ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2011, Vol. 40, No. 2

COMPLEXITY OF EMBEDDED CHAIN ALGORITHM FOR COMPUTING STEADY STATE PROBABILITIES OF MARKOV CHAIN

Henrikas Pranevičius

Department of Business Informatics, Kaunas University of Technology
Studentų St, 56-301, LT – 51368 Kaunas, Lithuania
e-mail: henrikas.pranevicius@ktu.lt

Eimutis Valakevičius, Mindaugas Šnipas

Department of Mathematical Research in Systems, Kaunas University of Technology
Studentų St. 50-203, LT – 51368 Kaunas, Lithuania
e-mail: eimval@ktu.lt, minsnip@ktu.lt

crossref http://dx.doi.org/10.5755/j01.itc.40.2.425

Abstract. The paper presents the theoretical evaluation of the complexity of an algorithm, based on embedded Markov chains, for computing steady state probabilities. Experimental research with different infinitesimal generator matrices was performed to support theoretical evaluations. Results showed that modified algorithm can be more effective for sparse matrices. An example of a queuing system is presented to demonstrate the automatic creation of the model of the system based on the proposed modelling method.

Keywords: Steady state probabilities, complexity of the algorithm, numerical model, queuing system.

1. Introduction

The Markov chains are widely used for creating functioning models of various systems: computer networks, communication systems, reliability models etc. It is known that the analytical solution of complex real world systems is very difficult or even impossible. Using numerical methods permits to model a wider class of systems. The process of creating numerical models for systems described by the Markov chain consists of the following stages:

1. Defining a state space of the Markov chain.
2. Generating a linear system of Kolmogorov-Chapman equations, describing the performance of the system as a set of states and transitions among them.
3. Solving the linear system of Kolmogorov-Chapman equations, i.e., computing steady state probabilities.
4. Computing probabilistic characteristics of the system performance.

It is necessary to emphasize that automation is not equally applicable for all the stages. The first stage is determined in a heuristic way. The description of the system is the original that determines the state space of the system. The principal requirement in that stage is to introduce a necessary number of coordinates to describe behavior of the system.

Most difficult stages in the construction of a numerical model are the creation and solution of the linear system of Kolmogorov-Chapman equations. The number of equations is usually large (counted in thousands). Thus an automatic construction of linear systems and effective numerical solution methods are very important.

In order to demonstrate creation of numerical model we present an example of queuing system. There are a few different approaches to model queuing systems. In some cases, an exact solution of queuing systems can be found using the matrix geometric or the matrix analytical approaches [5, 8, 15]. Simulation (which in some aspects can be seen as opposite of analytical approach) is a powerful and universal scientific method to estimate the performance measures of various stochastic systems, including those based on queuing theory [1, 9]. One of the drawbacks of simulation is that sometimes it requires large amount of CPU time in order to achieve high accuracy. Validation and verification of a model is also a difficult problem.

An alternative is to use special numerical solvers, based on Markov chains to compute necessary

performance measures of queuing system. Such a numerical-analytical approach can be seen as a compromise among strictly analytical models and simulation, since it can achieve computational accuracy in relatively small amount of time.

In this paper we used the created software, which is based on Markov chains, to generate linear system of Kolmogorov-Chapman equations, which is stored in CPU memory as infinitesimal generator matrix of Markov chain. In book [14], the system that automates the construction of Markov models is presented. The embedded Markov chains are used for computing steady state probabilities of the Markov process. We used PLA formalism ([10, 13],etc.) to describe the performance of the system.

The former modeling and computation method was presented in [16] etc. In this paper we propose theoretical evaluation of the complexity of the algorithm. Theoretical results were compared with similar computation techniques and supported by experimental results.

In the second section of the paper, the method of embedded Markov chains for computing steady state probabilities is presented. In the third section, we derive theoretical evaluation of the complexity of the algorithm for worst-case scenario. In Section 4, the modified algorithm for computing steady state probabilities is presented. It is proved that its slight modifycation improves the performance of the algorithm when infinitesimal generator matrix is sparse. In Section 5, an empirical research of the algorithm is presented. Experimental results with different infinitesimal generator matrices confirm theoretical evaluations of the complexity of the algorithm. An example of creating a numerical model of multi-class, multi-server queuing system is presented in the last section. In the paper, we use PLA formalism for description of the system performance and generation of Kolmogorov-Chapman equations [12].

2. An algorithm for computing steady state probabilities

Let $P_N = \left(p_{ij}^N\right)_{N\times N}$ be the transitions probability matrix of the embedded Markov chain $\left\{x_m^{(N)}, m \ge 0\right\}$ specified by $X_N = \{i_1, i_2, \ldots, i_N\}$. Let X_N, X_{N-1},..., X_1 be the sequence of the state spaces obtained by the following state space reduction routine:

$X_k = \{i_1, i_2, \ldots, i_k\}$,

$X_k = X_{k+1} / i_{k+1}\}$, $k = \overline{1, N-1}$.

Construct the sequence of Markov chains $\left\{x_m^{(N)}, m \ge 0\right\}, \left\{x_m^{(N-1)}, m \ge 0\right\}, \cdots, \left\{x_m^{1}, m \ge 0\right\}$ by embedding the chain $\left\{x_m^{(k)}, m \ge 0\right\}$ into the chain $\left\{x_m^{(k+1)}, m \ge 0\right\}$, $k = \overline{1, N-1}$. It is proved that in this case the Markov chain $\left\{x_m^{(k)}, m \ge 0\right\}$ is embedded into the chain $\left\{x_m^{(N)}, m \ge 0\right\}$ too. Since the Markov chain $\left\{x_m^{(n)}, m \ge 0\right\}$ is specified by the P_n and X_n then we need to construct the sequence of transition matrixes P_N, P_{N-1},..., P_1. In [14], it is shown that the elements of matrixes P_N, P_{N-1},..., P_1 are calculated according the formulas

$$p_{ij}^{(k)} = p_{ij}^{(k+1)} + p_{i,k+1}^{(k+1)} \frac{1}{1 - p_{k+1,k+1}^{(k+1)}} \cdot p_{k+1,i}^{(k+1)}, \qquad (1)$$

$$i, j = \overline{1,k}, k = \overline{N-1,1}.$$

Steady state probabilities of discrete time Markov chain are calculated by the formulas

$$\overline{p}_1^1 = 1;$$

$$\overline{p}_i^{k+1} = \begin{cases} \overline{p}_i^k, & i = \overline{1,k}, \\ \dfrac{\sum_{j=1}^{k} \overline{p}_j^k p_{ji}^{(k+1)}}{1 - p_{ii}^{(k+1)}}, & i = k+1;\ k = \overline{1, N-1}; \end{cases} \qquad (2)$$

$$p_j = p_j^{(N)} = \frac{\overline{p}_j^{(N)}}{\sum_{j=1}^{N} \overline{p}_j^{(N)}}, j = \overline{1, N}. \qquad (3)$$

Formulas (1)-(3) are not convenient when dealing with practical problems. Usually, real systems are described by continuous time Markov chains. Therefore transitions are given as an infinitesimal generator matrix $Q = \left(\lambda_{ij}^N\right)_{N\times N}$ instead of the transition probability matrix $P_N = \left(p_{ij}^N\right)_{N\times N}$. To apply the algorithm to continuous time Markov chains, we substitute

$$p_{ij} = \frac{\lambda_{ij}}{S_i}, S_i = \sum_{j=1}^{N} \lambda_{ij}$$

into (1). After the substitution we obtain the formulas

$$\lambda_{ij}^{(k)} = \lambda_{ij}^{(k+1)} + \frac{\lambda_{i,k+1}^{(k+1)} \cdot \lambda_{k+1,j}^{(k+1)}}{\overline{S}_i},$$

$$\overline{S}_i = \sum_{j=1, j\ne i}^{N} \lambda_{\ ij}, k = \overline{N-1,1}; i, j = \overline{1,k}.$$

The non-normalized probabilities can be calculated recursively. At first we get

$$r_1^{(1)} = 1.$$

The other non-normalized probabilities are

$$r_i^{(k+1)} = \begin{cases} r_i^{(k)}, i = \overline{1,k}, \\ \dfrac{\sum_{j=1}^{N-1} r_j^{(k)} \lambda_{ji}^{(k+1)}}{\overline{S}_i}, i = k+1; k = \overline{1, N-1}. \end{cases}$$

Finally, steady state probabilities $q_i, i \in \{1,\ldots,N\}$ are

$$q_i = \frac{r_i^{(N)}}{\sum_{j=1}^{N} r_j^{(N)}}, i = \overline{1,N}.$$

3. Complexity analysis of the algorithm for calculating steady state probabilities

The worst-case scenario arises when all elements of the generator **Q** are non-zero. In this case, all elements of the matrix Q are stored in a two-dimension array. We will analyze two stages of the embedded chains algorithm separately.

The first stage of the algorithm is an embedding stage. The recalculated intensities are stored for the second step. The code is written in C++ and shown below:

```
void TForm1::EmbeddingStep(int n)
{
 1 if ( n==2)
 2   { M[0]=A[0][1];
 3     sumn[0]=A[1][0]; }
 4 else
 5  { float sum = 0;
 6    for (int i=0;i<(n-1);i++)
 7     M[((n-1)*(n-1)-(n-1))/2+i]=Q[i][n-1] ;
 8    for (int i=0;i<(n-1);i++)
 9      sum = sum + Q[n-1][i];
10    sumn[n-2]=sum;
11     for ( int i=0;i<(n-1);i++)
12     for (int j = 0;j<(n-1);j++)
13  A[i][j]+=(Q[i][m-1]*Q[m-1][j]/sumn[n-2]);
14        EmbeddingStep(n-1);}
}
```

In order to evaluate the runtime complexity of the algorithm, we made a simplifying assumption, that different instructions (i.e. saving new variables, addition, and multiplication) require equal amount of time, which we denote as 1. In steps 1-4, two variables are saved, so they require 2 units of time. While $n > 2$, steps 5-14 are executed recursively. Step 5 requires 1 unit of time. In steps 6-7 we have the loop, which is executed $(n - 2)$ times, and each execution requires 1 unit of time. Analogically, in steps 8-9, the loop requires the same amount of time as in steps 6-7, and in step 10 one unit of time is required. In steps 11-13 we have the outer and inner loops which are executed $(n - 1)$ times each. Since each execution requires 3 units of time, steps 11-13 require $3(n\text{-}1)^2$ units of time in total.

We denote by $T(n)$ the runtime complexity of the whole algorithm, and by $T_1(n)$ and $T_2(n)$ we denote the runtime complexity functions of the first and the second stages of the algorithm, respectively.

In order to find $T_1(n)$, we need to solve the inhomogeneous difference equation of the first order

$$T_1(n) = (n-1) + n + 3(n-1)^2 + T_1(n-1)$$

or

$$T_1(n) - T_1(n-1) = 3n^2 - 4n + 2,\ n \in N \qquad (4)$$

with the seed value

$$T_1(2) = 2. \qquad (5)$$

The solution of a difference equation can be found by the method of undetermined coefficients. The characteristic polynomial is $q - 1 = 0$, so $q = 1$ is a root having multiplicity 1. The general solution of homogenous part is

$$T_H(n) = c_1 \cdot 1^n \Rightarrow T_H(n) = c_1,\ c_1 \in R.$$

The particular solution has the form

$$\varphi_0(n) = n\left(p_0 + p_1 n + p_2 n^2\right), \qquad (6)$$
$$p_0, p_1, p_2 \in R.$$

To find coefficients p_0, p_1, p_2, we substitute (3) into (1). We obtain the equation:

$$p_0 - p_1 + p_2 + 2p_1 n - 3p_2 n + 3p_2 n^2 = 3n^2 - 4n + 2.$$

Comparing both sides, we have the system linear of equations

$$\begin{cases} p_0 - p_1 + p_2 = 2 \\ 2p_1 - 3p_2 = -4 \\ 3p_2 = 3 \end{cases}$$

which has the solution

$$p_0 = \frac{1}{2},\ p_1 = -\frac{1}{2},\ p_2 = 1.$$

The general solution is the sum of the homogenous part and particular solution:

$$T_1(n) = n^3 - \frac{n^2}{2} + \frac{n}{2} + c.$$

Constant c can be found from the seed value (2) $c = -7$.

The second stage of the algorithm is the calculation of steady state probabilities. The code in C++ is written below:

```
void TForm1::StacTik()
{
1 float r[n];
2 r[0]=1;
3 for ( int i=1; i<n; i++)
4     { r[i]=0;
5       for ( int j=0;j<i;j++)
6          r[i]+=r[j]*M[(i*i-i)/2+j];
7       r[i] = r[i]/sumn[i-1]; }
}
```

In step 2, the algorithm performs a save operation. In steps 3-6 an outer loop is executed $(n - 1)$ times and it has an inner loop inside. The inner loop is governed by the outer loop. Using the arithmetical progression,

the running time of the second stage can be evaluated as follows:

$$T_2(n) = \sum_{i=1}^{n-1}\sum_{j=0}^{i} 2 + \sum_{i=1}^{n-1} 2 =$$

$$\left(1+2+\ldots+(n-1)\right)+2(n-1) =$$

$$\frac{n^2-n}{2}+2n-2 =$$

$$\frac{1}{2}n^2+\frac{3}{2}n-2.$$

The complexity of the whole algorithm is equal to $T(n)=T_1(n)+T_2(n)=n^3+2n-7$.

Eventually, using big O notation, the complexity of the algorithm is

$$T(n)\sim O(n^3). \qquad (7)$$

4. Complexity analysis of the modified algorithm

In the worst-case scenario, the algorithm has the run time complexity of $O(n^3)$. Matrices which arise in practical problems (e.g. queuing networks) usually are very sparse. The algorithm of embedded chains can be modified for that case. Assuming that the matrix **Q** is not very large (i.e., it is possible to store the matrix in random access memory and no sparse matrix representation is necessary), steps 11-13 of the embedding stage of the algorithm can be written as follows:

```
11 for ( int i=0;i<(n-1);i++)
12 if ( Q[i][n-1]>0)
13  {for (int j = 0;j<(n-1);j++)
14  Q[i][j]+=Q[i][n-1]*Q[n-1][j]/sumn[n-2]);}
```

In other words, we add an additional 'if' statement, in order to avoid superfluous instructions in step 14 of outer loop.

The run time complexity of the modified algorithm can be estimated using the recurrence relation. We denote by ν_i – the number of nonzero entries in an *i*-th column of the matrix **Q**, and ν is

$$\nu = \max_{i\in\{1,\ldots,N\}} \nu_i .$$

The modified algorithm has one additional instruction in each outer loop, but the inner loop will be executed just *v* times (at most). Obviously, until step 11, the modified algorithm requires the same amount of time (2(*n*-1)) as written in Section 3. 'If' statement in step 12 will be executed (*n*-1) times and steps 13-14 will be executed no more than *v*(*n*-1) times (because of the modification), and each execution requires 3 units of time. The run time complexity can be expressed using the following recurrence relation:

$$T_1(n) = 2(n-1)+(n-1)+3\nu(n-1)+T_1(n-1)$$

or

$$T_1(n)-T_1(n-1) = (3\nu+3)n-3\nu-3,\ n,\nu\in N$$

with seed value $T(2)=2$.

The general solution of homogenous part is

$$T_H(n)=c,\ c\in R.$$

If $\nu << n$, or even, for example $\nu<10$, the particular solution has the form

$$\varphi_0(n)=n(p_0+p_1 n),$$

$$p_0, p_1 \in R.$$

We can find p_0, p_1, p_2 using the method of undetermined coefficients. Again, we obtain the equation:

$$p_0 - p_1 + 2p_1 n = (3+\nu)n - 3\nu - 3.$$

The values p_0, p_1 are the solution of the system of linear equations:

$$\begin{cases} p_0 - p_1 = -3\nu - 3 \\ 2p_1 = 3+\nu. \end{cases}$$

So

$$p_0 = \frac{-5\nu-6}{2},\ p_1 = \frac{3+\nu}{2}.$$

In conclusion, the run time complexity of the embedding step of the modified algorithm is equal to

$$T_1(n) = \frac{3+\nu}{2}n^2 - \left(\frac{5\nu+6}{2}\right)n + c.$$

If we assume that $\nu << n$, then using big O notation, the complexity of the first stage is

$$T_1(n)\sim O(n^2).$$

(An exact value of the constant *c* can be found from the seed value, but it does not change the asymptotic evaluation).

Since the second step was not modified, we obtain the estimation of the run time complexity

$$T(n)=T_1(n)+T_2(n)\sim \mathrm{O}(n^2).$$

Therefore, for very sparse matrices, the modified embedded chains algorithm has the run time complexity of $\mathrm{O}(n^2)$, which is better than classical Gaussian elimination. In addition, the run time complexity of the embedded chains algorithm does not depend on the structure of the infinitesimal generator matrix **Q**.

5. Experimental investigation of the algorithm

We have compared the run time of the embedded chains (both ordinary and modified) algorithm and the LU decomposition (in particularly, we have used the Doolittle decomposition of the transposed matrix **Q**), which is a variant of the Gaussian elimination.

In order to compare the algorithms, we have used four different infinitesimal generator matrices of Markov chains with 1000 000 elements each (i.e., matrices represent Markov chains with 1000 states). Since the matrices are relatively small, they were stored in two-dimension arrays.

For the first experiment we chose a Markov chain with all intensities among the states equal to 1 (i.e., matrix **Q1** has all non-diagonal elements equal to 1). Obviously, in this trivial case the steady states probabilities are all equal to 0.001.

For the second experiment, we generated (i.e., using random number generator) infinitesimal generator matrix **Q05**. Matrix **Q05** was generated in such a way that all intensities among the states – i.e., non-diagonal elements of the matrix – are equal to 1 (with probability 0.5) or 0 (also with probability 0.5). We have also checked if the rank of the matrix **Q** is equal to (n-1). It is obvious that randomly generated matrix **Q05** will have on average 50 % nonzero entries, and entries in matrix are scattered without any noticeable pattern or structure. In Figure 1 a portion of the matrix **Q05** is shown**.**

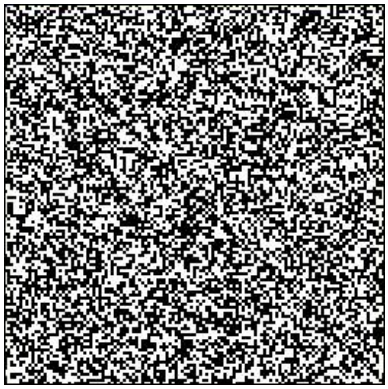

Figure 1. Structure of matrix Q05

In the third experiment, we have generated matrix **Q001** with the intensities among the states equal to 1 with probability 0.01, or 0 with probability 0.99. In this case, matrix **Q001** will have on average 1 % nonzero entries (so it is a sparse matrix), and the entries again are scattered without any noticeable structure, except for diagonal (see Figure 2).

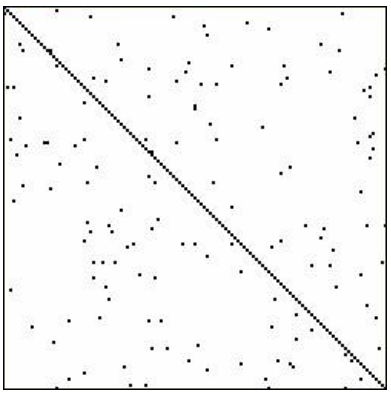

Figure 2. Structure of matrix Q001

For the last experiment, we use infinitesimal generator matrix **Q** of the Markov chain, which describes a queuing system with quality control [14].

We have generated an infinitesimal generator matrix **Q**, using the created software, which is based on Markov chains. It generates the set of possible states, the generator and the steady state probabilities. In this case, the matrix **Q** is very sparse, nonsymmetrical and highly structured (Figure. 3).

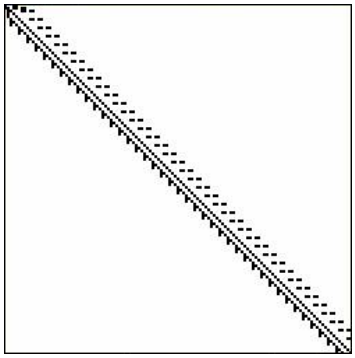

Figure 3. Structure of matrix Q

Experimental run time analysis of algorithms was performed using a PC with AMD Athlon 64 X2 dual core processor 4000+ 2.10 GHz, 896 MB of RAM physical address extensions.

The experimental results are shown are in Table 1.

Table 1. Comparison of modeling results (time in seconds)

Generator matrix	Embedded chains	Modified embedded chains	LU decomposition
Q1	7.032	6.984	5.101
Q05	6.984	6.921	5.093
Q001	6.969	3.688	5.100
Q	6.969	0.047	5.095

Embedded chains algorithm is slightly less effective than LU decomposition, if infinitesimal generator matrix is dense. However, if infinitesimal generator matrix is sparse (less than 1 percent non-zero entries) the modified algorithm outperforms LU decomposition, and it is much more effective if infinetisimal generator matrix is very sparse.

6. Example: queuing system with Markov modulated service time

Consider a queue whose service capacity varies over time. That is, the speed of the server is determined by an underlying stochastic process. In particular, we assume that the server speed changes according to a continuous time Markov chain that is independent of the arrival process and service requirements of the customer. Each customer brings a certain random amount of work, however, the rate at which this work is completed is time varying. We assume that the customers in the queue are served in a

First In First Out (FIFO) manner. Server speed can change at service completion.

The system is represented schematically in Figure 4.

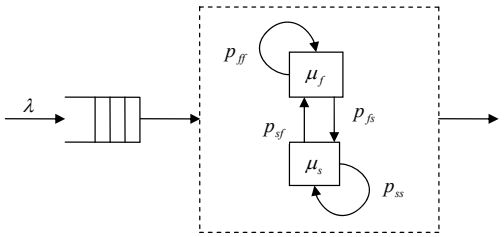

Figure 4. Scheme of the queuing system

Customers arrive into the queue according to a Poisson process with mean rate λ. Each arriving customer brings a certain amount of work distributed exponentially with mean which depends on some external Markov chain**.** In the following analysis we assume, that underlying Markov chain has two states, which we denote as slow S, and fast F. When the underlying Markov chain is in state S, server works at rate μ_S, and when Markov chain is in state F, server works at rate μ_f. We assume that $\mu_s < \mu_f$. If server is in state S and completes a service, it can remain in state S with probability p_{ss}, or it can transit in state F with probability $p_{sf} = 1 - p_{ss}$. Similarly, if server is in state *F*, it can remain in the same state with probability p_{ff}, or it can transit in the state S with probability $p_{fs} = 1 - p_{ff}$.

Since customer arrival and service times have exponential distribution, performance of the whole queuing system can be described as Markov chain with infinitesimal generator matrix. There are a few different approaches to create infinitesimal generator matrix automatically. One can use tools based on Petri nets ([4, 6, 7]), stochastic automata networks formalism, proposed by Plateau ([2, 3, 11]) or other methods. In any case, the user must precisely describe the performance of the system.

We used the PLA formalism for numerical models [12] to describe the performance of the queuing system.

Aggregate specification of the queuing system is:

1. The set of input signals $X = \emptyset$.
2. The set of output signals $Y = \emptyset$.
3. The set of external events $E' = \emptyset$.
4. The set of internal events:

$$E'' = \{e_1'', e_2'', e_3'', e_4'', e_5''\},$$

where e_1'' – a customer arrived at the system;

e_2'' – a customer was served at state *F* and the state of the server did not change;

e_3'' – a customer was served at state *F* and the state of the server changed into *S*;

e_4'' – a customer was served at state *S* and the state of the server did not change;

e_5'' - a customer was served at state *S* and the state of the server changed into *F*.

5. The transition rates between states of the system:

$e_1'' \to \lambda$, $e_2'' \to \mu_f \cdot p_{ff}$, $e_3'' \to \mu_f \cdot p_{fs}$,

$e_4'' \to \mu_s \cdot p_{ss}$, $e_5'' \to \mu_s \cdot p_{sf}$.

6. The discrete component of the state:

$v(t) = \{n_1(t), n_2(t)\}$,

where $n_1(t)$ – number of customers in the system;

$n_2(t)$ – indicates the state of the server (0, if state is *F* and 1 if it's S).

7. The continuous component of the state:

$z_v(t) = \{w(e_1'', t), w(e_2'', t), w(e_3'', t), w(e_4'', t), w(e_5'', t)\}$.

8. Initial state of the system:

$z(t) = \{0, 0, <\infty, \infty, \infty, \infty, \infty\}$.

9. Internal transition operators:

$H(e_1'')$: / a customer arrived at the system /

$$n_1(t+0) = \begin{cases} n_1(t)+1, \text{ if } n_1(t) < L, \\ n_1(t), \text{ otherwise}; \end{cases}$$

$$n_2(t+0) = n_2(t);$$

$$w(e_1'', t+0) = w(e_1'', t);$$

$$w(e_2'', t+0) = \begin{cases} \mu_f \cdot p_{ff}, \text{ if } n_2(t) = 0, \\ \infty, \text{ otherwise}; \end{cases}$$

$$w(e_3'', t+0) = \begin{cases} \mu_f \cdot p_{fs}, \text{ if } n_2(t) = 0, \\ \infty, \text{ otherwise}; \end{cases}$$

$$w(e_4'', t+0) = \begin{cases} \mu_s \cdot p_{ss}, \text{ if } n_2(t) = 1, \\ \infty, \text{ otherwise}; \end{cases}$$

$$w(e_5'', t+0) = \begin{cases} \mu_s \cdot p_{sf}, \text{ if } n_2(t) = 1, \\ \infty, \text{ otherwise}. \end{cases}$$

$H(e_2'')$ and $H(e_5'')$:

$$n_1(t+0) = n_1(t) - 1;$$

$$n_2(t+0) = 0;$$

$$w(e_1'', t+0) = w(e_1'', t);$$

$$w(e_2'', t+0) = \begin{cases} \mu_f \cdot p_{ff}, \text{ if } n_1(t) > 1, \\ \infty, \text{ otherwise}; \end{cases}$$

$$w\left(e_3'', t+0\right)=\begin{cases}\mu_f \cdot p_{fs}, \text{ if } n_1(t)>1,\\ \infty, \text{ otherwise};\end{cases}$$

$$w\left(e_4'', t+0\right)=\infty;$$

$$w\left(e_5'', t+0\right)=\infty.$$

$H\left(e_3''\right)$ and $H\left(e_4''\right)$:

$$n_1(t+0)=n_1(t)-1;$$

$$n_3(t+0)=1;$$

$$w\left(e_1'', t+0\right)=w\left(e_1'', t\right);$$

$$w\left(e_2'', t+0\right)=\infty;$$

$$w\left(e_3'', t+0\right)=\infty;$$

$$w\left(e_4'', t+0\right)=\begin{cases}\mu_s \cdot p_{ss}, \text{ if } n_1(t)>1,\\ \infty, \text{ otherwise};\end{cases}$$

$$w\left(e_5'', t+0\right)=\begin{cases}\mu_s \cdot p_{sf}, \text{ if } n_1(t)>1,\\ \infty, \text{ otherwise}.\end{cases}$$

The created software automatically builds infinitesimal generator matrix and estimates steady state probabilities $\pi(n_1, n_2)$. System performance characteristics can be calculated using steady state probabilities. For example, loss probability (arriving customer finds L customers in the queue) is estimated by the formula

$$P(L)=\sum_{n_1=L}\sum_{n_2}\pi(n_1,n_2),$$

when waiting space is limited by L.

We modeled queuing systems with Markov modulated service times and with limitation on the waiting space (*M/MMPP/*1*/N* by Kendall notation). In order to compare run time of the modified algorithm with different size infinitesimal generator matrices, the limit imposed on the waiting space L was varied.

Experimental results (in Table 2 and Figure 5) confirm theoretical complexity evaluation of the modified algorithm $O(n^2)$, since run time of modified algorithm increases about 4 times, when the number of system states increases 2 times, and trend line is consistent with second order function.

Table 2. Computation time of the modified algorithm

Number of states	CPU time, ms
600	16
800	31
1000	47
1200	62
1400	94
1600	141

Modeling results were compared with results obtained by analytical solution in [17]. Loss probability of the system with limited waiting space was calculated when limitation on waiting space is 7. In Table 3 we can see that numeric modeling results coincide with results obtained analytically. The results were obtained with the software created in C++ program language.

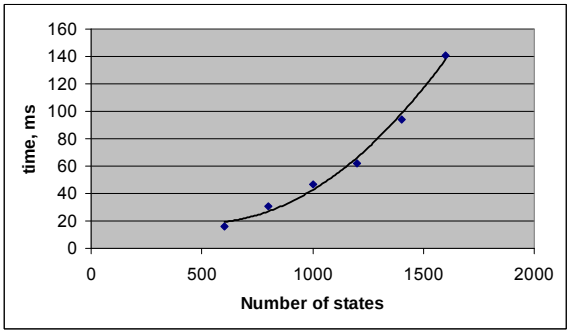

Figure 5. Computation time

Table 3. Comparison of numerical and analytical modeling results

	p_{sf}	p_{fs}	$P(L)_{an}$	$P(L)_{num}$
$\lambda=10$	0.6	0.3	3.11%	3.11%
$\mu_s=8$	0.3	0.6	11.05%	11.05%
$\mu_s=50$	0.9	0.15	0.78%	0.78%
$\lambda=10$	0.6	0.3	0.52%	0.52%
$\mu_s=12.5$	0.3	0.6	1.91%	1.91%
$\mu_s=50$	0.9	0.15	0.14%	0.14%

7. Conclusion

Embedded chains algorithm is designed specifically for Markov chains. It is a direct algorithm, so theoretically it will calculate the exact solution in a finite number of steps. Theoretical analysis showed that the complexity of the embedded chains algortihm is $O(n^3)$ in the worst-case scenario, so it cannot outperform Gaussian algortihm and its variants (LU decomposition) when infinetisimal generator matrix is dense. However, the modified algorithm becomes more effective when infinetisimal generator matrix is sparse. Theoretical and experimental results showed that the complexity of the modified algorithm is $O(n^2)$ when infinitesimal generator matrix is very sparse.

References

[1] **D. Bakšys, L. Sakalauskas**. Simulation and testing of FIFO clearing algortihms. *Information Technology and Control, Vol*. 39, *No.* 1, 2010, 25-31.

[2] **L. Brenner, P. Fernandes, B. Plateau, I. Sbeity**. PEPS2007 – Stochastic automata networks software tool. *Fourth International Conference on the Quantitive Evaluation of Systems, QEST 2007, Edinburgh,* 2007,163-164.

[3] **L. Brenner, P. Fernandes, J.M. Fourneau, B. Plateau**. Modelling GRID5000 point availability with SAN. *Electronic Notes in Theoretical Computer Science, Vol*. 232, 2009, 165-178.

[4] **K.S. Cheung.** Refinement of Petri-net-based system specification. *Information Technology and Control, Vol.* 35, *No*. 2, 2006, 137-143.

[5] **M. Harchol-Balter, T. Osogami, A. Scheller-Wolf, A. Wierman**. Multi-server queueing systems with multiple priority classes *Queueing System, Vol.* 51, 2005, 331-360.

[6] **S. Kounev.** Performance modelling and evaluation of distributed component-based systems using queueing Petri nets. *IEEE Transactions on Software Engineering, Vol.* 32, *No.* 7, 2006, 486-502.

[7] **S. Kounev, C. Dutz.** QPME: a performance modelling tool besed on queueing Petri nets. *ACM SIGMETRICS Performance Evaluation Review*, *Vol.* 36, *No.* 4, 2009, 46-51.

[8] **S.R. Mahabhashyam, N. Gautam.** On queues with Markov modulated service rates. *Queueing Systems, Vol.* 51, 2005, 89-113.

[9] **S. Minkevičius.** Simulation of the open message switching system. *Information Technology and Control, Vol.* 37, *No.* 1, 2008, 75-78.

[10] **Š. Packevičius, A. Kazla, H. Pranevičius.** Extension of PLA specification for dynamic system formalization. *Information Technology and Control, Vol.* 35, *No*. 3, 2006, 235-242.

[11] **B. Plateau, K. Atif**. Stochastic automata network of modeling parallel systems. *IEEE Transactions on Software Engineering, Vol.* 17, *No.* 10, 1991, 1093-1108.

[12] **H. Pranevičius, V. Germanavičius, G. Tumelis.** Automatic Creation of Numerical Model of Systems Specified by PLA Method. 19th *European Conference of Modelling and Simulation, ASMTA* 2005, 118-124.

[13] **H. Pranevičius, A. Paulauskaitė-Tarasevičienė, D. Makackas.** Application of abstract data type in dynamic PLA approach. *Information Technology and Control, Vol.* 38, *No*. 1, 2009, 7-13.

[14] **H. Pranevičius, E. Valakevičius.** Numerical models of systems specified by Markovian processes. *Technologija, Kaunas*, 1996.

[15] **A. Sleptchenko, A. van Harten, M. van der Heijden.** An exact solution for the state probabilities of the multi-class, multi-server queue with preemptive priorities. *Queueing Systems, Vol.* 50, 2005, 81-107.

[16] **E. Valakevičius, H. Pranevičius.** An algorithm for creating Markovian Models of Complex Systems, *Proceedings of the* 12th *World Multi-Conference on Systemics, Cybernetics and Informatics*, *WSMCI* 2008, *Orlando*, 2008. 258-262.

[17] **Y.P. Zhou, N. Gans.** A Single-Server Queue with Markov Modulated Service Times. *Working paper, the Wharton School, University of Pensilvania.*

Received November 2010.