Štefan Peško *

SQL ALGORITHM FOR SOLVING MARKOV MODELS BY GRAPH METHOD

A simple graph algorithm for finding stabilized probabilities of the finite Markov models implemented in SQL is presented. The algorithm generates systematically all oriented spanning trees of a transition graph. The method is demonstrated on the computation of probabilities in the $MMPP_2/M/1/K$ queue.

Keywords: *Markov models, queue, graph algorithm, SQL algorithm*

1. Introduction

The original graph study of steady-state distribution for stabile Markov process with finite state sets is presented by Markl [1]. Stabilized probabilities π of the stable Markov process associated with a weighted digraph (called transition graph) $G = (S, E, c)$ are given by formulas

$$\pi i = \frac{B_i}{\sum_{j \in S} B_j} \quad i \in S, \tag{1}$$

where B_i, $i \in S$ is the sum of weights of all the spanning trees of G that have their roots in vertex i.

For processes with many states and many possible transitions between them is not easy to find all spanning trees. We apply a simple algorithm that systematically generates all spanning trees with given roots in SQL.

2. Basic definitions

We start with giving the some definitions from graph theory that will be used in this paper. A (simple) *digraph* $G = (V, E)$ consists of a finite set V of vertices and set E of edges – ordered pairs of distinct vertices; that is, each edge (u, v) is directed from *tail* u to *head* v. A (directed) $u - v$ *path* from vertex u to vertex v is such a sequence $u = v_0, (v_0, v_1), v_1, (v_1, v_2), v_2, ..., (v_{k-1}, v_k), v_k = v$ that $v_i \in V$ for $i = 0. ..., k$ and $v_i \neq v_j$ for $0 < i < j < k$, and that $(v_{i-1}, v_i) \in E$ for $i = 1, ..., k$. If $u = v$ then $u - v$ path is called a *cycle*. A digraph in which each pair of vertices lies on a common cycle is called *strongly connected*. A digraph in which for each pair $\{u, v\} \in V$ exists $u - v$ path or $v - u$ path is called *connected*. The *component* of a digraph is maximal connected subgraph of a digraph.

A digraph that has no cycle is called a *directed acyclic graph* (DAG). A *rooted tree* is a DAG in which one vertex, the *root*, is distinguished and in which all edges are implicitly directed to the root. (Note that in the standard definition [7] of the rooted tree the orientation of the edges is away from the root.) The *weight* of the (edge) weighted rooted tree $T = (V, E_T, w)$ we mean number

$$w(T) = \prod_{h \in E_T} w(h) \tag{2}$$

A *spanning rooted tree* (T, r) with root r, shortly *spanning r-tree*, of a strongly connected digraph G is a rooted tree T that is a subgraph of G and that contains every vertex of G. The *weight* of the a spanning r-tree (T, r) of a (edge) weighted digraph $G = (V, E, w)$ is weight $w(T)$ of the rooted tree $T = (V, E_T)$ given by (2).

3. SQL algorithm

The strongly connected digraph $G = (V, E)$ is given. We may assume without loss of generality that $V = \{1, 2, ..., n\}$ and root $r = 1$. The cases where $r \neq 1$ can be transformed to the case $r = 1$ by renumbering of vertices V. Let the set E be represented by a table *Edge* with two columns:

- *Edge.u* is head of a directed edge (u, v)
- *Edge.v* is tail of a directed edge (u, v)

and a table *Solution* with two columns of *array*[1, ..., *n*]:

- *Solution.tree* is the subtree – subgraf of spanning 1-tree
- *Solution.comp* is the components of *tree.Solution*

In Figure 1 we have the example of the spanning 1-tree $T = (V, H_T)$ where $V = \{1, 2, 3, 4, 5\}$ and $H_T = \{(2, 4), (3, 5), (4, 1), (5, 4)\}$ are represented by array *tree* = [0, 4, 5, 1 4]. In Figure 2

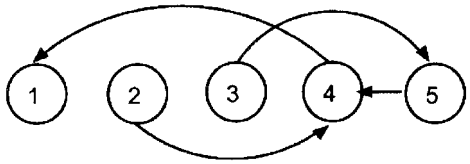

Fig. 1 The spanning 1-tree as array [0, 4, 5, 1, 4]

* **Štefan Peško**
Department of Mathematical Methods, Faculty of Management Science and Informatics, University of Žilina, Slovakia,
E-mail: pesko@frcatel.fri.utc.sk

we have the example of the subtree of the spanning 1-tree which is represented by array *tree* = [0, 4, 0, 1, 4] and has two components T_1 = {1, 2, 4, 5}, {(2, 4), (4, 1), (5, 4)}) and T_2 = ({3}, ∅) which are represented by array *comp* = [1, 1, 3, 1, 1].

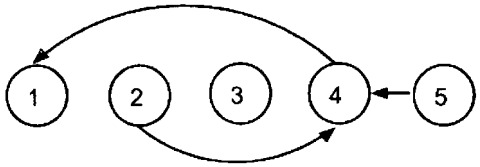

Fig. 2 The subtree of the 1-tree with two components

We can now describe the algorithm which generates all spanning 1-trees:

```
procedure SQL1Trees (Edge )
  for i = 1 to n do (* Initialization *)
    Solution.tree [i] = 0 , Solution.comp [i] = i
  for k = 1 to n − 1 do
    SELECT DISTINCT (* Subgraphs of spanning 1-trees *)
      SetValue (Solution.tree, Edge.u, Edge.v) AS tree
      SetValue (Solution.comp, Edge.u, 1) AS comp
      FROM Solution, Edge
      WHERE
        Solution.comp [Edge.u] ≠ 1 AND Solution.comp
        [Edge.v] = 1
      GROUP BY tree, comp
      TO FILE Solution
```

Edges of the transition graph *G* Tab. 1

u	1	1	2	3	3	3	4	4	4	5	5	6	6
v	2	3	1	1	4	5	2	3	6	3	6	4	5

Before we start the main SQL-algorithm, we calculate the values of *null subtree* (V,∅) with *n* components. One new edge (*u*, *v*) appends in the *k*-step, if it is possible else subtree is deleted. So the subtrees (V, H_k) where $|H_k| = k$ are generated step by step. After the last step we have all spanning 1-trees.

A new subtree and its components are stored in an array *tree* and *comp* which are updated by the function:

```
function x = SetValue(x, i, j)
  x[i] = j
```

Note that after the last step are *comp* = [1, 1, 1, ..., 1] because the spanning 1-trees are connected digraphs.

4. Example of $MMPP_2/M/1/K$ queue

We consider a queueing system studied by Peško [2] with *two-state Markov modulated Poisson process* $MMPP_2$, a single exponential distributed server and a finite waiting room. The problem of switch design and admission control in a high speed network is modeled in [6] as $MMPP_2/GI/1/\infty$ queue. The transition graph for the $MMPP_2/M/1/K$ queue is a weighted digraph

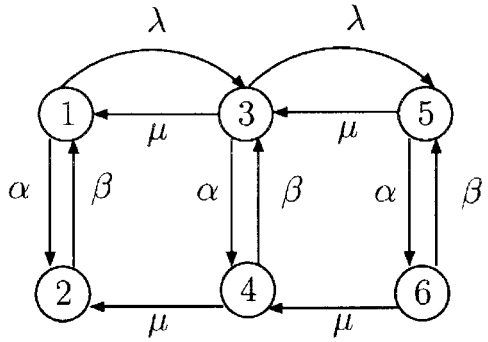

Fig. 3 Transition graph G of $MMPP_2/M/1/3$ queue

$G = (S, E, c)$. In Figure 3 we have the small example of the $MMPP_2/M/1/3$ queue with maximal 2 customers in waiting room where α and β are ON and OFF rate of source, λ is the arrival rate from ON source and μ the service rate.

We have a table *Edge* in the Table 1. The costs of the edges of the transition graph are omitted. Note that the table *Edge* is shown horizontally instead of a usual vertical layout. The initialization table *Solution* is in the first row of the Table 2. After k^{th} step (k = = 1, 2, 3, 4) we have a new *Solution* in Table 2 with added column of step marked k. All spanning 1-trees are generated in the last *Solution* table 3 and described in figure 4. And so B_1 – the sum of weights of all the spanning 1-trees is

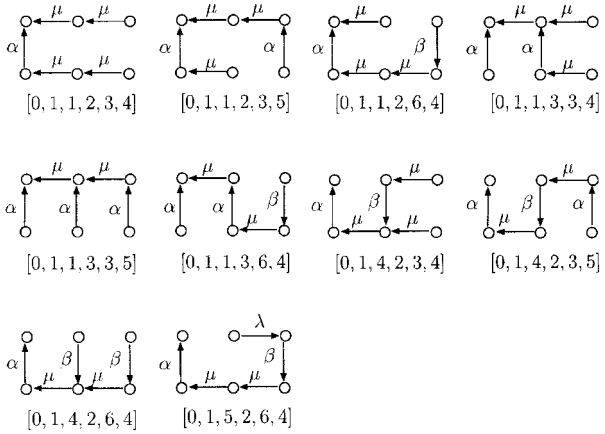

Fig. 4 All spanning 1-trees of G

$$B_1 = \alpha\mu^4 + 2\alpha^2\mu^3 + \alpha^3\mu^2 + 2\alpha\beta\mu^3 +$$

$$+ 2\alpha^2\beta\mu^2 + \alpha\beta^2\mu^2 + \lambda\alpha\beta\mu^2.$$

The sum of weights of all the spanning r-trees B_r for r = 2, ..., 6 can be computed by renumbering set of vertices S.

Solution after k^{th} step — Tab. 2

tree	comp	k
[0, 0, 0, 0, 0, 0]	[1, 2, 3, 4, 5, 6]	–
[0, 1, 0, 0, 0, 0]	[1, 1, 3, 4, 5, 6]	1
[0, 0, 1, 0, 0, 0]	[1, 2, 1, 4, 5, 6]	1
[0, 1, 1, 0, 0, 0]	[1, 1, 1, 4, 5, 6]	2
[0, 1, 0, 2, 0, 0]	[1, 1, 3, 1, 5, 6]	2
[0, 0, 1, 3, 0, 0]	[1, 2, 1, 1, 5, 6]	2
[0, 0, 1, 0, 3, 0]	[1, 2, 1, 4, 1, 6]	2
[0, 1, 1, 2, 0, 0]	[1, 1, 1, 1, 5, 6]	3
[0, 1, 1, 3, 0, 0]	[1, 1, 1, 1, 5, 6]	3
[0, 1, 1, 0, 3, 0]	[1, 1, 1, 4, 1, 6]	3
[0, 1, 4, 2, 0, 0]	[1, 1, 1, 1, 5, 6]	3
[0, 1, 0, 2, 0, 4]	[1, 1, 3, 1, 5, 1]	3
[0, 0, 1, 3, 3, 0]	[1, 2, 1, 1, 1, 6]	3
[0, 0, 1, 3, 0, 4]	[1, 2, 1, 1, 5, 1]	3
[0, 0, 1, 0, 3, 5]	[1, 2, 1, 4, 1, 1]	3
[0, 1, 1, 2, 3, 0]	[1, 1, 1, 1, 1, 6]	4
[0, 1, 1, 2, 0, 4]	[1, 1, 1, 1, 5, 1]	4
[0, 1, 1, 3, 3, 0]	[1, 1, 1, 1, 1, 6]	4
[0, 1, 1, 3, 0, 4]	[1, 1, 1, 1, 5, 1]	4
[0, 1, 1, 0, 3, 5]	[1, 1, 1, 4, 1, 1]	4
[0, 1, 4, 2, 3, 0]	[1, 1, 1, 1, 1, 6]	4
[0, 1, 4, 2, 0, 4]	[1, 1, 1, 1, 5, 1]	4
[0, 1, 0, 2, 6, 4]	[1, 1, 3, 1, 1, 1]	4
[0, 0, 1, 3, 3, 4]	[1, 2, 1, 1, 1, 1]	4
[0, 0, 1, 3, 3, 5]	[1, 2, 1, 1, 1, 1]	4
[0, 0, 1, 3, 6, 4]	[1, 2, 1, 1, 1, 1]	4

Solution after last 5^{th} step — Tab. 3

tree	comp
[0, 1, 1, 2, 3, 4]	[1, 1, 1, 1, 1, 1]
[0, 1, 1, 2, 3, 5]	[1, 1, 1, 1, 1, 1]
[0, 1, 1, 2, 6, 4]	[1, 1, 1, 1, 1, 1]
[0, 1, 1, 3, 3, 4]	[1, 1, 1, 1, 1, 1]
[0, 1, 1, 3, 3, 5]	[1, 1, 1, 1, 1, 1]
[0, 1, 1, 3, 6, 4]	[1, 1, 1, 1, 1, 1]
[0, 1, 4, 2, 3, 4]	[1, 1, 1, 1, 1, 1]
[0, 1, 4, 2, 3, 5]	[1, 1, 1, 1, 1, 1]
[0, 1, 4, 2, 6, 4]	[1, 1, 1, 1, 1, 1]
[0, 1, 5, 2, 6, 4]	[1, 1, 1, 1, 1, 1]

The research of the autor is supported by the Slovak Scientific Grant Agency under grant No. 1/0490/03.

References:

[1] MARKL, J.: *A Graph Method for Markov Models Solving,* Acta Mathematica et Informatica Universitatis Ostraviensis, Vol. 1, 1993, pp. 75-82

[2] PEŠKO, Š.: *Erlang ON/OFF Moduled Queueing Systems,* Proceedings of the 20'th International Conference, Mathematical Methods in Economics 2002, Ostrava, ISBN 80-248-0153-1, 2002, pp. 209-213

[3] ROSS, S. M.: *Stochastic Processes,* John Wiley & Sons, Inc., 1983, ISBN 0-471-09942-2

[4] KAUKIČ, M.: *Open Source Software in Mathematical Education*, 1. International conference, Aplimat, Bratislava 2002, pp. 233-238

[5] REBO, J., BARTL, O.: *Condition of Stability for Tandem Queues with Blocking and Exponential and Erlangian Service Time Distribution*, Studies of the Faculty of Management Science and Informatics, Vol. 8, Žilina, 1999, pp. 67-74

[6] JEAN-MARIA, A., LIU, Z., NAIN, P., TOWSLEY, D.: *Computational Aspects of the Workload Distribution in the MMPP/GI/1 Queue,* IEEE Journal on Selected Areas in Communications, Vol.16, No.5, 1998, pp.640-652

[7] PALÚCH, S.: *Graph Theory* (*Teória grafov),* Žilinská univerzita, EDIS, 2001, ISBN 80-7100-874-5

[8] MATIAŠKO, K.: *Database Systems (Databázové systémy)*, Žilinská univerzita, EDIS, 2002, ISBN 80-7100-968-7.