An Application of Exact Linear Algebra to Capacity Planning Models

Giuliano Casale
College of William and Mary
Department of Computer Science
140 McGlothlin-Street Hall
23187-8795 Williamsburg, Virginia
email: casale@cs.wm.edu

1 Introduction

This note illustrates a novel application of exact linear algebra to performance evaluation and stochastic modeling. We focus on queueing network models, which are high-level abstractions of Markov chains used in capacity planning of computer and communication systems [6, 7]. Until recently, it was prohibitively expensive to compute exact solutions for these models when they describe hundreds or thousands of users interacting with a network of servers, a case of large practical application when sizing web architectures. Here, we overview a new approach, which we have recently proposed [3, 4], that overcomes this limitation by means of a linear matrix difference equation that strictly requires exact linear algebra to be evaluated.

Exact linear algebra is required in our method because of uncontrollable numerical instabilities that arise if round-off errors are introduced in the recursive evaluation of the matrix difference equation. This difficulty is also exasperated by the "astronomical" growth of the number of digits of the operands, which can be as large as $10^{1000} - 10^{10000}$. The application presented in this paper shows a case where accepting the computational costs of exact algebra to stabilize the numerical evaluation leads to massive computational savings compared to established approaches based on standard (inexact) linear algebra.

The remainder of this work is as follows. In Section 2, we give minimal background about queueing network models and explain how they can be solved recursively. We point to textbooks such as [7] for extensive background on queueing networks. In Section 3, we discuss the new solution approach based on a matrix difference equation. The numerical properties of the method are discussed in Section 3.1, where we argue that exact algebra is the only viable approach to prevent numerical instability. For the reader interested in experimenting with the problem, we report an example in Section 3.2. Finally, we draw conclusions in Section 4. Additional material, including a MAPLE implementation of the matrix difference equation approach to queueing networks, can be obtained by contacting the author or from his homepage.

2 Recursive Evaluation of Queueing Network Models

A queueing network model is a Markov chain in which a state describes the position of a population of requests across a network of servers that process them. We here focus on closed multiclass models [1], in which a constant population of $N \in \mathbb{N}$ users is partitioned into classes according to the characteristics of the requests they submit. Each class is composed by N_r users, $\sum_r N_r = N$, $N_r \in \mathbb{N}$. Solving closed queueing networks amounts to computing a certain *normalizing constant* G which enables the evaluation by closed-form formulas of all model properties, see [1] for additional information. This normalizing constant G is too expensive to be computed directly, but it can be described recursively in terms of difference equations [2, 5, 8]. In fact, consider a model with M servers and where users are partitioned into R classes, define the function $g(\vec{n}, \vec{m})$ as the solution of any of the following difference

equations:

$$g(\vec{n},\vec{m}) = g(\vec{n},\vec{m}-1_i) + \sum_{r=1}^{R} D_{i,r}\, g(\vec{n}-1_r,\vec{m}), \qquad 1 \le i \le M, \tag{1}$$

$$n_r g(\vec{n},\vec{m}) = Z_r g(\vec{n}-1_r,\vec{m}) + \sum_{k=1}^{M} m_k D_{k,r}\, g(\vec{n}-1_r,\vec{m}+1_k), \qquad 1 \le r \le R. \tag{2}$$

where $\vec{n} = (n_1,\ldots,n_r,\ldots,n_R)$ and $\vec{m} = (m_1,\ldots,m_k,\ldots,m_M)$ are two vectors of nonnegative integers, $D_{i,r} \in \mathbb{R}$ and $Z_r \in \mathbb{R}$ are known model parameters [1], 1_l indicates a vector composed by all 0's except for a one in the l-th position, and where (1)-(2) are subject to two termination conditions: (i) $g(\vec{n},\vec{m}) = 0$ if any entry in $\vec{n}$ or $\vec{m}$ is negative; (ii) $g(\vec{0},\vec{0}) = 1$, $\vec{0} = (0,0,\ldots,0)$. Then, it is possible to show that the normalizing constant G is immediately available if one is able to evaluate $g(\vec{n},\vec{m})$ for $\vec{n} = (N_1,N_2,\ldots,N_R)$ and $\vec{m} = (1,1,\ldots,1)$. This value $g((N_1,N_2,\ldots,N_R),(1,1,\ldots,1))$ is equal to the normalizing constant G.

From a computational standpoint, the $g(\vec{n},\vec{m})$ values can be obtained by recursively evaluating one between (1) and (2) until termination conditions are met. Classic solution algorithms proposed in the literature [2, 5, 8] follow this approach, which implies time and space requirements that grow roughly as $O(N^R)$ if (1) is used and as $O(N^M)$ if (2) is used. In practice, these costs are prohibitive since it is not difficult to have N of the order of hundreds or thousands and $\min\{M,R\} \ge 5-6$, which makes the storage requirement of the order of hundreds of gigabytes regardless of the recursion used. However, until recently, this type of recursive evaluation was the only one known to compute the normalizing constant, which therefore remained infeasible to evaluate on models with the aforementioned characteristics. In the next section, we overview the new recursive solution methods we have recently proposed in [3,4] which use *both* (1) and (2) simultaneously and which leverage on exact linear algebra for numerical computations.

3 Solution Based on Matrix Difference Equations

When solving (1) or (2) in isolation, the summations make the recursion tree branch exponentially with R and M, respectively. We have recently observed in [3] that if one considers a certain subset of values $\vec{V}(\vec{n})$, called *basis*, of the function $g(\vec{n},\vec{m})$, then this basis can be computed recursively by a matrix difference equation defined by jointly using (1) and (2) into a linear system

$$\mathbf{A}(\vec{n})\vec{V}(\vec{n}) = \mathbf{B}(\vec{n})\vec{V}(\vec{n}-1_r), \tag{3}$$

for any $1 \le r \le R$, where $\vec{V}(\vec{0})$ is known from the termination conditions of (1)-(2), and the matrices $\mathbf{A}(\vec{n})$ and $\mathbf{B}(\vec{n})$ are square of identical size. The matrices $\mathbf{A}(\vec{n})$ and $\mathbf{B}(\vec{n})$ are defined by the coefficients of the equations (1)-(2) that relate all and only the values of $g(\vec{n},\vec{m})$ in $\vec{V}(\vec{n})$ with those in $\vec{V}(\vec{n}-1_r)$. The interest for (3) is that the matrix recursion is linear and does not branch exponentially like (1)-(2), since we can progressively remove the elements from the vector $\vec{n}$ until the termination condition $\vec{V}(\vec{0})$ is reached. Note that the vectors $\vec{V}(\vec{n})$ and $\vec{V}(\vec{n}-1_r)$ can be formulated to have constant size during the matrix recursion, and that the knowledge of $\vec{V}(N_1,\ldots,N_R)$ is sufficient to compute the normalizing constant G. The matrix $\mathbf{A}(\vec{n})$ is usually invertible and admits a fine-grained block triangular form decomposition that greatly reduces the computational costs of solving the linear system (3). In addition, we have recently found in [4] that $\mathbf{A}(\vec{n})$ can be redefined in a way that admits a very different block triangular form compared to the one found in [3]; in the rest of this paper, we refer to (3) meaning the formulation in [3]. We discuss in Section 3.1 computational costs and numerical solution of (3) and we provide in Section 3.2 an example that illustrates (3) based on the methodology in [3].

3.1 Numerical Evaluation

We argue that the numerical evaluation of (3) requires exact linear algebra. In fact, the recursive solution of (3) is subject to two numerical issues: (i) floating-point range exceptions of the values of $g(\vec{n},\vec{m})$ which can be as large as $10^{1000} - 10^{10000}$, see [3] for an example; (ii) round-off error accumulation while recurring hundreds or thousands of times[2] on the sequence of linear systems (3). In line of principle, floating-point range exceptions may be addressed

[1] In a queueing network model, $D_{i,r}$ is the service demand of class r users at server i expressed in seconds, and Z_r is the time before submission of two requests by a class-r user.

[2] In the methodology we propose, the target $\vec{n}$ in (3) is set equal to $(N_1,N_2,\ldots,N_R)$ which means that the recursion terminates after N steps, where N is the total number of users in the queueing network model. Typical values of N are hundreds or thousands.

Table 1: *Computational costs of the classic MVA solution algorithm [9] versus (3) solved by exact LU decomposition using the GNU GMP 4.2.1 mpq_∗ library.*

$M = 3$, $R = 3$	*cpu*	*mem*	*cpu*	*mem*	*cpu*	*mem*	*cpu*	*mem*	*cpu*	*mem*
$N =$	10 users		100 users		1000 users		10000 users		100000 users	
MVA	0.04s	20MB	4.68s	180MB	144.6s	950MB	*mem limit*	$>$ 1GB	*mem limit*	$>$ 1GB
Eq.(3) + GMP	0.03s	5MB	0.24s	10MB	0.57s	15MB	9.1s	30MB	35.4s	70MB

using extended precision arithmetic. However, we have observed that the error accumulation problem in the solution of (3) remains uncontrollable with inexact algebra. The instability is visible also if each individual linear system (3) has small condition number; the error accumulation can only be slowed, but not avoided, if extended precision arithmetic is used. For instance, in [4] we report experiments that show critical error accumulations with Gaussian elimination and iterative algorithms such as CG, BiCG, GMRES, and QMR implemented in 32-bit and 128-bit arithmetic. We have found that significant error accumulation starts as soon as the operands exceed the floating-point range, which usually happens after a few tens of iterations.

After extensive experimentation, we have not found approaches to reliably solve (3) other than exact linear algebra. Instead, we have correctly solved (3) using either exact Gaussian elimination implemented with the GNU GMP libraries or the Wiedemann algorithm in the LinBox library implementation. In particular, we have found that if one accepts the costs of an exact solution, then (3) can be solved reliably in roughly $O(N^2 \log N)$ time and $O(N \log N)$ space, compared to the much larger $O(N^M)$ or $O(N^R)$ in both time and space of the best-available methods. To make the point, we are now able to solve models with $R = 7$ classes, $M = 5$ servers, and $N = 10,000$ users in about five minutes with a memory occupation of 456MBs; the best-existing algorithm would require instead 10^{10} GBs of memory and similarly prohibitive computational time. This makes a strong point in favor of adopting exact linear algebra and provides a useful example on the benefits in performance modeling of exact linear algebra, which has enabled the accurate solution of the key equation (3) that would instead be of little practical value with standard inexact linear algebra solvers.

As an additional example of the positive impact of the numerical stabilization, Table 1 compares (3) with the classic Mean Value Analysis (MVA) algorithm for closed queueing networks [9] on a model with three servers, three classes, and increasing number of users. The results are obtained on a Xeon 2.80GHz machine with 1GB of available RAM and using an optimized C implementation that runs on a single processor. The linear system (3) is solved using block Gaussian elimination applied to the block triangular form given in [3]. Indeed, the impact of multi-precision overheads due to the rational multi-precision libraries (GNU GMP 4.1.2) is negligible with respect to the strong computational gains of (3) over MVA. This further stresses the importance of exact linear algebra for numerical evaluation of closed queueing network models.

3.2 Example

We conclude the paper by illustrating the structure of (3) on a small queueing network with $M = 2$ servers, $R = 2$ classes, a population of users (N_1, N_2), and where $m_k = 1$, $1 \leq k \leq M$. To simplify, we start from an intermediate step of the recursion where we know $\vec{V}(N_1, 0)$ and we need to compute $\vec{V}(N_1, N_2)$ by progressively computing with (3) the vectors $\vec{V}(N_1, 1)$, $\vec{V}(N_1, 2)$, …, $\vec{V}(N_1, n_2)$, …, $\vec{V}(N_1, N_2 - 1)$, $V(N_1, N_2)$. In this case, only few elements of $\mathbf{A}(\vec{n})$ and $\mathbf{B}(\vec{n})$ change at each step and letting $t_{z,k,s} = (m_k + z) \cdot D_{k,s}$ we can write the two matrices as

$$\mathbf{A}(N_1,n_2) \equiv \begin{bmatrix}
1 & -D_{1,1} & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & -D_{1,1} & \cdot & \cdot & \cdot & \cdot & -1 & \cdot \\
\cdot & \cdot & 1 & -D_{2,1} & \cdot & \cdot & -1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & -D_{2,1} & \cdot & \cdot & -1 & \cdot \\
\cdot & -t_{1,1,1} & \cdot & -t_{0,2,1} & \cdot & \cdot & N_1 & -Z_1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & -t_{0,1,1} & \cdot & -t_{1,2,1} & \cdot & \cdot & N_1 & -Z_1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & n_2 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & n_2 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & n_2 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & n_2
\end{bmatrix}, \quad \mathbf{B}(N_1,n_2) \equiv \begin{bmatrix}
D_{1,2} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & D_{1,2} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & D_{2,2} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & D_{2,2} & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
t_{1,1,2} & \cdot & t_{0,2,2} & \cdot & \cdot & \cdot & Z_2 & \cdot & \cdot & \cdot \\
\cdot & t_{1,1,2} & \cdot & t_{0,2,2} & \cdot & \cdot & \cdot & Z_2 & \cdot & \cdot \\
\cdot & \cdot & t_{0,1,2} & \cdot & t_{1,2,2} & \cdot & \cdot & \cdot & Z_2 & \cdot \\
\cdot & \cdot & \cdot & t_{0,1,2} & \cdot & t_{1,2,2} & \cdot & \cdot & \cdot & Z_2
\end{bmatrix}$$

which setting, e.g., $D_{1,1} = 10$, $D_{1,2} = 5$, $D_{2,1} = 4$, $D_{2,2} = 9$, $Z_1 = 0$, $Z_2 = 0$, $N_1 = 3$, are computed as

$$\mathbf{A}(3,n_2) \equiv \begin{bmatrix} 1 & -10 & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & -10 & \cdot & \cdot & \cdot & \cdot & -1 & \cdot \\ \cdot & \cdot & 1 & -4 & \cdot & \cdot & -1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & -4 & \cdot & \cdot & -1 & \cdot \\ \cdot & -20 & \cdot & -4 & \cdot & \cdot & 3 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & -10 & \cdot & -8 & \cdot & \cdot & 3 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & n_2 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & n_2 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & n_2 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & n_2 \end{bmatrix}, \quad \mathbf{B}(3,n_2) \equiv \begin{bmatrix} 5 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 5 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 9 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 10 & \cdot & 9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 10 & \cdot & 9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 5 & \cdot & 18 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 5 & \cdot & 18 & \cdot & \cdot & \cdot & \cdot \end{bmatrix}.$$

Here, the variable n_2 is incremented throughout the recursion starting from $n_2 = 1$. Note that, in this example, $\mathbf{B}(N_1, n_2)$ is independent of both N_1 and n_2. Furthermore, the intermediate vector we start from to apply (3) is

$$\vec{V}(N_1, 0) = \left[12944,\ 736,\ 7616,\ 508,\ 3800,\ 316,\ 5584,\ 396,\ 2536,\ 228\right]^T.$$

Using (3) for $n_2 = 1$, we immediately compute

$$\vec{V}(N_1, 1) = \mathbf{A}^{-1}(3,1)\mathbf{B}(3,1) = \left[519024,\ 25632,\ 347840,\ 20328,\ 198760,\ 14520,\ 197984,\ 11932,\ 106480,\ 8228\right]^T,$$

and similarly for $n_2 = 2$ we find

$$\vec{V}(N_1, 2) = \left[12031680,\ 527616,\ 9219840,\ 482220,\ 6023840,\ 394140,\ 4160400,\ 219636,\ 2658440,\ 181500\right]^T.$$

These are correct evaluations of the entries of the $V(N_1, n_2)$ vector. However, working with inexact linear algebra, around $n_2 = 15$ the entries of the $V(N_1, n_2)$ vector cannot be represented exactly anymore and error accumulation starts. For instance, for $n_2 = 100$ the maximum relative error is $6.92 \cdot 10^{-4}$, for $n_2 = 120$ it grows to $1.18 \cdot 10^{-1}$, and for $n_2 \geq 140$ negative entries appear in $V(N_1, n_2)$ although it is possible to prove that all elements of the vector must be always positive. Indeed, exact linear algebra is able to address these numerical instabilities and provide correct results in all the above cases.

4 Conclusions

In this abstract, we have reported a recent development in queueing theory which benefits of exact linear algebra. The algorithm is based on the matrix difference equation (3) that cannot be solved reliably by inexact linear solvers. As exemplified by the numerical results in Table 1, the approach made viable by exact linear algebra achieves remarkable computational gains with respect to existing schemes. Clearly, new research results that reduce the requirements of exact linear algebra algorithms will continue to benefit queueing network modeling by reducing the computational costs of solving (3). In addition, further investigations regarding the nature of the numerical instability of the matrix difference equation (3) are needed and breakthroughs in the numerical stabilization of (3) would be of high practical importance.

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

[3] 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.

[4] G. Casale. CoMoM: Efficient Class-Oriented Evaluation of Multiclass Performance Models. *IEEE Transactions on Software Engineering*, 14 pages, to appear in 2009.

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

[6] S. S. Lavenberg. A perspective on queueing models of computer performance. *Performance Evaluation*, 10(1):53–76, 1989.

[7] D. Menasce and V. A. F. Almeida. *Capacity Planning for Web Performance: Metrics, Models, and Methods*. Prentice Hall, 1998.

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

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