Markovian Models for Performance and Dependability Evaluation

Boudewijn R. Haverkort

Laboratory for Performance Evaluation and Distributed Systems
Department of Computer Science, RWTH Aachen, 52056 Aachen, Germany
haverkort@cs.rwth-aachen.de

Abstract. Markovian models have been used for about a century now for the evaluation of the performance and dependability of computer and communication systems. In this paper, we give a concise overview of the most widely used classes of Markovian models, their solution and application.
After a brief introduction to performance and dependability evaluation in general, we introduce discrete-time Markov chains, continuous-time Markov chains and semi-Markov chains. Stepwisely, we develop the main equations that govern the steady-state and the transient behaviour of such Markov chains. We thereby emphasise on intuitively appealing explanations rather than on mathematical rigor. The relation between the various Markov chain types is explained in detail. Then, we discuss means to numerically solve the systems of linear equations (both direct and iterative ones) and the systems of differential equations that arise when solving for the steady-state and transient behaviour of Markovian models.

1 Introduction

Markovian models have inherited their name form the pioneering work by the Russian mathematician A.A. Markov around the turn of the twentieth century (see Figure 1). He introduced finite-state Markov chains in [49]; a translation of another important article of his hand appears in Appendix B of [37]. In fact, his work launched the area of stochastic processes. In the first two decades of the twentieth century, the Danish mathematician A.K. Erlang (see Figure 2) applied Markovian techniques (then not yet named as such) to solve capacity planning problems for the Copenhagen Telephone Company [20]. His models were soon adapted by others, among others by the Britisch Post Office; one of his first models will be presented later in this paper. The Russian mathematician A.N. Kolmogorov (see Figure 3) developed the theory for Markov chains with infinite (denumerable and continuous) state spaces in the 1930's [46].

Throughout the twentieth century the work of these pioneers became better understood and more wide-spread. These days, Markov chains and stochastic processes form the basis for model-based system evaluations in many areas of science and engineering. For instance, in biology to model growth and decay of

Fig. 1. Andrei Andreevich Markov (* June 14, 1856; † July 20, 1922)

Fig. 2. Agner Krarup Erlang (* January 1, 1878; † February 3, 1929)

Fig. 3. Andrey Nikolaevich Kolmogorov (* April 25, 1903; † October 20, 1987)

populations, in physics to model interactions between elementary particles, in chemical engineering to model (chain) reactions between molecules or to model mixing processes, in management science to model the flow of commodities in logistic or flexible manufacturing systems or to model the availability of production lines and, most notably, in computer and communication science and engineering to model system performance and dependability in a wide variety of settings. In this paper, we focus on *the use of Markovian models for the performance and dependability evaluation of computer and communication systems.*

But let us first take one step back, and address the question what performance or dependability evaluation really is. Performance or dependability evaluation is the craft that tries to answer questions related to the performance or dependability of systems. Typical questions take the following form:

(i) how many clients can this server adequately support?
(ii) what is the typical response time to load a WWW page from MIT?
(iii) how large should the buffer space in this IP router be to guarantee a packet loss ratio of less than 10^{-6}?
(iv) how many jobs can be processed before a system failure occurs?
(vi) how long does it take before this system crashes?

The above questions can only be answered when they are made more exact, i.e., when some of the informally stated requirements or aims are made concrete. For instance, we will have to define what "adequately" really means in question (i), or what "typical" means in question (ii). More precisely, we have to define clearly what the *measure of interest* is, in order to express the performance or dependability criterion we are interested in, in the best possible way. Before we will come to this issue, we will restrict the class of evaluation techniques considerably. In this paper, we will only address so-called *model-based* evaluation techniques, meaning that we are not addressing measurement-based techniques such as benchmarking. Even though measurement-based evaluation techniques are very important and accurate (they address the real system, or at least a prototype) these methods are also very costly, and sometimes even inappropriate, for instance when the interest is in very rare events; a measurement-based approach then takes too long to be practically feasible. Hence, we restrict ourselves to model-based evaluation techniques, meaning that we have to develop models, in order to evaluate the system under study. According to [36], a *model* is a "*small-scale reproduction or representation of something*" and *modelling* is "*the* art *of making models*". The latter definition states an important aspect of model-based performance and dependability evaluation: the construction of appropriate models is a challenging task for which, as of yet, no general recipe is available.

Let us now come back to the measures of interest in a performance or dependability evaluation. The choice of measure strongly depends on the standpoint of the model user (the person using the results of the model evaluation). System engineers are most often interested in *system-oriented measures* like queue lengths, component utilisations or the number of operational components. For system users, seeing the system as a service-providing black box, these measures are not that interesting; for them what counts is how fast or how many service invocations can be performed per unit of time. Examples of such *user-oriented measures* are waiting times, throughput and downtime minutes per year. On top of this classification comes the question how detailed the measure of interest should be evaluated: does a mean value suffice, or is knowledge of higher moments or even of the complete distribution necessary? Furthermore, is the measure to be evaluated for a particular time instance in the operation of the system, that is, is there an interest in so-called *transient measures*, or is the interest more in long-term average values, that is, in so-called *steady-state measures.*

With the class of Markovian models we will address in the rest of this paper, we have available a versatile modelling formalism, allowing us (i) to model system performance and dependability at various levels of detail, (ii) to study a wide variety of user- and system-oriented measures, at (iii) either in steady-state or at some time instance t.

It should be noted that by the availability of good software tools for performance and dependability evaluation, the actual Markov chains being used and solved often remain hidden from the end-user. That is, users specify their performance or dependability model using some high-level modelling language, for instance based on queueing networks, stochastic Petri nets or stochastic process algebras of some sort, from which the underlying Markov chain can automatically be generated and analysed. The analysis results are then presented such that they can be interpreted correctly in the context of the high-level model, so that, in fact, the translations to and from the Markov chain level remain transparant. High-level specification techniques for Markov chains are an active area of research, but are not addressed in more detail in this paper; the interested reader is referred to a number of surveys addressing these issues [31,30,29] as well as to three other papers in this volume [6,10,60].

A final point of notice is the following. Model-based performance and dependability evaluation necessarily have to be based on abstractions of the real system. In that sense, it is intrinsically approximate. This is both a strength, as it can be applied always (albeit more or less accurate), and a weakness, as its accuracy is not known in advance. In any case, the results from an evaluation should be interpreted with care; the results are never more accurate than the numerical parameters used in the models! Furthermore, note that even though model-based performance and dependability evaluation provides us with numbers, the insight gained in the (functional) operation of the system under study is often even more important. As Alan Scherr, IBM's time-sharing pioneer, puts it in an interview with *Communications of the ACM* [22]: "model-based performance evaluation is about finding those 10% of the system that explains 90% of its behaviour".

After this more general introduction, the rest of this paper completely focusses on Markov chains. In Section 2 we introduce discrete-time Markov chains, followed by the introduction of semi-Markov chains and continuous-time Markov chains in Section 3 and Section 4, respectively. We then address techniques to solve these Markov chains with respect to their steady-state and their transient-state probabilities in Section 5 and Section 6, respectively. A variety of important issues not addressed in this paper is presented in Section 7. The paper ends with Section 8.

2 Discrete-Time Markov Chains

We define discrete-time Markov chains in Section 2.1, followed by a derivation of the steady-state and transient state probabilities in Section 2.2. We comment on the state residence time distribution in Section 2.3 and discuss convergence properties in Section 2.4.

2.1 Definition

Discrete-time Markov chains (DTMCs) are a class of stochastic processes. A stochastic process can be regarded as a family of random variables $\{X_t, t \in \mathcal{T}\}$, of which each instance X_t is characterisedby a distribution function. The index set $\mathcal{T}$ is mostly associated with the passing of time. In DTMCs, time passes in discrete steps, so that the subsequent time instances are denumerable and can be seen as elements of $\mathbb{N}$, hence, one typically denotes a DTMC as $\{X_n, n \in \mathbb{N}\}$. The continuous counterpart to DTMCs, that is, continuous-time Markov chains, will be discussed in Section 4.

The set of values the random variables X_n can assume is denoted the *state space* $\mathcal{I}$ of the DTMC. In performance and dependability evaluation, most often the state space of a DTMC is denumerable. We do restrict ourselves to that case here. Given a denumerable set $\mathcal{I}$, it can either be finite or infinitely large. We will only address the finite case here; a few remarks with respect to denumerable infinite state space will be given in the examples in Section 4.

The fact that we deal with a finite-state discrete-time stochastic process does not directly imply that we are dealing with a DTMC. The distinctive property of a DTMC is that the *Markov property* has to hold for it. This means that given the current state of the DTMC, the future evolution of the DTMC is totally described by the current state, and is independent of past states. This property is intuitively so appealing, that one sometimes tends to forget that it is a very special property. Mathematically, the Markov property can be described as follows. Assuming $\mathcal{T} = \{0, 1, 2, \cdots\}$ and $\mathcal{I} = \{i_0, i_1, \cdots\}$ we have

$$\Pr\{X_{n+1} = i_{n+1} | X_0 = i_0, \cdots, X_n = i_n\} = \Pr\{X_{n+1} = i_{n+1} | X_n = i_n\}.$$

From this equation we see that the future (at time instance $n+1$) only depends on the current state (at time instance n) and is independent of states assumed in the past (time instances 0 through $n-1$). At this point, we also note that the DTMCs we study are *time-homogeneous*, which means that the actual time instances are not important, only their relative differences, that is:

$$\Pr\{X_{n+1} = i | X_n = j\} = \Pr\{X_{m+1} = i | X_m = j\}, \quad \text{for all } n, m \in \mathbb{N},$$

with $i, j \in \mathcal{I}$. This means that in a time-homogeneous DTMC the state-transition behaviour itself does not change over time.

We now define the conditional probability $p_{j,k}(m,n) = \Pr\{X_n = k | X_m = j\}$, for all $m = 0, \cdots, n$, i.e., the probability of going from state $j \in \mathcal{I}$ at time $m \in \mathbb{N}$ to state $k \in \mathcal{I}$ at time $n \in \mathbb{N}$. Since we deal with time-homogeneous Markov chains, these *transition probabilities* only depend on the time difference $l = n - m$. We can therefore denote them as $p_{j,k}(l) = \Pr\{X_{m+l} = k | X_m = j\}$, the so-called *$l$-step transition probabilities.* The *1-step transition probabilities* are simple denoted $p_{j,k}$ (the parameter 1 is omitted). The 0-step probabilities are defined as $p_{j,k}(0) = 1$, whenever $j = k$, and 0 elsewhere. The initial distribution $\underline{\pi}(0)$ of the DTMC is defined as $\underline{\pi}(0) = (\pi_0(0), \cdots, \pi_{|\mathcal{I}|}(0))$. By iteratively applying the rule for conditional probabilities, it can easily be seen that

$$\Pr\{X_0 = i_0, X_1 = i_1, \cdots, X_n = i_n\} = \pi_{i_0}(0) \cdot p_{i_0,i_1} \cdots p_{i_{n-1},i_n}. \tag{1}$$

This implies that the DTMC is totally described by the initial probabilities and the 1-step probabilities $p_{i,j}$. The 1-step probabilities are summarised in the *state-transition probability matrix* $\mathbf{P} = (p_{i,j})$. The matrix $\mathbf{P}$ is a *stochastic matrix* because all its entries $p_{i,j}$ satisfy $0 \leq p_{i,j} \leq 1$, and $\sum_j p_{i,j} = 1$, for all i.

A DTMC can be visualised as a labeled directed graph with the elements of $\mathcal{I}$ as vertices. A directed edge with label $p_{i,j}$ exists between vertices i and j whenever $p_{i,j} > 0$. Such representations of Markov chains are called *state* *transition diagrams.* Notice the similarity with the usual graphical notation for finite-state machines (FSMs). In fact, a DTMC can be viewed as an FSM in which the successor function is specified in a probabilistic manner, that is, given state i, the next state will be state j with probability $p_{i,j}$.

Example 1. Graphical representation of a DTMC. In Figure 4 we show the state transition diagram for the DTMC with state-transition probability matrix

$$\mathbf{P} = \frac{1}{10}\begin{pmatrix} 6 & 2 & 2 \\ 1 & 8 & 1 \\ 6 & 0 & 4 \end{pmatrix}. \tag{2}$$

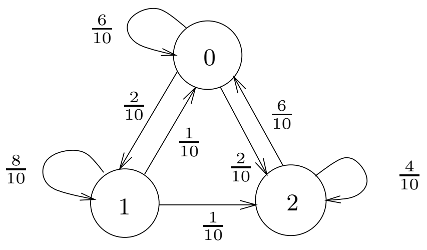

Fig. 4. State transition diagram for the example DTMC (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

2.2 Transient and Steady-State Probabilities

We can now proceed to calculate the 2-step probabilities of a DTMC with state-transition probability matrix $\mathbf{P}$. We have

$$p_{i,j}(2) = \Pr\{X_2 = j | X_0 = i\} = \sum_{k \in \mathcal{I}} \Pr\{X_2 = j, X_1 = k | X_0 = i\}, \tag{3}$$

since in going from state i to state j in two steps, any state $k \in \mathcal{I}$ can be visited as intermediate state. Now, due to the rule of conditional probability as well as the Markov property, we can write

$$\begin{aligned} p_{i,j}(2) &= \sum_{k \in \mathcal{I}} \Pr\{X_2 = j, X_1 = k | X_0 = i\} \\ &= \sum_{k \in \mathcal{I}} \Pr\{X_1 = k | X_0 = i\} \Pr\{X_2 = j | X_1 = k, X_0 = i\} \\ &= \sum_{k \in \mathcal{I}} \Pr\{X_1 = k | X_0 = i\} \Pr\{X_2 = j | X_1 = k\} \\ &= \sum_{k \in \mathcal{I}} p_{i,k} p_{k,j}. \end{aligned} \tag{4}$$

In the last summation we recognise the matrix product. Thus, the 2-step probabilities $p_{i,j}(2)$ are elements of the matrix $\mathbf{P}^2$. The above derivation can be applied iteratively, yielding the n-step probabilities $p_{i,j}(n)$ as elements of the matrix $\mathbf{P}^n$. For the 0-step probabilities we find the matrix $\mathbf{I} = \mathbf{P}^0$. The equation that establishes a relation between the $(m+n)$-step probabilities and the m- and n-step probabilities, that is,

$$\mathbf{P}^{m+n} = \mathbf{P}^m \mathbf{P}^n, \tag{5}$$

is known as the *Chapman-Kolmogorov equation.*

The probability of residence in state j after n steps, that is $\pi_j(n)$, can be obtained by conditioning:

$$\begin{aligned}\pi_j(n) = \Pr\{X_n = j\} &= \sum_{i \in \mathcal{I}} \Pr\{X_0 = i\} \Pr\{X_n = j | X_0 = i\} \\ &= \sum_{i \in \mathcal{I}} \pi_i(0) p_{i,j}(n).\end{aligned} \tag{6}$$

Writing this in matrix-vector notation, with $\underline{\pi}(n) = (\pi_0(n), \pi_1(n), \cdots)$, we arrive at

$$\underline{\pi}(n) = \underline{\pi}(0)\mathbf{P}^n. \tag{7}$$

Since the index n in (7) can be interpreted as the step-count in the DTMC, this equation expresses the time-dependent or transient behaviour of the DTMC.

Example 2. Transient behaviour of a DTMC. Let us compute $\underline{\pi}(n) = \underline{\pi}(0)\mathbf{P}^n$ for $n = 1, 2, 3, \cdots$ with $\mathbf{P}$ as given in (2), and $\underline{\pi}(0) = (1, 0, 0)$. Clearly, $\underline{\pi}(1) = \underline{\pi}(0)\mathbf{P} = (0.6, 0.2, 0.2)$. Then, $\underline{\pi}(2) = \underline{\pi}(0)\mathbf{P}^2 = \underline{\pi}(1)\mathbf{P} = (0.50, 0.28, 0.22)$. We proceed with $\underline{\pi}(3) = \underline{\pi}(2)\mathbf{P} = (0.460, 0.324, 0.216)$. We observe that the successive values for $\underline{\pi}(n)$ seem to converge somehow, and that the elements of all the vectors $\underline{\pi}(n)$ always sum to 1.

For many DTMCs (but not all; we will discuss conditions in Section 2.4) all the rows in $\mathbf{P}^n$ converge to a common limit when $n \to \infty$. For the time being, we assume that such a limit indeed exists. Defining $\underline{v} = (\cdots, v_j, \cdots)$ as

$$v_j = \lim_{n\to\infty} \pi_j(n) = \lim_{n\to\infty} \Pr\{X_n = j\} = \lim_{n\to\infty} \sum_{i \in \mathcal{I}} \pi_i(0) p_{i,j}(n). \tag{8}$$

Writing this in matrix-vector notation, we obtain

$$\underline{v} = \lim_{n\to\infty} \underline{\pi}(n) = \lim_{n\to\infty} \underline{\pi}(0)\mathbf{P}^n. \tag{9}$$

However, we also have

$$\underline{v} = \lim_{n\to\infty} \underline{\pi}(n+1) = \lim_{n\to\infty} \underline{\pi}(0)\mathbf{P}^{n+1} = \left(\lim_{n\to\infty} \underline{\pi}(0)\mathbf{P}^n\right)\mathbf{P} = \underline{v}\mathbf{P}. \tag{10}$$

Hence, whenever the limiting probabilities $\underline{v}$ exist, they can be obtained by solving the system of linear equations

$$\underline{v} = \underline{v}\mathbf{P} \quad \Rightarrow \quad \underline{v}(\mathbf{I} - \mathbf{P}) = \underline{0}, \tag{11}$$

with, since $\underline{v}$ is a probability vector, $\sum_i v_i = 1$, and $0 \leq v_i \leq 1$. The equivalent form on the right, i.e., $\underline{v}(\mathbf{I} - \mathbf{P}) = \underline{0}$, will be discussed in Section 4 in relation to CTMCs.

The vector $\underline{v}$ is called the *stationary* or *steady-state probability vector* of the DTMC, which, for the DTMCs we will encounter, will most often uniquely exist. Furthermore, in most of the practical cases we will encounter, this steady-state probability vector will be independent of the initial state probabilities $\underline{\pi}(0)$.

Example 3. Steady-state probability vector calculation. Let us compute $\underline{v} = \underline{v}\mathbf{P}$ with $\mathbf{P}$ as in (2) and compare it to the partially converged result obtained there. Denoting $\underline{v} = (v_0, v_1, v_2)$ we derive from the system of three linear equations that $v_0 = v_1$ and $v_2 = v_0/2$. Using the fact that $v_0 + v_1 + v_2 = 1$ (normalisation) then gives us $\underline{v} = (\frac{4}{10}, \frac{4}{10}, \frac{2}{10})$.

The steady-state probabilities can be interpreted in two ways. One way is to see them as the long-run proportion of time the DTMC "spends" in the respective states. The other way is to regard them as the probabilities that the DTMC would be in a particular state if one would take a snapshot after a very long time. It is important to note that for large values of n state changes do still take place!

2.3 State-Residence Time Distribution

The matrix $\mathbf{P}$ describes the 1-step state transition probabilities. If, at some time instance n, the state of the DTMC is i, then, at time instance $n + 1$, the state will still be i with probability $p_{i,i}$, and will be $j \neq i$ with probability $1-p_{i,i} = \sum_{j\neq i} p_{i,j}$. For time instance $n+1$, a similar reasoning holds, so that the probability of still residing in state i at time instance $n+2$ (given residence there at time instance n and $n + 1$) equals $p_{i,i}^2$. Taking this further, the probability to reside in state i for exactly m consecutive epochs equals $(1 - p_{i,i})p_{i,i}^{m-1}$, that is, there are $m - 1$ steps in which the possibility (staying in i) with probability $p_{i,i}$ is taken, and one final step with probability $1-p_{i,i}$ where indeed a step towards another state $j \neq i$ is taken. Interpreting leaving state i as a success and staying in state i as a failure, i.e., a failure to leave, we see that the state residence times in a DTMC obey a *geometric distribution.* The expected number of epochs in state i then equals $1/(1-p_{i,i})$ and the variance of the number of epochs in state i then equals $p_{i,i}/(1 - p_{i,i})^2$.

The fact that the state residence times in a DTMC are geometrical distributions need not be a surprise. From the Markov property, we know that only the actual state, at any time instance, is of importance in determining the future, irrespective of the residence time in that state. The geometric distribution is the only discrete distribution exhibiting this *memoryless property*, that is, when a random variable M is geometrically distributed the following holds:

$$\Pr\{M = n + m | M > n\} = \Pr\{M = m\}, \quad m \geq 1.$$

2.4 Convergence Properties

Previously, we stated that the steady-state probability distribution of a DTMCs can be determined when the DTMC fulfills certain conditions. In this section we discuss concisely a number of properties of DTMCs that help us in deciding whether a DTMC has a unique steady-state probability distribution or not.

Let us start with a classification of the states in a DTMC. A state $j \in \mathcal{I}$ is said to be *reachable* from state $i \in \mathcal{I}$ if, for some value n, $p_{i,j}(n) > 0$, which means that there is a step number for which there is a nonzero probability of going from state i to j. For such a pair of states, we write $i \rightarrow j$. If $i \rightarrow j$ and $j \rightarrow i$, then i and j are said to be *communicating states*, denoted $i \sim j$. Clearly, the communicating relation ($\sim$) is (i) transitive: if $i \sim j$ and $j \sim k$ then $i \sim k$, (ii) symmetric: by its definition in terms of $\rightarrow$, $i \sim j$ is equivalent to $j \sim i$, and (iii) reflexive: for $n = 0$, we have $p_{i,i}(0) = 1$, so that $i \rightarrow i$ and therefore $i \sim i$. Consequently, $\sim$ is an *equivalence relation* which partitions the state space in communicating classes. If all the states of a DTMC belong to the same communicating class, the DTMC is said to be *irreducible*. If not, the DTMC is *reducible*.

The *period* $d_i \in \mathbb{N}$ of state i is defined as the greatest common divisor of those values n for which $p_{i,i}(n) > 0$. When $d_i = 1$, state i is said to be *aperiodic*, in which case, at every time step there is a non-zero probability of residing in state i. It has been proven that within a communicating class all states have the same period. Therefore, one can also speak of periodic and aperiodic communicating classes, or, in case of an irreducible DTMC, of an *aperiodic* or *periodic DTMC*.

A state i is said to be *absorbing* when $\lim_{n\rightarrow\infty} p_{i,i}(n) = 1$. Recall that for an absorbing state i we have $\sum_{j \neq i} p_{i,j} = 0$. When there is only a single absorbing state, the DTMC will, with certainty, reach that state for some value of n.

A state is said to be *transient* or *non-recurrent* if there is a nonzero probability that that state is not visited again at some point in the future. If this is not the case, the state is said to be recurrent. For *recurrent* states, we can address the time between successive visits. Let $f_{i,j}(n)$ denote the probability that exactly n steps after leaving state i, state j is visited for the first time. Consequently, $f_{i,i}(n)$ is the probability that exactly n steps are taken between two successive visits to state i. Defining $f_{i,i} = \sum_n f_{i,i}(n)$, it follows that if $f_{i,i} = 1$, then state i is recurrent. If state i is nonrecurrent then $f_{i,i} < 1$. In the case $f_{i,i} = 1$ we can make a further classification based upon the mean recurrence time m_i of state i, defined as $m_i = \sum_{n=1}^{\infty} n f_{i,i}(n)$. A recurrent state i is said to be *positive recurrent* (or recurrent non-null) if the mean recurrence time m_i is finite. If m_i is infinite, state i is said to be *null recurrent*.

Having defined the above properties, the following theorem expresses when a DTMC has a (unique) steady-state probability distribution.

Theorem 1. Steady-state probability distributions in a DTMC. *In an irreducible and aperiodic DTMC with positive recurrent states:*

- *for all j, the limiting probability $v_j = \lim_{n\to\infty} \pi_j(n) = \lim_{n\to\infty} p_{i,j}(n)$ does exist;*
- *$\underline{v}$ is independent of the initial probability distribution $\underline{\pi}(0)$;*
- *$\underline{v}$ is the unique steady-state probability distribution.*

In typical performance and dependability models, the Markov chains will be irreducible and aperiodic. When dealing with continuous-time Markov chains, similar conditions apply as for DTMC.

3 Semi-Markov Chains

We define semi-Markov chains in Section 3.1 and give an alternative interpretation of their dynamics in Section 3.2.

3.1 SMCs as Generalisation of DTMCs

We can leave the discrete-time domain and move to the continuous-time domain by associating with every state in a DTMC a positive residence time distribution $F_i(t)$ and density $f_i(t)$. In doing so, we end up with a *semi-Markov chain* (SMC), which is fully described by the matrix with 1-step probabilities $\mathbf{P}$ (as known from DTMCs), the initial probability vector $\underline{\pi}(0)$ and the vector of state residence distributions $\underline{F}(t) = (F_1(t), \cdots, F_{|\mathcal{I}|}(t))$. A simple interpretation of an SMC is the following. At epochs when the state changes take place (transition epochs), the SMC behaves as a DTMC in the sense that the state changes are completely governed by the state transition probability matrix $\mathbf{P}$, and are independent of the past. When state i is entered, a random amount of time has to be passed, distributed according to $F_i(t)$, before the next state transition takes place.

To obtain the steady-state probabilities of an SMC, we first solve the steady-state probabilities for the so-called *embedded DTMCs* characterisedby $\mathbf{P}$. Since the SMC behaviour at transition epochs is the same as for this DTMC, we can compute the steady-state probabilities $\underline{v}$ for the embedded DTMC in the usual way. Now, we have to compute the average state residence times h_i for all states i in the SMC. We do this directly from the state residence time distributions:

$$h_i = \int_0^\infty t f_i(t) dt.$$

We then obtain the steady-state probabilities in the SMC by taking these residence times into account, as follows:

$$\pi_i = \frac{v_i h_i}{\sum_j v_j h_j}, \quad \text{for all } i.$$

Note that for the steady-state probabilities of the SMC only the mean state residence times h_i are of importance. Hence, in many applications, these mean values are given directly, so that the explicit integration above does not need to be performed.

The computation of transient state probabilities for an SMC is far more complex than for DTMCs (and for CTMCs). A derivation of the relevant equations as well as their solution go beyond the scope of this paper, but can be found in [48].

3.2 Alternative View on SMCs

We can also view an SMCs in a slightly different, but equivalent, way. Consider a DTMC in which the transition probabilities are dependent on the time already spent in (current) state i since the last entrance there, but not on states visited before entering state i nor on any previous residence times. Thus, we deal with a time-dependent probability matrix $\mathbf{K}(t)$ known as the *kernel of the SMC*, where an entry $k_{i,j}(t)$ provides the probability that, after having entered state i, it takes at most t time units to switch to state j, given that no transition to any other state takes place.

From $\mathbf{K}(t)$, we can derive two well-known other quantities. First of all, the limit $p_{i,j} = \lim_{t\to\infty} k_{i,j}(t)$ expresses the probability that once state i has been entered, the next state will be j. The thus resulting probabilities indeed coincide with the entries of $\mathbf{P}$ for the embedded DTMC in Section 3.1. Furthermore, the state residence time distribution $F_i(t)$ can be written as $F_i(t) = \sum_j k_{i,j}(t)$.

Hence, once $\mathbf{K}(t)$ is known, both $\mathbf{P}$ and $\underline{F}(t)$ can be derived and the solution procedure of Section 3.1 can be applied.

4 Continuous-Time Markov Chains

In this section, we focus on continuous-time Markov chains (CTMCs). We first present how CTMCs can be seen as generalisations of DTMCs, by enhancing them with negative exponential state residence time distributions in Section 4.1. We then present the evaluation of the steady-state and transient behaviour of CTMCs in Section 4.2.

4.1 From DTMC to CTMC

In DTMCs time progresses in abstract steps. With CTMCs, as for SMCs, we associate positive state-residence time distributions with each state; hence we address Markov chains in continuous-time. In SMCs, we associated general residence time distributions with states. As a result, the state transition probability matrix $\mathbf{K}(t)$ became time dependent, so that the *complete state* of an SMC is given by the current state number i *and* the time already spent in that state (denoted in the sequel as t_{res}).

With CTMCs, we strive for a considerably more simple notion of state. We will chose the state residence time distribution such that the current state index i describes the state of the chain completely. This can only be achieved when the chosen state residence time distribution is memoryless, so that it does not matter what the actual value of t_{res} is. In doing so, the Markov property is valid, and reads for the case at hand, for all non-negative $t_0 < t_1 < \cdots < t_{n+1}$ and $x_0, x_1, \cdots, x_{n+1}$:

$$\begin{array}{c} \Pr\{X(t_{n+1}) = x_{n+1} | X(t_0) = x_0, \cdots, X(t_n) = x_n\} \\ = \\ \Pr\{X(t_{n+1}) = x_{n+1} | X(t_n) = x_n\}, \end{array} \tag{12}$$

hence, the probability distribution for the $(n+1)$-th state residence time only depends on the current (n-th) state and neither on the time passed in the current state, nor on states visited previously.

The only memoryless continuous-time distribution is the exponential distribution. Thus, we have to associate with every state i in a CTMC a parameter μ_i describing the rate of an exponential distribution; the residence time distribution in state i then equals

$$F_i(t) = 1 - e^{-\mu_i t}, \ t \geq 0. \tag{13}$$

The vector $\underline{\mu} = (\cdots, \mu_i, \cdots)$ thus describes the state residence time distributions in the CTMC. To be precise, this vector describes the rates of these distributions, but these rates fully characterise the distribution. We can still use the state transition probability matrix $\mathbf{P}$ to describe the state transition behaviour. The initial probabilities remain $\underline{\pi}(0)$. The dynamics of the CTMC can now be interpreted as follows. When state i is entered, this state will remain for a random amount of time, distributed according to the state residence distribution $F_i(t)$. After this delay, a state change to state j will take place with probability $p_{i,j}$. To ease understanding at this point, assume that $p_{i,i} = 0$ for all i; we come back to this issue below.

Instead of associating with every state just one negative exponentially distributed delay, it is also possible to associate as many delays with a state as there are transition possibilities. We therefore define the matrix $\mathbf{Q}$ with $q_{i,j} = \mu_i p_{i,j}$, in case $i \neq j$, and $q_{i,i} = -\sum_{j \neq i} q_{i,j} = -\mu_i$. Since $p_{i,i} = 0$, we have $q_{i,i} = -\mu_i$. Using this notation allows for the following interpretation. When entering state i, for those states j that can be reached from i, i.e., for those with $q_{i,j} > 0$, a random variable is thought to be drawn, according to the (negative exponential) distributions $F_{i \to j}(t) = 1 - e^{-q_{i,j} t}$. These distributions model the delay perceived in state i when going to j. One of the "drawn" delays will be the smallest, meaning that the transition corresponding to that delay will actually occur before any of the others (race condition: the faster one wins). This interpretation is correct due to the special properties of the negative exponential distribution. Let us first address the state residence times. In state i, the time it takes to reach state j is exponentially distributed with rate $q_{i,j}$. When there is more than one possible successor state, the next state will be such that the residence time in state i is minimised (race condition). However, the minimum value of a set of of exponentially distributed random variables with rates $q_{i,j}$ $(j \neq i)$ is again an exponentially distributed random variable, with as rate the sum $\sum_{j \neq i} q_{i,j}$ of the original rates. This sum is, however, exactly equal to the rate μ_i of the residence time in state i.

A second point to verify is whether the state-transition behaviour is still the same. In general, if we have n negative exponentially distributed random variables X_k (with rates l_k), then X_i will be the minimum of them with probability $l_i/\sum_k l_k$. In our case, we have a number of competing delays when starting from state i, which are all negative exponentially distributed random variables (with rates $q_{i,j}$). The shortest one will then lead to state j with probability

$$\frac{q_{i,j}}{\sum_{k \neq i} q_{i,k}} = \frac{p_{i,j}\mu_i}{\mu_i} = p_{i,j}, \tag{14}$$

so that also the state-transition behaviour is as required.

Let us now discuss the case where $p_{i,i} > 0$, that is, the case where, after having resided in state i for an exponentially distributed period of time (with rate μ_i), there is a positive probability of staying in i for another period. In particular, we have seen in Section 2 that the state residence distributions in a DTMC obey a geometric distribution (measured in "visits"), with mean $1/(1-p_{i,i})$ for state i. Hence, if we decide that the expected state residence *time* in the CTMC constructed from the DTMC is $1/\mu_i$, the time spent in state i *per visit* should on average be $(1-p_{i,i})/\mu_i$. Hence, the rate of the negative exponential distribution associated with that state should equal $\mu_i/(1-p_{i,i})$. Using this rate in the above procedure, we find that we have to assign the following transition rates for $j \neq i$:

$$q_{i,j} = \frac{\mu_i p_{i,j}}{1-p_{i,i}} = \mu_i \frac{p_{i,j}}{1-p_{i,i}} = \mu_i \Pr\{\text{jump } i \to j | \text{leave } i\}, \quad j \neq i, \tag{15}$$

that is, we have renormalised the probabilities $p_{i,j}$ $(j \neq i)$ such that they make up a proper distribution. To conclude, if we want to associate a negative exponential residence time with rate μ_i to state i, we can do so by just normalising the probabilities $p_{i,j}$ $(j \neq i)$ appropriately.

4.2 Evaluating the Steady-State and Transient Behaviour

CTMCs can be depicted conveniently using state-transition diagrams. i.e., as labelled directed graphs, with the states of the CTMC represented by the vertices and an edge between vertices i and j $(i \neq j)$ whenever $q_{i,j} > 0$. The edges in the graph are labelled with the corresponding rates.

Formally, a CTMC can be described by an *(infinitesimal) generator matrix* $\mathbf{Q} = (q_{i,j})$ and initial state probability vector $\underline{\pi}(0)$. Denoting the system state at time $t \in \mathcal{T}$ as $X(t)$, we have, for $h \to 0$:

$$\Pr\{X(t+h) = j | X(t) = i\} = \Pr\{X(h) = j | X(0) = i\} = q_{i,j}h + \mathbf{o}(h), \quad i \neq j, \tag{16}$$

where $\mathbf{o}(h)$ is a term that goes to zero faster than h. This result follows because the CTMC is time-homogeneous and the fact that the state residence times are negative exponentially distributed; in fact, (16) represents the first-order

Taylor/MacLaurin series expansion of $1-e^{-q_{i,j}h}$ around 0. The value $q_{i,j}$ $(i \neq j)$ is the rate at which the current state i changes to state j. Denote with $\pi_i(t)$ the probability that the state at time t equals i: $\pi_i(t) = \Pr\{X(t) = i\}$. Given $\pi_i(t)$, we can compute the evolution of the Markov chain in the period $[t, t+h)$ as follows:

$$\begin{aligned}\pi_i(t+h) &= \pi_i(t) \Pr\left\{\begin{array}{c}\text{do not depart from } i\\ \text{during } [t,t+h)\end{array}\right\} + \sum_{j \neq i} \pi_j(t) \Pr\left\{\begin{array}{c}\text{go from } j \text{ to } i\\ \text{during } [t,t+h)\end{array}\right\}\\ &= \pi_i(t)\left(1 - \sum_{j\neq i} q_{i,j}h\right) + \left(\sum_{j \neq i} \pi_j(t) q_{j,i}\right) h + \mathbf{o}(h). \qquad (17)\end{aligned}$$

Now, using the earlier defined notation $q_{i,i} = -\sum_{j\neq i} q_{i,j}$, we have

$$\pi_i(t+h) = \pi_i(t) + \left(\sum_{j\in\mathcal{I}} \pi_j(t) q_{j,i}\right) h + \mathbf{o}(h). \qquad (18)$$

Rearranging terms, dividing by h and taking the limit $h \to 0$, we obtain

$$\pi_i'(t) = \lim_{h\to 0} \frac{\pi_i(t+h) - \pi_i(t)}{h} = \sum_{j \in \mathcal{I}} q_{j,i}\pi_j(t), \qquad (19)$$

which in matrix-vector notation has the following form:

$$\underline{\pi}'(t) = \underline{\pi}(t)\mathbf{Q}, \quad \text{given } \underline{\pi}(0). \qquad (20)$$

We thus find that the time-dependent or *transient state probabilities* in a CTMC are described by a system of linear differential equations.

In many cases, however, the *transient behaviour* $\underline{\pi}(t)$ of the Markov chain is more than we really need. For performance evaluation purposes we are often already satisfied when we are able to compute the long-term or *steady-state probabilities* $\pi_i = \lim_{t\to\infty} \pi_i(t)$. When we assume that a steady-state distribution exists, this implies that the above limit exists, and thus that $\lim_{t\to\infty} \pi_i'(t) = 0$. Consequently, for obtaining the steady-state probabilities we only need to solve the system of linear equations:

$$\underline{\pi}\mathbf{Q} = \underline{0}, \quad \sum_{i\in\mathcal{I}} \pi_i = 1. \qquad (21)$$

The right part (normalisation) is added to ensure that the obtained solution is indeed a probability vector; the left part alone has infinitely many solutions, which upon normalisation all yield the same probability vector.

Note that the equation $\underline{\pi}\mathbf{Q} = \underline{0}$ is of the same form as the equation $\underline{v} = \underline{v}\mathbf{P}$ we have seen for DTMCs. Since this latter equation can be rewritten as $\underline{v}(\mathbf{P}-\mathbf{I}) = \underline{0}$, the matrix $(\mathbf{P} - \mathbf{I})$, as already encountered in (11), can be interpreted as a generator matrix.

Note that we can also solve the steady-state probabilities of a CTMC by seeing it as a special case of an SMC, with embedded DTMC described by the probabilities $p_{i,j} = q_{i,j}/|q_{i,i}|$ and mean state residence times $h_i = |q_{i,i}|^{-1}$.

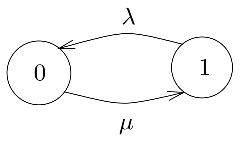

Fig. 5. A simple 2-state CTMC (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

Example 4. Evaluation of a 2-state CTMC. Consider a component that is either operational or not. The time it is operational is exponentially distributed with mean $1/\lambda$. The time it is not operational is also exponentially distributed, with mean $1/\mu$. Signifying the operational state as state "1", and the down state as state "0", we can model this system as a 2-state CTMC with generator matrix $\mathbf{Q}$ as follows:

$$\mathbf{Q} = \begin{pmatrix} -\mu & \mu \\ \lambda & -\lambda \end{pmatrix}.$$

Furthermore, it is assumed that the system is initially fully operational so that $\underline{\pi}(0) = (0, 1)$. In Figure 5 we show the corresponding state-transition diagram. Solving (21) yields the following steady-state probability vector:

$$\underline{\pi} = \left(\frac{\lambda}{\lambda + \mu}, \frac{\mu}{\lambda + \mu} \right). \tag{22}$$

This probability vector can also be computed from the embedded DTMC which is given as:

$$\mathbf{P} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Solving for $\underline{v}$ yields us $\underline{v} = (\frac{1}{2}, \frac{1}{2})$, indicating that both states are visited equally often. However, these visits are not equally long. Incorporating the mean state residence times, being respectively $1/\mu$ and $1/\lambda$, yields

$$\underline{p} = \left(\frac{\frac{1}{2}\frac{1}{\mu}}{\frac{1}{2}\left(\frac{1}{\mu} + \frac{1}{\lambda}\right)}, \frac{\frac{1}{2}\frac{1}{\lambda}}{\frac{1}{2}\left(\frac{1}{\mu} + \frac{1}{\lambda}\right)} \right) = \left(\frac{\lambda}{\lambda + \mu}, \frac{\mu}{\lambda + \mu} \right), \tag{23}$$

which is the solution we have seen before.

For the transient behaviour of the CTMC we have to solve the corresponding system of linear differential equations. Although this is difficult in general, for this specific example we can obtain the solution explicitly. We obtain (see also Section 6):

$$\underline{\pi}(t) = \underline{\pi}(0) e^{\mathbf{Q}t}, \tag{24}$$

from which we can derive

$$\pi_0(t) = \frac{\lambda}{\lambda+\mu} - \frac{\lambda}{\lambda+\mu}e^{-(\lambda+\mu)t},$$
$$\pi_1(t) = \frac{\mu}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu}e^{-(\lambda+\mu)t}. \quad (25)$$

Notice that $\pi_0(t) + \pi_1(t) = 1$ (for all t) and that the limit of the transient solutions for $t \to \infty$ indeed equals the steady-state probability vectors derived before. In Figure 6 we show the transient and steady-state behaviour of the 2-state CTMC for $3\lambda = \mu = 1$.

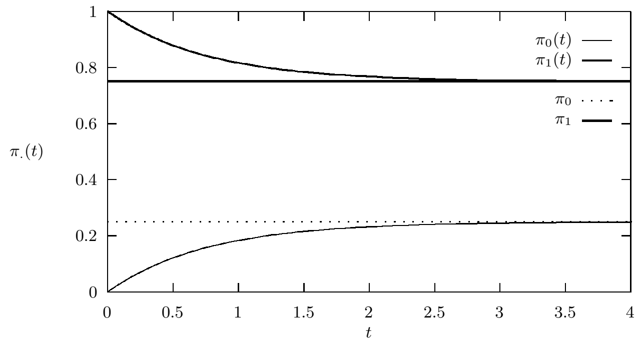

Fig. 6. Steady-state and transient behaviour of a 2-state CTMC (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

Example 5. Availability evaluation of a fault-tolerant system. Consider a fault-tolerant computer system consisting of three computing nodes and a single voting node. The three computing nodes generate results after which the voter decides upon the correct value (by selecting the answer that is given by at least two computing nodes). Such a fault-tolerant computing system is also known as a triple-modular redundant system (TMR). The failure rate of a computing node is λ and of the voter ν failures per hour (fph). The expected repair time of a computing node is $1/\mu$ and of the voter is $1/\delta$ hours. If the voter fails, the whole system is supposed to have failed and after a repair (with rate δ) the system is assumed to start "as new". The system is assumed to be operational when at least two computing nodes and the voter are functioning correctly.

To model the availability of this system as a CTMC, we first have to define the state space: $\mathcal{I} = \{(3,1),(2,1),(1,1),(0,1),(0,0)\}$, where state (i,j) specifies that

i computing nodes are operational as well as j voters. Note that the circumstance of the computing nodes does not play a role any more as soon as the voter goes down; after a repair in this down state the whole system will be fully operational, irrespective of the past state. Using the above description, the state-transition diagram can be drawn easily, as given in Figure 7. The corresponding generator matrix is given as:

$$\mathbf{Q} = \begin{pmatrix} -(3\lambda+\nu) & 3\lambda & 0 & 0 & \nu \\ \mu & -(\mu+2\lambda+\nu) & 2\lambda & 0 & \nu \\ 0 & \mu & -(\mu+\lambda+\nu) & \lambda & \nu \\ 0 & 0 & \mu & -(\mu+\nu) & \nu \\ \delta & 0 & 0 & 0 & -\delta \end{pmatrix}. \quad (26)$$

We assume that the system is fully operational at $t = 0$. The following numerical parameters are given: $\lambda = 0.01$ fph, $\nu = 0.001$ fph, $\mu = 1.0$ repairs per hour (rph) and $\delta = 0.2$ rph.

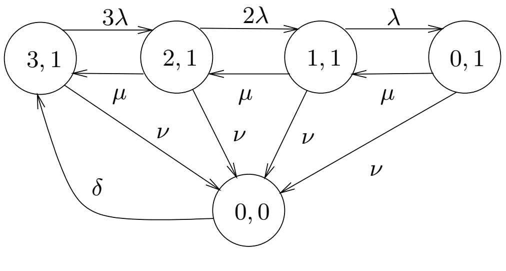

Fig. 7. CTMC for the TMR system (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

We can now compute the steady-state probabilities by solving the linear system $\underline{\pi}\mathbf{Q} = \underline{0}$ under the condition that $\sum_i \pi_i = 1$ (see Section 5) which yields the following values (note that we use the tuple (i, j) as state index here):

(i,j)	$(3,1)$	$(2,1)$	$(1,1)$	$(0,1)$	$(0,0)$
$\pi_{(i,j)}$	9.6551×10^{-1}	2.8936×10^{-2}	5.7813×10^{-4}	5.7755×10^{-6}	4.9751×10^{-3}

The probability that the system is operational can thus be computed as 0.99444. Although this number looks very good (it is very close to 100%) for a non-stop transaction processing facility, it would still mean an expected down-time of 48.7 hours a year ($(1 - 0.99444) \times 24 \times 365$).

The transient behaviour of this small CTMC can be obtained by numerically solving the differential equation for $\underline{\pi}(t)$ with a technique known as uniformisation (see Section 6). In Figure 8 we show the probability $\pi_{(3,1)}(t)$ for the first 10 hours of system operation. As can be observed, the transient probability reaches the steady-state probability relatively fast. A similar observation can be made for the other transient probabilities in Figure 9 (note the logarithmic scale of the vertical axis).

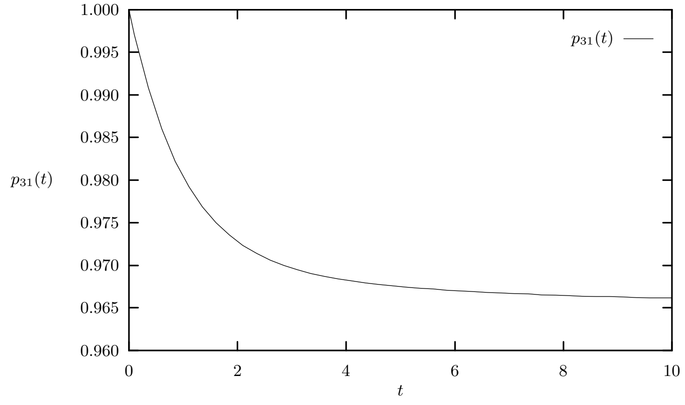

Fig. 8. Transient probability $\pi_{(3,1)}$ for the TMR system (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

Example 6. A finite-buffer queueing station. Consider a single server that accepts requests to be processed in first-come first-served order. The processing time is assumed to be exponentially distributed with mean $1/\mu$ and the interarrival times are exponentially distributed with mean $1/\lambda$. An arrival process in which the interarrival times are independent and negative exponentially distributed is called a *Poisson process*. The number of arrivals taking place over a finite time interval $[0, t)$ in a Poisson process with rate λ follows a Poisson distribution with mean λt; $\Pr\{n \text{ arrivals in } [0,t)\} = e^{-\lambda t}\frac{(\lambda t)^n}{n!}$, $n \in I\!N$, named after Professor Siméon Denis Poisson, who lived in France from 1781 through 1840. Being an excellent mathematician, he published largely over 300 articles, devoted to a wide variety of topics. His name is attached to a wide area of concepts, e.g., as in the probability-related examples above, but also in the Poisson integral, the Poisson equation for potential energy and Poisson's constant in electricity.

The state of the server is, due to the involved memoryless distributions, completely given by the number of requests in the server. If we assume that the server can hold at most K requests, (including the one actually being processed) the state of the server is governed by a CTMC, as given in Figure 10. In fact, we are dealing here with the CTMC underlying the so-called M|M|1|K queue, in which both the interarrival and service times are memoryless (explaining the two "M"s), hence negative exponentially distributed, there is 1 server and there

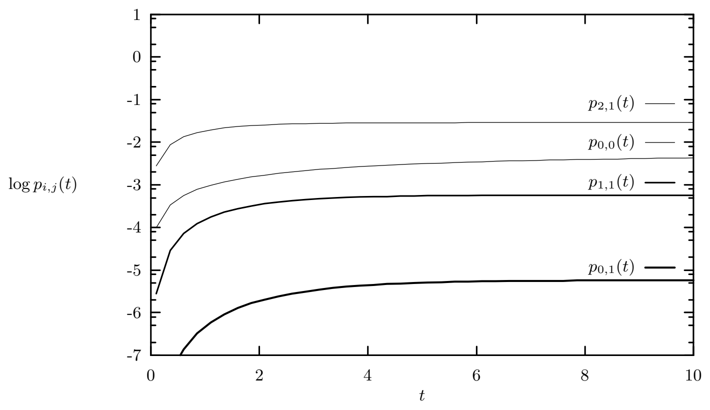

Fig. 9. Transient probabilities $\pi_{(i,j)}$ for the TMR system (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

are K buffer spots (including the server itself). The notation employed here to denote the particular queueing system is due to D.G. Kendall [43] (professor emeritus of Oxford University since 1989).

By the fact that the CTMC has a structure in which only left and right "neighbouring" states can be reached, this type of CTMC is called a *birth-death process*. The $(K+1)\times(K+1)$ generator matrix for the CTMC is given as follows:

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

The special *tridiagonal* structure of $\mathbf{Q}$ is typical for birth-death processes. Exploiting the birth-death structure of the CTMC, we can solve the equation $\underline{\pi}\mathbf{Q} = \underline{0}$ explicitly to reveal that $\pi_i = \pi_0 \cdot \rho,\ i = 1, \cdots, K$. Here, $\rho = \lambda/\mu$ is the ratio of the arrival rate and the service rate, which is also called the *traffic intensity* or the *utilisation.* We observe that all steady-state probabilities are related to the probability that the server is empty (π_0). The normalisation $\sum_i \pi_i = 1$ then yields π_0, in the following way:

$$\pi_0 \sum_{i=1}^{K} \rho^i = 1 \quad \Rightarrow \quad \pi_0 = \frac{1-\rho}{1-\rho^{K+1}},$$

where the latter equality follows from the *geometric series*: $\sum_{i=0}^{K} a^i = (1 - a^{K+1})/(1-a)$ (with $a > 0, a \neq 1$). Note at this point that only the steady-state probabilities can be obtained explicitly; the transient probabilities can only be obtained numerically.

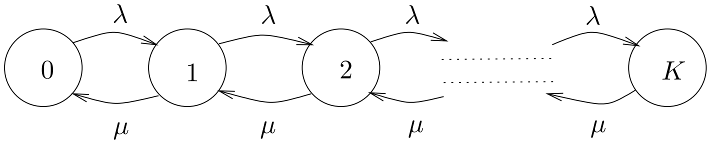

Fig. 10. CTMC underlying the M|M|1|K queue

Example 7. An infinite-buffer queueing station. We can extend the previous example by making the buffering capacity of the server unbounded. Surprisingly, a closed-form solution for the steady-state probabilities then still exists. The state space of the corresponding CTMC then equals $\mathbb{N}$ and we still have $\pi_i = \pi_0\rho$. Furthermore, if we require $\lambda < \mu$, i.e., the average number of requests arriving per unit of time is smaller than the average number of jobs that can be handled per unit of time, we have $\rho < 1$, so that π_i becomes smaller for increasing i. Moreover, the sum $\sum_{i=0}^{\infty} \rho^i = (1-\rho)^{-1}$, so that we find for all $i \in \mathbb{N}$: $\pi_i = (1-\rho)\rho^i$ (which is a geometric distribution). Furthermore, we can simply obtain a closed-form solution for the mean number of requests in the server:

$$E[N] = \sum_{i=0}^{\infty} i\pi_i = \sum_{i=0}^{\infty} i(1-\rho)\rho^i = \frac{\rho}{1-\rho}, \quad 0 \leq \rho < 1.$$

This example shows that, provided a regular structure exists in the Markov chain, steady-state probabilities can still be obtained explicitly, even if the state space is infinitely large. For more information on this topic, refer for instance to [27,67].

Example 8. Erlang's loss model. As stated in the introduction, Erlang studied Markovian models of telephone exchanges. In Kendall's notation, his model can now be described as an M|M|K|K queueing model, in which calls arrive according to a Poisson process with rate λ and take an exponentially distributed time with length $1/\mu$. Furthermore, the telephone switch considered can accommodate K simultaneous calls ("there are K lines") and cannot put calls on hold. Clearly, when all K lines are busy, an arriving call will be lost; the caller will hear a busy tone. The problem to be solved then, is to compute the required number of lines K so that, given traffic characteristics in terms of λ and μ, the call loss probability remains under some threshold.

Erlang's model describes a birth-death process, as illustrated in Figure 11, where the state number denotes the number of calls in progress. The call rate λ is constant for all states. The service rate linearly depends on the number of calls underway. For this birth-death process, the matrix $\mathbf{Q}$ has again a tridiagonal structure and we can easily solve the steady-state probabilities explicitly. Defining $\rho = \lambda/\mu$, we find:

$$\pi_i = \pi_0 \frac{\rho^i}{i!}, \quad i = 0, \cdots, K, \quad \text{with } \pi_0 = \left(\sum_{j=0}^{K} \frac{\rho^j}{j!} \right)^{-1},$$

where the expression for π_0 follows from the normalisation equation. The probability that an arriving call is lost, is now given by the probability for state K, that is:

$$\Pr\{\text{arriving call lost}; K, \rho\} = \frac{\rho^K/K!}{\sum_{i=0}^{K} \rho^i/i!}.$$

This result is also known as *Erlang's loss formula* $B(K, \rho)$. As part of his studies, Erlang published large tables with these loss probabilities, which were used to dimension telephone switches.

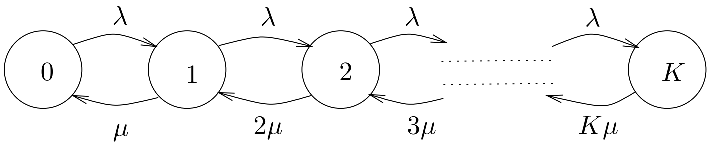

Fig. 11. CTMC underlying the M|M|K|K queue

5 Solution Methods for Steady-State Probabilities

As has become clear from the previous sections, in order to obtain the steady-state probabilities of finite DTMCs and CTMCs (with N states; numbered 1 through N) we need to solve a system of N linear equations which takes the following form (here given for a CTMC, but similar in the DTMC case):

$$\underline{\pi}\mathbf{Q} = \underline{0}, \quad \sum_{i=1}^{N} \pi_i = 1, \tag{27}$$

We assume here that the Markov chain is irreducible and aperiodic such that $\underline{\pi}$ does exist and is independent from $\underline{\pi}(0)$. Notice that the left part of (27) in fact does not uniquely define the steady-state probabilities; however, together with the normalisation equation a unique solution is found. For the explanations that follow, we will transpose the matrix $\mathbf{Q}$ and denote it as $\mathbf{A}$. Hence, we basically have to solve the following system of linear equations:

$$\mathbf{A}\underline{\pi}^T = \underline{b}, \quad \text{with} \quad \mathbf{A} = \mathbf{Q}^T \quad \text{and} \quad \underline{b} = \underline{0}^T. \tag{28}$$

Starting from this system of equations, two solution approaches can be chosen: *direct methods* or *iterative methods*. These methods will be discussed in Section 5.1 and 5.2, respectively.

5.1 Direct Methods

The main characteristic of a so-called direct method is that it aims at rewriting the system of equations in such a form that explicit expressions for the steady-state probabilities are obtained. The rewriting procedure costs an *a priori* known number of operations, given the number of states N.

Gaussian Elimination Perhaps the best-known direct solution technique is *Gaussian elimination*, named after the famous German mathematician Johann Carl Friedrich Gauss (1777–1855). The Gaussian elimination procedure consists of two phases: a reduction phase and a substitution phase.

In the *reduction phase* repetitive subtractions of equations from one another are used to make the system of equations upper-triangular (see also Figure 12). To do so, let the i-th equation be $\sum_j a_{i,j}p_j = 0$ (this equals $\sum_j p_j q_{j,i} = 0$ in the non-transposed system). We now vary i from 1 to N. The j-th equation, with $j = i+1, \cdots, N$, is now changed by subtracting the i-th equation $m_{j,i}$ times from it, where $m_{j,i} = a_{j,i}/a_{i,i}$, that is, we reassign the $a_{j,k}$ values as follows:

$$a_{j,k} := a_{j,k} - m_{j,i}a_{i,k}, \quad j,k > i.$$

Clearly, $a_{j,i} := a_{j,i} - m_{j,i}a_{i,i} = 0$, for all $j > i$. To avoid round-off errors, it is important to *set* $a_{j,i}$ to zero. By repeating this procedure for increasing i, the linear system of equations is transformed, in $N-1$ steps, to an upper-triangular system of equations. The element $a_{i,i}$ that acts as a divisor is called the *pivot.* If a pivot is encountered that equals 0, an attempt to divide by 0 results, which indicates that the system of equations does not have a solution. Since $\mathbf{Q}$ is a generator matrix of an irreducible ergodic CTMC, this problem will not occur. Moreover, since $\mathbf{A}$ is weakly diagonal dominant ($a_{i,i}$ is as large as the sum of all the values $a_{j,i}$ ($j \neq i$) in the same column) we have that $m_{j,i} < 1$ so that overflow problems are unlikely to occur.

At the end of the reduction phase, the N-th equation will always reduce to a trivial one ($0 = 0$). This is no surprise, since the system of equations without normalisation is not of full rank. We might even completely ignore the last equation. Since the right-hand side of the linear system of equations equals $\underline{0}$, nothing changes there either. When the right-hand side is a non-zero vector $\underline{b}$, we would have to set $b_j := b_j - m_{j,i}b_i$, for all $j > i$ in each step in the reduction process.

After the reduction has been performed, the *substitution phase* can start. The equation for π_N does not help us any further; we therefore assume a value $\alpha > 0$ for π_N, which can be substituted in the first $N-1$ equations, thus yielding a system of equations with one unknown less. We implement this by setting $b_j := b_j - a_{j,N}\pi_N$. Now, the $(N-1)$-th equation will have only one unknown left which we can directly compute as $\pi_{N-1} = b_{N-1}/a_{N-1,N-1}$. This new value can be substituted in the $N-2$ remaining equations, after which the $(N-2)$-th equation has only one unknown. This procedure can be repeated until all probabilities have been computed explicitly in terms of π_N (or α). We then use

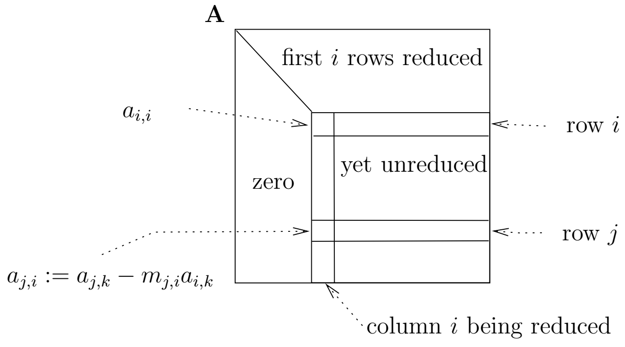

Fig. 12. Schematic representation of the i-th reduction step in the Gaussian elimination procedure (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

the normalisation equation to compute α to obtain the true probability vector, that is, we compute $\sigma = \sum_{i=1}^{N} \pi_i$ and set $\pi_i := \pi_i/\sigma$, for all i.

Instead of assuming the value α for π_N, we can also directly include the normalisation equation in the Gaussian elimination procedure. The best way to go then, is to replace the N-th equation with the equation $\sum_i \pi_i = 1$. In doing so, the last equation will directly give us π_N. The substitution phase can proceed as before.

Complexity Considerations for Gaussian Elimination The computational complexity for Gaussian elimination is $\mathcal{O}(N^3)$. By a more careful study of the algorithm, one will find that about $N^3/3 + N^2/2$ multiplications and additions have to be performed, as well as $N(N+1)/2$ divisions. Clearly, these numbers increase rapidly with increasing N. The main problem with Gaussian elimination, however, lies in its storage requirements. Although $\mathbf{A}$ will initially be sparse for most models, the reduction procedure normally increases the number of non-zeros in $\mathbf{A}$. At the end of the reduction phase, most entries of the upper half of $\mathbf{A}$ will be non-zero. The non-zero elements generated during this phase are called *fill-ins*. They can only be inserted efficiently when direct storage structures (arrays) are used. To store the upper-triangular matrix $\mathbf{A}$, $N^2/2$ floats have to be stored, each taking at least 8 bytes, plus 2 to 4 bytes for the corresponding indices. For moderately sized models, generated from some high-level model specification, N can easily be as large as 10^5 or even 10^6. This then precludes the use of Gaussian elimination. Fortunately, there are methods to compute $\underline{\pi}$ that do not change $\mathbf{A}$ and that are very fast as well. We will discuss these methods after we have discussed one alternative direct method.

LU Decomposition A method known as LU decomposition is advantageous to use when multiple systems of equations have to be solved, all of the form $\mathbf{A}\underline{x} = \underline{b}$, for different values of $\underline{b}$. This occurs, for instance when one tries to invert $\mathbf{A}$ by solving $\mathbf{A}\underline{x}_i = \underline{e}_i$, where the vectors $\underline{e}_i$ have as single nonzero element a 1 at the i-th position; the matrix $\mathbf{A}^{-1} = (\underline{x}_1, \underline{x}_2, \cdots)$.

The LU method starts by decomposing $\mathbf{A}$ such that it can be written as the *product* of two matrices $\mathbf{L}$ and $\mathbf{U}$, where the former is lower-triangular, and the latter is upper-triangular. We have:

$$\mathbf{A}\underline{x} = \underline{b} \quad \Rightarrow \quad \mathbf{L}\,\underbrace{\mathbf{U}\underline{x}}_{\underline{z}} = \underline{b}. \tag{29}$$

After the decomposition has taken place, we solve $\mathbf{L}\underline{z} = \underline{b}$, after which we solve $\mathbf{U}\underline{x} = \underline{z}$. Since the last two systems of equations are triangular, their solution can be found by a simple forward- and back-substitution.

The main question then lies in the computation of suitable matrices $\mathbf{L}$ and $\mathbf{U}$. Since $\mathbf{A}$ is the product of these two matrices, we know that

$$a_{i,j} = \sum_{k=1}^{N} l_{i,k} u_{k,j}, \quad i, j = 1, \cdots, N. \tag{30}$$

Given the fact that $\mathbf{L}$ and $\mathbf{U}$ are lower- and upper-triangular, we have to find $N^2 + N$ unknowns:

$$\begin{cases} l_{i,j}, \; i = 1, \cdots, N, \; k = 1, \cdots, i, \\ u_{k,j}, \, k = 1, \cdots, N, \, j = k, \cdots, N. \end{cases} \tag{31}$$

Since (30) only consists of N^2 equations, we have to assume N values to determine a unique solution. Two well-known schemes for this are [66]: (i) the Doolittle decomposition where one assumes $l_{i,i} = 1$, $i = 1, \cdots, N$; (ii) the Crout decomposition where one assumes $u_{i,i} = 1$, $i = 1, \cdots, N$.

Let us consider the Doolittle variant. First notice that in (30) many of the terms in the summation are zero, since one of the multiplicants is zero. In fact, we can rewrite (30) in a more convenient form as follows:

$$\begin{cases} i \leq j : a_{i,j} = u_{i,j} + \sum_{k=1}^{i-1} l_{i,k} u_{k,j}, \\ i > j : a_{i,j} = l_{i,j} u_{j,j} \sum_{k=1}^{j-1} l_{i,k} u_{k,j}. \end{cases} \tag{32}$$

From this system of equations, we can now iteratively compute the entries of $\mathbf{L}$ and $\mathbf{U}$ as follows:

$$\begin{cases} i \leq j : u_{i,j} = a_{i,j} - \sum_{k=1}^{i-1} l_{i,k} u_{k,j}, \\ i > j : l_{i,j} = \frac{1}{u_{j,j}} \left(a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} u_{k,j} \right), \end{cases} \tag{33}$$

by increasing i from 1 until N is reached.

Example 9. LU decomposition after Doolittle. Suppose we want to decompose

$$\mathbf{A} = \begin{pmatrix} 3 & 2 & 5 \\ -6 & 1 & 8 \\ -7 & 2 & -3 \end{pmatrix},$$

using a Doolittle LU decomposition. We then know that

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ \cdot & 1 & 0 \\ \cdot & \cdot & 1 \end{pmatrix} \text{ and } \mathbf{U} = \begin{pmatrix} \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot \\ 0 & 0 & \cdot \end{pmatrix}.$$

We start to compute $u_{1,1} = a_{1,1} = 3$. We then compute $l_{2,1} = a_{2,1}/u_{1,1} = -2$ and $u_{1,2} = a_{1,2} = 2$. From this, we find $u_{2,2} = a_{2,2} - l_{2,1}u_{1,2} = 5$. We then compute $l_{3,1} = -\frac{7}{3}$ and find $l_{3,2} = \frac{4}{3}$. Via $u_{1,3} = a_{1,3} = 5$ and $u_{2,3} = 18$ we find $u_{3,3} = a_{3,3} - \sum_{k=1}^{2} l_{3,k}u_{k,3} = -\frac{46}{3}$. We thus have:

$$\mathbf{A} = \mathbf{LU} \text{ with } \mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -2\frac{1}{3} & 1\frac{1}{3} & 1 \end{pmatrix} \text{ and } \mathbf{U} = \begin{pmatrix} 3 & 2 & 5 \\ 0 & 5 & 18 \\ 0 & 0 & -15\frac{1}{3} \end{pmatrix}.$$

To solve $\mathbf{A}\underline{x} = \underline{1}$, we first solve for $\underline{z}$ in $\mathbf{L}\underline{z} = \underline{1}$. A simple substitution procedure yields $\underline{z} = (1, 3, -\frac{2}{3})$. We now continue to solve $\mathbf{U}\underline{x} = \underline{z}$; also here a substitution procedure suffices to find $\underline{x} = \frac{1}{115}(-4, 51, 5)$.

Example 10. LU decomposition for a CTMC. We reconsider the CTMC for which the matrix $\mathbf{Q}$ is given by

$$\mathbf{Q} = \begin{pmatrix} -4 & 2 & 2 \\ 1 & -2 & 1 \\ 6 & 0 & -6 \end{pmatrix}.$$

We form $\mathbf{A} = \mathbf{Q}^T$ and directly include the normalisation equation. To find the steady-state probabilities we thus have to solve:

$$\begin{pmatrix} -4 & 1 & 6 \\ 2 & -2 & 0 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{34}$$

We now decompose $\mathbf{A}$ using the Doolittle decomposition as follows:

$$\mathbf{A} = \mathbf{LU} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{4} & -\frac{10}{12} & 1 \end{pmatrix} \begin{pmatrix} -4 & 1 & 6 \\ 0 & -\frac{3}{2} & 3 \\ 0 & 0 & 5 \end{pmatrix}. \tag{35}$$

The solution of $\mathbf{L}\underline{z} = (0, 0, 1)^T$ now reveals, via a simple substitution, that $\underline{z} = (0, 0, 1)$. We now have to find $\underline{\pi}$ from $\mathbf{U}\underline{\pi} = \underline{z}$, from which we, again via a substitution procedure, find $\underline{\pi} = (\frac{2}{5}, \frac{2}{5}, \frac{1}{5})$, as we have seen before.

In the above example, we took a specific way to deal with the normalisation equation: we replaced one equation from the "normal" system with the normalisation equation. In doing so, the vector $\underline{b}$ changes to $\underline{b} = (0, 0, 1)$ and after the solution of $\mathbf{L}\underline{z} = \underline{b}$, we found $\underline{z} = (0, 0, 1)^T$. This is not only true for the above example; if we replace the last equation, the vector $\underline{z}$ *always* has this value, so that we do not really have to solve the system $\mathbf{L}\underline{z} = (0, 0, 1)^T$. Hence, after the LU decomposition has been performed, we can directly solve $\underline{\pi}$ from $\mathbf{U}\underline{\pi} = (0, \cdots, 0, 1)^T$.

Opposed to the above variant, we can also postpone the normalisation. We then decompose $\mathbf{A} = \mathbf{Q}^T = \mathbf{LU}$, for which we will find that the last row of $\mathbf{U}$ contains only 0's. The solution of $\mathbf{L}\underline{z} = \underline{0}$ will then *always* yield $\underline{z} = \underline{0}$, so that we can immediately solve $\mathbf{U}\underline{\pi} = \underline{0}$. This triangular system of equations can easily be solved via a back-substitution procedure; however, we have to assume $\pi_N = \alpha$ and compute the rest of $\underline{\pi}$ relative to α as well. A final normalisation will then yield the ultimate steady-state probability vector $\underline{\pi}$.

Postponing the normalisation is preferred in most cases for at least two reasons: (i) it provides an implicit numerical accuracy test in that the last row of $\mathbf{U}$ should equal 0; and (ii) it requires less computations than the implicit normalisation since the number of non-zeros in the matrices that need to be handled is smaller. Of course, these advantages will become more apparent for larger values of N.

Complexity Considerations for LU Decomposition The LU decomposition solution method has the same computational complexity of $\mathcal{O}(N^3)$ as the Gaussian elimination procedure. The decomposition can be performed with only one data structure (typically an array). Initially, the matrix $\mathbf{A}$ is stored in it, but during the decomposition the elements of $\mathbf{L}$ (except for the diagonal elements from $\mathbf{L}$, but these are equal to 1 anyway) and the elements of $\mathbf{U}$ replace the original values.

Under- and Overflow We finally comment on the occurrence of over- and underflow during the computations. Underflow can be dealt with by setting intermediate values smaller than some threshold, say 10^{-24}, equal to 0. Overflow is unlikely to occur during the reduction phase in the Gaussian elimination since the pivots are the largest (absolute) quantities in every column. If in other parts of the algorithms overflow tends to occur, which can be observed if some of the values grow above a certain threshold, e.g., 10^{10}, then an intermediate normalisation of the solution vector is required. A final normalisation then completes the procedures.

5.2 Iterative Methods

Although direct methods are suitable to solve the system of equations (28), for reasons of computational and memory efficiency they cannot be used when the number of states N grows beyond about a thousand. Instead, we use iterative methods in these cases. With iterative methods, the involved matrices do not change (fill-in is avoided), so that they can be stored efficiently using sparse matrix methods. Moreover, these methods can be implemented such that in the matrix-multiplications only the multiplications involving *two* non-zero operands are taken into account.

Iterative procedures do not result in an explicit solution of the system of equations. A key characteristic of iterative methods is that it is not possible to state *a priori* how many computational steps are required. Instead, a simple numerical procedure (the iteration step) is performed repeatedly until a desired level of accuracy is reached.

The Power Method We have already seen the simplest iterative method to solve for the steady-state probabilities of a DTMC in Section 2: the *Power method*. The Power method performs successive multiplication of the steady-state probability vector $\underline{v}$ with $\mathbf{P}$ until convergence is reached. The Power method can also be applied to CTMCs. Given a CTMC with generator matrix $\mathbf{Q}$, we can compute the DTMC transition matrix $\mathbf{P} = \mathbf{I} + \mathbf{Q}/\lambda$. If we take $\lambda \geq \max_i\{|q_{i,i}|\}$, the matrix $\mathbf{P}$ is a stochastic matrix and describes the evolution of the CTMC in time-steps of mean length $1/\lambda$ (see Section 6 for a more precise formulation). Using $\mathbf{P}$ and setting $\underline{\pi}^{(0)} = \underline{\pi}(0)$ as initial estimate for the steady-state probability vector, we can compute $\underline{\pi}^{(k+1)} = \underline{\pi}^{(k)}\mathbf{P}$ and find that $\underline{\pi} = \lim_{k\to\infty} \underline{\pi}^{(k)}$.

In practice, the Power method is not very efficient. Since more efficient methods do exist, we do not discuss the Power method any further.

The Jacobi Method Two of the best-known (and simple) iterative methods are the Jacobi and the Gauss-Seidel iterative methods. For these methods, one first rewrites the i-th equation of the linear system (28) in the following way:

$$\sum_{j=1}^{N} a_{i,j}\pi_j = 0 \;\Rightarrow \pi_i = -\frac{1}{a_{i,i}}\left(\sum_{j<i}\pi_j a_{i,j} + \sum_{j>i}\pi_j a_{i,j}\right).$$

We clearly need $a_{i,i} \neq 0$; when the linear system is used to solve for the steady-state probabilities of an irreducible aperiodic Markov chain, this is guaranteed.

The iterative procedures now proceed with assuming a first guess for $\underline{\pi}$, denoted $\underline{\pi}^{(0)}$. If one does know an approximate solution for $\underline{\pi}$, it can be used as initial guess. In other cases, the uniform distribution is a reasonable choice, i.e., $\pi_i^{(0)} = 1/N$. The next estimate for $\underline{\pi}$ is then computed as follows:

$$\pi_i^{(k+1)} = -\frac{1}{a_{i,i}}\left(\sum_{j\neq i}\pi_j^{(k)} a_{i,j}\right). \tag{36}$$

This is the *Jacobi iteration* scheme. We continue to iterate until two successive estimates for $\underline{\pi}$ differ less than some ϵ from one another, i.e., when $||\underline{\pi}^{(k+1)} - \underline{\pi}^{(k)}|| < \epsilon$ (difference criterion). Notice that when this difference is very small, this does *not always* imply that the solution vector has been found. Indeed, it might be the case that the convergence towards the solution is very slow. Therefore, it is good to check whether $||\mathbf{A}\underline{\pi}^{(k)}|| < \epsilon$ (residual criterion). Since this way of checking convergence is more expensive, often a combination of these two methods is used: use the difference criterion normally; once it is satisfied use the residual criterion. If the convergence is really slow, two successive iterates might be very close to one another, although the actual value for $\underline{\pi}$ is still "far away". To avoid the difference criterion to stop the iteration process too soon, one might instead check on the difference between non-successive iterates, i.e., $||\underline{\pi}^{(k)} - \underline{\pi}^{(k-d)}|| < \epsilon$, with $d \in I\!N^+$ (and $d \leq k$).

The Gauss-Seidel Method The Jacobi method requires the storage of both $\underline{\pi}^{(k)}$ and $\underline{\pi}^{(k+1)}$ during an iteration step. If, instead, the computation is structured such that the $(k+1)$-th estimates are used as soon as they have been computed, we obtain the *Gauss-Seidel* scheme:

$$\pi_i^{(k+1)} = -\frac{1}{a_{i,i}} \left(\sum_{j<i} \pi_j^{(k+1)} a_{i,j} + \sum_{j>i} \pi_j^{(k)} a_{i,j} \right), \tag{37}$$

where we assume that the order of computation is from $\pi_1^{(k+1)}$ to $\pi_N^{(k+1)}$. This scheme then requires only one probability vector to be stored, since the $(k+1)$-th estimate for π_i immediately replaces the k-th estimate in the single stored vector.

The SOR Method The last method we mention is the *successive over-relaxation method* (SOR). SOR is an extension of the Gauss-Seidel method, in which the vector $\underline{\pi}^{(k+1)}$ is computed as the weighted average of the vector $\underline{\pi}^{(k)}$ and the vector $\underline{\pi}^{(k+1)}$ that would have been used in the (pure) Gauss-Seidel iteration. That is, we have, for $i = 1, \cdots, N$:

$$\pi_i^{(k+1)} = (1-\omega)\pi_i^{(k)} - \frac{\omega}{a_{i,i}} \left(\sum_{j<i} \pi_j^{(k+1)} a_{i,j} + \sum_{j>i} \pi_j^{(k)} a_{i,j} \right),$$

where $\omega \in (0,2)$ is the relaxation factor. When $\omega = 1$, this method reduces to the Gauss-Seidel iteration scheme; however, when we take $\omega > 1$ (or $\omega < 1$) we speak over over-relaxation (under-relaxation). With a proper choice of ω, the iterative solution process can be accelerated significantly. Unfortunately, the optimal choice of ω cannot be determined *a priori*. We can, however, estimate ω during the solution process itself; for details, refer to Stewart [66] or Hageman and Young [26].

Example 11. Comparing the Power, the Jacobi and the Gauss-Seidel method. We reconsider the CTMC for which the matrix $\mathbf{Q}$ is given by

$$\mathbf{Q} = \begin{pmatrix} -4 & 2 & 2 \\ 1 & -2 & 1 \\ 6 & 0 & -6 \end{pmatrix}.$$

As starting vector for the iterations we take $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. In the Jacobi and Gauss-Seidel method we renormalised the probability vector after every iteration. In Table 1 we show the first ten iteration vectors for these methods. As can be seen, the Power method convergest slowest, followed by the Jacobi and the Gauss-Seidel method.

#	Power	Jacobi	Gauss-Seidel
1	(0.5000, 0.3333, 0.1667)	(0.5385, 0.3077, 0.1538)	(0.5833, 0.5833, 0.2917)
2	(0.3889, 0.3889, 0.2222)	(0.4902, 0.3137, 0.1961)	(0.4000, 0.4000, 0.2000)
3	(0.4167, 0.3889, 0.1944)	(0.3796, 0.4213, 0.1991)	(0.4000, 0.4000, 0.2000)
4	(0.3981, 0.3981, 0.2037)	(0.3979, 0.4023, 0.1998)	:
5	(0.4028, 0.3981, 0.1991)	(0.4001, 0.3999, 0.2000)	:
6	(0.3997, 0.3997, 0.2006)	(0.4000, 0.4000, 0.2000)	:
7	(0.4005, 0.3997, 0.1998)	(0.4000, 0.4000, 0.2000)	:
8	(0.3999, 0.3999, 0.2001)	(0.4000, 0.4000, 0.2000)	:
9	(0.4001, 0.3999, 0.2000)	:	:
10	(0.4000, 0.4000, 0.2000)	:	:

Table 1. The first few iteration vectors for three iterative solution methods (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

Complexity Considerations Iterative methods can be used to solve the linear systems arising in the solution of the steady-state probabilities for Markov chains, with or without the normalisation equation. Quite generally we can state that it is better *not* to include the normalisation equation; if the normalisation equation is included, the second largest eigenvalue of the coefficient matrix **A** generally increases (the largest one is 1) which normally reduces the speed of convergence.

All iterative methods require the storage of the matrix **A**. For larger modelling problems, **A** has to be stored sparsely; it is then important that the sparse storage structure is structured such that row-wise access is very efficient since all methods require the product of a row of **A** with the (column) iteration vector $\underline{\pi}^{(k)}$. The Power and the Jacobi method require two iteration vectors to be stored, each of length N. The Gauss-Seidel and the SOR method only require one such vector. In all the iteration schemes the divisions by $-a_{i,i}$ (and for SOR the multiplication with ω) need to be done only once, either before the actual iteration process starts or during the first iteration step, by changing the matrix **A** accordingly. This saves N divisions (and N multiplications for SOR) per iteration. A single iteration can then be interpreted as a single matrix-vector multiplication (MVM). In a non-sparse implementation, a single MVM costs $\mathcal{O}(N^2)$ multiplications and additions. However, in a suitably chosen sparse storage structure only $\mathcal{O}(\eta)$ multiplications and additions are required, where η is the number of non-zero elements in **A**. Typically, the number of nonzero elements per column in **A** is limited to a few dozen. For example, if the CTMC is derived from a high-level model specification, the number of nonzero elements per row in **Q** equals the number of enabled activities in a particular state. This number is normally much smaller than N. Hence, it is reasonable to assume that one iteration step only takes $\mathcal{O}(N)$ operations.

An important difference between iterative methods is the number of required iterations. Typically, the Power method converges slowest, and the Gauss-Seidel method typically outperforms the Jacobi method. With the SOR method, a proper choice of the relaxation factor ω accelerates the iteration process, so that it often is the fastest method. In practical modelling problems, the required number of iterations can range from just a few to a few thousands.

There do exist more advanced methods to solve linear systems of equations which often convergence in less iteration steps. This then mostly comes at the cost of either more complex iteration steps (more computation time required per step) or iteration steps requiring much more intermediate solution vectors, or both. A fast method, for instance, requiring 7 iteration vectors is CGS (conjugate gradient squared), an example of a so-called Krylov subspace method [66, Chapter 4]. It goes beyond the scope of the current paper to go in more detail here.

6 Solution Methods for Transient-State Probabilities

In this section, we discuss the solution of the time-dependent behaviour of Markov chains. As we have seen in Section 2, the time-dependent behaviour of a DTMC is simply obtained by successive matrix-vector multiplications and is therefore not further considered here. The time-dependent behaviour of an SMC is much more complex; it goes beyond the scope of this paper. Hence, we focus on the transient behaviour of CTMCs in this section.

In Section 6.1 we explain why transient behaviour is of interest and which equations we need to solve for that purpose. We discuss "traditional" methods to solve these equations in Section 6.2 and continue with the use of uniformisation in Section 6.3. Finally, in Section 6.4, we comment on the use of uniformisation to compute so-called cumulative measures.

6.1 Introduction

Steady-state measures (probabilities) do suffice for the evaluation of the performance of most systems. There are, however, exceptions to this rule, for instance

- when the system life-time is so short that steady-state is not reached;
- when the period towards the steady-state situation itself is of interest;
- when temporary overload periods, for which no steady-state solution exists, are of interest;

– when reliability and availability properties are taken into account in the model, e.g., non-repairable systems that are failure-prone are of no interest in steady-state, since then they will have completely failed.

The time-dependent state probabilities of a CTMC are specified by a linear system of differential equations (as already given in (20)):

$$\underline{\pi}'(t) = \underline{\pi}(t)\mathbf{Q}, \quad \text{given} \ \underline{\pi}(0). \tag{38}$$

Measures that are specified in terms of $\underline{\pi}(t)$ are called *instant-of-time measures*. If we associate a *reward* r_i with every state, the expected reward at time t can be computed as

$$E[X(t)] = \sum_{i=1}^{N} r_i \pi_i(t). \tag{39}$$

The rewards express the amount of gain (or costs) that is accumulated per unit of time in state i; $E[X(t)]$ then expresses the speed of gain accumulation (per time-unit).

In many modelling applications, not only the values of the state probabilities at a time instance t are of importance, but also the total time spent in any state up to some time t, as expressed in so-called *cumulative measures*. We define the cumulative state vector $\underline{l}(t)$ as

$$\underline{l}(t) = \int_0^t \underline{\pi}(s) ds. \tag{40}$$

Notice that the entries of $\underline{l}(t)$ are no longer probabilities; $l_i(t)$ denotes the overall time spent in state i during the interval $[0, t)$. Integrating (38), we obtain

$$\int_0^t \underline{\pi}'(s) ds = \int_0^t \underline{\pi}(s)\mathbf{Q} ds, \tag{41}$$

which can be rewritten as

$$\underline{\pi}(t) - \underline{\pi}(0) = \underline{l}(t)\mathbf{Q}, \tag{42}$$

which can, after having substituted $\underline{l}'(t) = \underline{\pi}(t)$, be written as

$$\underline{l}'(t) = \underline{l}(t)\mathbf{Q} + \underline{\pi}(0). \tag{43}$$

hence, $\underline{l}(t)$ follows from the solution of a linear non-homogeneous system of differential equations. If r_i is the reward obtained per time-unit in state i, then

$$Y(t) = \sum_{i=1}^{N} r_i l_i(t) \tag{44}$$

expresses the total amount of reward gained over the period $[0, t)$. The distribution $F_Y(y, t) = \Pr\{Y(t) \leq y\}$ has been defined by Meyer as the *performability distribution* [50,51]; it expresses the probability that a reward of at most y is gained in the period $[0, t)$. Meyer developed his performability measure in order to express the effectiveness of use of computer systems in failure prone environments.

Example 12. Measure interpretation. Consider a three-state CTMC with generator matrix

$$\mathbf{Q} = \begin{pmatrix} -2f & 2f & 0 \\ r & -(f+r) & f \\ 0 & r & -r \end{pmatrix}.$$

This CTMC models the availability of a computer system with two processors. In state 1 both processors are operational but can fail with rate $2f$. In state 2 only one processor is operational (and can fail with rate f); the other one is repaired with rate r. In state 3 both processors have failed; one of them is being repaired. Note that we assume that both the processor life-times and the repair times are negative exponentially distributed. Since in state 1 both processors operate, we assign a reward 2μ to state 1, where μ is the effective processing rate of a single processor. Similarly, we assign $r_2 = \mu$ and $r_3 = 0$. We assume that the system is initially fully operational, i.e., $\underline{\pi}(0) = (1, 0, 0)$. The following measures can now be computed:

- Steady-state reward rate $\sum_i r_i \pi_i$: the expected processing rate of the system in steady-state, i.e., the long-term average processing rate of the system;
- Expected instant reward rate $\sum_i r_i \pi_i(t)$: the expected processing rate at a particular time instance t;
- Expected accumulated reward $\sum_i r_i l_i(t)$: the expected number of jobs (of length 1) processed in the interval $[0, t)$;
- Accumulated reward distribution $F_Y(y, t)$: the probability that at most y jobs (of length 1) have been processed during $[0, t)$.

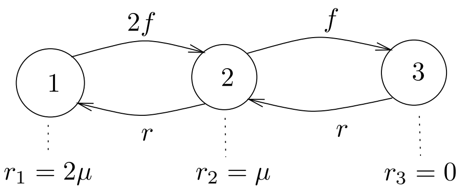

Fig. 13. A three-state CTMC (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

6.2 Runge-Kutta Methods

The numerical solution of systems of differential equations of type (38) and (43) has since long been an important topic in numerical mathematics. Many numerical procedures have been developed for this purpose, all with specific strengths and weaknesses. Below, we will present one such method in a concise way, thereby focusing on the computation of $\underline{\pi}(t)$; for details, see [66].

With *Runge-Kutta methods* (RK-methods) the continuous vector function $\underline{\pi}(t)$ that follows from the differential equation $\underline{\pi}'(t) = \underline{\pi}(t)\mathbf{Q}$, given $\underline{\pi}(0)$, is approximated by a discrete function $\tilde{\underline{\pi}}_i$ ($i \in \mathbb{N}$), where $\tilde{\underline{\pi}}_i \approx \underline{\pi}(ih)$, i.e., h is the fixed step-size in the discretisation; the smaller h, the better (but more expensive) the solution.

With RK-methods, the last computed value for any point $\tilde{\underline{\pi}}_i$ is used to compute $\tilde{\underline{\pi}}_{i+1}$. The values $\tilde{\underline{\pi}}_0$ through $\tilde{\underline{\pi}}_{i-1}$ are not used to compute $\tilde{\underline{\pi}}_{i+1}$. For this reason, RK-methods are called *single-step methods*. They are always stable, provided the step-size h is taken sufficiently small. Unlike Euler-methods, RK-methods do not require the computation of derivatives of the function of interest, which keeps them fairly efficient. RK-methods are distinguished on the basis of their *order*: a RK-method is of order k if the exact Taylor series for $\underline{\pi}(t+h)$ and the solution of the RK-scheme for time instance $t+h$ coincide as far as the terms up to h^k are concerned.

One of the most widely used RK-methods is the 4th-order RK-method (normally denoted as "RK4"). For a vector-differential equation $\underline{\pi}'(t) = \underline{\pi}(t)\mathbf{Q}$, given $\underline{\pi}(0)$, successive estimates for $\tilde{\underline{\pi}}_i$ are computed as follows:

$$\tilde{\underline{\pi}}_{i+1} = \tilde{\underline{\pi}}_i + \frac{h}{6}(\underline{k}_1 + 2\underline{k}_2 + 2\underline{k}_3 + \underline{k}_4), \tag{45}$$

with

$$\begin{cases} \underline{k}_1 = \tilde{\underline{\pi}}_i \mathbf{Q}, \\ \underline{k}_2 = (\tilde{\underline{\pi}}_i + \frac{h}{2}\underline{k}_1)\mathbf{Q}, \\ \underline{k}_3 = (\tilde{\underline{\pi}}_i + \frac{h}{2}\underline{k}_2)\mathbf{Q}, \\ \underline{k}_4 = (\tilde{\underline{\pi}}_i + h\underline{k}_3)\mathbf{Q}. \end{cases} \tag{46}$$

Since the RK4 method provides an explicit solution to $\tilde{\underline{\pi}}_i$, it is called an *explicit 4th-order method*. Per iteration step of length h, it requires 4 matrix-vector multiplications, 7 vector-vector additions and 4 scalar-vector multiplications. Furthermore, apart from $\mathbf{Q}$ and $\tilde{\underline{\pi}}$ also storage for at least two intermediate probability vectors is required.

In contrast, *implicit* RK-methods yield a system of linear equations in which the vector $\tilde{\underline{\pi}}_i$ appears implicitly. Such methods are normally more expensive to employ and can therefore only be justified in special situations, e.g., when the CTMC under study is stiff, meaning that the ratio of the largest and smallest rate appearing in $\mathbf{Q}$ is very large, say of the order of 10^4 or higher.

6.3 Uniformisation for Transient Measures

Consider the *scalar* differential equation $p'(t) = p(t)Q$, given $p(0)$ and scalar constant Q. From elementary analysis we know that the solution to this differential equation is $p(t) = p(0)e^{Qt}$. When dealing with CTMCs, the transient behaviour is defined by the linear system of differential equations (20); the transient behaviour then can still be computed as an exponential, however, now in terms of vectors and matrices, that is,

$$\underline{\pi}(t) = \underline{\pi}(0)e^{\mathbf{Q}t}. \tag{47}$$

Direct computation of this matrix exponential, e.g., via a Taylor/MacLaurin series expansion as $\sum_{i=0}^{\infty}(\mathbf{Q}t)^i/i!$, is in general not a good idea [52]: (i) the infinite summation that appears in the Taylor series cannot be truncated efficiently; (ii) severe round-off errors usually will occur due to the fact that $\mathbf{Q}$ contains positive as well as negative entries; and (iii) the matrices $(\mathbf{Q}t)^i$ become non-sparse, thus requiring too much storage capacity for practically relevant applications. To avoid these problems, a method known as *uniformisation*, also known as Jensen's method or randomisation, is regarded as the method of choice [24,25,39]. To use uniformisation, we define the matrix

$$\mathbf{P} = \mathbf{I} + \frac{\mathbf{Q}}{\lambda} \quad \Rightarrow \quad \mathbf{Q} = \lambda(\mathbf{P} - \mathbf{I}). \tag{48}$$

If λ is chosen such that $\lambda \geq \max_i\{|q_{i,i}|\}$, then the entries in $\mathbf{P}$ are all between 0 and 1, whereas the rows of $\mathbf{P}$ sum to 1. In other words, $\mathbf{P}$ is a stochastic matrix and describes a DTMC. The value of λ is called *uniformisation rate*.

Example 13. Uniformising a CTMC. Consider the CTMC given by

$$\mathbf{Q} = \begin{pmatrix} -4 & 2 & 2 \\ 1 & -2 & 1 \\ 6 & 0 & -6 \end{pmatrix}. \tag{49}$$

and initial probability vector $\underline{\pi}(0) = (1, 0, 0)$. For the uniformisation rate we find by inspection: $\lambda = 6$, so that the corresponding DTMC is given by:

$$\mathbf{P} = \frac{1}{6}\begin{pmatrix} 2 & 2 & 2 \\ 1 & 4 & 1 \\ 6 & 0 & 0 \end{pmatrix}. \tag{50}$$

The CTMC and the DTMC are given in Figure 14.

The process of uniformising a CTMC can be understood as follows. In the CTMC, the state residence times are exponentially distributed. The state with the shortest residence times provides us with the uniformisation rate λ. For that state, one epoch in the DTMC corresponds to one negative exponentially distributed delay with rate λ, after which one of the successor states is selected probabilistically. For the states in the CTMC that have total outgoing rate λ, the corresponding states in the DTMC will have no self-loops. For states in the CTMC having a state residence time distribution with a rate smaller than λ (these states have on average a longer state residence time), one epoch in the DTMC might not be long enough; hence, in the next epoch these states might

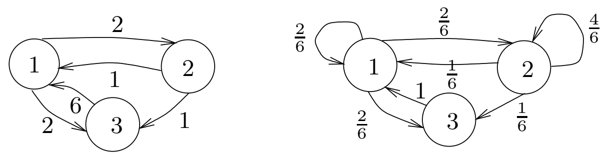

Fig. 14. A small CTMC (left) and the corresponding DTMC (right) after uniformisation (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

be revisited. This is made possible by the definition of **P**, in which these states have self-loops, i.e., $p_{i,i} > 0$. Using (48) we can write

$$\underline{\pi}(t) = \underline{\pi}(0)e^{\mathbf{Q}t} = \underline{\pi}(0)e^{\lambda(\mathbf{P}-\mathbf{I})t} = \underline{\pi}(0)e^{-\lambda\mathbf{I}t}e^{\lambda\mathbf{P}t} = \underline{\pi}(0)e^{-\lambda t}e^{\lambda\mathbf{P}t}. \tag{51}$$

We now employ a Taylor-series expansion for the matrix exponential as follows:

$$\underline{\pi}(t) = \underline{\pi}(0)e^{-\lambda t}\sum_{n=0}^{\infty}\frac{(\lambda t)^n\mathbf{P}^n}{n!} = \underline{\pi}(0)\sum_{n=0}^{\infty}\psi(\lambda t;n)\mathbf{P}^n, \tag{52}$$

where

$$\psi(\lambda t;n) = e^{-\lambda t}\frac{(\lambda t)^n}{n!}, \quad n \in \mathbb{N}, \tag{53}$$

are Poisson probabilities, i.e., $\psi(\lambda t;n)$ is the probability of n events occurring in $[0,t)$ in a Poisson process with rate λ. Of course, we still deal with a Taylor series approach here, however, the involved **P**-matrix is a probabilistic matrix with all its entries between 0 and 1, as are the Poisson probabilities. Hence, this Taylor series "behaves nicely", as we will discuss below.

Equation (52) can be understood as follows. At time t, the probability mass of the CTMC, initially distributed according to $\underline{\pi}(0)$ has been redistributed according to the DTMC with state-transition matrix **P**. During the time interval $[0,t)$, with probability $\psi(\lambda t;n)$ exactly n jumps have taken place. The effect of these n jumps on the initial distribution $\underline{\pi}(0)$ is described by the vector-matrix product $\underline{\pi}(0)\mathbf{P}^n$. Weighting this vector with the associated Poisson probability $\psi(\lambda t;n)$, and summing over all possible numbers of jumps in $[0,t)$, we obtain, by the law of total probability, the probability vector $\underline{\pi}(t)$.

Uniformisation allows for an iterative solution without matrix-matrix multiplications, so that matrix fill-in do not occur. Instead of directly computing the n-th Power of **P** as suggested by (52) one considers the following sum of vectors:

$$\underline{\pi}(t) = \sum_{n=0}^{\infty}\psi(\lambda t;n)\left(\underline{\pi}(0)\mathbf{P}^n\right) = \sum_{n=0}^{\infty}\psi(\lambda t;n)\underline{\hat{\pi}}(n), \tag{54}$$

where $\underline{\hat{\pi}}_n$ is the state probability distribution vector after n epochs in the DTMC with transition matrix **P**, which can be derived recursively as

$$\underline{\hat{\pi}}(0) = \underline{\pi}(0) \text{ and } \underline{\hat{\pi}}(n) = \underline{\hat{\pi}}(n-1)\mathbf{P}, \quad n \in \mathbb{N}^+. \tag{55}$$

Clearly, the infinite sum in (54) has to be truncated, say after k_ϵ epochs in the DTMC. The actually computed state probability vector $\underline{\tilde{\pi}}(t)$ then equals:

$$\underline{\tilde{\pi}}(t) = \sum_{n=0}^{k_\epsilon} \psi(\lambda t; n)\underline{\hat{\pi}}(n). \tag{56}$$

The number of terms that has to be added to reach a prespecified accuracy ϵ can now be computed *a priori* as follows. It can be shown that the difference between the computed and the exact value of the transient probability vector is bounded as follows:

$$||\underline{\pi}(t) - \underline{\tilde{\pi}}(t)|| \leq 1 - \sum_{n=0}^{k_\epsilon} e^{-\lambda t}\frac{(\lambda t)^n}{n!}. \tag{57}$$

Thus, we have to find that value of k_ϵ such that $1 - \sum_{n=0}^{k_\epsilon} e^{-\lambda t}(\lambda t)^n/n! \leq \epsilon$. Stated differently, we need the smallest value of k_ϵ that satisfies

$$\sum_{n=0}^{k_\epsilon} \frac{(\lambda t)^n}{n!} \geq \frac{1-\epsilon}{e^{-\lambda t}} = (1-\epsilon)e^{\lambda t}. \tag{58}$$

For reasons that will become clear below, k_ϵ is called the *right truncation point.*

Example 14. How large should we take k_ϵ? In Table 2 we show the number of required steps k_ϵ as a function of ϵ and the product λt in the uniformisation procedure. As can be observed, k_ϵ increases sharply with increasing λt and decreasing ϵ.

If the product λt is large, k_ϵ tends to be of order $\mathcal{O}(\lambda t)$. On the other hand, if λt is large, the DTMC described by $\mathbf{P}$ might have reached steady-state along the way, so that the last matrix-vector multiplications do not need to be performed any more. Such a steady-state detection can be integrated in the computational procedure (see [57] and the example below).

	λt						
ϵ	0.1	0.2	1	2	4	8	16
0.0005	2	3	6	8	12	19	31
0.00005	3	3	7	10	14	21	34
0.000005	3	4	8	11	16	23	37

Table 2. The number of required steps k_ϵ as a function of ϵ and the product λt (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

Example 15. Transient solution of a three-state CTMC. We consider the transient solution of the CTMC given in Figure 14; we already performed the uniformisation to form the matrix $\mathbf{P}$ with uniformisation rate $\lambda = 6$.

We first establish how many steps we have to take into account for increasing t. This number can be computed by checking the inequality (58) and taking $\epsilon = 10^{-4}$. We find:

t	0.1	0.2	0.5	1	5	10	20	50	100
k_ϵ	5	7	11	17	52	91	163	367	693

We then continue to compute $\underline{\tilde{\pi}}(t)$ according to (56) to find the curves for $\pi_i(t)$ as indicated in Figure 15. As can be observed, for $t \geq 2$ steady-state is reached. Although for larger values of t we require very many steps to be taken, the successive vectors $\underline{\hat{\pi}}(n)$ do not change any more. Denote with $k_{\text{ss}} < k_\epsilon$ the value after which $\underline{\hat{\pi}}_i$ does not change any more. Instead of explicitly computing the sum (56) for all values of n, the last part of it can then be computed more efficiently as follows:

$$\underline{\tilde{\pi}}(t) = \sum_{n=0}^{k_\epsilon} \psi(\lambda t; n)\underline{\hat{\pi}}(n) = \sum_{n=0}^{k_{\text{ss}}} \psi(\lambda t; n)\underline{\hat{\pi}}(n) + \underbrace{\left(\sum_{n=k_{\text{ss}}+1}^{k_\epsilon} \psi(\lambda t; n) \right)}_{1-\sum_{n=0}^{k_{\text{ss}}} \psi(\lambda t;n)} \underline{\hat{\pi}}(k_{\text{ss}}), \quad (59)$$

thus saving the computation intensive matrix-vector multiplications in the last part of the sum. The point k_{ss} is called the *steady-state truncation point.*

If the product λt is very large, the first group of Poisson probabilities is very small, often so small that the corresponding vectors $\underline{\hat{\pi}}(n)$ do not really matter. We can exploit this by only starting to add the weighted vectors $\underline{\hat{\pi}}(n)$ after the Poisson weighting factors become reasonably large. Of course, we still have to compute the matrix-vector products (55). The point where we start to add the probability vectors is called the *left truncation point.*

Finally, we note that the Poisson probabilities $\psi(\lambda t; n), n = 0, \cdots, N$, can be computed efficiently when taking into account the following recursive relations:

$$\psi(\lambda t; 0) = e^{-\lambda t}, \quad \text{and} \quad \psi(\lambda t; n+1) = \psi(\lambda t; n)\frac{\lambda t}{n+1}, \quad n \in I\!N. \quad (60)$$

When λt is large, say larger than 25, overflow might easily occur. However, for these cases, the normal distribution can be used as an approximation. Fox and Glynn report on a stable algorithm to compute Poisson probabilities [21].

To use uniformisation, the sparse matrix $\mathbf{P}$ has to be stored, as well as two probability vectors. Given an N-state Markov chain, two probability vectors of length N have to be stored. Given that the matrix $\mathbf{P}$ is sparse, which typically is the case, the cost to store it is of order N. Hence, the overall storage complexity is $\mathcal{O}(N)$.

The main computational complexity lies in the $\min\{k_{\text{ss}}, k_\epsilon\}$ matrix-vector multiplications that need to be performed (plus the subsequent multiplication

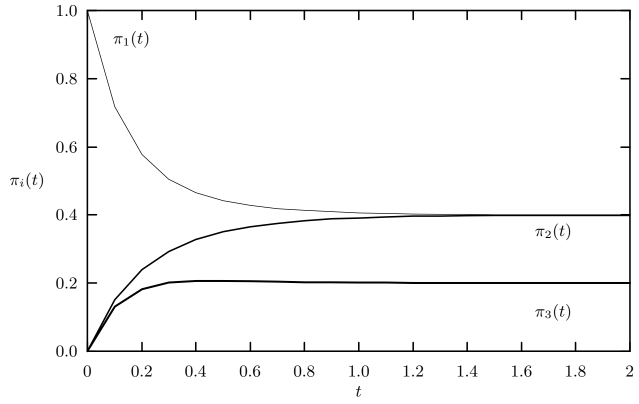

Fig. 15. First two seconds in the evolution of the three-state CTMC (B.R. Haverkort, *Performance of Computer Communication Systems*, 1998. © John Wiley & Sons Limited. Reproduced with Permission.)

of these vectors with the precomputed Poisson probabilities). As we have seen above, for large λt, k_ϵ is of order $\mathcal{O}(\lambda t)$. A single matrix-vector multiplication costs, in case of a sparse matrix $\mathbf{P}$ only $\mathcal{O}(N)$, and in case of a non-sparse matrix $\mathcal{O}(N^2)$. Taking the sparse case, we arrive at an overall time complexity of $\mathcal{O}(\lambda t N)$.

To increase the efficiency of uniformisation in specific situations, various variants have been developed. A good overview can be found in [53,55,54].

6.4 Uniformisation for Cumulative Measures

Let us now address a uniformisation-based efficient procedure for computing the expected accumulated reward over $[0, t)$, that is:

$$E[Y(t)] = E\left[\sum_{i=1}^{N} r_i l_i(t)\right]. \tag{61}$$

We first note that in the interval $[0, t)$, that is, an interval of length t, the expected time between two jumps, when k jumps have taken place according to a Poisson process with rate λ equals $t/(k+1)$. Interpreting λ as the uniformisation rate, the expected accumulated reward until time t, given k jumps, in the uniformised chain equals

$$\frac{t}{k+1}\sum_{i=1}^{N} r_i \sum_{m=0}^{k} \hat{\pi}_i(m).$$

This expression can be explained as follows. The right-most sum expresses the sum of the probabilities to reside in state i over the $k+1$ intervals addressed;

multiplied with the mean interval length (left-most factor), this gives the expected time spent in state i. The first summation weights these times with the corresponding reward and adds over all states.

We can now sum the above expression over all possible number of jumps that might occur during the interval $[0, t)$ and weight them with the usual Poisson probabilities, to arrive at:

$$E[Y(t)] = \sum_{k=0}^{\infty} \psi(\lambda t; k) \frac{t}{k+1} \sum_{i=1}^{N} r_i \sum_{m=0}^{k} \hat{\pi}_i(m). \tag{62}$$

Based on this expression, efficient numerical procedures can be devised as follows (see also [63,64]). First define

$$\phi(\lambda t; k) = \psi(\lambda t; k) \frac{t}{k+1} = e^{-\lambda t} \frac{\lambda^n t^{n+1}}{(k+1)!},$$

which can be computed recursively in a similar way as $\psi(\lambda t; k)$. When we define the diagonal matrix $\mathbf{R} = \text{diag}(\underline{r})$, i.e., $\mathbf{R}$ is a matrix with on the diagonal the rewards r_i, we can rewrite (62) by transforming the summation over all states in a matrix-vector multiplication as follows:

$$E[Y(t)] = \underline{1} \left(\sum_{k=0}^{\infty} \phi(\lambda t; k) \sum_{m=0}^{k} \underline{\hat{\pi}}(m) \right) \mathbf{R}, \tag{63}$$

where ϕ is computed recursively and $\underline{\hat{\pi}}(m) = \underline{\hat{\pi}}(m-1)\mathbf{P}$, with $\mathbf{P}$ the transition matrix for the uniformised DTMC. By defining the vector $\underline{C}(k)$, which denotes the cumulative probability over k steps to reside at each of the states, in the following way: $\underline{C}(0) = \underline{\hat{\pi}}(0)$ and $\underline{C}(k) = \underline{C}(k-1) + \underline{\hat{\pi}}(k)$, we finally arrive at

$$E[Y(t)] = \underline{1} \left(\sum_{k=0}^{\infty} \phi(\lambda t; k) \underline{C}(k) \right) \mathbf{R}. \tag{64}$$

As for the transient measures, a truncation criterion for the infinite summation can be easily developed. Similar storage and computational complexity considerations apply as in Section 6.3.

We finally comment on the solution of the performability distribution $F_Y(y, t)$, i.e., the probability distribution $\Pr\{Y(t) \leq y\}$ [50,51]. Also here, uniformisation can be employed; however, a direct summation over all states does not suffice any more. Instead, we have to sum the accumulated reward over all paths of length l (given a starting state) that can be taken through the DTMC, after which we have to compute a weighted sum over all these paths and their occurrence probabilities; for details we refer to [63,64,59].

7 Other Issues

In this section, a number of important issues not covered in detail in this chapter will be addressed briefly; pointers to relevant literature will be provided.

Phase-Type Distributions In this paper, we did not further address absorbing Markov chains, even though their applicability is substantial. Given a CTMC with a single absorbing state, the time from the initial state to absorption in that absorbing state has a so-called *phase-type distribution*, a distribution that can be seen as the sum of a possibly infinite number of exponential phases. Many well-known distributions are indeed of this type, e.g., the Erlang or an hyperexponential distribution. Moreover, almost any other distribution can be approximated very well with phase-type distributions. The birth-death processes we encountered in the examples, can be extended such that instead of exponential distributions, phase-type distributions are used. The thus resulting Markov chains are of so-called *quasi-birth-death* type and can still be solved efficiently using *matrix-geometric methods*, even when the state spece is infinitely large. For details, we refer to [58], [66, Chapter 5] or [27, Chapter 8].

Product-Form Solutions There is a large class of Markov chains that exhibits a so-called product-form solution. Most often such Markov chains arise when modelling systems not directly at the Markov chain level by identifying states and state-transitions, but when modelling systems as networks of queues. The structure of the Markov chain underlying the queueing network then results in an overall steady-state probability vector that can be written as the product of steady-state probabilities over smaller parts of the model. The book by Van Dijk on queueing networks and product-forms [18] is an excellent source on this topic. Also the more general books on performance evaluation mentioned above address product-form models. Hillston addresses product-form results for stochastic process algebras [35].

Distributed Solution of Markovian Models Especially when Markov chains are automatically generated from high-level specifications, these Markov chains tend to become very large. To cope with Markov chains with several millions (or more) states, specialised data structures have to be employed that are efficient both from a memory and a computational point of view. Recent advances in the use of tensor algebra and binary decision diagrams (and variants) should be mentioned here [15,13]. Furthermore, recently also the use of parallel and distributed computer systems has been advocated for both the generation of large Markov chains from high-level model specifications, as well as their numerical solution. Early work in this area can be found in [11,14]. With the PARSECS prototype tool, the generation and solution of Markov chains with more than 750 million states has recently been reported [7,28].

Tools for Markovian Modelling The practical application of Markovian modelling techniques has become widespread since the beginning of the 1980's. At that time, powerful workstations with larger memories became available for daily use. Since then, a large number of software tools has been built that support, in one way or another, the generation and solution of Markovian models.

Typically, the Markovian models are constructed using either general-purpose high-level modelling formalisms such as queueing networks (*cf.* QNAP2 [68], NUMAS [56] and MACOM [47]), stochastic Petri nets (*cf.* GreatSPN [12] and SPNP [16]), the "balls and buckets" formalism (*cf.* MARCA [65] and [66, Chapter 10.2–3]) stochastic activity networks (*cf.* UltraSAN [61,62]) or stochastic process algebras (*cf.* the PEPA workbench [34], TIPPtool [32] or TwoTowers [8]) or are more application-specific formalisms (*cf.* SAVE [23] for availability evaluation). It goes beyond the scope of the current paper to give an overview of all these tools; the interested reader is referred to a number of surveys: [31,30] and [29, Chapter 10].

Model Checking Markovian Models Recently, there has been an increased interest in the merging of Markovian modelling and evaluation techniques (as described in this paper) and techniques for formal system verification, in particular model checking [19,17,41]. Where previously timing aspects were not addressed in model checking, this becomes a necessity when model checking systems and protocols for real-time systems. By adding time in a specific stochastic manner to a finite-state machine, it can be interpreted as a Markov chain. By extending the logic to express time-related properties over the finite-state machine, as has been done with the logic CSL, a stochastically timed extension of CTL, such properties can be checked efficiently using evaluation techniques for Markov chains. Seminal work in this direction has been reported by Aziz *et al.* [2,1]; more recent developments can be found in [5,4,3,33].

8 Concluding Remarks

In the preceding sections we have addressed in a nutshell a large number of aspects of the use and solution of Markovian models. However, the amount of literature on Markovian models, their solution and application is vast. To conclude, let me refer to a number of well-known textbooks in the field. An absolute "must-read" on the numerical solution of Markov chains is Stewart's textbook [66]. Very readable is the two-volume work by Howard [37,38] as is the book by Kemeny and Snell [42]. A variety of books on performance evaluation in general address Markov chains in more or less detail, most often providing numerous examples [9,27,40,44,45,67].

References

1. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In R. Alur and T. Henzinger, editors, *Lecture Notes in Computer Science 1102*, pages 269–276, 1996.
2. A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A. Sangiovanni-Vincentelli. It usually works: the temporal logic of stochastic systems. In P. Wolper, editor, *Lecture Notes in Computer Science 939*, pages 155–165, 1995.

3. C. Baier, B.R. Haverkort, J.-P. Katoen, and H. Hermanns. On the logical characterisation of performability properties. In U. Montanari, J.D.P. Rolin, and E. Welzl, editors, *Lecture Notes in Computer Science 1853*, pages 780–792, 2000.
4. C. Baier, B.R. Haverkort, J.-P. Katoen, and H. Hermanns. Model checking continuous-time Markov chains by transient analysis. In E.A. Emerson and A.P. Sistla, editors, *Lecture Notes in Computer Science 1855*, pages 358–372, 2000.
5. C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J.C.M. Baeten and S. Mauw, editors, *Lecture Notes in Computer Science 1664*, pages 146–161, 1999.
6. G. Balbo. Introduction to stochastic Petri nets. This volume.
7. A. Bell and B.R. Haverkort. Serial and parallel out-of-core solution of linear systems arising from generalised stochastic Petri net models. In *Proceedings High Performance Computing 2001*. Society for Computer Simulation, 2001.
8. M. Bernardo, R. Cleaveland, S. Sims, and W. Stewart. TwoTowers: a tool integrating functional and performance analysis of concurrent systems. In *Proceedings FORTE/PSTV 1998*, pages 457–467, 1998.
9. G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi. *Queueing Networks and Markov Chains*. John Wiley & Sons, 1998.
10. E. Brinksma and H. Hermanns. Process algebra and Markov chains. This volume.
11. S. Caselli, G. Conte, and P. Marenzoni. Parallel state space exploration for GSPN models. In G. De Michelis and M. Diaz, editors, *Applications and Theory of Petri Nets 1995*, volume 935 of *Lecture Notes in Computer Science*, pages 181–200. Springer-Verlag, 1995.
12. G. Chiola. A software package of the analysis of generalized stochastic Petri nets. In *Proceedings of the 1st International Workshop on Timed Petri Nets*, pages 136–143, Torino, Italy, July 1985. IEEE Computer Society Press.
13. G. Ciardo. Distributed and structured analysis approaches to study large and complex systems. This volume.
14. G. Ciardo, J. Gluckman, and D. Nicol. Distributed state space generation of discrete-state stochastic models. *INFORMS Journal of Computing*, 10(1):82–93, 1998.
15. G. Ciardo and A.S. Miner. Storage alternatives for large structured state spaces. In R. Marie, B. Plateau, M. Calzarossa, and G. Rubino, editors, *Computer Performance Evaluation*, Lecture Notes in Computer Science 1245, pages 44–57. Springer Verlag, 1997.
16. G. Ciardo, J. Muppala, and K. S. Trivedi. SPNP: Stochastic Petri net package. In *Proceedings of the 3rd International Workshop on Petri Nets and Performance Models*, pages 142–151. IEEE Computer Society Press, 1989.
17. E.M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 1999.
18. N.M. van Dijk. *Queueing Networks and Product Form: A Systems Approach*. John Wiley & Sons, 1993.
19. E.M.Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
20. A.K. Erlang. Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges. *The Post Office Electrical Engineer's Journal*, 10:189–197, 1917.
21. B.L. Fox and P.W. Glynn. Computing Poisson probabilities. *Communications of the ACM*, 31(4):440–445, 1988.
22. K.A. Frenkel. Allan L. Scherr — Big Blue's time-sharing pioneer. *Communications of the ACM*, 30(10):824–828, 1987.

23. A. Goyal, S.S. Lavenberg, and K.S. Trivedi. The system availability estimator. *Annals of Operations Research*, 8:285–306, 1987.
24. W.K. Grassmann. Finding transient solutions in Markovian event systems through randomization. In W.J. Stewart, editor, *Numerical Solution of Markov Chains*, pages 357–371. Marcel Dekker, 1991.
25. D. Gross and D.R. Miller. The randomization technique as a modeling tool and solution procedure for transient Markov processes. *Operations Research*, 32(2):343–361, 1984.
26. A.L. Hageman and D.M. Young. *Applied Iterative Methods.* Academic Press, 1981.
27. B. R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach.* John Wiley & Sons, 1998.
28. B. R. Haverkort, A. Bell, and H. Bohnenkamp. On the efficient sequential and distributed generation of very large Markov chains from stochastic Petri nets. In *Proceedings of the 8th International Workshop on Petri Nets and Performance Models*, pages 12–21. IEEE Computer Society Press, 1999.
29. B. R. Haverkort, R. Marie, G. Rubino, and K.S. Trivedi (editors). *Performability Modelling: Techniques and Tools.* John Wiley & Sons, 2001.
30. B.R. Haverkort and I.G. Niemegeers. Performability modelling tools and techniques. *Performance Evaluation*, 25:17–40, 1996.
31. B.R. Haverkort and K.S. Trivedi. Specification and generation of Markov reward models. *Discrete-Event Dynamic Systems: Theory and Applications*, 3:219–247, 1993.
32. H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with the TIPPTOOL. *Performance Evaluation*, 39:5–35, 2000.
33. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov chain model checker. In S. Graf and M. Schwartzbach, editors, *Lecture Notes in Computer Science 1785*, pages 347–362. Springer-Verlag, 2000.
34. J. Hillston. *A Compositional Approach to Performance Modelling.* PhD thesis, University of Edinburgh, 1994.
35. J. Hillston. Exploiting structure in solution: decomposing compositional models. This volume.
36. A.S. Hornby. *Oxford Advanced Learner's Dictionary of Current English.* Oxford University Press, 1974.
37. R.A. Howard. *Dynamic Probabilistic Systems; Volume I: Markov models.* John Wiley & Sons, 1971.
38. R.A. Howard. *Dynamic Probabilistic Systems; Volume II: Semi-Markov and decision processes.* John Wiley & Sons, 1971.
39. A. Jensen. Markov chains as an aid in the study of Markov processes. *Skand. Aktuarietidskrift*, 3:87–91, 1953.
40. K. Kant. *Introduction to Computer System Performance Evaluation.* McGraw-Hill, 1992.
41. J.-P. Katoen. *Concepts, Algorithms and Tools for Model Checking.* Universität Erlangen-Nürnberg, 1999.
42. J.G. Kemeny and J.L. Snell. *Finite Markov chains.* Van Nostrand, Princeton, 1960.
43. D.G. Kendall. Some problems in the theory of queues. *Journal of the Royal Statistical Society, Ser. B*, 13:151–185, 1951.
44. L. Kleinrock. *Queueing Systems; Volume 1: Theory.* John Wiley & Sons, 1975.
45. L. Kleinrock. *Queueing Systems; Volume 2: Computer Applications.* John Wiley & Sons, 1976.

46. A.N. Kolmogorov. Anfangsgründe der Theorie der Markoffschen Ketten mit unendlichen vielen möglichen Zuständen. *Mat. Sbornik N.S.*, pages 607–610, 1936.
47. U. Krieger, B. Müller-Clostermann, and M. Sczittnick. Modelling and analysis of communication systems based on computational methods for Markov chains. *IEEE Journal on Selected Areas in Communications*, 8(9):1630–1648, 1990.
48. V.G. Kulkarni. *Modeling and Analysis of Stochastic Systems.* Chapman & Hall, London, Glasgow, Weinheim, 1995.
49. A.A. Markov. Investigations of an important case of dependent trails. *Izvestia Acad. Nauk VI, Series I (in Russian)*, 61, 1907.
50. J.F. Meyer. On evaluating the performability of degradable computing systems. *IEEE Transactions on Computers*, 29(8):720–731, 1980.
51. J.F. Meyer. Closed-form solutions of performability. *IEEE Transactions on Computers*, 31(7):648–657, 1982.
52. C. Moler and C.F. van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20(4):801–835, 1978.
53. A.P.A. van Moorsel. *Performability Evaluation Concepts and Techniques.* PhD thesis, University of Twente, 1993.
54. A.P.A. van Moorsel and B.R. Haverkort. Probabilistic evaluation for the analytical solution of large Markov models: Algorithms and tool support. *Microelectronics and Reliability*, 36(6):733–755, 1996.
55. A.P.A. van Moorsel and W.H. Sanders. Adaptive uniformization. *Stochastic Models*, 10(3):619–648, 1994.
56. B. Müller-Clostermann. NUMAS – a tool for the numerical analysis of computer systems. In D. Potier, editor, *Proceedings of the International Conference on Modelling Techniques and Tools for Performance Analysis*, pages 141–154. North-Holland, 1985.
57. J.K. Muppala and K.S. Trivedi. Numerical transient solution of finite Markovian queueing systems. In U. Bhat, editor, *Queueing and Related Models.* Oxford University Press, 1992.
58. M.F. Neuts. *Matrix Geometric Solutions in Stochastic Models: An Algorithmic Approach.* Johns Hopkins University Press, 1981.
59. M.A. Qureshi and W.H. Sanders. A new methodology for calculating distributions of reward accumulated during a finite interval. In *Proceedings of the 26th Symposium on Fault-Tolerant Computer Systems (Sendai, Japan, June 1996)*, pages 116–125. IEEE Computer Society Press, 1996.
60. W.H. Sanders and J.-F Meyer. Stochastic activity networks: Formal definitions and concepts. This volume.
61. W.H. Sanders and J.F. Meyer. Reduced-base model construction for stochastic activity networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, 1991.
62. W.H. Sanders, W.D. Obal, M.A. Qureshi, and F.K. Widnajarko. The UltraSAN modeling environment. *Performance Evaluation*, 24:89–115, 1995.
63. E. de Souza e Silva and H.R. Gail. Calculating availability and performability measures of repairable computer systems using randomization. *Journal of the ACM*, 36(1):171–193, 1989.
64. E. de Souza e Silva and H.R. Gail. Performability analysis of computer systems: from model specification to solution. *Performance Evaluation*, 1:157–196, 1992.
65. W.J. Stewart. MARCA: Markov chain analyzer. a software package for Markov modelling. In W.J. Stewart, editor, *Numerical Solution of Markov Chains*, pages 37–62. Marcel Dekker, 1991.

66. W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.
67. K.S. Trivedi. *Probability and Statistics with Reliability, Queueing and Computer Science Applications*. Prentice-Hall, 1982.
68. M. Veran and D. Potier. QNAP2: A portable environment for queueing system modelling. In D. Potier, editor, *Modelling Techniques and Tools for Computer Performance Evaluation*, pages 25–63. North-Holland, 1984.