Abstract

Interval availability is a dependability measure defined by the fraction of time during which a system is in operation over a finite observation period. The computation of its distribution allows the user to ensure that the probability that its system will achieve a given availability level is high enough.

As usual, the system is assumed to be modeled by a finite Markov process. We propose in this paper two new algorithms to compute this measure and we compare them with respect of the input parameters of the model, both through the storage requirement and the execution time points of view. We show that one of them is an improvement of a well known one. Both algorithms are based on the uniformization technique.

Index terms - Repairable computer systems, dependability, interval availability, Markov processes, uniformization.

1 Introduction

In the dependability analysis of repairable computing systems, there is an increasing interest in evaluating cumulative measures, in particular the availability over a given period. In highly available systems, steady state measures can be very poor, even if the mission time is not small. The use of expectations also suffers from similar drawbacks. Considering, for instance, critical applications, it is crucial for the user to ensure that the probability that its system will achieve a given availability level is high enough. This paper deals with the computation of the distribution of the interval availability which is defined by the fraction of time during which a system is in operation over a finite observation period.

Formally, if the system is modeled by a Markov process, its state space is divided into two disjoint sets representing the up states in which the system delivers the specified service and the down states in which there is no more service delivered. Transitions from the up (resp. down) states to the down (resp. up) states are called failures (resp. repairs). The interval availability over (0, t) is the fraction of the interval (0, t) during which the process is in the up states. This random variable has been studied in previous papers as for instance in [1], where its distribution is calculated recursively by discretizing the observation period (0, t) into intervals of length \( \Delta t \) small enough so that the probability of two or more events occurring during \( \Delta t \) is negligible. However, no error bounds were found for this approximation method. In [2], a particular algorithm has been developed in the case where the two sequences of sojourn times in the subsets of operational and unoperational states are both independent and independent of each other. It is also shown that this property can be checked directly on the transition rate matrix of the process. In [3] the evaluation of the distribution of the interval availability is based on the uniformization technique. This approach is interesting because it has good numerical properties and, moreover, it allows the user to perform the computation with an error as small as desired.

In this paper we develop two new methods to compute the interval availability distribution. They are based on the uniformization technique and the starting point is the work performed in [3].

The remainder of the paper is organized as follows. In the following section, we recall the main relation and the corresponding algorithm of [3]. In Section 3, we give a first algorithm which is an improvement of the previous one. It needs the same memory space but it is faster. In Section 4, we propose a second method. Its main characteristics are that it needs less memory space than the previous methods and that its space requirements are known at the beginning of the execution. In Section 5 we compare the complexity of these algorithms and we illustrate the results with some numerical values.

2 Interval Availability Distribution

Consider a continuous-time homogeneous Markov process \( X = \{X_t, t \geq 0\} \), over a finite state space denoted by \( S \). The states of \( S \) are divided into two disjoint subsets:
$U$, the set of the operational states (or the up states) and $D$, the set of the unoperational states (or the down states). We assume that the system, modeled by such a process, is considered during a finite interval of time denoted by $(0, t)$. Of interest is the cumulative amount of operational time (up time) during $(0, t)$, defined by
\[
O(t) = \int_0^t I(s)ds
\]
where $I(s)$ denotes the indicator random variable
\[
I(s) = \begin{cases} 1 & \text{if } X_s \in U, \\ 0 & \text{otherwise}. \end{cases}
\]
From this, the interval availability over $(0, t)$, that is, the fraction of time the system is in operation during $(0, t)$, is defined by
\[
\text{IAV}(t) = \frac{O(t)}{t}.
\]
The process $X$ is, as usual, given by its infinitesimal generator, denoted by $A$, in which the $i$th diagonal entry $A(i,i)$ verifies $A(i,i) = -\sum_{j \neq i} A(i,j)$ and by its initial probability distribution $\alpha$.

Let us denote by $Z$ the uniformized Markov chain with respect to the uniformization rate $\lambda$ and by $P$ its transition probability matrix [4]. The uniformization rate $\lambda$ verifies $\lambda \geq \max(-A(i,i), i \in E)$ and $P$ is related to $A$ by $P = I + A/\lambda$, where $I$ denotes the identity matrix. We decompose $P$ and the initial probability vector $\alpha$ with respect to the partition $\{U, D\}$ of $S$ as follows:
\[
P = \begin{pmatrix} P_U & P_{UD} \\ P_{DU} & P_D \end{pmatrix}, \quad \alpha = (\alpha_U, \alpha_D).
\]
Let us now briefly recall how the distribution of $O(t)$ is derived in [3]. Using results on order statistics of identically and uniformly distributed random variables in $(0, t)$, it is shown that for $p < 1$ (and thus for $k \leq n$),
\[
\mathbb{P}(\text{IAV}(t) \leq p \mid n \text{ transitions in } (0, t), \ k \text{ states of } U \text{ visited}) = \sum_{i=k}^{n} C_n^i p^i (1-p)^{n-i}.
\]
Let $\Omega(n, k)$, $0 \leq k \leq n + 1$, be the probability that the uniformized Markov chain $Z$ visits $k$ states of $U$ during its $n$ first transitions. Unconditioning with respect to the number of visited states of $U$, we have
\[
\mathbb{P}(\text{IAV}(t) \leq p \mid n \text{ transitions in } (0, t)) = \sum_{k=0}^{n} \Omega(n, k) \sum_{i=k}^{n} C_n^i p^i (1-p)^{n-i}.
\]
Finally, unconditioning on the number of transitions in $(0, t)$, 
\[
\mathbb{P}(\text{IAV}(t) \leq p) = \sum_{n=0}^{+\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \sum_{k=0}^{n} \Omega(n, k) \sum_{i=k}^{n} C_n^i p^i (1-p)^{n-i}.
\]
Let $e(N)$ be the error obtained when the previous series is truncated at step $N$. We write
\[
\mathbb{P}(\text{IAV}(t) \leq p) = e(N) + \sum_{n=N+1}^{+\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \sum_{k=0}^{n} \Omega(n, k) \sum_{i=k}^{n} C_n^i p^i (1-p)^{n-i}.
\]
It is immediately checked that $e(N)$ can be bounded in the following manner:
\[
e(N) = \sum_{n=N+1}^{+\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \sum_{k=0}^{n} \Omega(n, k) \sum_{i=k}^{n} C_n^i p^i (1-p)^{n-i} \leq 1 - \sum_{n=0}^{N} e^{-\lambda t} \frac{(\lambda t)^n}{n!}
\]
and so $N$ can be evaluated beforehand for a given specified error tolerance.

In order to evaluate the values of $\Omega(n, k)$, let $\Omega_U(n, k)$ (resp. $\Omega_U(n, k)$, $\Omega_D(n, k)$) be the row vector so that its $i$th entry, $i \in S$ (resp. $i \in U$, $i \in D$), is the probability that the Markov chain $Z$ visits $k$ states of subset $U$ during the first $n$ transitions and the $n$th transition leads to state $i$. Thus,
\[
\Omega(n, k) = \Omega_U(n, k)1^T \quad \text{where } 1 = (1 \ 1 \ldots \ 1) \text{ and } (.)^T \text{ denotes the transpose operator.}
\]
Using the backward renewal equations on the Markov chain $Z$, we have
\[
\Omega_U(n, k) = \Omega_U(n-1, k-1)P_U + \Omega_D(n-1, k-1)P_{UD} \quad \text{and} \quad \Omega_D(n, k) = \Omega_U(n-1, k)P_{UD} + \Omega_D(n-1, k)P_D
\]
with initial conditions,
\[
\Omega_U(0, 0) = 0, \quad \Omega_D(0, 0) = \alpha_D, \quad \Omega_U(0, 1) = \alpha_U, \quad \Omega_D(0, 1) = 0.
\]
Finally, changing the summation order on relation (2) and performing a second truncation,
\[
\mathbb{P}(\text{IAV}(t) \leq p) = e(N) + e'(N, C) + \text{...}
\]
\[
\sum_{k=1}^{N} \sum_{n=k}^{n} e^{-\lambda t} \left(\frac{\lambda t}{n!}\right)^n \frac{n!}{n} \Omega(n, n - k + 1) \times \\
\sum_{i=n-k+1}^{n} c_i p^i (1 - p)^{n-i} 
\]

where \( c' (N, C) \) verifies

\[
e' (N, C) \leq 1 - \sum_{k=0}^{C} \Omega(n, N - k + 1). 
\] (7)

This bound is obtained from the following relation which is proved in the Appendix of [3]: for every integers \( C, n, m \) such that \( 0 \leq C \leq m \leq n \),

\[
\sum_{k=0}^{C} \Omega(n, n - k + 1) \leq \sum_{k=0}^{C} \Omega(m, m - k + 1). 
\] (9)

The following figure shows how the computation of \( \Omega(n, k) \) is done in a column by column manner.

The contents of cell \((n, k)\) is the vector \( \Omega_{S}(n, n - k + 1) \).

Cell \((n, k)\) is filled from cells \((n-1, k)\) and \((n-1, k-1)\) using the recurrences (5) and (6). This means that it is necessary to store a whole column to compute the next one. The algorithm proceeds in the following manner. Given a tolerance error \( \varepsilon \) specified by the user, the first truncation step is \( N \), computed from (3) and (4) as

\[
N = \min \left\{ n \in \mathbb{N} \left| \sum_{j=0}^{n} e^{-\lambda t} \left(\frac{\lambda t}{j!}\right)^j \geq 1 - \frac{\varepsilon}{2} \right. \right\}.
\] (10)

Then, the table of Figure 1 is computed column by column. The algorithm stops after column \( C \) where, using the bound of (8),

\[
C = \min \left\{ c \in \mathbb{N} \left| \sum_{k=0}^{c} \Omega(n, N - k + 1) \geq 1 - \frac{\varepsilon}{2} \right. \right\}.
\] (11)

To do this, the sum of the scalars \( \Omega_{N,N-k+1} \) is computed from the elements of the last line of the table that have been already computed.

3 Algorithm I

In this section, we improve the previous algorithm from the computational point of view. We start from relation (1) which can be rewritten, for \( p < 1 \), as

\[
\mathbb{P}(IAV(t) \leq p) = \\
\sum_{n=0}^{+\infty} e^{-\lambda t} \left(\frac{\lambda t}{n!}\right)^n \sum_{k=0}^{n} c_{n} p^{k} (1 - p)^{n-k} W(n, k)
\] (12)

where \( W(n, k) \) is defined, for \( 0 \leq k \leq n + 1 \), by

\[
W(n, k) = \sum_{i=0}^{k} \Omega(n,i),
\] (13)

that is, \( W(n, k) \) is the probability that \( Z \) visits at most \( k \) states of \( U \) during the first \( n \) transitions. To evaluate the values of \( W(n, k) \), we use the forward renewal equations (instead of the backward ones). Let \( W_{S}(n, k) \) (resp. \( W_{T}(n, k), W_{D}(n, k) \)) be the column vector so that its \( i \)th entry, \( i \in S \) (resp. \( i \in U, i \in D \)), is the probability that \( Z \) visits at most \( k \) states of \( U \) during the first \( n \) transitions, given that the initial state is \( i \). We have

\[
W(n, k) = \alpha W_{S}(n, k).
\]

Using the forward renewal equations on the Markov chain \( Z \), we get

\[
W_{T}(n, k) = P_{D} W_{T}(n - 1, k - 1) + P_{U} W_{T}(n - 1, k) + P_{D} W_{D}(n - 1, k)
\] (14)

with initial conditions \( W_{T}(n, 0) = 0 \) and \( W_{D}(0, 0) = 1^T \). Note also that \( W_{S}(n, n + 1) = 1^T \). These equations are more interesting than the backward ones since, in practice, the initial probability is almost always concentrated in only one state \( l \), and so \( W(n, k) \) is equal to the \( l \)th component of the vector \( W_{S}(n, k) \).
Let us rewrite relation (12), by making the variable change \( k \rightarrow n - k \) in the second sum:

\[
P(IAV(t) \leq p) = \sum_{n=0}^{+\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \sum_{k=0}^{n} C_n^k p^{n-k} (1 - p)^k W(n, n - k).
\]

As in the previous section, the series is truncated at step \( N \) and we write

\[
P(IAV(t) \leq p) = e(N) + \sum_{n=N}^{+\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \sum_{k=0}^{n} C_n^k p^{n-k} (1 - p)^k W(n, n - k)
\]

This relation can be written as

\[
P(IAV(t) \leq p) = e(N) + e_1(N, C_1) + \sum_{k=0}^{N} \sum_{n=k}^{N} e^{-\lambda t} \frac{(\lambda t)^n}{n!} C_n^k p^{n-k} (1 - p)^k W(n, n - k)
\]

where

\[
e_1(N, C_1) = \sum_{k=C_1}^{N} \sum_{n=k}^{N} e^{-\lambda t} \frac{(\lambda t)^n}{n!} C_n^k p^{n-k} (1 - p)^k W(n, n - k).
\]

The term \( e_1(N, C_1) \) is bounded in the following way:

\[
e_1(N, C_1) \leq \sum_{k=C_1}^{N} W(N, N - k) \sum_{n=k}^{N} e^{-\lambda t} \frac{(\lambda t)^n}{n!} C_n^k p^{n-k} (1 - p)^k.
\]

This comes from the inequality \( W(n, k) \leq W(n + 1, k + 1) \), which is an equivalent form of (9). The following second relation:

\[
W(n, k) \leq W(n, k + 1),
\]

a trivial consequence of the definition of \( W(n, k) \), allows us to obtain

\[
e_1(N, C_1) \leq W(N, N - C_1) \sum_{k=C_1}^{N} \sum_{n=k}^{N} e^{-\lambda t} \frac{(\lambda t)^n}{n!} C_n^k p^{n-k} (1 - p)^k.
\]

Finally,

\[
e_1(N, C_1) \leq W(N, N - C_1) \times 
\sum_{k=C_1+1}^{N} \sum_{n=k}^{N} e^{-\lambda t} \frac{(\lambda t)^n}{n!} C_n^k p^{n-k} (1 - p)^k \frac{e^{-\lambda(1-p)(\lambda t(1-p))^k}}{k!}
\]

\[
= W(N, N - C_1) \times 
\sum_{k=C_1+1}^{N} e^{-\lambda t(1-p)} \frac{(\lambda t(1-p))^k}{k!} \sum_{n=0}^{N-k} e^{-\lambda t(1-p)} \frac{(\lambda t(1-p))^n}{n!}
\]

\[
\leq (1 - \sum_{k=0}^{C_1} e^{-\lambda t(1-p)} \frac{(\lambda t(1-p))^k}{k!}) \times (1 - \sum_{k=0}^{C_1} e^{-\lambda t(1-p)} \frac{(\lambda t(1-p))^k}{k!}) \leq \frac{e}{2}.
\]

Remark that, since for every \( l \) such that \( 0 \leq l \leq N \),

\[
W(N, N - l) = \sum_{k=l+1}^{N+1} \Omega(N, N - k + 1),
\]

we have \( C_1 \leq C \), which proves that this algorithm runs faster than the algorithm in [3]; moreover, the difference becomes important when \( p \) is near from 1 since \( C_1 \) decreases when \( p \) increases, while \( C \) does not depend on \( p \).

### 4 Algorithm II

As we have seen, the two previous algorithms need a large amount of memory to run. Once \( N \) is known, the workspace is composed basically by \( N \) vectors having the size of the whole state space. In this section, a different approach is followed to derive a new method with a better space complexity. Let us come back to relation (12), that we write as

\[
P(IAV(t) > p) = \sum_{n=0}^{+\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} \sum_{k=0}^{n} C_n^k p^k (1 - p)^{n-k} Y(n, k)
\]

where

\[
Y(n, k) = 1 - W(n, k).
\]
The usual first truncation leads to
\[
P(IAV(t) > p) = e(N) + \sum_{n=0}^{N} e^{-\lambda t} \frac{(\lambda t)^n}{n!} C_{n}^{k} p^k (1 - p)^{n-k} Y(n, k).
\]  

As previously, we change the order of the sums to obtain
\[
P(IAV(t) > p) = e(N) + \sum_{n=k}^{N} e^{-\lambda t} \frac{(\lambda t)^n}{n!} C_{n}^{k} p^k (1 - p)^{n-k} Y(n, k)
\]

and then, we perform a second truncation:
\[
P(IAV(t) > p) = e(N) + e_2(N, C_2) +
\]

\[
\sum_{k=0}^{N-C_2+k} \sum_{n=k}^{N} e^{-\lambda t} \frac{(\lambda t)^n}{n!} C_{n}^{k} p^k (1 - p)^{n-k} Y(n, k)
\]

where \(e_2(N, C_2)\) verifies
\[
e_2(N, C_2) = \sum_{k=0}^{N-C_2-1} \sum_{n=C_2+k}^{N} e^{-\lambda t} \frac{(\lambda t)^n}{n!} C_{n}^{k} p^k (1 - p)^{n-k} Y(n, k)
\]

Let \(N\) be computed as in the previous algorithms and \(C_2\) as
\[
C_2 = \min \{ c \leq N \mid 1 - \sum_{k=0}^{c} e^{-\lambda (1-p)} \frac{(\lambda (1-p))^k}{k!} \leq \frac{\varepsilon}{2} \}.
\]

Then, the total error introduced when the double truncation is performed is less than \(\varepsilon\). As for \(C_1\), the truncation step \(C_2\) decreases when \(p\) increases.

The recurrences to compute the \(Y(n, k)\’s\) can be derived from (14) or directly by writing the corresponding forward equations. They are
\[
Y_D(n, k) = P_D Y_D(n - 1, k - 1) + P_{UD} Y_D(n - 1, k - 1) + P_Y Y_D(n - 1, k)
\]

with initial conditions \(Y_D(n, 0) = 1^T\) and \(Y_D(0, 0) = 0\).

To evaluate the two first terms in (21), as we know the value of \(C_2\), we need to store only \(C_2\) vectors having the size of the whole state space, in contrast with the \(N\) vectors needed by the previous algorithms. The key feature of this technique is then the fact that we are able to compute the index \(C_2\) beforehand. In the previous algorithms, the second truncation steps \(C\) and \(C_1\) are known only at the end of the execution.

A further improvement can be made in the following way. Consider the greatest integer \(C_3\) such that
\[
\sum_{n=0}^{C_3} e^{-\lambda (1-p)} \frac{(\lambda (1-p))^n}{n!} \leq \frac{\varepsilon}{3}.
\]

Such a value of \(C_3\) exists if \(e^{-\lambda (1-p)} \leq \frac{\varepsilon}{3}\). Assume that this is the case. Relation (21) can be written as
\[
P(IAV(t) > p) = e(N) + e_2(N, C_2) + e_3(N, C_2, C_3)
\]

\[
+ \sum_{k=0}^{N-C_2+k} \sum_{n=k}^{N} e^{-\lambda t} \frac{(\lambda t)^n}{n!} C_{n}^{k} p^k (1 - p)^{n-k} Y(n, k)
\]

\[
+ \sum_{k=N-C_3+1}^{N-C_2-1} \sum_{n=k}^{N} e^{-\lambda t} \frac{(\lambda t)^n}{n!} C_{n}^{k} p^k (1 - p)^{n-k} Y(n, k).
\]

Note that if \(C_3 = C_2 - 1\) then the second term in the previous relation is equal to 0. The error \(e_3(N, C_2, C_3)\) verifies
\[
e_3(N, C_2, C_3) = \sum_{k=N-C_3}^{N-C_2-1} \sum_{n=k}^{N} e^{-\lambda t} \frac{(\lambda t)^n}{n!} C_{n}^{k} p^k (1 - p)^{n-k} Y(n, k)
\]

52
The general computational scheme is shown in Figure 2, before from (10) and (22), by changing
serve that the following inequalities hold:

\[ C_1 \leq C_2 \leq N. \]

The general computational scheme is shown in Figure 2, where we indicate the cells that are effectively filled. When \( C_3 \) does not exist, we can define \( C_3 = -1 \) and still use relation (24).

![Figure 2](image)

**Figure 2.** The contents of cell \((n, k)\) is the vector \(Y_s(n, k)\).

### 5 Complexity analysis and numerical examples

In this section, we compare the different techniques from the complexity point of view and we give some numerical results. The number of states of the system is denoted by \( M \).

#### 5.1 Storage complexity

The method proposed in [3] and Algorithm I basically need the storage of \( N \) vectors of dimension \( M \), that is, \( N M \) real numbers. Algorithm II necessitates the storage of \( C_2 \) vectors of dimension \( M \), that is \( C_2 M \) real numbers.

#### 5.2 Time complexity

In the three considered algorithms, the complexity of the computation of each cell is the same: it requires a matrix-vector product of dimension \( M \). Observe that a compact representation of the involved matrices must be used since, in general, they are sparse. The number of cells constructed in the algorithm proposed in [3] is

\[ \#_0 = (C + 1) \left( N + 1 - \frac{C}{2} \right); \]

in Algorithm I it is equal to

\[ \#_1 = (C_1 + 1) \left( N + 1 - \frac{C_1}{2} \right) \]

and in Algorithm II its value is

\[ \#_2 = (C_2 + 1) \left( N' + 1 - \frac{C_2}{2} \right) - \frac{(C_3 + 1)(C_3 + 2)}{2} \]

where \( N' \) is the truncation step computed from Relation 10 by changing \( \varepsilon/2 \) by \( \varepsilon/3 \). We have \( N' = N \) if \( C_3 = -1 \) and \( N' \geq N \) otherwise. It is clear that \( \#_1 \leq \#_0 \) since \( C_1 \leq C \), as we pointed out before. Concerning Algorithm II, the number of cells can be anywhere, from values greater than \( \#_0 \) to values less than \( \#_1 \). In practice, it is in general near \( \#_1 \), leading to better computational times than those of the algorithm in [3]. In the next subsection, we give some execution times to illustrate the behaviour of the different algorithms.

#### 5.3 Numerical examples

Let us consider a hardware system which consists of \( n \) identical components. As for most hardware devices, failure is a reflection of component failure. Let us assume that our system operates on a “\( k \) out of \( n \)” basis, that is,
the system is up when at least \( k \) components are up. Furthermore, it is assumed that the failure of any component would occur independently of the operation of the others. Repair times are stochastically independent of component lives and maintenance policy is unrestricted (i.e., the number of repairmen available is equal to the number of system components). These assumptions lead to a Markov model in which the number of system states results to be \( M = 2^n \). The failure and the repair rate of each component are taken to be 0.01 and 1 respectively. For numerical results, we will consider the case of \( k \) respectively equal to \( n, n - 1 \) and \( n - 2 \), which leads to a number of operational states respectively equal to \( 1, 1 + n, 1 + n + n(n - 1)/2 \).

We compute the probability of achieving an availability of at least \( p \), that is \( P(LAV(t) > p) \), for different values of the parameters \( p \) and \( t \): \( p = 0.95, 0.97, 0.99, 0.995; t = 100, 1000 \). The global error \( \varepsilon \) chosen for each algorithm is equal to 0.00001.

The following tables show the computation time (in seconds) of the two algorithms proposed in this paper as a function of the input parameters. The programs were run on a Sun 4/50. The number of components are successively \( n = 5 \) and \( n = 6 \), and the values of \( k \) are \( k = n \) and \( k = n - 1 \). The computational time of the algorithm described in [3] being independent of \( p \), it is given in each table header between parentheses.

### Table 1: Computation Time for \( n = 5, k = 5 \)

<table>
<thead>
<tr>
<th>( n = 5, k = 5 )</th>
<th>( t = 100 ) (44)</th>
<th>( t = 1000 ) (1563)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p = 0.950 )</td>
<td>16 18</td>
<td>950 1024</td>
</tr>
<tr>
<td>( p = 0.970 )</td>
<td>12 13</td>
<td>631 677</td>
</tr>
<tr>
<td>( p = 0.990 )</td>
<td>6 7</td>
<td>260 297</td>
</tr>
<tr>
<td>( p = 0.995 )</td>
<td>4 5</td>
<td>154 192</td>
</tr>
</tbody>
</table>

### Table 2: Computation Time for \( n = 5, k = 4 \)

<table>
<thead>
<tr>
<th>( n = 5, k = 4 )</th>
<th>( t = 100 ) (33)</th>
<th>( t = 1000 ) (889)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p = 0.950 )</td>
<td>15 18</td>
<td>785 1019</td>
</tr>
<tr>
<td>( p = 0.970 )</td>
<td>11 13</td>
<td>576 675</td>
</tr>
<tr>
<td>( p = 0.990 )</td>
<td>6 7</td>
<td>263 304</td>
</tr>
<tr>
<td>( p = 0.995 )</td>
<td>4 5</td>
<td>154 192</td>
</tr>
</tbody>
</table>

### Table 3: Computation Time for \( n = 6, k = 6 \)

<table>
<thead>
<tr>
<th>( n = 6, k = 6 )</th>
<th>( t = 100 ) (146)</th>
<th>( t = 1000 ) (5537)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p = 0.950 )</td>
<td>49 51</td>
<td>2995 3067</td>
</tr>
<tr>
<td>( p = 0.970 )</td>
<td>35 36</td>
<td>1950 2002</td>
</tr>
<tr>
<td>( p = 0.990 )</td>
<td>18 19</td>
<td>792 862</td>
</tr>
<tr>
<td>( p = 0.995 )</td>
<td>12 13</td>
<td>467 513</td>
</tr>
</tbody>
</table>

We see in these examples the better performance of Algorithm I with respect to the algorithm of [3]. Observe that in the case of \( n = 5, k = 4, t = 1000 \) and \( p = 0.95 \), Algorithm II is the worst one in computing time. Observe also the monotonicity of the execution times of Algorithms I and II when \( p \) increases.

We can remark that for a given model (\( n \) fixed), if \( t \) and \( p \) do not change, the execution time of Algorithm II does not depend on \( k \), as expected.

Concerning the storage complexity, let us give some examples. If \( n = 6, k = 6 \) and \( t = 1000 \), we have \( N = 6357 \) and the value \( p = 0.995 \) gives \( C_2 = 59 \). The number of states being equal to 64 in this case, Algorithm I needs to store \( 6357 \times 64 = 406848 \) real numbers and Algorithm II needs to store only \( 59 \times 64 = 3776 \) real numbers.

For \( n = 7, k = 5 \) and \( t = 1000 \), we obtain \( N = 7385 \) and the value \( p = 0.950 \) gives \( C_2 = 440 \). The number of states being equal to 128 in this case, Algorithm I needs to store \( 7385 \times 128 = 945280 \) real numbers and Algorithm II needs to store only \( 440 \times 128 = 56320 \) real numbers.

### References

