$	$\left[\begin{array}{r}1536\\ 2160\\ 2156\\ 2156\\ -2156\\ -2156\end{array}\right]$	$\left[\begin{array}{c}172\\ 14\\ 0\\ 260\\ 16\\ 0\end{array}\right]$	$\left[\begin{array}{r}5100\\ 348\\ 218\\ 5680\\ 292\\ 214\end{array}\right]$
$d=2$, $\vec{m}=(1,0,1)$, $\vec{n}=(1,1,1)$				
$\left[\begin{array}{cccccc}1 & -10 & -3 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1 & -2 & -4\\ \hline 1 & -2 & -4 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1 & -10 & -3\\ \cdot & -2 & \cdot & \cdot & -10 & \cdot\\ \cdot & \cdot & -4 & \cdot & \cdot & -3\end{array}\right]$	$\left[\begin{array}{cccccc}6 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 8 & \cdot & \cdot\\ \hline 8 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 6 & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\end{array}\right]$	$\left[\begin{array}{r}1536\\ 4320\\ 3328\\ 3328\\ -3328\\ -3328\end{array}\right]$	$\left[\begin{array}{c}200\\ 11\\ 0\\ 288\\ 10\\ 0\end{array}\right]$	$\left[\begin{array}{r}7176\\ 324\\ 400\\ 9464\\ 268\\ 576\end{array}\right]$
$d=2$, $\vec{m}=(0,1,1)$, $\vec{n}=(1,1,1)$				
$\left[\begin{array}{cccccc}1 & -10 & -3 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1 & -5 & -6\\ \hline 1 & -5 & -6 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1 & -10 & -3\\ \cdot & -5 & \cdot & \cdot & -10 & \cdot\\ \cdot & \cdot & -6 & \cdot & \cdot & -3\end{array}\right]$	$\left[\begin{array}{cccccc}6 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 3 & \cdot & \cdot\\ \hline 3 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 6 & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\end{array}\right]$	$\left[\begin{array}{r}2160\\ 4320\\ 3510\\ 3510\\ -3510\\ -3510\end{array}\right]$	$\left[\begin{array}{c}390\\ 15\\ 0\\ 390\\ 12\\ 0\end{array}\right]$	$\left[\begin{array}{r}7830\\ 234\\ 330\\ 9720\\ 234\\ 510\end{array}\right]$

Table A.1
Elements of the MoM(M) recursion for an example model with $M = 3$ and $R = 3$

Acknowledgement

This work was partially supported by the Imperial College Junior Research Fellowship. A shorter, preliminary, version of this paper appeared in the Proceedings of the 6th International Conference on Quantitative Evaluation of Systems (QEST) 2009.

References

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

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

[3] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains*. Wiley and Sons, 1998.

[4] S. C. Bruell and G. Balbo. *Computational Algorithms for Closed Queueing Networks*. North-Holland, 1980.

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

[6] G. Casale. An efficient algorithm for the exact analysis of multiclass queueing networks with large population sizes. In *Proc. of joint ACM SIGMETRICS/IFIP Performance*, pages 169–180. ACM Press, 2006.

[7] G. Casale. CoMoM: Efficient class-oriented evaluation of multiclass performance models. *IEEE Trans. on Software Engineering*, 35(2):162-177, March/April 2009.

[8] G. Casale. The Method of Moments for Queueing Networks. Under review, journal extension of [6].

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

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

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

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

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

[14] A. E. Conway and E. de Souza e Silva and S. S. Lavenberg. Mean Value Analysis by Chain of Product Form Queueing Networks. *IEEE Trans. on Computers*, 38(3):432–442, 1989.

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

[16] P. J. Denning and J. P. Buzen. The operational analysis of queueing network models. *ACM Computing Surveys*, 10(3):225–261, 1978.

[17] GNU MP Bignum library. `http://gmplib.org`

[18] G. H. Golub and C. F. Van Loan. Matrix Computations (3rd ed.), Johns Hopkins, 1996.

[19] W. J. Gordon and G. F. Newell. Closed queueing systems with exponential servers. *Oper. Res.*, 15(2):254–265, 1967.

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

[21] S. Kounev and A. Buchmann. Performance modeling and evaluation of large-scale J2EE applications. In *Proc. of CMG Conference*, pages 273–283, 2003.

[22] B. A. LaMacchia and A. M. Odlyzko Solving Large Sparse Linear Systems over Finite Fields. *Proc. of CRYPTO Conf.*, 109133, 1990.

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

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

[25] D. Mitra and J. McKenna. Asymptotic expansions for closed markovian networks with state-dependent service rates. *Journal of the ACM*, 33(3):568–592, July 1985.

[26] M. Reiser and H. Kobayashi. Queueing networks with multiple closed chains: Theory and computational algorithms. *IBM J. Res. Dev.*, 19(3):283–294, 1975.

[27] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queueing networks. *Journal of the ACM*, 27(2):312–322, 1980.

[28] K. W. Ross, J. Wang. Implementation of Monte Carlo Integration for the Analysis of Product-Form Queueing Networks. *Performance Evaluation*, 273–292, May 1997.

[29] P. J. Schweitzer. Approximate analysis of multiclass closed networks of queues. In *Proc. of the Int'l Conf. on Stoch. Control and Optim.*, pages 25–29, Amsterdam, 1979.

[30] K. C. Sevcik and I. Mitrani. The Distribution of Queuing Network States at Input and Output Instants. *Journal of the ACM*, 28(2):358–371, 1981.

[31] D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32:54–62, 1986.