MGMtool: A Performance Modelling Tool Based on Matrix Geometric Techniques

Boudewijn R. Haverkort, Aad P.A. van Moorsel, Arvid Dijkstra

University of Twente, Tele-Informatics and Open Systems
P.O. Box 217, 7500 AE Enschede, the Netherlands
tel: +31 53 894018, fax: +31 53 333815, e-mail: haverkor@cs.utwente.nl

Abstract

Over the last two decades a considerable amount of effort has been put in the development and application of matrix geometric techniques for the analysis of queueing systems of which the (embedded) Markov chain exhibits a regular structure. Most of this work however has been presented in either a mathematical context or in a purely application oriented context.

In this paper we present MGMtool, a performance analysis tool which allows its users to easily specify queueing systems in terms of interarrival and service time distributions. MGMtool then takes care of the translation of this description to an underlying Markov chain that exhibits a matrix geometric solution. Subsequently, MGMtool takes care of the derivation of the measures specified by the user. A particular measure of interest that can be derived is the caudal curve. This measure gives insight in the "tail behaviour" of the queue.

With MGMtool models are specified via C procedure calls. This turns out to be a very flexible approach for modelling queueing systems; it also allows for the easy evaluation of models over parameter ranges.

We present the underlying mathematics of MGMtool and various architectural and implementation issues. In the examples we show how MGMtool can be used for modelling Markov modulated Poisson processes and various other interarrival and service time distributions.

1 Introduction

Over the last two decades a considerable amount of research has been done in the field of queueing system analysis. For obtaining the average behaviour of queueing systems many closed-form solutions have been obtained, both for single queueing stations and for networks of them (see e.g. [10, 15, 18]). However, these solutions generally only apply under a number of restrictions which in practice often do not hold. Especially when modern day communication systems are analyzed, the usual Poisson arrivals and exponential packet lengths are no longer valid assumptions.

Also when one is interested in other than average behaviour of the queueing systems, analytical techniques often do not apply. For instance, when the interest is in distributions of e.g. waiting times, queue lengths etc., few results are available. Moreover, whenever they are available they often come as a Laplace-Stieltjes transform, which is not very insightfull for practioneers, nor are these solutions easily transformed in (stable) computer algorithms [12].

Some of the above mentioned shortcomings are solved by using numerical techniques or simulation. In the former case, often a Markov chain representation underlying the queueing system is constructed, which is subsequently solved numerically, generally by an iterative procedure.

This brute force method has proven to be very powerful [17], but still has his disadvantages. Among others we mention the required finiteness of the Markov chain and the difficulty of acquiring a good insight in the behaviour of the system by the lack of convenient functional relations between system parameters and performance metrics of interest. Simulation of course is always applicable, however, it suffers the drawback that it is rather costly, especially for the more intricate systems and scenarios we want to analyze today.

For a wide class of systems that do not conform the requirements for a "classical" closed form solution, *matrix geometric methods* might still be applicable and also lead to closed form solutions [12, 14]. Roughly spoken, for queueing systems that have an underlying (embedded) Markov chain that can be seen as a two dimensional generalization of an elementary M|G|1 or a G|M|1 queue, matrix geometric methods apply. Moreover, these methods are mathematically very elegant, insightful, and allow for a computationally efficient implementation.

Up till now, matrix geometric methods have not been used very widely for practical modelling studies. This is partly due to the fact that the mathematics behind the method is of an advanced level, but maybe even more to the fact that only a very limited amount of software tools is available that support the use of these techniques. Moreover, we are only aware of one tool for matrix geometric methods that has been used for real modelling studies, namely the tool MAGIC developed by Squillante [16]. To really exploit the potential of matrix geometric techniques, it is needed that more modelling tools support these techniques.

In the areas of ATM switching and workload modelling there seems to be an awareness of the potential of matrix geometric methods. We are aware of a number of modelling studies using these methods (see e.g. [1, 3]), however, all of them by using specialized, application oriented algorithms, instead of a general tool.

In this paper we present MGMtool, a tool that can be used to easily specify and solve queueing models that fall in the category that can be solved by matrix geometric techniques. MGMtool makes extensive use of the language C[1] and is, as we will see, easily extendible with new user-interface components and with new algorithms. We have tried to require the user only to input system characteristics and to hide the mathematics underlying the method as much as possible. Of course, whenever needed, it is possible to control or even change the underlying solution engine with little effort.

This paper is organized as follows. In Section 2 we discuss the basic mathematics needed for understanding the methods implemented in MGMtool. The architecture of MGMtool is described in Section 3. The modelling primitives available for the user are presented in Section 4. Implementation aspects and an application are discussed in Section 5 and 6 respectively. Section 7 concludes the paper.

2 MGMtool — Underlying mathematics

In this section we briefly discuss the mathematical aspects of matrix geometric techniques. We do not give an in-depth survey here. We therefore refer to the original work by Neuts [12, 14] or to the recent mathematical tutorial by Nelson [11].

We introduce scalar state processes in Section 2.1. After having introduced phase type distributions in Section 2.2, we generalize scalar state processes to vector state processes in Section 2.3, in which we explain the matrix geometric method along the lines of a PH|PH|1 queueing system. We discuss the derivation of the matrix $\mathbf{R}$, which is of key importance to matrix geometric methods, in Section 2.4. Standard measures that can be obtained are discussed

[1] A similar use of C can be observed in the SPNP package [4].

in Section 2.5. Finally, in Section 2.6 we elaborate on the caudal curve, a measure specific to matrix geometric methods.

2.1 Scalar state processes

Consider an M|M|1 queueing model with arrival rate λ and service rate μ. The Markov chain underlying this queueing model is a simple birth-death process where the state variable denotes the total number of packets in the queue and server. Since the state variable is a scalar, we call such a process a *scalar state process*. The steady state probability distribution of the number of packets N in the queue has the following form:

$$z_i = \Pr\{N = i\} = z_0 \rho^i, \quad i = 1, 2, \cdots, \text{ and } z_0 = 1 - \rho, \tag{1}$$

with $\rho = \lambda/\mu$, the utilisation or traffic intensity of the queue. For stability we require $0 \leq \rho < 1$. We observe that the number of packets N in the queue satisfies a *geometric distribution*. From the distribution $z_i,\ i = 0, 1, \cdots$, we can derive all kinds of interesting performance metrics, such as

- the average number of packets in the queue: $E[N] = \sum_{i=0}^{\infty} i z_i = \rho/(1-\rho)$;
- the average response time (via Little's law): $E[R] = E[N]/\lambda = E[S]/(1-\rho)$, with the average service time $E[S] = 1/\mu$;
- the probability of at least j packets in the queue: $B_j = \sum_{i=j}^{\infty} z_i$.

The generator matrix $\mathbf{Q}$ of the Markov chain underlying the M|M|1 queue has the following form:

$$\mathbf{Q} = \begin{pmatrix} -\lambda & \lambda & 0 & \cdots & \cdots & \cdots \\ \mu & -(\lambda+\mu) & \lambda & 0 & \cdots & \cdots \\ 0 & \mu & -(\lambda+\mu) & \lambda & 0 & \cdots \\ \vdots & 0 & \mu & -(\lambda+\mu) & \lambda & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{2}$$

We observe that, apart from the first column, all the columns are basically the same, except that "they are shifted down one row". We call the first column a *boundary column* and the other ones *repeating columns*. The fact that the steady state distribution of the number of packets N in the M|M|1 queue exhibits a geometric solution has turned out to be intimately related to the regular structure of the matrix $\mathbf{Q}$.

2.2 Phase type distributions

Consider a continuous time Markov chain on the state space $\{1, \cdots, m, m+1\}$ with generator matrix

$$\mathbf{Q} = \begin{pmatrix} \mathbf{T} & \underline{T}^0 \\ \underline{0} & 0 \end{pmatrix}, \tag{3}$$

where size$(\mathbf{T}) = m \times m$, $T_{ii} < 0$ $(i = 1, \cdots, m)$, and $T_{ij} \geq 0$ $(i \neq j)$. The row sums of $\mathbf{Q}$ equal zero, i.e. $\mathbf{T}\underline{1} + \underline{T}^0 = \underline{0}$. The initial probability vector is given as $(\underline{\alpha}, \alpha_{m+1})$ with $\underline{\alpha}\,\underline{1} + \alpha_{m+1} = 1$. Furthermore, assume that the states $1, \cdots, m$ are transient. Consequently, state $m+1$ is the one and only absorbing state, regardless of the initial probability vector. In Figure 1 we visualize this. The probability distribution $F(x)$ of the time until absorption in state $m+1$ is given by

$$F(x) = 1 - \underline{\alpha} e^{\mathbf{T}x} \underline{1}, \quad x \geq 0. \tag{4}$$

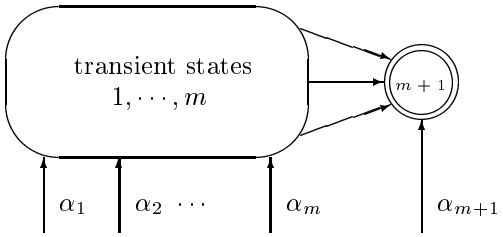

Figure 1: Schematic view of a PH distribution

We define a distribution $F(x)$ on $[0, \infty)$ to be of phase type (PH) if and only if it is the distribution of the time to absorption in a CTMC as defined above. The pair $(\underline{\alpha}, \mathbf{T})$ is called *a representation of* $F(x)$. Then, the following properties hold:

- The distribution $F(x)$ has a jump of α_{m+1} at $x = 0$ and its density on $(0, \infty)$ equals $f(x) = F'(x) = \underline{\alpha} e^{\mathbf{T}x} \underline{T}^0$.
- The moments μ_i of $F(x)$ are finite and given by $\mu_i = (-1)^i i! (\underline{\alpha} \mathbf{T}^{-i} \underline{1})$, $i = 0, 1, \cdots$.

Examples of PH distributions are the exponential distribution, the Erlang distribution, and the hyper- and hypoexponential distribution. Important to note is the fact that these well-known PH distributions are acyclic. The definition above however, allows also for cyclic Markov chains. We will see an example of such a PH distribution in Section 6.

2.3 Vector state processes

Now, consider the PH|PH|1 queue, i.e. a generalization of an M|M|1 queueing model in which both the service and the interarrival times have a phase type distribution. The state of the underlying Markov chain is not totally described by the number of packets in the queueing system. Part of the state description now is the phase of the arrival process, and the service phase of the packet in service, if any. Consequently, the state is a vector of three elements. The underlying Markov chain is therefore referred to as a *vector state process.*

Suppose that the representation of the arrival process is given by $(\underline{\alpha}, \mathbf{T})$ with size$(\mathbf{T}) = m_a$, and that the representation of the service process is given by $(\underline{\beta}, \mathbf{S})$ with size$(\mathbf{S}) = m_s$. Ordered lexicographically, the states of the Markov chain underlying this PH|PH|1 queueing system are of the form (n, a, s) where n is the number of packets in the queue, a the phase of the arrival process ($a = 1, \cdots, m_a$), and s the phase of the service process ($s = 1, \cdots, m_s$):

$$(0,1,0), (0,2,0), \cdots, (0,m_a,0), (1,1,1), \cdots, (1,1,m_s), (1,2,1), \cdots, (1,2,m_s),$$
$$\cdots, (1,m_a,1), (1,m_a,2), \cdots, (1,m_a,m_s), \cdots, (2,m_a,m_s), \cdots, (\infty,m_a,m_s).$$

All the states that have the same number of packets in the queue are said to belong to one *level.* The level is indicated by its corresponding number of packets in the queue. For example, level i, $i = 1, 2, \cdots$, corresponds to the set of states

$$\{(i,1,1), \cdots, (i,1,m_s), (i,2,1), \cdots, (i,2,m_s), \cdots, (i,m_a,1), \cdots, (i,m_a,m_s)\}.$$

Level 0 consists of the states $\{(0,1,0), (0,2,0), \cdots, (0,m_a,0)\}$. The generator matrix $\mathbf{Q}$ of the

underlying Markov chain has the following structure:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{B}_{00} & \mathbf{B}_{01} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{B}_{10} & \mathbf{A}_1 & \mathbf{A}_0 & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{A}_2 & \mathbf{A}_1 & \mathbf{A}_0 & \cdots \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_2 & \mathbf{A}_1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{5}$$

The square matrices $\mathbf{A}_i$ are of size $m_a m_s \times m_a m_s$ and have the following form [12]:

$$\begin{aligned} \mathbf{A}_0 &= \underline{T}^0 \underline{\alpha} \otimes \mathbf{I}, \\ \mathbf{A}_1 &= (\mathbf{T} \otimes \mathbf{I}) + (\mathbf{I} \otimes \mathbf{S}), \\ \mathbf{A}_2 &= \mathbf{I} \otimes \underline{S}^0 \underline{\beta}. \end{aligned} \tag{6}$$

The matrices $\mathbf{B}_{ij}$ are all differently dimensioned and have the following form:

$$\begin{aligned} \mathbf{B}_{00} &= \mathbf{T} \text{ (size } m_a \times m_a), \\ \mathbf{B}_{01} &= \underline{T}^0 \underline{\alpha} \otimes \underline{\beta} \text{ (size } m_a \times m_a m_s), \\ \mathbf{B}_{10} &= \mathbf{I} \otimes \underline{S}^0 \text{ (size } m_a m_s \times m_a). \end{aligned} \tag{7}$$

Note that the binary operator $\otimes$ is used to represents the tensor or Kronecker product of two matrices. Let $\underline{z}_i$ denote the vector of lexicographically ordered steady state probabilities of states in level i, i.e. for $i = 1, 2, \cdots$, we have

$$\underline{z}_i = (\pi_{(i,1,1)}, \cdots, \pi_{(i,1,m_s)}, \pi_{(i,2,1)}, \cdots, \pi_{(i,2,m_s)}, \cdots, \pi_{(i,m_a,1)}, \cdots, \pi_{(i,m_a,m_s)}). \tag{8}$$

It can now be shown that the steady state probability vectors $\underline{z}_i,\ i \geq 1$ have the form

$$\underline{z}_i = \underline{z}_1 \mathbf{R}^{i-1},\ i \geq 1. \tag{9}$$

The length of the vectors $\underline{z}_i,\ i = 1, 2, \cdots$, is $m_a m_s$. The square matrix $\mathbf{R}$ of size $m_a m_s \times m_a m_s$ follows from the matrix quadratic equation

$$\sum_{i=0}^{\infty} \mathbf{R}^i \mathbf{A}_i = \mathbf{R}^2 \mathbf{A}_2 + \mathbf{R}^1 \mathbf{A}_1 + \mathbf{R}^0 \mathbf{A}_0 = \mathbf{0}. \tag{10}$$

To start the recursive relation (9), the following boundary equations must be solved to obtain $\underline{z}_0$ and $\underline{z}_1$:

$$(\underline{z}_0, \underline{z}_1) \begin{pmatrix} \mathbf{B}_{00} & \mathbf{B}_{01} \\ \mathbf{B}_{10} & \mathbf{A}_1 + \mathbf{R}\mathbf{A}_2 \end{pmatrix} = (\underline{0}, \underline{0}). \tag{11}$$

As can be seen, the length of $\underline{z}_0 = (\pi_{(0,1,0)}, \cdots, \pi_{(0,m_a,0)})$ is m_a. Finally, the normalization equation has to be used to come to a unique solution:

$$\sum_{i=0}^{\infty} \underline{z}_i \underline{1} = \underline{z}_0 \underline{1} + \sum_{i=1}^{\infty} \underline{z}_i \underline{1} = \underline{z}_0 \underline{1} + \underline{z}_1 \left(\sum_{i=0}^{\infty} \mathbf{R}^i \right) \underline{1} = \underline{z}_0 \underline{1} + \underline{z}_1 (\mathbf{I} - \mathbf{R})^{-1} \underline{1} = 1. \tag{12}$$

2.4 The derivation of R

As we have seen, the matrix $\mathbf{R}$ plays a central role in the matrix geometric method. Then, how is $\mathbf{R}$ calculated? In a general setting, $\mathbf{R}$ is the solution of the equation

$$\sum_{i=0}^{\infty} \mathbf{R}^i \mathbf{A}_i = \mathbf{0}. \tag{13}$$

Normally, $\mathbf{R}$ is calculated iteratively. Starting with

$$\mathbf{R}(0) = -\mathbf{A}_0 \mathbf{A}_1^{-1}, \tag{14}$$

we obtain successive approximations of $\mathbf{R}$ as follows:

$$\mathbf{R}(k+1) = -\sum_{l=0,\ l\neq 1}^{\infty} \mathbf{R}^l(k) \mathbf{A}_l \mathbf{A}_1^{-1}, \quad k = 0, 1, \cdots \tag{15}$$

The iteration is stopped whenever $||\mathbf{R}(k) - \mathbf{R}(k+1)|| < \epsilon$. It can be shown that the sequence $\{\mathbf{R}(k),\ k = 0, 1, \cdots\}$ is entry-wise nondecreasing, and that it converges monotonically to the matrix $\mathbf{R}$ [11]. For the problems we study, we have $\mathbf{A}_i = \mathbf{0}$ for $i \geq I$. This implies that the suggested infinite summation in Equation (15) does not occur in practice.

2.5 Performance metrics

Once the matrix $\mathbf{R}$ and the two boundary vectors $\underline{z}_0$ and $\underline{z}_1$ have been obtained, all types of interesting performance metrics can be derived:

- Arrival and service process characteristics:
  - the average interarrival time $E[A] = (-\underline{\alpha}\mathbf{T}^{-1}\underline{1})^{-1}$; we define $\lambda = 1/E[A]$;
  - the average service time $E[S] = (-\underline{\beta}\mathbf{S}^{-1}\underline{1})^{-1}$; we define $\mu = 1/E[S]$.
- Average performance measures:
  - the utilisation $E[N_s] = \rho = \lambda/\mu$;
  - the average number of packets in the queue and server $E[N] = \sum_{i=0}^{\infty} i\underline{z}_i\underline{1} = \underline{z}_0\mathbf{R}(\mathbf{I} - \mathbf{R})^{-2}\underline{1}$;
  - the average number of packets in the queue $E[N_q] = E[N] - \rho$;
  - the average response time $E[R] = E[N]/\lambda$;
  - the average waiting time $E[W] = E[N_q]/\lambda$.
- Detailed performance measures:
  - the probability z_i of having i packets in the queue: $z_i = \underline{z}_i\underline{1}$;
  - the probability B_j of having at least j packets in the queue: $B_j = \sum_{i=j}^{\infty} z_i$.

2.6 The caudal curve

A special type of measure that can be obtained is the so-called *caudal curve* [13]. Consider again the matrix $\mathbf{R}$. The largest eigenvalue of $\mathbf{R}$, also called the spectral radius of $\mathbf{R}$, is denoted η. The graph of η as a function of ρ is called the caudal curve. This name stems from the Latin *cauda*, meaning tail, a name that will become clear shortly. It can be shown that

$$\lim_{i\to\infty} \frac{\underline{z}_{i+1}\underline{1}}{\underline{z}_i\underline{1}} = \lim_{i\to\infty} \frac{\underline{z}_1\mathbf{R}^i\underline{1}}{\underline{z}_1\mathbf{R}^{i-1}\underline{1}} = \eta. \tag{16}$$

This means that for large i the ratio of the expected time spent at level $i+1$ to that at level i is approximately equal to η. A similar limit theorem holds for any two corresponding elements of the probability vectors $\underline{z}_i$ and $\underline{z}_{i+1}$. Recalling that a level corresponds to the set of states for which the number of packets in the queue is the same, it is clear that Equation (16), for large i, says something about the tail of the queue length distribution. In a similar way, the equality

$$\Pr\{N_q > k\} = h\eta^k + o(\eta^k), \quad \text{for } k \to \infty, \tag{17}$$

holds. So, we observe that η "rules" the rate of decay of the steady state queue length distribution.

Only for very few queueing systems the caudal curve can be obtained with little effort. For M|M|c queueing systems we have $\eta(\rho) = \rho$. We could regard an M|M|c queueing system as a reference queueing system.

For $E_r|E_r|1$ queueing systems, it can be derived that $\eta(\rho) = \rho^r$, which implies that $\eta(\rho) \leq \rho$. Intuitively, one might have expected the latter inequality. Erlang-r distributions have a smaller coefficient of variation than exponential distributions. Less variance often implies better performance, i.e. smaller waiting times and smaller queues. Thus, η will generally be smaller than ρ, so as to accelerate the decay in Equation (17).

In a similar way, for an $H_2|M|1$ queueing system, by a suitable parameter choice for the 2-phase hyperexponential distribution, the explicit solution for the caudal curve reveals that $\eta(\rho) \geq \rho$. Again, this is intuitively appealing since the hyperexponential distribution is known to introduce more variability in the system, which often implies worse performance, i.e. longer queues and longer waiting times. Then, a value of η larger than ρ will cause a shift in the queue length probability distribution towards higher queue length values.

For more intricate queueing systems, the caudal curve can only be obtained numerically from $\mathbf{R}$. Its form can be rather suprising, thus revealing interesting system behaviour that would not have been revealed by the average performance metrics only.

3 MGMtool — Requirements and architectural issues

As stated already in Section 1, one of the reasons why matrix geometric techniques are not that widely used for applied performance analysis studies, might be the fact that the required mathematics is non-trivial. In order to proliferate the usage we should therefore aim at providing high level constructs for the definition of queueing systems. These constructs should allow users to easily specify a queueing system, without having to bother about the underlying mathematical engine. Furthermore, the users should have a large flexibility in specifying experiment series. The translation to the underlying solution modules should be both automatic and transparant. The output of the tool should be given in terms of the model as specified by the user. If more specialized users want so, they should be able to control the underlying mathematical engine and to view intermediate results.

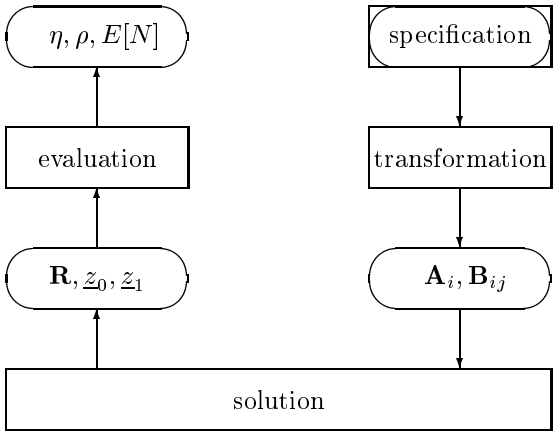

Figure 2: The four functional parts and their interfaces of MGMtool

The above sketched requirements are very general and also applicable to other analysis tools, and can also be found in Berson *et al.* [2], Haverkort and Niemegeers [7] and Szczittnick and Müller-Clostermann [17]. Keeping these requirements in mind, a performance analysis tool based on matrix geometric methods can be thought to consist of four functional parts:

- *A specification part.* The functionality of this part of a tool is such that it allows the user to specify queueing models in an easy and flexible way. It must allow the user to easily specify which measures to obtain and for which parameter values. Furthermore, if the user should want so, it should allow for the control of the underlying mathematics. One can think of accuracy control or the choice of specific numerical procedures.

- *A transformation part.* The functionality of this part of a tool is such that it "translates" the description of the queueing model made by the user to an internal data structure that is directly amenable for numerical procedures.

- *A solution part.* The functionality of this part of a tool is that it solves the mathematically specified model resulting form the transformation part. It results in an abstract "solution" of the model, typically in some internal format.

- *An evaluation part.* The functionality of this part of a tool is to enhance the already obtained abstract solution to user oriented measures of interest. Results generated by this tool part are typically tables and graphs, presented in a way that is suitable to human users.

The separation in four parts as given above is still very abstract. For MGMtool, a more concrete description of the four parts is as follows:

- The specification part. Here modelling primitives are provided that allow users to specify phase type interarrival and service time distributions, the number of servers and the measures that must be calculated. Furthermore, it allows to control internal quantities such as the required accurracy or the maximum number of iterations.

- The transformation part. Here procedures are provided that transform the user description of the queueing system to the $\mathbf{A}_i$ and $\mathbf{B}_{ij}$ matrices, characteristic for the matrix geometric approach.

- The solution part. Here procedures are provided that derive the matrix $\mathbf{R}$ and the boundary vectors $\underline{z}_0$ and $\underline{z}_1$ from the already derived matrices $\mathbf{A}_i$ and $\mathbf{B}_{ij}$.
- The evaluation part. Here procedures are provided that derive user oriented performance measures from the matrix $\mathbf{R}$ and the boundary vectors $\underline{z}_0$ and $\underline{z}_1$.

The four parts and their relation are depicted in Figure 2 in which boxes represent activa (programs) and ovals represent passiva (files, data structures)[2].

4 MGMtool — Modelling primitives

MGMtool has been built as a library of C procedures and functions that can be used in a C program to specify the queueing system model and the desired performance measures, as well as to control the transformation, the solution and the evaluation process. We introduce the modelling primitives by giving an example of the specification of a PH|PH|1 queue in Section 4.1. We then discuss in more detail the possibilities of the specification primitives in Section 4.2.

4.1 An example specification: a PH|PH|1 queueing system

In the description that follows, we continuously refer to the C program in Figure 3. A C program meant as input for MGMtool may include all normal C code, but no `main()`. It should however include the header-file `mgmproj.h` and a function `project()`. In the body of `project()` the actual model specification takes place.

First, phase type representations of the interarrival and service time distributions have to be given. To accomplish this, three real variables are declared (`lambda`, `mu1`, and `mu2`), as well as two matrices for the PH representations of the interarrival and the service time distributions (`Arr` and `Ser` respectively). Then, the variables are assigned their value, followed by the assignment of a value to the matrices. Here, the interarrival time distribution is of exponential type, with rate `lambda`, and the service time distribution is of hypo-exponential type, with two phases. The first phase has rate of completion `mu1`, and the second `mu2`.

The measures that are to be derived by the tool are specified with the procedure `COMPUTE()`. This procedure has a variable length parameter list that should end with the special parameter `END`. In the example, we specify that ρ, η and $E[N]$ have to be calculated.

The iterative process for finding $\mathbf{R}$ is controlled by assigning values to a number of "loop-control" variables. The procedure `DEFAULT()` assigns a set of values that normally work fine.

The procedure `PH2MGM()` requires as parameters the interarrival and service time distribution and the number of servers, in a Kendall notation look-alike way. It transforms them in an internal representation which basically consists of the matrices $\mathbf{A}_i$ and $\mathbf{B}_{ij}$. The procedure `MGMSOLVE()` then first derives the matrix $\mathbf{R}$ and the boundary vectors $\underline{z}_0$ and $\underline{z}_1$. Then, depending on what has been specified by the `COMPUTE()` procedure, the measures of interest are calculated. Finally, the procedure `TOFILE()` specifies which measures have to be included in the standard report of MGMtool.

[2] In a sense, the specification is a file, and should therefore be represented as an oval. On the other hand, in the implementation of MGMtool, the specification is an executable program that generates the input to the transformation part. From that viewpoint, the specification should be represented as a box.

```
#include "mgmproj.h"           /* inclusion of header file                  */

project()                      /* start of project definition               */
{
   double lambda, mu1, mu2;    /* declaration of reals                      */
   mat_type Arr, Ser;          /* declaration of matrices                   */

   lambda = 1.0;               /* value assignment to variables             */
   mu1 = 8.0; mu2 = 3.0;
   EXPV(&Arr, lambda);         /* definition of the arrival process:  an    */
                               /* exponential distribution with rate lambda */
   HYPOV(&Ser, 2, mu1, mu2);   /* definition of the service process:  an    */
                               /* hypo-exponential distribution with two    */
                               /* phases, the first with rate mu1, the      */
                               /* second with rate mu2                      */
   DEFAULT();                  /* set iteration control parameters          */
   COMPUTE(RHO, ETA, EN, END); /* specification which measures to obtain    */
   PH2MGM(Arr, Ser, 1);        /* transformation of arrival and service     */
                               /* processes to the A_i and B_ij matrices    */
   MGMSOLVE();                 /* derivation of R, z_0 and z_1              */
   TOFILE(RHO, ETA, EN, END);  /* write performance measures to file        */
}
```

Figure 3: Specifying a PH|PH|1 queueing system

4.2 Details of the specification primitives

In this section we discuss the modelling primitives that can be used. We first discuss the various ways to define interarrival and service time distributions as well as the allowed overall model class. Then we discuss the capabilities of the underlying mathematical engine, i.e. which measures can be obtained, by which techniques etc. We finally discuss the various output facilities that are provided.

Defining interarrival and service time distributions

MGMtool allows for the direct use of a number of standard PH distributions: exponential (`EXPV()`), Erlang (`ERLV()`), hypo-exponential (`HYPOV()`), hyper-exponential (`HYPERV()`) and Coxian (`COXV()`). These distributions are used through specific procedure calls. By these procedures, a PH representation is built, using the parameters specific for the distribution[3].

Starting from these standard distributions, more complicated ones can be constructed by expanding any single state in which the state residence time is exponentially distributed, in a state in which the residence time is PH distributed by using the `EXPAND()` procedure. This expanding can be applied repeatedly. In Figure 4 we show an hyperexponential distribution in which one of the states is expanded to an Erlang-3 distribution. The corresponding C code is given in Figure 5.

[3]Note that since this number of parameters is variable, these procedures employ variable argument lists. This explains the `V` in their name. A different set of procedures works without these variable argument lists (and without the `V` in their names). Experience up till now has revealed that the variable argument list procedures work the most comfortable.

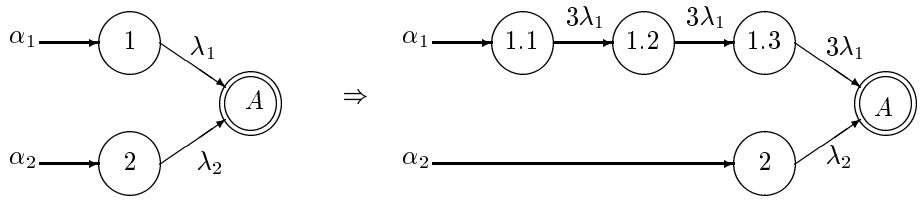

Figure 4: Expanding a state

```
project()
{
 double alpha1, alpha2, lambda1, lambda2; /* declare program variables */
 mat_typ Arr, Help;                       /* declare representation matrices */

 HYPERV( &Arr, 2, lambda1, lambda2,
       alpha1, alpha2);                   /* define the basic hyperexponential */
 ERLV( &Help, 3, 3*lambda1);              /* define a ``Help'' Erlangian */
 EXPAND( &Arr, Arr, Help, 1);             /* expand state 1 of 'Arr' with 'Help', */
                                          /* i.e. as a 3-stage Erlangian */
}
```

Figure 5: C code to expand a state

MGMtool also allows for the use of cyclic PH distributions. By using the `BACKTRANS()` procedure, "backward transitions" can be added to PH distributions that have already been constructed by one of the methods above.

When none of the standard procedures apply, one can resort to the C language, to make procedures that correctly fill the representation matrices of the required PH distributions.

Model class

From the PH representations of the interarrival and service time distributions the matrices $\mathbf{A}_i$ and $\mathbf{B}_{ij}$ are derived. Depending on the type of interarrival and service time distribution, this derivation is more or less complex. The procedure `PH2MGM()` recognizes the used distributions and selects the most appropriate way to proceed. Model classes that are currently distinguished are the PH|M|1 queue, the PH|PH|1 queue and the PH|M|2 queue.

Measures to obtain

The measures that a user of MGMtool wants to be derived have to be indicated in the argument list of the procedure `COMPUTE()`. The following standard measures are available (between brackets the corresponding measure index): the mean arrival rate (`LAMBDA`), the mean service rate (`MU`), the utilization (`RHO`), the largest eigenvalue of $\mathbf{R}$ (`ETA`), the mean number of packets in the queue (`ENq`), the mean number of packets in the total system (`EN`), the mean waiting time (`EW`), the mean response time (`ER`), the steady state probabilities per level (`Z`, `Z+1`, `Z+2`, $\cdots$), and the blocking probabilities per level (`B`, `B+1`, `B+2`, $\cdots$).

ρ	η	$E[N]$
0.458333	0.394325	0.769195

Table 1: Results of `mgm2latex` for the PH|PH|1 specification

Apart from these queueing system oriented measures one can also obtain measures related to the computations themselves: the required CPU time (`CPU`), and the number of steps needed in the iteration to obtain $\mathbf{R}$ (`STEPS`).

The actual calculations are done via the call `MGMSOLVE()`. Afterwards, the desired measures are available in the array `METRIC[]`. The measure indices also serve as indices in this array. One can also define "user defined measures" in such a way that they are handled in a similar way as the standard measures, i.e. one can include them in the compute and output procedures.

Mathematical methods used

The derivation of the measures requires the execution of a number of algorithms that heavily rely on matrix operations.

Linear systems (see Equations (11) and (12)) are solved directly with Gaussian elimination or iteratively. Among the used iterative algorithms are the Jacobi method and the Gauss Seidel method. Depending on the stability criteria for these algorithms, the most suitable one is chosen. The stop criterion for the iterative methods allows for tests on the number of iterations, on the used CPU time, or on the accuracy of the results. The latter can be specified per element of the solution vector or for the vector as a whole.

Square matrices are inverted (see Equation (14)) by solving a linear system of equations as many times as the size of the matrix requires.

The matrix $\mathbf{R}$ is solved iteratively (see Equation (15)). When the number of $\mathbf{A}_i$ matrices is large, we use Horner's algorithm, since it provides the most efficient way of evaluating "long" matrix polynomials [12, p.38]. For the case that we only have to deal with the matrices $\mathbf{A}_0$ through $\mathbf{A}_2$, we employ a direct and therefore more efficient multiplication scheme [5].

The maximum eigenvalue η of the matrix $\mathbf{R}$ is calculated by using the Power method or Elsner's algorithm. Up till now, the results of some test runs indicate that the Power method is preferable.

For the iterative process to obtain $\mathbf{R}$, the following stop criteria can be specified: the maximum number of iteration steps (`MAXSTEPS`), the maximum amount of CPU time in seconds (`MAXCPU`), and the required accuracy (`EPSILON`). The procedure `DEFAULT()` assignes the values 50, 3600 and 10^{-5} respectively. The iterative process stops whenever at least one of the criteria is met.

Output facilities

The compute procedure is used to indicate which measures are to be computed. This is specified separately from the way they are output. Obtained measures can be used in further calculations or `fprintf()` statements by accessing the array `METRIC[]`. The procedure `TOFILE()` generates an ASCII table of the measures indicated in its argument list.

With the program `mgm2latex` the standard output of MGMtool can be converted to a LaTeX formatted table (see Table 1). Similarly, the program `mgm2gnu` generates a Gnuplot plotfile from the standard output file produced by MGMtool.

5 MGMtool — Implementation issues

In this section we discuss various implementation aspects of MGMtool. MGMtool has been developed for Sun 4 workstations, running the Unix operating system. All code has been written in C. In Section 5.1 we discuss the internal structure of MGMtool and in Section 5.2 we discuss how MGMtool is used, given that a valid specification has been made. Finally, in Section 5.3 we discuss how we tested MGMtool.

5.1 Internal structure of MGMtool

The directory structure of MGMtool is rather straightforward. The main directory (`mgmtool`) contains 6 subdirectories and a number of small files (programs). The directory `headers` contains a number of files in which the used matrix datatypes, some global variables and the used constants (macros) are defined. The directory `libraries` contains the programs to handle matrices as well as the linear system solvers. The directory `spectrans` contains programs that allow for the specification of the PH distributions as well as for the transformation of these distributions to the required matrices (the $\mathbf{A}_i$ and $\mathbf{B}_{ij}$ matrices). In the directory `soleval` programs are present that solve for $\mathbf{R}$ and $\underline{z}_i$, $i = 0, 1, \cdots$, and programs that take care of the evaluation of the desired measures. The directory `mgmproj` contains programs that control the actual progress of project evaluations. Finally, the directory `interfaces` contains procedures that take care of reading and writing intermediate data structures. Depending on the compilation options, data interchange happens via files or via global variables[4].

The directory `mgmtool` also contains 4 programs (`mgm2latex.c`, `mgm2gnu.c`, `mgm.c` and `mgmmain.c`) and 2 makefiles (`mgmmake` and `makefile`). Only the `makefile` is of interest here. With the Unix command `make` a static library is constructed, containing all the procedures, functions and data types of all the subdirectories of the directory `mgmtool`. Procedures, functions and data types from this library can be used by any `project()` to be specified later.

Experience with MGMtool has revealed that the calculation of $\mathbf{R}$ is the main performance determining factor. The size of $\mathbf{R}$ equals $m_a m_s$. In the current implementation of MGMtool, this size is limited to $50^2 = 2500$. For matrices this large, special techniques need to be implemented to keep computation times within bounds.

5.2 MGMtool as perceived by its users

For using MGMtool, a path has to exist to the directory `mgmtool` discussed above. Once a user has made a `project()`, say in the file `phph1.c`, the call

```
mgm -o resultfile phph1
```

executes the program `mgm.c` from the mgmtool directory. This program checks parameters and options of the call. Then the procedure `project()` as specified by the user is embedded in the program `mgmmain.c`. This program is then compiled via a `make` command, using the makefile `mgmmake`. In this compilation, all the required procedures and functions are statically taken from the library. The resulting executable is placed in the calling directory and immediately invoked. The results (of the `TOFILE()` procedure) are written in the file indicated by the `-o` option. Default is a file with name equal to the project file but with extension `out`. When supplying `screen` as filename, the output is written on the screen.

[4] For debugging purposes, data interchange via files is preferable. Once the programs run smoothly, data interchange via global variables is far preferable, due to its much lower delay.

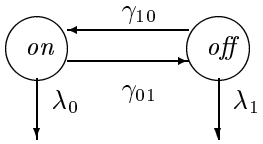

Figure 6: A Markov modulated Poisson process

The above call also creates two subdirectories in the calling directory: `params` and `matrices`. In the former some model parameters are stored in files. In the latter files are created that contain the matrices $\mathbf{A}_i$, $\mathbf{B}_{ij}$ and $\mathbf{R}$ as well as the boundary vectors $\underline{z}_0$ and $\underline{z}_1$. The following call removes these directories as well as their contents:

```
mgm -r phph1
```

5.3 Testing MGMtool

The correctness of MGMtool has been tested by running a large set of examples and comparing the results with known analytical results or simulation [5].

Examples that were run to test for the correct calculation of $E[N]$ included an M|M|c ($c = 1, \cdots, 4$) and various M|G|1 queues. Examples that were run to test for the correct computation of η included the M|M|c queue, the $E_k|E_k|1$ queue and the $HE_2|M|1$ queue. In all cases the results were satisfactory, given the required accuracy.

6 MGMtool — An application

In this section, we show the use of MGMtool for the analysis of a queueing system with Markov modulated Poisson arrivals and Erlangian service times.

In modern day computer and communication networks, packets do not arrive at the system as a Poisson stream. Generally, there is much correlation between successive packets. Furthermore, packets tend to have a constant length.

To accomplish the latter in a model, one can employ Erlangian service times (packet lengths). The (squared) coefficient of variation C_v^2 of an Erlang-k distribution is known to be $1/\sqrt{k}$. Distributions with $C_v^2 \leq 0.5$ are often assumed to be already "pretty deterministic".

To obtain a model with correlated arrivals, one can employ the packet train model of Jain [9]. A particular implementation of this model is a Markov modulated Poisson process (MMPP). Gusella also shows that MMPPs are well-suited for modelling realistic arrival processes [6].

Consider an MMPP in which the modulating Markov chain has only two states, "off" (0) and "on" (1). The transition rate from state i to state j equals γ_{ij}. In state i, packets are generated as a Poisson process with rate λ_i. In Figure 6 we show such an MMPP. When we choose $\lambda_0 = 0$, the time between successive packet arrivals of the Poisson stream with rate λ_1 is PH distributed (since we only deal with a non-zero value for λ_1, we omit the subscript 1 from now on). The absorbing Markov chain, corresponding to this PH distribution, is given in Figure 7. The generator matrix is given as

$$\mathbf{Q} = \begin{pmatrix} -\gamma_{01} & \gamma_{01} & 0 \\ \gamma_{10} & -(\gamma_{10}+\lambda) & \lambda \\ 0 & 0 & 0 \end{pmatrix}, \tag{18}$$

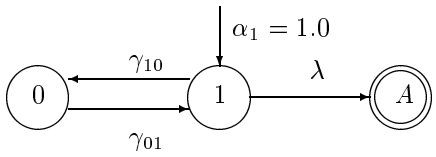

Figure 7: PH representation of a Markov modulated Poisson process

```
void MMPP(phmat, a0, a1, g01, g10, l) /* procedure heading */
mat_typ *phmat;                       /* procedure parameters */
double a0, a1, g01, g10, l;
{
     PHMATINIT(phmat,3);              /* creation of ``phmat'' */
     phmat->element[0] = -g01;        /* filling the elements of ``phmat'' */
     phmat->element[1] = g01;
     phmat->element[2] = 0.0;
     phmat->element[3] = g10;
     phmat->element[4] = -(g10+l);
     phmat->element[5] = l;
     phmat->element[6] = a0;          /* the vector ``alpha'' is stored in */
     phmat->element[7] = a1;          /* the last row of ``phmat'' */
     phmat->element[8] = 0.0;
} /* end MMPP() */
```

Figure 8: Code of the `MMPP()` procedure

and the vector $(\underline{\alpha}, \alpha_A) = (0, 1, 0)$. Note that due to the choice of $\underline{\alpha}$, we have established that after a packet arrival (an absorption) the PH distribution starts anew in state 1, i.e. the arrival process remains in a burst. An important parameter for MMPP is the *burstiness* b. It is defined as the ratio between the arrival rate in a burst and the overall average arrival rate [8]:

$$b = \frac{\text{arrival rate in a burst}}{\textit{overall average arrival rate}} = \frac{\lambda}{\lambda\gamma_{01}/(\gamma_{01}+\gamma_{10})} = \frac{\gamma_{01}+\gamma_{10}}{\gamma_{01}}. \tag{19}$$

MGMtool does not provide a standard procedure to input this MMPP. We can however resort to the C functionality to accomplish this. We simply construct a procedure which, given the paramaters of this MMPP, assigns them correctly to a PH representation matrix. In Figure 8 we show this procedure. The overall MGMtool specification for this example is given in Figure 9. In this figure we present the specification that is used to calculate the values of η, $E[N]$ and $E[W]$ for $\rho = 0.05, \cdots, 0.95$.

In the first set of model evaluations, we will vary the burstiness of the MMPP, while keeping the utilisation of the communication channel constant. This allows us to quantitatively investigate the influence of b on the performance. As performance measures of interest we have selected η and $E[N]$. We keep the number of Erlangian stages fixed to 4 in these cases.

Varying ρ from 0.05 to 0.95 (with steps of 0.05) we obtain the caudal curves as in Figure 10. For clarity we only show the curves with $b = 2$, $b = 4$ and $b = 10$. As could be expected, when b increases, the caudal curve tends to the line $\eta(\rho) = 1,\ \rho > 0$. In Figure 11(a) we show the average number of packets in the queueing system as a function of ρ, again for the three mentioned burstiness factors. The burstiness clearly has a bad influence on the average number

```
project()                          /* start of project definition */
{
    int k;                         /* k for the erlang-k distribution */

    double mu,                     /* service time parameter */
        alpha0, alpha1,            /* entry probabilities for the MMPP */
        gamma01, gamma10,          /* transition rates in the MMPP */
        lambda,                    /* arrival rate in state 1 of the MMPP */
        rho,                       /* rho as required by user */
        b;                         /* burstiness */
    mat_typ ARR, SER;              /* declaration of matrices */

    k = 10;                        /* Erlang-10 service time distribution */
    b = 4.0                        /* required burstiness */
    mu = 100.0;
    alpha0 = 0.0;
    alpha1 = 1.0;
    gamma10 = 1.0;                 /* burst time is 1 second */
    DEFAULT();                     /* set iteration control parameters */
    for (r=1; r<=19; r++)          /* let r run from 1 to 19 */
    {
        rho = r/20.0;              /* set the required rho */
        gamma01 = gamma10/(b-1.0); /* set gamma01 such that b is the burstiness */
        lambda = rho * b * mu;     /* set lambda such that RHO will equal rho */

        MMPP( &ARR, alpha0, alpha1, gamma01, gamma10, lambda );
                                   /* definition of the MMPP arrival process */
        ERLV( &SER, k, mu*k);      /* definition of the service time distribution:  */
                                   /* k-stage Erlang with rate mu*k per stage */
        COMPUTE(RHO, ETA, EN, EW, END);
        PH2MGM(ARR, SER, 1);
        MGMSOLVE();
        TOFILE(RHO, ETA, EN, EW, END);
    } /* end for r */
} /* end project() */
```

Figure 9: Specification of the MMPP example model

Figure 10: Caudal curves for the MMPP$|E_4|1$ queue for $b = 2$, 4 and 10

of customers in the queue. The value of ρ for which the $E[N]$ curve clearly starts to deviate from the ρ-axis, roughly corresponds to the value of ρ for which $\eta \approx 0.8$. This is confirmed by Figure 11(b), in which we show $E[N]$ as a function of η. The three curves almost coincide. The value of η seems to be a better characterization of the intensity of the traffic through the queue than ρ.

In the second set of evaluations we investigate how important it is to make the service time distribution more deterministic. We therefore keep the burstiness $b = 4$ and vary the number of stages in the Erlangian service time distribution over 1, 5, 10 and 20.

In Figure 12(a) we show the caudal curves. Similarly, we show $E[N]$ as a function of ρ in Figure 12(b). What can be observed is that both η and $E[N]$ are governed more by the arrival process than by the service time distribution. Less variability of the latter does not seem to be able to increase the performance of the queueing system significantly. This clearly indicates the importance of using realistic arrival process representations.

From the application discussed in this section, it has become clear that the influence of traffic burstiness (variability) on the system performance is enormous. Moreover, by the fact that the number of stages in the Erlangian service time distribution did not seem to matter that much, we have the feeling that variability in the interarrival time distribution determines the system performance more than variability in the service time distribution. More research needs to be done to make this statement more exact.

7 Concluding remarks and future work

In this paper we have described a performance analysis tool based on matrix geometric methods. After a short resume of these methods we have discussed the general tool architecture as well as many implementation characteristics. MGMtool is, to the best of our knowledge, the only general performance analysis tool that relies on matrix geometric methods and that does not bother the tool user with mathematical details of this method. Instead, it allows users to specify queueing systems by only specifying a PH representation of the interarrival and service time distribution. MGMtool then takes care of the underlying mathematics. Also on a relatively high level, the user specifies the desired output measures. Apart from average performance measures, more

(a) (b)

Figure 11: $E[N]$ as a function of ρ and η for the MMPP$|$E$_4|1$ queue for $b = 2$, 4 and 10

(a) (b)

Figure 12: Results for the MMPP$|$E$_k|1$ queue for $b = 4$ and $k = 1$, 5, 10 and 20

detailed performance measures can also be derived. A special measure that can be obtained is the caudal curve which helps to understand the variability in the queueing process. With a number of examples we have shown the usability of the tool. Integration with the programming language C has turned out to be very powerful.

For the (near) future, we see a number of directions that can be taken to enhance the work presented here:

- to extend the model class of MGMtool by allowing:
  - series of queues;
  - general PH|PH|c queues;
  - general Markov modulated Poisson processes;
  - queues with an embedded M|G|1 or GI|M|1 structure;
- to extend the underlying mathematical engine, e.g. by including routines that allow for the calculation of waiting time distributions or more advanced routines for the determination of $\mathbf{R}$, especially when the model class is enlarged;
- to built an X-windows environment for MGMtool, or to allow for the graphical input of certain parts of the model specification;
- to develop new input paradigms;
- to use MGMtool for more and more realistic modelling studies.

References

[1] H. Ahmadi, R. Guérin, "Analysis of a Class of Buffer Storage Systems with Markov-Correlated Input and Bulk Service", *Proceedings of the Fourth International Conference on Data Communication Systems and Their Performance*, pp.67–84, Barcelona, June 20–22, 1990.

[2] S. Berson, E. de Souza e Silva, R.R. Muntz, "An Object Oriented Methodology for the Specification of Markov Models", *UCLA Technical Report* CSD-870030, 1987.

[3] C. Blondia, O. Casals, "Statistical Multiplexing of VBR Sources: A Matrix-Analytical Approach", *Workshop proceedings "Performance Aspects in ATM"*, PTT Research Laboratories, Leidschendam, the Netherlands, October 1991.

[4] G. Ciardo, J. Muppala, K.S. Trivedi, "SPNP: Stochastic Petri Net Package", *Proceedings of the Third International Workshop on Petri Nets and Performance Models*, IEEE Computer Society Press, pp.142–151, 1989.

[5] A. Dijkstra, *MGM: A tool for the performance analysis of* PH|PH|c *queues, based on matrix geometric techniques*, M.Sc. thesis, University of Twente, Department of Electrical Engineering, 1992.

[6] R. Gusella, "Characterizing the Variability of Arrival Processes with Indexes of Dispersion", *IEEE Journal on Selected Areas in Communications*, **9**(2), pp.203–211, 1991.

[7] B.R. Haverkort, I.G. Niemegeers, " Performability Modelling Tools: A Survey", *Memoranda Informatica* 91-82, University of Twente, Department of Computer Science, 1991.

[8] G.J. Heijenk, A.P.A. van Moorsel, I.G. Niemegeers, "Performance of a Connectionless Protocol over ATM", *Proceedings of the International Workshop on Advanced Communications and Applications for High Speed Networks*, pp.123–130, Munich, March 1992.

[9] R. Jain, "Packet Trains—Measurements and a New Model for Computer Network Traffic", *IEEE Journal on Selected Areas in Communications*, **4**(6), pp.986-995, 1986.

[10] R. Jain, *The Art of Computer System Performance Analysis*, John Wiley & Sons, 1991.

[11] R. Nelson, "Matrix Geometric Solutions in Markov Models: A Mathematical Tutorial", *IBM Research Report* RC 16777, 1991.

[12] M.F. Neuts, *Matrix Geometric Solutions in Stochastic Models: An Algorithmic Approach*, Johns Hopkins University Press, Baltimore, 1981.

[13] M.F. Neuts, "The Caudal Characteristic Curve of Queues", *Advances in Applied Probability* **18**, pp.221–254, 1986.

[14] M.F. Neuts, *Structured Stochastic Matrices of* M|G|1 *Type and Their Applications*, Marcel Dekker Inc., New York, 1989.

[15] L. Kleinrock, *Queueing Systems, Volume 1: Theory*, John Wiley & Sons, 1975.

[16] M.F. Squillante, "MAGIC: A Computer Performance Modelling Tool Based on Matrix-Geometric techniques", in *Computer Performance Evaluation: Modelling Techniques and Tools*, Editors: G. Balbo, G. Serazzi, North-Holland, pp.411–425, 1992.

[17] M. Sczittnick, B. Müller-Clostermann, "MACOM: A Tool for the Markovian Analysis of Communication Systems", *Proceedings of the Fourth International Conference on Data Communication Systems and Their Performance*, pp.456–470, Barcelona, June 20–22, 1990.

[18] K.S. Trivedi, *Probability & Statistics with Reliability, Queueing and Computer Science Applications*, Prentice-Hall, 1982.