A recursive algorithm for computing steady state probabilities

Eimutis VALAKEVICIUS
Department of Mathematical Research in Systems
Kaunas University of Technology
Kaunas, LT - 51368, Lithuania

and

Mindaugas SNIPAS
Department of Mathematical Research in Systems
Kaunas University of Technology
Kaunas, LT - 51368, Lithuania

ABSTRACT

One of the most important problems using Markov chains for modeling of the systems is computation of steady state probabilities. One of three major classes of numerical solution methods of this problem are direct methods, which usually are perceived as variants of Gaussian elimination. In this paper we introduce the embedded chains algorithm, which is a recursive direct algorithm, designed specifically to compute steady state probabilities of Markov chains. We have analytically evaluated run time complexity of embedded chains algorithm for the worst-case scenario. We have also introduced the modified embedded chains algorithm and showed that it is effective when infinitesimal generator matrix is very sparse. Empirical run time analysis of the algorithms was performed using infinitesimal generator matrixes with different number of nonzero elements.

Keywords: Markov chain; steady state probabilities; direct methods; recursion; analysis of algorithm;

1. INTRODUCTION

Markov chains can arise in many different areas – engineering, computer science, social sciences, economics, biology etc. One of the main problems using Markov chain models is computing steady state probabilities of continuous time Markov chain (CTMC). Basically, this means solving system of linear equations:

$$\pi \cdot Q = 0 \ , \ \ \forall i : \pi_i > 0$$

$$\sum_i \pi_i = 1$$

here: $\boldsymbol{\pi}$ – a vector of steady state probabilities; $\mathbf{Q}$ – an infinitesimal generator matrix.

We consider CTMC with countable space of states. Denote by n – the number of states of Markov chain. The rank of matrix $\mathbf{Q}$ is equal to (n-1).

Direct methods are one of three major classes of solution methods for this problem (the other two would be direct iterative methods and projection methods) [1,2,3]. Direct methods are usually faster (solving the problem of same amount of data), and their computation time can be evaluated accurately. Usually, direct methods are preferable, if only there are enough storage places in the computer memory available.

In this paper, we consider a recursive direct algorithm to compute steady state probabilities which is called an algorithm of embedded Markov chains. We think that in some cases it could be an alternative to direct methods, based on Gaussian elimination. Also, embedded chains algorithm is not universal solver of linear systems, and it is designed specifically for Markov chains.

The algorithm was successfully used to model various queuing systems.

2. EMBEDDED CHAINS ALGORITHM

The algorithm can be divided in two different stages.

The first is embedding stage. In this stage elements of the matrix $\mathbf{Q}$ are recalculated, and necessary variables are saved for the next step. Recalculation at the k-th step of embedding stage is the following

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

,

here $\overline{S}_{k+1}^{(k+1)} = \sum_{j=1, j\neq k+1}^{N} \lambda_{k+1,j}^{(k+1)}$.

The second stage – estimation of steady states probabilities $\boldsymbol{\pi}$:

$$\overline{r}_1^{(1)} = 1$$

$$\overline{r}_i^{(k+1)} = \begin{cases} \overline{r}_i^{(k)}, & i = \overline{1,k} \\ \dfrac{\sum_{j=1}^{k} \overline{r}_j^{(k)} \lambda_{ji}^{(k+1)}}{\overline{S}_i}, & i = k+1 \end{cases}, \quad k = \overline{1,N-1}$$

$$\pi_i = \frac{r_i^{(N)}}{\sum_{j=1}^{N} \left(r_j^{(N)}\right)}, \quad i = \overline{1,N}$$

3. ANALYSIS OF ALGORITHM

The worst-case scenario arises when all elements of matrix **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 embedded chains algorithm separately.

The first step of the algorithm – embedding stage, when intensities are recalculated, and necesary variables are stored for the second step. The code is written in C++ and shown bellow:

```
void TForm1::EmbeddingStep(int m)
{
 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          Idetosios(n-1);  }
}
```

In order to evaluate the run time complexity of the algorithm, we made a simplifying assumption, that different instructions (i.e. saving new variables, addition, multiplication) requires equal amount of time, which we denote as 1.

In steps 1-4, two variables are saved, so it requires 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 executes $(n-2)$ times, and each execution requires 1 unit of time. Analogically, in steps 8-9 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 both. Since each execution requires 3 units of time, steps 11-13 require $3(n-1)^2$ units of time.

We denote by $T(n)$ the run time complexity of algorithm, and by $T_1(n)$ and $T_2(n)$ are denoted the run time complexity functions of first and second steps 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

$$T_1(n) - T_2(n-1) = 3n^2 - 4n + 2 \quad , n \in N \qquad (1)$$

with the seed value $T_1(2) = 2$. (2)

The solution of 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

$$\varphi_0(n) = n\left(p_0 + p_1 n + p_2 n^2\right) \quad p_0, p_1, p_2 \in R \qquad (3)$$

To find coefficients p_0, p_1, p_2 we substitute (3) into the (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 linear system 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

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

and the general solution is the sum of the homogenous part and particular solution $T_1(n)$.

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

Constant c can be found from applying the seed value and equal to $c = -7$.

The second step of the algortihm is calculation of steady state probabilities. Code in is C++ is written bellow:

```
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 executes one saving, and in steps 3-6 where is one outer loop, which makes (n – 1) passes and 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 equal to

$$T(n) = T_1(n) + T_2(n) = n^3 + 2n - 7$$

Finally we obtain that the run time complexity of the algorithm of embedded chains is equal to

$$T(n) \sim O\left(n^3\right) \qquad (4)$$

4. MODIFIED ALGORITHM

In the worst-case scenario the algorithm has run time complexity of $O\left(n^3\right)$. Matrices which arise in practical problems (e.g. queuing networks) 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., were is enough an operative memory to store the whole matrix 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' sentence, in order to avoid superfluous instructions in the step 14 of outer loop. (As research showed, additional "if" sentence in the inner loop does not have positive effect).

Run time complexity of the modified algorithm can be estimated using the recurrent relation. We denote v – maximum number of nonzero elements in any column of **Q**. The modified algorithm has one additional instruction in each outer loop, but the inner loop will execute just v times. Obviously, until step 11, the modified algorithm requires the same amount of time (2*(n-1)) as written in section 3.

'If' sentence in step 12 will execute (n-1) times and steps 13-14 will be executed 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:

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

or

$$T(n) - T(n-1) = (3v+3)n - 3v - 3\ , \qquad (5)$$

$n, v \in N$,

with seed value $T(2) = 2$. (6)

The general solution of homogenous part is

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

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

$$\varphi_0(n) = n(p_0 + p_1 n) \quad p_0, p_1 \in R \qquad (7)$$

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

$$p_0 - p_1 + 2p_1 n = (3+v)n - 3v - 3$$

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

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

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 $\nu << n$, the complexity is $T_1(n) \sim O(n^2)$. (Again, the constant *c* can be found from the seed value (6), but it does not change the evaluation).

Since the second step was not modified, we can use the evaluation (2), and to get the run time estimation equal to

$$T(n) = T_1(n) + T_2(n) \sim \mathrm{O}(n^2) \qquad (8)$$

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, run time complexity of the embedded chains algorithm does not depend on the structure of the infinitesimal generator matrix **Q**.

5. EXPERIMENTAL ANALYSIS

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 matrixes of Markov chains with 1000 000 elements each (i.e., matrices represents 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 nondiagonal 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 so, that all intensities among the states – i.e., nondiagonal 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 elements, and elements in matrix are scattered without any noticeable pattern or structure. In Fig. 1 a portion of the matrix **Q05** is shown**.**

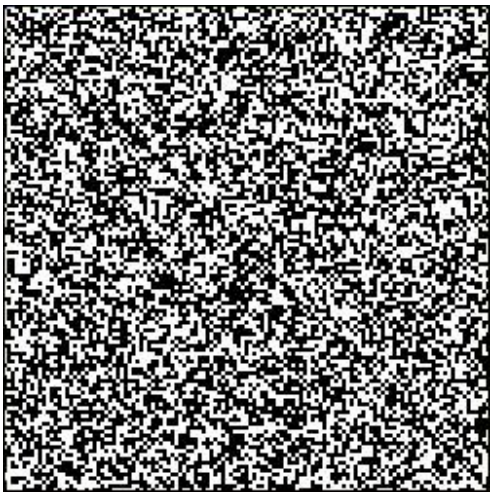

Fig. 1. Structure of matrix Q05

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 elements (so it is a sparse matrix), and the elements again are scattered without any noticeable structure, except for diagonal (see Fig. 2.).

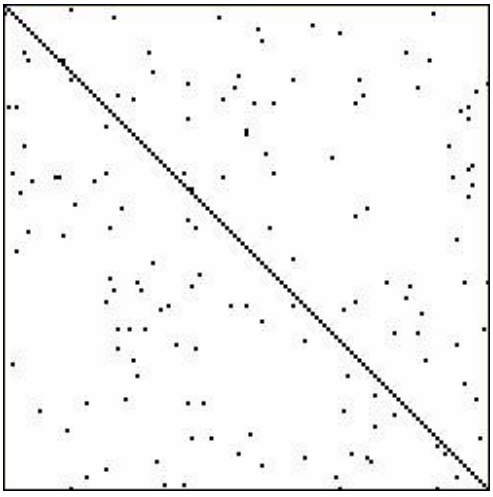

Fig. 2. Structure of matrix Q001

For the last experiment, we use infinitesimal generator

matrix **Q** of the Markov chain, which describes a queuing system with quality control.

In some cases, an exact solution of queuing systems can be found using matrix geometric or matrix analytical approaches [4,5,6]. An alternative is to use special numerical solvers (Petri nets tools, or tools based on Markov chains) to compute necessary performance measures of queuing systems [1,7]. We have generated infinitesimal generator matrix **Q**, using created software, which is based on Markov chains. It generates the set of possible states and computes both transition and steady state probabilities. The method is presented in [8,9].

In this case, the matrix **Q** is very sparse, nonsymmetrical and highly structured (Fig. 3).

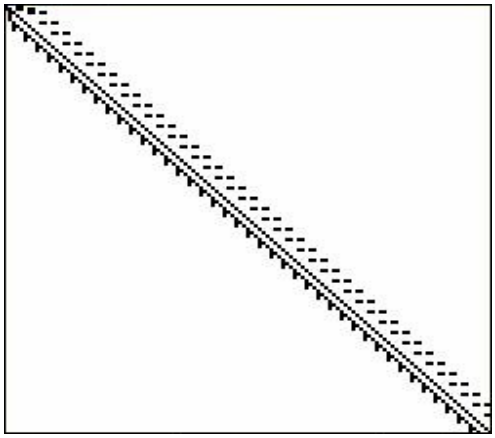

Fig. 3. Structure of matrix Q

6. RESULTS

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. Comparing of modeling results

Generator matrix	Embedded chains	Modified embedded chains	LU decomposition
Q1	7 s 032 ms	6 s 984 ms	5 s 101 ms
Q05	6 s 984 ms	6 s 921 ms	5 s 093 ms
Q001	6 s 969 ms	3 s 688 ms	5 s 100 ms
Q	6 s 969 ms	0 s 047 ms	5 s 095 ms

7. CONCLUSION

The results show that the embedded chains algorithm is less effective than LU decomposition (also than classical Gaussian elimination, since its run time complexity is practically the same as LU decomposition) in the worst-case scenario, but modified embedded chains algorithm becomes more effective, when sparsity of the generator matrix increases. We expect that an implementation of the algorithm for sparse matrices representation would enable to model large Markov chains. The further investigation is needed.

REFERENCES

[1] W. J. Stewart, Numerical Solution of Markov Chains, Princeton, New Jersey, 1994.

[2] P. Buchholz , Structured analysis techniques for large Markov chains, Ins, Institute ACM International Conference Proceeding Series,Vol. 201 (2006)

[3] W.S. Barge, W.J. Stewart Autonomous Solution Methods for Large Markov Chains, http://www4.ncsu.edu/~billy/Publications/PS_files /Shep.pdf

[4] A. Sleptchenko, A. Harten, M. Heijden, An exact solution for the state probabilities of the multi-class, multi-server queue with preemptive priorities, Queuing systems, Queueing Systems: Theory and Applications vol. 50, (2005), 81-107.

[5] M. Harchol-Balter, T. Osogami, A. Scheller-Wolf, A. Wierman, Multi-server queueing systems with multiple priority classes *Queueing Systems: Theory and Applications* vol. 51, (2005), 331-360

[6] E. Valakevicius, "Numerical Modeling of Queuing Systems Based on Phase Type Distribution", WMSCI 2007, Orlando, Florida, USA, pp.103-107.

[7] G. Bolch, S. Greiner, H. de Meer, K. S. Trivedi, Queuing Networks and Markov Chains. Modelling and Performance evaluation with Computer Science applications, New York, Wiley-Interscience, 2006

[8] E. Valakevicius, H. Pranevicius, An algorithm for creating Markovian Models of Complex Systems, In Proceedings of the 12th World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando, USA, (2008), 258-262

[9] H. Pranevičius, E. Valakevičius. Numerical Models of Systems Specified by Markovian processes, Kaunas, Technologija, 1996.