On the Cost of Generating PH-distributed Random Numbers

Philipp Reinecke, Katinka Wolter
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany
{preineck,wolter}@informatik.hu-berlin.de

Levente Bodrog, Miklós Telek
Budapest University of Technology and Economics
Department of Telecommunications
1521 Budapest, Hungary
{bodrog,telek}@webspn.hit.bme.hu

***Abstract*—Phase-type (PH) distributions are proven to be very powerful tools in modelling and analysis of a wide range of phenomena in computer systems. The use of these distributions in simulation studies requires efficient methods for generating PH-distributed random numbers. In this work, we consider the cost of PH-distributed random-number generation.**

I. Introduction

Phase-type distributions (PH) distributions have been widely used in modelling various phenomena such as response-times, inter-arrival times and failure times in computer systems. The fact that there are simple and elegant solution techniques available for PH distributions has made them appealing for analytic solutions.

PH distributions can also be employed in simulation studies. In this case the efficiency of generating PH-distributed random numbers plays a crucial role. In this work we investigate the efficiency of generating random numbers from continuous PH distributions. Due to the fact that the Markovian representation of PH distributions is not unique the key issue to investigate is which representation of a PH distribution is most efficient for random-number generation.

II. Definitions and Notation

Continuous phase-type (PH) distributions represent the time to absorption in a continuous-time Markov chain with one absorbing state [4]. PH distributions are commonly specified as a tuple $(\boldsymbol{\alpha}, \mathbf{A})$ of the initial probability vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ and the transient generator matrix $\mathbf{A} = \{a_{ij}\}, 1 \leq i, j \leq n$. The probability density function, the cumulative distribution function, and the kth moment, respectively, are defined as follows [2], [4], [7]:

$$\begin{aligned} f(x) &= \boldsymbol{\alpha} \mathrm{e}^{\mathbf{A}x} \mathbf{a}, \\ F(x) &= 1 - \boldsymbol{\alpha} \mathrm{e}^{\mathbf{A}x} \mathbb{1}, \\ E\left[X^k\right] &= k! \boldsymbol{\alpha} (-\mathbf{A})^{-k} \mathbb{1}. \end{aligned}$$

where $\mathbf{a} = -\mathbf{A}\mathbb{1}$, and $\mathbb{1}$ is the column vector of ones of appropriate size.

Definition 1: The $(\boldsymbol{\alpha}, \mathbf{A})$ representation is called Markovian if $\boldsymbol{\alpha} \geq 0$, $a_{ij} \geq 0, 1 \leq i \neq j \leq n$ and $\mathbf{a} = -\mathbf{A}\mathbb{1} \geq 0$. Then, the generator matrix of the associated CTMC is

$$\overline{\mathbf{A}} = \begin{pmatrix} \mathbf{A} & \mathbf{a} \\ \mathbf{0} & 0 \end{pmatrix}.$$

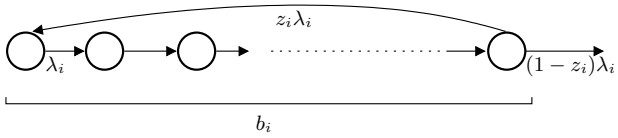

Fig. 1. A Feedback-Erlang block.

Definition 2: The size of the $(\boldsymbol{\alpha}, \mathbf{A})$ representation is the size of the vector $\boldsymbol{\alpha}$, which is equal to the size of the square matrix $\mathbf{A}$.

The $(\boldsymbol{\alpha}, \mathbf{A})$ representation is not unique. When $\mathbf{B}$ is invertible and $\mathbf{B}\mathbb{1} = \mathbb{1}$, then $(\boldsymbol{\alpha}\mathbf{B}, \mathbf{B}^{-1}\mathbf{A}\mathbf{B})$ is another representation of the same distribution, since its CDF is

$$1 - \boldsymbol{\alpha}\mathbf{B}\mathrm{e}^{\mathbf{B}^{-1}\mathbf{A}\mathbf{B}x}\mathbb{1} = 1 - \boldsymbol{\alpha}\mathbf{B}\mathbf{B}^{-1}\mathrm{e}^{\mathbf{A}x}\mathbf{B}\mathbb{1} = 1 - \boldsymbol{\alpha}\mathrm{e}^{\mathbf{A}x}\mathbb{1}.$$

The sizes of the $(\boldsymbol{\alpha}, \mathbf{A})$ and the $(\boldsymbol{\alpha}\mathbf{B}, \mathbf{B}^{-1}\mathbf{A}\mathbf{B})$ representations are the same in this case, but it is also possible to generate representations of the same distribution with any larger size.

Based on the $(\boldsymbol{\alpha}, \mathbf{A})$ representation, we distinguish the following classes of phase-type distributions:

- $\mathrm{HEx}(n)$*:* The Hyper-Exponential distributions of order n. For these, $a_{ii} < 0$, and $a_{ij} = 0$ for $i \neq j$.
- $\mathrm{HErD}(\boldsymbol{\beta}, m, \boldsymbol{b}, \boldsymbol{\lambda})$*:* The Hyper-Erlang distributions with initial distribution $\boldsymbol{\beta}$ (of size m) and m Erlang branches with length b_i and parameter λ_i, i.e., $\mathrm{Erl}(b_i, \lambda_i), i = 1, \ldots, m$. The order of the Hyper-Erlang distribution is $n = \sum_{i=1}^{m} b_i$.
- $\mathrm{APH}(n)$*:* The Acyclic Phase-type distributions of order n, represented in the CF-1 form [1]: $a_{ii} < 0$, $a_{ii+1} = -a_{ii}$ and $a_{ij} = 0$ for $j < i$ and $j > i + 1$.
- $\mathrm{PH}(n)$*:* Any Markovian $(\boldsymbol{\alpha}, \mathbf{A})$ of size n.

Apart from these traditional PH structures we make use of the monocyclic representations of PH distributions introduced in [3]. Monocyclic PH distributions are composed of a series of Feedback-Erlang distributions (Figure 1), which are Erlang blocks with a feedback transition from the last phase of the block to the first one. The degenerate cases with no feedback (true Erlang distribution) and Erlang block of size one (Exponential distribution) are allowed as well. Similarly to the above we thus define:

- $\mathrm{Mono}(\boldsymbol{\alpha}, m, \boldsymbol{b}, \boldsymbol{\lambda}, \boldsymbol{z})$*:* The monocyclic distributions with initial distribution $\boldsymbol{\alpha}$ (of size n) and m Feedback-Erlang

blocks of order b_i, parameter λ_i and feedback probability z_i. The order of the monocyclic distribution is $n = \sum_{i=1}^{m} b_i$.

Any PH distribution has a monocyclic representation [3]. If the representation of the PH distribution is PH-simple [6] and of size n, then the size of the monocyclic representation is $n' \geq n$. This potential size expansion makes the monocyclic representation less efficient in analytical studies, but its simple and still Markovian structure makes it promising for simulation studies.

III. PH-DISTRIBUTED RANDOM-NUMBER GENERATION

The simulation of PH-distributed random numbers is based on the following simple elementary operations:

- Drawing an exponentially distributed sample with parameter λ

$$\mathrm{Exp}(\lambda) = -\frac{1}{\lambda}\ln(U),$$

- Generating an Erlang-distributed sample with degree b and parameter λ

$$\mathrm{Erl}(b,\lambda) = -\frac{1}{\lambda}\ln\left(\prod_{i=1}^{b} U_i\right)$$

- Obtaining a geometrically distributed sample (starting from 0) with parameter p

$$\mathrm{Geo}(p) = \left\lfloor \frac{\ln(U)}{\ln(p)} \right\rfloor,$$

where U denotes a $[0,1]$ uniformly distributed pseudo-random number. The $\mathrm{Erl}(b,\lambda)$ sampling is more efficient than drawing b exponentially distributed samples and summing them up, because the $\ln$ operation is applied only once.

Based on the structural properties of the above-listed PH classes, $\mathrm{HEx}(n) \subset \mathrm{HErD}(n) \subset \mathrm{APH}(n) \subset \mathrm{PH}(n)$ holds, but the subset membership is far from visible based on an arbitrary representation, e.g., the CF-1 form of a $\mathrm{HEx}(n)$ distribution does not indicate the $\mathrm{HEx}(n)$ membership. A random-number generator for one class of phase-type distributions can generate random numbers also for all of the subclasses, which allows a comparison of the computational complexity.

The most natural way to simulate a PH-distributed random number is to play the CTMC until absorption. By 'play' we mean to simulate the state transitions of the CTMC according to the following basic steps. Let $\boldsymbol{e_i}$ denote the row vector with 1 at position i, and 0 everywhere else.

`Procedure Play:`

1) clock= 0, draw an $\boldsymbol{\alpha}$-distributed discrete sample for the initial state,
2) the chain is in state i
   - draw an $\boldsymbol{e}_i(-\mathrm{diag}\langle 1/a_{ii}, 0\rangle\overline{\mathbf{A}} + \mathbf{I})$-distributed discrete sample for the next state,
   - clock += $\mathrm{Exp}(-a_{ii})$,
   - if the next state is the absorbing one go to 3), otherwise go to 2)
3) return the clock value

Instead of individually sampling exponentially distributed samples with the same parameter, [5] proposed the following approach:

`Procedure Count:`

1) clock= 0, count$[i]$ = 0, $(i = 1,..,n)$, draw an $\boldsymbol{\alpha}$-distributed discrete sample for the initial state,
2) the chain is in state i
   - count$[i]$ += 1,
   - draw an $\boldsymbol{e}_i(-\mathrm{diag}\langle 1/a_{ii}, 0\rangle\overline{\mathbf{A}} + \mathbf{I})$-distributed discrete sample for the next state,
   - if the next state is the absorbing one go to 3) otherwise to 2)
3) for $i = 1,\ldots,n$, clock += $\mathrm{Erl}(\text{count}[i], -a_{ii})$ and return the clock value.

The structural properties of the monocyclic representation lead to another approach, where we consider the single exponential phase to be a degenerate Feedback-Erlang block with $z_i = 0$ and $b_i = 1$:

`Procedure Monocyclic:`

1) clock= 0, draw an $\boldsymbol{\alpha}$-distributed discrete sample for the initial state,
2) the chain is in state l of block i (for the left-most state of the block, $l = b_i$)
   - $c = \mathrm{Geo}(z_i)$,
   - clock += $\mathrm{Erl}(cb_i + l, \lambda_i)$ sample,
   - if the next block is the absorbing state go to 3), otherwise $l = b_{i+1}$, $i = i + 1$ and go to 2)
3) return the clock value.

The CF-1 form represents distributions of the $\mathrm{APH}(n)$ class as a chain of phases. For each phase, there is exactly one successor phase. This structural restriction allows the following simplification of `Play`:

`Procedure SimplePlay:`

1) clock= 0, draw an $\boldsymbol{\alpha}$-distributed discrete sample for the initial state.
2) The chain is in state i.
   - clock += $\mathrm{Exp}(-a_{ii})$,
   - i += 1,
   - if the next state is the absorbing state go to 3), otherwise go to 2).
3) Return the clock value.

PH Class	Worst Case #uni	Worst Case #ln	Average Case #uni	Average Case #ln
HEx(n) `SimpleCount`	2	1	2	1
HErD(n) `SimpleCount`	$\max b_i + 1$	1	$\boldsymbol{\beta}\boldsymbol{b}^\mathsf{T} + 1$	1
APH(n) `SimplePlay`	$n+1$	n	$\boldsymbol{\alpha}\boldsymbol{\nu}^\mathsf{T} + 1$	$\boldsymbol{\alpha}\boldsymbol{\nu}^\mathsf{T}$
PH(n) `Play`	∞	∞	$2n^* + 1$	n^*
PH(n) `Count`	∞	n	$2n^* + 1$	n
`Monocyclic`	∞	$3m$	$\boldsymbol{\omega}\boldsymbol{\varphi}^\mathsf{T} + \boldsymbol{\alpha}\boldsymbol{\psi}^\mathsf{T}$	$\boldsymbol{\omega}\boldsymbol{\vartheta}^\mathsf{T}$

TABLE I
THEORETICAL COSTS (WHERE $\boldsymbol{\nu} = (n, n-1, \ldots, 1)$, $n^* = \boldsymbol{\alpha}(\text{DIAG}\langle 1/a_{ii}\rangle\mathbf{A})^{-1}\mathbb{1}$).

For the HErD($\boldsymbol{\beta}, m, \boldsymbol{b}, \boldsymbol{\lambda}$) class, we can simplify the procedure `Count`:

`Procedure SimpleCount:`

1) Draw a $\boldsymbol{\beta}$-distributed discrete sample to choose an Erlang branch i.
2) Return $\text{Erl}(b_i, \lambda_i)$.

IV. COST OF GENERATING PH-DISTRIBUTED NUMBERS

We consider two complexity metrics:

- *#uni*, the number of required uniform random variates, and
- *#ln*, the number of logarithms that need to be computed.

A. Worst-Case Costs

For the APH(n) class and its subclasses we can compute the worst-case cost by considering the longest possible path through the CTMC. We denote the length of this path as $\tilde{n}$. For HEx(n), $\tilde{n} = 1$, for HErD($\boldsymbol{\beta}, m, \boldsymbol{b}, \boldsymbol{\lambda}$), $\tilde{n} = \max b_i$, and for APH(n), $\tilde{n} = n$.

Then, the worst-case costs can be computed as follows: For every class, we need one uniform random variate to choose the initial state. With the HEx(n) class we need another uniform and one logarithm to generate a random number from the chosen exponential distribution. Similarly, for the HErD(n) class we need $\tilde{n}$ additional random variates and one logarithm to obtain an Erlang-distributed random number. Finally, when using the APH(n) class in CF-1 form we need $\tilde{n}$ uniforms and $\tilde{n}$ logarithms for the consecutive phases. These results are summarised in the left half of Table I.

B. Average Costs

As general PH(n) and monocyclic distributions may contain cycles, we cannot construct a worst case for these classes. Instead, we compute the cost of the average case, which is based on the average number of state transitions up to absorption,

$$n^* = \boldsymbol{\alpha}(\text{diag}\langle 1/a_{ii}\rangle\mathbf{A})^{-1}\mathbb{1}.$$

For subclasses of APH(n), n^* is straightforward: $n^* = 1$ for HEx(n), $n^* = \boldsymbol{\beta}\boldsymbol{b}^\mathsf{T}$ for HErD($\boldsymbol{\beta}, m, \boldsymbol{b}, \boldsymbol{\lambda}$), and $n^* = \boldsymbol{\alpha}\boldsymbol{\nu}^\mathsf{T}$ APH(n), where $\boldsymbol{\nu} = (n, n-1, \ldots, 1)$. Using the `SimpleCount` and `SimplePlay` procedures, respectively, we then need one uniform random variate to choose the initial state, and one uniform per traversed phase.

For the general PH(n) class, in each step we need two uniforms because the next phase is chosen randomly. With the `Count` procedure the number of logarithms is n instead of n^* for `Play`.

For Mono($\boldsymbol{\alpha}, m, \boldsymbol{b}, \boldsymbol{\lambda}, \boldsymbol{z}$) we introduce vector $\boldsymbol{\omega}$ of size m, whose ith element is the probability of starting from Feedback-Erlang block i (e.g. $\omega_1 = \sum_{j=1}^{b_1} \alpha_j$), vector $\boldsymbol{\varphi}$ of size m, whose ith element is $\varphi_i = \frac{z_i b_i}{1-z_i} + \sum_{j=i+1}^{m} \frac{b_j}{1-z_j}$ (the mean number of steps spent in a Feedback-Erlang block from the first feedback, i.e. excluding the steps from the initial state to the feedback state in the first passage through the initial block), vector $\boldsymbol{\psi}$ of size n whose ith element indicates how many phases are needed to reach the next Feedback-Erlang block (e.g. if $b_1 \geq 2$ then $\psi_1 = b_1, \psi_2 = b_1 - 1$).

Using these notations the mean number of steps till absorption is

$$n^* = \boldsymbol{\omega}\boldsymbol{\varphi}^\mathsf{T} + \boldsymbol{\alpha}\boldsymbol{\psi}^\mathsf{T},$$

where $\boldsymbol{\alpha}\boldsymbol{\psi}^\mathsf{T}$ contains the number of steps if there is no feedback (i.e., if $z_i = 0$, for $i = 1, \ldots, m$) and $\boldsymbol{\omega}\boldsymbol{\varphi}^\mathsf{T}$ contains the additional number of steps due to the loops in the Feedback-Erlang block.

The mean number of $\ln$ operations is

$$\ell^* = \boldsymbol{\omega}\boldsymbol{\vartheta}^\mathsf{T},$$

where $\boldsymbol{\vartheta}$ is a row vector of size m whose ith element indicates the number of required $\ln$ operations starting from block i. $\vartheta_i = \sum_{j=i}^{m}(1 + 2\ \text{sgn}(z_j))$, since a degenerate Feedback-Erlang block with $z_i = 0$ is Erlang(l, λ_i) distributed which requires one $\ln$ operation and a non degenerate ($z_i > 0$) Feedback-Erlang block requires three $\ln$ operations, two $\ln$ operations for $c = \text{Geo}(z_i)$ and one for $\text{Erl}(cb_i + l, \lambda_i)$.

We can summarise the complexity (in terms of #uni and #ln) of PH classes as follows: For HEx(n) and HErD(n) `SimpleCount` is efficient, because this procedure reduces #ln to 1 and does not draw uniform random variates for the choice of the next state (which is fixed). For APH(n) in CF-1 form, there is again no choice between successor states, and therefore `SimplePlay` is more efficient than `Play`. For the general PH(n) class, `Count` is more efficient than `Play`.

Note that by exploiting structural limitations the cost of random-number generation can be reduced. For instance, generating HErD(n) random variates using `SimplePlay` requires $\sum_{i=1}^{m} b_i$ random variates and n logarithm operations, while the specialised `SimpleCount` procedure has worst-case costs of $\max b_i + 1 \leq \sum_{i=1}^{m} b_i$ uniforms and 1 logarithm.

V. A MOTIVATING NUMERICAL EXAMPLE

In the above discussion we used the number of uniforms and the number of logarithms as measures of the cost of generating a random number. The effect of these measures on the computational cost is very much implementation-dependent. For illustration purposes, we measured the time required to generate a large number of PH-distributed samples on a particular hardware.

Procedure	Play	Count	Monocyclic
$(\boldsymbol{\alpha}, \mathbf{A})$	217 s	155 s	-
$(\boldsymbol{\delta}, \mathbf{D})$	196 s	142 s	-
$(\boldsymbol{\gamma}, \mathbf{G})$	21 s	23 s	22 s

TABLE II
THE RUNNING-TIME OF GENERATING 10^7 SAMPLES IN SECONDS

Starting from representation $(\boldsymbol{\alpha}, \mathbf{A})$, where $\boldsymbol{\alpha} = \{0.7, 0.1, 0.2\}$ and

$$\mathbf{A} = \begin{pmatrix} -1 & 0.9 & 0.1 \\ 1.5 & -2 & 0.5 \\ 2.5 & 0 & -3 \end{pmatrix},$$

we have

$$n^*(\boldsymbol{\alpha}, \mathbf{A}) = \boldsymbol{\alpha}(\text{diag}\langle 1/a_{ii}\rangle \mathbf{A})^{-1}\mathbb{1} = 39.8615.$$

Applying a similarity transformation with

$$\mathbf{B} = \begin{pmatrix} 1.04 & 0 & -0.04 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

results in $\boldsymbol{\delta} = \boldsymbol{\alpha}\mathbf{B} = \{0.728, 0.1, 0.172\}$ and

$$\mathbf{D} = \mathbf{B}^{-1}\mathbf{A}\mathbf{B} = \begin{pmatrix} -0.9 & 0.865385 & 0.0153846 \\ 1.56 & -2 & 0.44 \\ 2.6 & 0 & -3.1 \end{pmatrix},$$

whose associated complexity measure is

$$n^*(\boldsymbol{\delta}, \mathbf{D}) = \boldsymbol{\delta}(\text{diag}\langle 1/d_{ii}\rangle \mathbf{D})^{-1}\mathbb{1} = 36.7209.$$

We used the MOMI tool of Mocanu to obtain the monocyclic representation [3] of the same distribution: $\boldsymbol{\gamma} = \{0.944558, 0.013656, 0.003824, 0.037962\}$ and

$$\mathbf{G} = \begin{pmatrix} -0.0353682 & 0.0353682 & 0 & 0 \\ 0 & -2.66883 & 2.66883 & 0 \\ 0 & 0. & -2.66883 & 2.66883 \\ 0 & 0.034605 & 0. & -2.66883 \end{pmatrix}.$$

For this representation $\ell^* = 3.944558$ and

$$n^*(\boldsymbol{\gamma}, \mathbf{G}) = \boldsymbol{\gamma}(\text{diag}\langle 1/g_{ii}\rangle \mathbf{G})^{-1}\mathbb{1} = 3.90422.$$

This way we obtained three different representations, $(\boldsymbol{\alpha}, \mathbf{A})$, $(\boldsymbol{\delta}, \mathbf{D})$ and $(\boldsymbol{\gamma}, \mathbf{G})$ of the same PH distribution. These representations differ in their analytical parameters and in their size. To evaluate the optimal way of drawing samples from this PH distribution we implemented the simulation procedures `Play`, `Count`, and `Monocyclic` in C++ and measured their performance for the three representations.

The simulations ran under Slackware Linux on an Intel Pentium 4 2.4 GHz-based machine. The measurement simply determines the time difference between the start and the end of each program generating 10^7 samples. To reduce environmental impact as much as possible, our programs do not store the samples in memory or on disk. The results are shown in Table II

The running-times reported in Table II mirror the analytical results (Table I), as the computational cost increases with n^*. For large n^* ($n^* > 30$), the reduced number of $\ln$ operations of the `Count` procedure results in faster computation. For the monocyclic $(\boldsymbol{\gamma}, \mathbf{G})$ representation, the `Count` procedure does not offer an advantage over `Play`, since it requires more logarithm operations than the latter ($n^* < n$), and because it has some book-keeping overhead.

VI. OPTIMISING THE PH-REPRESENTATION FOR RANDOM-NUMBER GENERATION

We can define optimisation problems to minimise the cost of PH-distributed random-number generation. First of all, we need to recall that all of the introduced procedures are based on *Markovian* representations. The authors are not aware of efficient simulation methods for generating PH-distributed random numbers based on a non-Markovian representation. Consequently, the considered representations of the PH distribution should be restricted to be Markovian.

In order to optimise the Markovian representation of a PH-distribution for efficient random-number generation we define the following optimisation problem:

`Starting from a PH-distribution defined by a Markovian representation` $(\boldsymbol{\alpha}, \mathbf{A})$ `find the Markovian representation` $(\boldsymbol{\delta}, \mathbf{D})$ `that minimises` $n^* = \boldsymbol{\delta}(\text{diag}\langle 1/d_{ii}\rangle \mathbf{D})^{-1}\mathbb{1}$, `where the size of` $(\boldsymbol{\delta}, \mathbf{D})$ `might differ from the size of` $(\boldsymbol{\alpha}, \mathbf{A})$.

The solution of this optimisation problem is not available according to the authors' current knowledge. We think it is an interesting research problem which might have application in numerical procedures as well.

Based on the experiences of our motivating example we think that the monocyclic representation is a good heuristic for a representation with low n^* value, but we are at the beginning of this investigation and we need to evaluate several further examples to verify this experience.

VII. CONCLUSION AND FUTURE WORK

In this paper, we considered some factors of the complexity of PH-distributed random-number generation. We collected procedures for general $\text{PH}(n)$ distributions (`Play` and `Count`) and also for PH distributions with specific structure (`Monocyclic`, `SimplePlay` and `SimpleCount`) and compared the `Play`, `Count` and `Monocyclic` procedures using a numeric example.

The main finding of the paper is that the complexity of PH-distributed random-number generation strongly depends on the representation of the PH distribution. Based on this finding we pose a research problem, but did not go far in its solution.

In the near future we intend to attack the posed optimisation problem and find efficient ways for PH-distributed random-number generation. Apart from the solution of the optimisation problem it requires numerical experimentation with the possible simulation procedures and software environments.

ACKNOWLEDGEMENTS

This work was supported by DFG grant Wo 898/2-1 and OTKA grant no. K-61709.

REFERENCES

[1] A. Cumani. On the canonical representation of homogeneous Markov processes modelling failure-time distributions. *Microelectronics and Reliability*, 22:583–602, 1982.

[2] A. Horváth and M. Telek. PhFit: A General Phase-Type Fitting Tool. In *TOOLS '02: Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, pages 82–91, London, UK, 2002. Springer-Verlag.

[3] S. Mocanu and C. Commault. Sparse representations of phase-type distributions. *Commun. Stat., Stochastic Models*, 15(4):759 – 778, 1999.

[4] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models. An Algorithmic Approach*. Dover Publications, Inc., New York, 1981.

[5] M. F. Neuts and M. E. Pagano. Generating random variates from a distribution of phase type. In *WSC '81: Proceedings of the 13th conference on Winter simulation*, pages 381–387, Piscataway, NJ, USA, 1981. IEEE Press.

[6] C. A. O'Cinneide. Phase-type distributions and invariant polytopes. *Advances in Applied Probability*, 23(3):515–535, 1991.

[7] M. Telek and A. Heindl. Matching Moments for Acyclic Discrete and Continous Phase-Type Distributions of second order. *International Journal of Simulation Systems, Science & Technology*, 3(3–4):47–57, Dec. 2002.