Reducing the Cost of Generating APH-distributed Random Numbers

Philipp Reinecke[1], Miklós Telek[2], and Katinka Wolter[1]

[1] Freie Universität Berlin
Institut für Informatik
Takustr. 9
14195 Berlin, Germany
{philipp.reinecke, katinka.wolter}@fu-berlin.de
[2] Budapest University of Technology and Economics
Department of Telecommunications
1521 Budapest, Hungary
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 discuss algorithms for generating random numbers from PH distributions and propose two algorithms for reducing the cost associated with generating random numbers from Acyclic Phase-Type distributions (APH).

1 Introduction

Phase-type (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, where they allow the introduction of realistic response-time distributions obtained from measurements into simulations without modification of the typically Markovian simulation tool. Here, 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. In [1] we posed the following optimisation problem: Starting from a Markovian representation of a PH distribution, find the (not necessarily minimal) Markovian representation that minimises the cost associated with generating random numbers. In this paper we study this optimisation problem for Acyclic Phase-Type (APH) distributions. We provide a result on the optimal

representation and propose a number of algorithms to transform a given APH representation into a representation with lower simulation cost.

The paper is structured as follows. In the next section we introduce the considered model class the notation used throughout the paper. We then describe a number of algorithms for generating random numbers from phase-type distributions and derive average costs. In Section 5 we study the problem of optimising bi-diagonal representations for random-number generation. Section 6 illustrates the application of our algorithms to phase-type distributions fitted to measurement data. Finally, in Section 7 we conclude with an outlook on future work.

2 Definitions and Notation

Continuous phase-type (PH) distributions represent the time to absorption in a continuous-time Markov chain with one absorbing state [2]. 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 [3, 2, 4]:

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

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 $\boldsymbol{\alpha} \geq 0$, $\boldsymbol{\alpha}\mathbb{1} = 1$, $a_{ij} \geq 0, 1 \leq i \neq j \leq n$ and $\mathbf{a} = -\mathbf{A}\mathbb{1} \geq 0$. Then, the generator matrix of the associated CTMC is*

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

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. In particular, another representation of the same size can be obtained by a similarity transformation using a matrix $\mathbf{B}$:

Definition 3. *When $\mathbf{B}$ is invertible and $\mathbf{B}\mathbb{1} = \mathbb{1}$, then the similarity transform*

$$(\boldsymbol{\alpha}\mathbf{B}, \mathbf{B}^{-1}\mathbf{A}\mathbf{B})$$

provides another representation of the same distribution, since its CDF is

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

In the following we refer to a PH representation as being *bi-diagonal* (cf. Figure 1) if it meets the following requirements:

Fig. 1. A bi-diagonal representation.

Definition 4. *A bi-diagonal representation of a PH distribution* $(\boldsymbol{\alpha}, \mathbf{A})$ *has* $a_{ii} < 0, a_{ii+1} = -a_{ii}$ *and* $a_{ij} = 0$ *for* $j < i$ *and* $j > i+1$. *An alternative notation is* $(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$, *with* $\boldsymbol{\Lambda} = (a_1, \ldots, a_n)$ *a row vector of length* n *and* $a_i = -a_{ii}$.

As shown by Cumani [5], every APH can be represented in the CF-1 form, which is a special bi-diagonal representation where the rates a_{ii} are in increasing order:

Definition 5. *CF-1 form [5] The CF-1 form represents an acyclic phase-type distribution as a bi-diagonal matrix with* $a_{ii} < 0, a_{ii+1} = -a_{ii}$ *and* $a_{ij} = 0$ *for* $j < i$ *and* $j > i+1$. *Rates are located along the diagonal in increasing order of magnitude:* $|a_{ii}| \leq |a_{i+1i+1}|$. *That is, in* $\boldsymbol{\Lambda}$, $a_1 \leq a_2 \leq \cdots \leq a_n$.

3 Generation of PH-distributed Random Numbers

We now discuss several methods for generating random variates from a general PH distribution given in Markovian form. These algorithms all rely on the following two 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

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

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.

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

Observe that `Play` draws one exponentially distributed sample for each visit to a state. If a state is visited multiple times, this means repeatedly drawing exponentially distributed samples with the same parameter and then summing them up. [6] proposed the following approach, which replaces the individual exponential samples by drawing one Erlang sample for repeated visits for each state:

`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 $\boldsymbol{e}_i(-\text{diag}\langle 1/a_{ii}, 0\rangle\overline{\mathbf{A}}+\mathbf{I})$-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 += Erl(count$[i], -a_{ii}$) and return the clock value.

We point out that both approaches are suited to general PH distributions in an arbitrary form, where each phase may have several successor phases. With bidiagonal representations such as the CF-1 form we can make use of the following structural restriction: For each phase, there is exactly one successor phase; consequently, there is no need to randomly choose the next state. This observation 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 += 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.

---

4 Average Costs of Generating PH-Distributed Random Numbers

As we saw in the previous section, PH random number generation requires uniform random variates, both for state selection and for generating Erlang and exponential random variates. Furthermore, for Erlang and exponential random variates logarithm operations must be performed. Therefore, we consider the following complexity metrics:

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

The average cost associated with drawing a random variate from a phase-type distribution depends on the average number of state transitions up to absorption,

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

For APH in bi-diagonal form this reduces to

$$n^* = \boldsymbol{\alpha}\boldsymbol{\nu}^{\mathsf{T}},$$

where $\nu = (n, n-1, \ldots, 1)$. Thus $n^* = \sum_{i=1}^{n} \alpha_i(n-i+1)$.

All three procedures require one uniform random variate to choose the initial state. The `Play` and `Count` procedures then need two uniforms per step, because the next phase is chosen randomly. Both `Play` and `Count` therefore require $\#uni = 2n^* + 1$ uniform random variates, while `SimplePlay` requires only $\#uni = n^* + 1$ uniforms. The number of logarithms required for the `Play` and `SimplePlay` procedures is $\#ln = n^*$, since in each phase an exponentially distributed random variate is drawn. Using `Count`, $\#ln$ is equal to the number of visited states, because `Count` counts repeated visits to the same state and draws one Erlang random variate per state instead. Thus $\#ln \leq n$ for `Count`. However, as APH distributions do not contain cycles, all three procedures require the same number of logarithms. We can thus conclude that for generating random numbers from an APH efficiently we should transform the distribution into a bi-diagonal representation and then apply the `SimplePlay` procedure.

5 Optimal Representations for APH-PRNG

As illustrated in Section 4, average costs for random-number generation depend mainly on the number of visited states, n^*. In [1] we posed the problem of finding a Markovian representation that minimises n^*.

In the following we tackle this optimisation problem for acyclic phase-type (APH) distributions $(\boldsymbol{\alpha}, \mathbf{A})$ in CF-1 form. We try to find a bi-diagonal representation $(\boldsymbol{\alpha}^*, \mathbf{A}^*)$ for which the average number of traversed states,

$$n^* = \boldsymbol{\alpha}\boldsymbol{\nu} = \sum_{i=1}^{n} \alpha_i(n-i+1). \tag{1}$$

is minimal.

From the right side of (1) it is immediately obvious that, in order to reduce n^*, probability mass must be shifted to the higher indices of the initial probability vector $\boldsymbol{\alpha}$. Formally, the new probability vector $\boldsymbol{\alpha}'$ must be stochastically larger than $\boldsymbol{\alpha}$:

Definition 6. *The stochastic ordering [7] on the set of stochastic vectors of size n is defined as follows:*

$$\boldsymbol{\alpha} \leq_{st} \boldsymbol{\alpha}' \Leftrightarrow 1 - \Pr\{\boldsymbol{\alpha} \leq k\} \leq 1 - \Pr\{\boldsymbol{\alpha}' \leq k\} \text{ for } k = 1, \ldots, n,$$

where

$$\Pr\{\boldsymbol{\alpha} \leq k\} := \sum_{i=1}^{k} \alpha_i.$$

At the same time, we know that in order to represent the same distribution the matrices $\mathbf{A}$ and $\mathbf{A}^*$ must have the same eigenvalues. Since in the bi-diagonal form the eigenvalues are found on the diagonals, we consider shifting probability mass to the right by modifying the order of the rates along the diagonals. We propose the following operator:

Definition 7. *The* $\mathtt{Swap}(\boldsymbol{\alpha}, \mathbf{A}, i)$ *operator exchanges the* i*th rate with the* $(i+1)$*th rate* $(1 \leq i \leq n-1)$ *on the diagonals in a bi-diagonal representation by swapping the* i*th and* $(i+1)$*th entry in the vector* $\boldsymbol{\Lambda}$*. The associated similarity transformation matrix* $\mathbf{B}$ *has the form*

$$\mathbf{B} = \begin{pmatrix} \ddots & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & b_{i+1,i} & b_{i+1,i+1} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \ddots \end{pmatrix}$$

where

$$b_{i+1,i} = \frac{a_i - a_{i+1}}{a_i}, \text{ and } b_{i+1,i+1} = \frac{a_{i+1}}{a_i} \text{ for } 1 \leq i \leq n-1.$$

Where appropriate, we also use the $\mathtt{Swap}(\boldsymbol{\alpha}, \boldsymbol{\Lambda}, i)$ notation to denote the same operator.

Let $(\boldsymbol{\alpha}', \mathbf{A}')$ denote the result of applying $\mathtt{Swap}(\boldsymbol{\alpha}, \mathbf{A}, i)$ on $(\boldsymbol{\alpha}, \mathbf{A})$. Since $(\boldsymbol{\alpha}', \mathbf{A}')$ is derived by applying a similarity transformation to $(\boldsymbol{\alpha}, \mathbf{A})$, both tuples represent the same distribution. Recall from Definition 3 that

$$\boldsymbol{\alpha}' = \boldsymbol{\alpha}\mathbf{B}.$$

Then, the following properties of the result of the `Swap` operation are immediately obvious:

$$\forall j \notin \{i, i+1\} : \alpha'_j = \alpha_j \tag{2}$$

$$\alpha'_i = \alpha_i + \alpha_{i+1}\frac{a_i - a_{i+1}}{a_i} = \alpha_i + \alpha_{i+1}(1 - \frac{a_{i+1}}{a_i}) \tag{3}$$

$$\alpha'_{i+1} = \alpha_{i+1}\frac{a_{i+1}}{a_i} \tag{4}$$

$$n^*(\boldsymbol{\alpha}', \mathbf{A}') = n^*(\boldsymbol{\alpha}, \mathbf{A}) + \alpha_{i+1}(1 - \frac{a_{i+1}}{a_i}). \tag{5}$$

In order to have $\boldsymbol{\alpha} <_{st} \boldsymbol{\alpha}'$, $\alpha'_i < \alpha_i$ and $\alpha_{i+1} < \alpha'_{i+1}$ must hold. From (3) and (4) we see that $1 < \frac{a_{i+1}}{a_i}$ (and thus $n^*(\boldsymbol{\alpha}', A') < n^*(\boldsymbol{\alpha}, A)$, (5)), i.e. $\boldsymbol{\alpha} <_{st} \boldsymbol{\alpha}'$ if we move the higher rate to the left. Consequently, by repeatedly exchanging

adjacent rates $a_i < a_{i+1}$ until no such operations are possible anymore, we can obtain a representation that has minimal costs n^*.

However, the `Swap` operator is limited by the fact that it will result in a non-stochastic vector $\boldsymbol{\alpha}'$ if $\alpha_i < \alpha_{i+1}(1-\frac{a_{i+1}}{a_i})$, since then the resulting $\boldsymbol{\alpha}'_i < 0$. In this case $(\boldsymbol{\alpha}', \mathbf{A})$ is a non-Markovian matrix-exponential (ME) representation of the original phase-type distribution. This representation is not suitable as input for the random-number generation algorithms discussed in Section 3. Furthermore, both the stochastic ordering and n^* are only defined for stochastic $\boldsymbol{\alpha}$. We must therefore avoid `Swap` operations that will result in non-stochastic $\boldsymbol{\alpha}'$. Based on these observations we propose the following

Lemma 1. *Given a Markovian representation $(\boldsymbol{\alpha}, \mathbf{A})$ in CF-1 form, the representation $(\boldsymbol{\alpha}^*, \mathbf{A}^*)$ that reverses the order of the rates is optimal with respect to n^* if $\boldsymbol{\alpha}^*$ is a stochastic vector.*

Proof. By contradiction: Assume the representation $(\boldsymbol{\alpha}', \mathbf{A}')$ with rates

$$a'_1, \ldots, a'_i, a'_{i+1}, \ldots, a'_n$$

ordered such that $a'_i < a'_{i+1}$ is optimal. Then from (5) it follows that by exchanging a'_i, a'_{i+1} using the `Swap` operator we can obtain a representation $(\boldsymbol{\alpha}'', \mathbf{A}'') = \mathtt{Swap}(\boldsymbol{\alpha}', \mathbf{A}', i)$ for which $n^*(\boldsymbol{\alpha}'', \mathbf{A}'') < n^*(\boldsymbol{\alpha}', \mathbf{A}')$. □

5.1 Algorithms for Computing Optimal APH Representations

We will now develop algorithms for finding the Markovian APH representation that is optimal (with respect to n^*) for generating random numbers. The most obvious approach proceeds by enumerating all permutations of the rate vector $\boldsymbol{\Lambda}$ and minimising n^* over the subset of permutations for which the associated initial vector is stochastic. This approach is easily implemented as a modification to the Steinhaus-Johnson-Trotter algorithm for enumerating permutations [8] and is guaranteed to find the optimum. Unfortunately, the approach requires exploring $n!$ permutations for an APH of size n, which makes it infeasible for APH of realistic size.

The two algorithms presented in this section both implement a directed search strategy based on Lemma 1 and are thus more efficient than an exhaustive search. On the other hand, they are not guaranteed to find the optimum.

The first algorithm follows directly from Lemma 1 and is a variant of the Bubblesort algorithm [9] on the vector $\boldsymbol{\Lambda}$:

```
Algorithm BubblesortOptimise(α, Λ):
For i = 1, ..., n − 1 do
    For j = 1, ..., n − 1 do
        If Λ[j] < Λ[j + 1] ∧ (α', Λ') := Swap(α, Λ, i) is Markovian then
            (α, Λ) := (α', Λ')
        Else
```

```
            break
    done
done
Return (α, Λ)
```

The algorithm attempts to re-order the rates such that the reversed CF-1 form is obtained, i.e. it tries to sort the rates in descending order. Because the algorithm only performs `Swap` operations if the result is Markovian, we cannot guarantee that it finds the optimal Markovian representation, because some Markovian representations can be hidden 'behind' non-Markovian ones.

The second algorithm starts from the reversed CF-1 and tries to find a Markovian representation by successively eliminating negative entries in $\boldsymbol{\alpha}$ if they exist:

```
Algorithm FindMarkovian:
Let (α', A') be the reversed CF-1 of (α, A).
While ∃i ∈ {1, ..., n − 1} : α'_i < 0
      i := argmin_i {α'_i < 0}
      i := max(1, i)
      While α' is not Markovian ∧∃k : Λ[k] ≥ Λ[k + 1]
            k := argmin_j{i − 1 ≤ j ≤ n − 1 : Λ[j] ≥ Λ[j + 1]}
            (α', Λ') := Swap(α', Λ', k)
      end
end
Return (α', Λ')
```

6 Illustrative Examples

We will now illustrate our results on several APH distributions.

Example 1: Consider the generalised Erlang distribution with $\boldsymbol{\Lambda}_A = (1, 2, 3, 4)$ and $\boldsymbol{\alpha}_1 = (1, 0, 0, 0)$. For this distribution, every order of rates in $\boldsymbol{\Lambda}_A$ has costs $n^* = 4$, since no probability mass can be shifted to the right. As expected, both `BubblesortOptimise` and `FindMarkovian` identify $(\boldsymbol{\alpha}'_1 = (1, 0, 0, 0), \boldsymbol{\Lambda}'_A = (4, 3, 2, 1)$ as the optimal case.

Example 2: We assign the initial probability vector $\boldsymbol{\alpha}_2 = (0.7, 0.15, 0.09, 0.06)$ to $\boldsymbol{\Lambda}_A$. Then, the average number of visited states is

$$n^*(\boldsymbol{\alpha}_2, \boldsymbol{\Lambda}_A) = 3.49.$$

Application of `BubblesortOptimise` results in the reversed CF-1 form with $\boldsymbol{\Lambda}''_A = (4, 3, 2, 1)$, $\boldsymbol{\alpha}_2 = (0.46, 0.12, 0.18, 0.24)$ and costs

$$n^*(\boldsymbol{\alpha}'_2, \boldsymbol{\Lambda}''_A) = 2.8.$$

Since the reversed CF-1 is Markovian, `FindMarkovian` gives the same result. We observe that probability mass in the initial probability vector has been shifted towards higher indices.

Example 3: We study $(\boldsymbol{\alpha}_3, \mathbf{A})$ with $\boldsymbol{\alpha} = (0.5, 0.4, 0.05, 0.05)$ and again $\boldsymbol{\Lambda}_A = (1, 2, 3, 4)$. This representation has costs

$$n^*(\boldsymbol{\alpha}_3, \mathbf{A}) = 3.35.$$

The reversed CF-1, $(\boldsymbol{\alpha}'_3, \mathbf{A}')$ has initial vector $\boldsymbol{\alpha}' = (-0.6, 1.4, 0, 0.2)$ and is therefore non-Markovian. Applying the `BubblesortOptimise` algorithm provides us with a representation $(\boldsymbol{\alpha}''_3, \boldsymbol{\Lambda}''_A)$ with $\boldsymbol{\Lambda}''_A = (2, 4, 3, 1)$ and $\boldsymbol{\alpha}''_3 = (0.1, 0.7, 0, 0.2)$, for which

$$n^*(\boldsymbol{\alpha}''_3, \mathbf{A}'') = 2.7.$$

`FindMarkovian` starts on the non-Markovian reversed CF-1 representation and generates the Markovian representation $\boldsymbol{\Lambda}'''_A = (2, 3, 4, 1)$ and $\boldsymbol{\alpha}'''_3 = (0.1, 0.7, 0, 0.2)$, which has the same costs of 2.7. A complete enumeration of all permutations shows that both orderings are optimal with respect to n^*.

Example 4: As the last example, we fit an APH(8) to the `loss1-50-opc-1` dataset from [10] using the PhFit tool [3]. This data set contains response-time measurements from a SOA system under high load and with network packet loss. The resulting APH has initial probability vector $\boldsymbol{\gamma} = (0.019, 0.006, 0.069, 0.104, 0.164, 0.371, 0.216, 0.051)$ and rate vector

$$\begin{aligned}\boldsymbol{\Lambda}_G = (&7.181e-05, 2.4280e-04, 5.854e-04, 5.863e-04,\\ &5.956e-04, 5.965e-04, 6.178e-04, 6.332e-04).\end{aligned}$$

For this representation,

$$n^*(\boldsymbol{\gamma}, \mathbf{G}) = 3.381.$$

Again, the reversed CF-1 for this representation has negative entries in the initial vector. Application of `BubblesortOptimise` results in $(\boldsymbol{\gamma}', \mathbf{G}')$ with initial probability vector $\boldsymbol{\gamma}' = (0.006, 0.019, 0.061, 0.093, 0.133, 0.391, 0.242, 0.055)$ and

$$\begin{aligned}\boldsymbol{\Lambda}'_G = (&2.4280e-04, 7.181e-05, 6.332e-04, 6.178e-04,\\ &5.965e-04, 5.956e-04, 5.863e-04, 5.854e-04),\end{aligned}$$

which has $n^*(\boldsymbol{\gamma}', \mathbf{G}') = 3.257$. `FindMarkovian` terminates with $\boldsymbol{\gamma}'' = (0.006, 0.019, 0.069, 0.104, 0.164, 0.371, 0.216, 0.051)$,

$$\begin{aligned}\boldsymbol{\Lambda}''_G = (&2.4280e-04, 7.181e-05, 5.854e-04, 5.863e-04,\\ &5.956e-04, 5.965e-04, 6.178e-04, 6.332e-04)\end{aligned}$$

and $n^*(\boldsymbol{\gamma}', \mathbf{G}') = 3.357$.

6.1 Discussion

In our examples we observe that the effectiveness of the algorithms depends strongly on the initial representations. Representations with (generalised) Erlang structure are invariant to re-ordering of rates. The same holds for blocks of subsequent phases with initial probability 0, as illustrated by the third example. For representations where the probability mass is already concentrated at the higher indices in the CF-1, there is also little room for improvement.

In general, our examples indicate that there are phase-type distributions for which re-ordering of rates does indeed lead to a cost reduction. On the other hand, we can identify (generalised) Erlang structure and large probability mass at the higher indices as two properties of representations that are not susceptible to the proposed optimisation. However, so far we have not been able to find more formal criteria for when and why the optimisation procedures fail. Such criteria would not only help in improving the optimisation algorithms, but may also enable the development of specialised PH-fitting methods that give APH distributions suited for efficient random-number generation.

7 Conclusion and Future Work

In this paper we considered the complexity of generating random numbers from phase-type distributions. Our focus lay on bi-diagonal representations of acyclic phase-type distributions, whose structural limitations enable the `SimplePlay` procedure which is more effective than the more general `Play` and `Count` procedures. By re-ordering rates along the diagonal we provided a first attempt at optimising the bi-diagonal representation for efficient random-number generation. We provide a limited result for the optimal ordering and propose two algorithms for finding the optimal representation, given an APH in CF-1 form.

We note that the effectiveness of our approach depends on the given APH. While we can provide a number of intuitive guidelines, formal criteria for deciding when re-ordering rates may provide an advantage are still future work. Furthermore, in the near future we will extend our approach to eliminate the limitations of our result, and we will apply the approach to general phase-type distributions in Monocyclic form.

Acknowledgements

This work was supported by DFG grant Wo 898/2-1, Wo 898/3-1, and OTKA grant no. K-61709.

References

1. Reinecke, P., Wolter, K., Bodrog, L., Telek, M.: On the Cost of Generating PH-distributed Random Numbers. In Horváth, G., Joshi, K., Heindl, A., eds.: Proceedings of the Ninth International Workshop on Performability Modeling of Computer

and Communication Systems (PMCCS-9), Eger, Hungary (September 17–18, 2009 2009)
2. Neuts, M.F.: Matrix-Geometric Solutions in Stochastic Models. An Algorithmic Approach. Dover Publications, Inc., New York (1981)
3. Horváth, A., Telek, M.: PhFit: A General Phase-Type Fitting Tool. In: TOOLS '02: Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools, London, UK, Springer-Verlag (2002) 82–91
4. Telek, M., Heindl, A.: Matching Moments for Acyclic Discrete and Continous Phase-Type Distributions of Second Order. International Journal of Simulation Systems, Science & Technology **3**(3–4) (December 2002) 47–57
5. Cumani, A.: On the canonical representation of homogeneous Markov processes modelling failure-time distributions. Microelectronics and Reliability **22** (1982) 583–602
6. Neuts, M.F., Pagano, M.E.: Generating random variates from a distribution of phase type. In: WSC '81: Proceedings of the 13th conference on Winter simulation, Piscataway, NJ, USA, IEEE Press (1981) 381–387
7. Szekli, R.: Stochastic Ordering and Dependence in Applied Probability. Springer Verlag (1995)
8. Johnson, S.M.: Generation of Permutations by Adjacent Transposition. Mathematics of Computation **17**(83) (July 1963) 282–285
9. Knuth, D.E.: The Art of Computer Programming. Volume 3. Addison-Wesley (1997)
10. Reinecke, P., Wittkowski, S., Wolter, K.: Response-time Measurements Using the Sun Java Adventure Builder. In: QUASOSS '09: Proceedings of the 1st International Workshop on Quality of Service-oriented Software Systems, New York, NY, USA, ACM (2009) 11–18