RAIRO Operations Research
RAIRO Oper. Res. **35** (2001) 229-250

FAST SIMULATION FOR ROAD TRAFFIC NETWORK

Roberta Jungblut-Hessel[1], Brigitte Plateau[2],
William J. Stewart[3] and Bernard Ycart[4]

Abstract. In this paper we present a method to perform fast simulation of large Markovian systems. This method is based on the use of three concepts: Markov chain uniformization, event-driven dynamics, and modularity. An application of urban traffic simulation is presented to illustrate the performance of our approach.

Résumé. Dans cet article, nous présentons une méthode pour réaliser des simulations rapides de grands systèmes Markoviens. Cette méthode est basée sur l'utilisation de trois concepts : l'uniformisation de chaîne de Markov, une dynamique liée aux événements et la modularité. Une application de trafic urbain illustre les performances de notre approche.

Keywords: Markov chains, stochastic automata networks, simulation, stochastic modeling.

Received May, 1999.

[1] ID-IMAG, 55 avenue Jean Kuntzmann, 38330 Montbonnot, France. Research supported by the (CNRS – INRIA – INPG – UJF) joint project *Apache*, CAPES-COFECUB Agreement (Project 140/93), Brazil and EC Transport RTD Program (Contract No. RO-97-SC-1005) joint project *HIPERTRANS*.

[2] ID-IMAG, 55 avenue Jean Kuntzmann, 38330 Montbonnot, France. Research supported by the (CNRS – INRIA – INPG – UJF) joint project *Apache* and EC Transport RTD Program (Contract No. RO-97-SC-1005) joint project *HIPERTRANS*.

[3] North Carolina State University, Raleigh, NC 27695-8206, U.S.A. Research supported in part by NSF (DDM-8906248 and CCR-9413309).

[4] UFR Math-Info, 45 rue des Saints-Pères, 75270 Paris Cedex 06, France. Research supported by EC Transport RTD Program (Contract No. RO-97-SC-1005) joint project *HIPERTRANS*.

1. Introduction

The problems of modelling very large systems are well documented. They principally include state space generation and storage, and the complexity of the analysis to be undertaken. In the context of Markov models, the theory provides elegant theorems for numerically computing both steady state and transient solutions. These theorems have been exploited in the context of Stochastic Automata Networks (SAN) [11] where the transition matrices can be expressed in a very dense manner and ad-hoc numerical algorithms can benefit from this structure [3, 4]. Such results have also been applied in the context of Petri nets [2, 6].

Such numerical solutions allow models with a few millions states to be solved, which means a few tens of automata at most. Parallel execution, if available, only helps to gain a linear factor. If one wants to model systems with a few thousands of automata, simulation is the only possibility. In this paper we show that we can simulate SAN systems with of the order of 10^4 automata by using two arguments: the system is a Markov chain and is described by a set of interacting components that operate more or less independently, requiring only infrequent interactions such as synchronizing their actions, or operating at different rates depending on the state of parts of the overall system.

We show that this approach is efficient compared to the standard event list algorithm. Indeed, the event list approach has a number of steps that can be very costly, namely, the time step implementation which often requires the computation of a *log* function and insertion into the event list for very large systems.

The paper is organized as follows. Section 2 is a review of basic results and methods for simulating Markov chains using a uniformization transformation. Section 3 presents the adaptation of these basic results in the context of SANs. Section 4 presents an application of urban road traffic.

2. Uniformized Markov chain and simulation

2.1. Basic simulation

Consider a continuous time Markov chain $X = (X_t)\,,\ t \in \mathbb{R}$, with finite[5] state space S and transition matrix (or generator) Q. A non-diagonal element x, y of Q, denoted by λ_{xy}, is the (non-negative) transition rate from x to y. The diagonal element x, x is denoted by $-\lambda_{xx}$, with $\lambda_{xx} = \sum_{y \neq x} \lambda_{xy}$. The basic simulation algorithm for this chain is depicted below:

[5]These results are also valid for countably infinite state spaces, provided that the various quantities exist.

Markov-1

- $t := 0$; Initial state is x;
- Repeat
  - choose the next state equal to y with probability $\frac{\lambda_{xy}}{\lambda_{xx}}$; $x := y$;
  - $t := t - (log(random))/\lambda_{xx}$;
- until the end of the simulation.

In this simulation, y is always different from x. *random* is a call to a function that returns an independent and uniformly distributed real number in $[0, 1]$. Let us define the uniformized Markov chain $Y = (Y_n), n \in \mathcal{N}$ of this initial chain. It is a discrete time Markov chain defined by its transition matrix P (I denotes the identity matrix with appropriate dimension):

$$P = I + \frac{1}{\lambda}Q \quad \text{with} \quad \lambda = \max_{x \in S} \lambda_{xx} \ .$$

It is well known that X_t and Y_n have the same stationary solution (if it exists) and that the transient solution of X_t can be computed from the transient solution of Y_n by a randomization (or uniformization) of the transition instant of the chain Y_n by a Poisson process of rate λ [5]. The basic simulation algorithm for the chain Y_n is:

Markov-2

- $n := 0$; Initial state is x;
- Repeat
  - choose the next state equal to $y \neq x$ with probability $\frac{\lambda_{xy}}{\lambda}$ or $y = x$ with probability $1 - \frac{\lambda_{xx}}{\lambda}$; $x := y$;
  - $n := n + 1$;
- until the end of the simulation.

If each time step of this simulation is scaled to an independent exponentially distributed random variable of rate λ, denote $Y_\lambda = (Y_{\lambda,t})$ the process simulated by **Markov-2**. Then, the two algorithms **Markov-1** and **Markov-2** simulate stochastic processes with identical distributions. If the simulation **Markov-2** runs for n steps (n large), the law of large numbers allows us to estimate the correponding time on the continuous time scale by $t = \frac{n}{\lambda}$, with a relative precision of order $1/\sqrt{n}$. For this reason, we call λ the *clock rate* of the chain Y_λ.

Let us now compare the efficiency of the two simulations: an iteration of the second algorithm is cheaper because the time increment avoids a *log* call. On the other hand, the second simulation has more iterations (on average) than the first one over the same period of simulated time because each time increment is smaller ($\lambda = \max_{x \in S} \lambda_{xx}$): In **Markov-1** the mean time increment is $\frac{1}{\lambda_{xx}}$ in state x, and in **Markov-2** it is always $\frac{1}{\lambda}$. If **Markov-2** has more iterations for the same simulated time, observe that it has *dummy transitions* to the same state, with probability $1 - \frac{\lambda_{xx}}{\lambda}$, which never occurs in **Markov-1**.

If we consider the M/M/1 queue as an example, almost all the diagonal entries of the transition matrix are equal to the same value, except for the state with no customers. So the probability of dummy transitions is zero, except for the zero state. If this zero state is rarely reached, dummy transitions are seldom. On the other hand, if the diagonal entries of Q are very different, the second simulation might compare poorly to the first: a poor case is one very large entry on the diagonal compared to all the others.

Note that any value h greater than λ leads to a different Markov Y_h chain with transition matrix $P_h = I + \frac{Q}{h}$, and a uniformized process $Y_{h,t}$ with a larger clock rate h, but obviously with an identical stationary solution. The transient solution of X can also be computed from the transient solution of Y_h by a uniformization process of rate h. This can easily be seen: let $\pi(t)$ be the transient probability vector of the continuous time Markov chain at time t:

$$\pi(t) = \pi(0)\exp(Qt)\ .$$

Thus

$$\pi(t)\exp(ht) = \pi(0)\exp(Qt)\exp(ht) = \pi(0)\exp\left(ht\left(\frac{Q}{h} + I\right)\right) = \pi(0)\exp(htP_h)$$

and because $h \geq \lambda$, P_h is stochastic and its exponential can be expanded

$$\pi(t) = \pi(0)\exp(-ht)\exp(htP_h) = \pi(0)\sum_{k=1}^{\infty} P_h^k \frac{(ht)^k}{k!}\exp(-ht).$$

This concludes the proof. In what follows, we shall call Y_h the h-uniformized Markov chain associated with X (abbreviated to h-HM).

From the point of view of algorithm speed, the optimal value for h is obviously $\lambda = \max_{x\in S}\ \lambda_{xx}$: the simulation (of type **Markov-2**) of Y_h is less efficient in number of iterations than the simulation of Y, as it leads to more dummy transitions: the rate of dummy transitions in state x is $1 - \frac{\lambda_{xx}}{h} = \frac{h-\lambda_{xx}}{h}$. Nevertheless, as will be explained later, this option may ease the computation of a clock.

Basically in this paper, we use h-uniformized Markov chains to estimate by simulation the steady state probabilities of ergodic systems and not transient probabilities. On the other hand, the traditional use of a uniformized Markov chain is the numerical computation of transient probabilities [5] and that is why these short explanations are given here.

2.2. Simulation with multiple components

Assume now that the chain X is a vector of N components $\left(X_t^{(1)}, \ldots, X_t^{(N)}\right)$. To simplify this introduction, assume first that the components are independent Markov chains on the state space $S^{(i)}$, each with the optimal clock rate $\lambda^{(i)}$ and

transition matrix $Q^{(i)}$. The clock rate of X is then $\lambda = \sum_{i=1}^{N} \lambda^{(i)}$, as[6]

$$Q = \bigoplus_{i=1}^{N} Q^{(i)} .$$

The traditional simulation of X, using the event list method is as follows:

Markov-3

- $t := 0$; Initial state is $x^{(1)}, \ldots, x^{(N)}$;
- Initialize the first possible transitions and transition dates in the sorted event list for each component. An event has the format (component identity, next state, transition instant);
- Repeat
  - pop the first event $(i, y^{(i)}, t^{(i)})$ from the event list;
  - $t := t^{(i)}$; $x^{(i)} := y^{(i)}$;
  - choose the next state of component i equal to $z^{(i)}$ with probability $\frac{\lambda^{(i)}_{y^{(i)} z^{(i)}}}{\lambda^{(i)}_{y^{(i)} y^{(i)}}}$;
  - compute the transition instant $t^{(i)} := t - (log(random))/\lambda^{(i)}_{y^{(i)} y^{(i)}}$;
  - insert the event $(c, z^{(i)}, t^{(i)})$ in the sorted event list;
- until the end of the simulation.

The cost of the *log* function call appears again plus the cost of insertion in a sorted event list. The discrete choice for the next state on component i operates on a small number of choices (at maximum the size of $S^{(i)}$).

The simulation of the corresponding uniformized chain is:

Markov-4

- $n := 0$; Initial current state is $x^{(1)}, \ldots, x^{(N)}$;
- Repeat
  - choose the component to move i and the next state equal to
    $y^{(i)} \neq x^{(i)}$ with probability $\frac{\lambda^{(i)}_{x^{(i)} y^{(i)}}}{\lambda}$ and $x^{(i)} := y^{(i)}$
    or dummy transition with probability $1 - \sum_{i=1}^{N} \frac{\lambda^{(i)}_{x^{(i)} x^{(i)}}}{\lambda}$;
  - $n := n + 1$;
- until the end of the simulation.

Markov-4 is an adaptation of **Markov-2** taking into account the vector structure of X. The complexity of the event list insertion in **Markov-3** is reported in **Markov-4** in the discrete choice, which has a number of options proportional to the sum of the $S^{(i)}$ sizes. This main discrete choice can be done in two steps, which is more efficient if N is large:

[6] $\bigoplus$ denotes the tensor product of matrices [3].

Markov-5

- $n := 0$; Initial current state to $x^{(1)}, \ldots, x^{(N)}$;
- Repeat
  - choose the component to move i with probability $\frac{\lambda^{(i)}}{\lambda}$;
  - choose the next state equal to $y^{(i)} \neq x^{(i)}$ with probability $\frac{\lambda^{(i)}_{x^{(i)}y^{(i)}}}{\lambda^{(i)}}$ and $x^{(i)} := y^{(i)}$ or dummy transition with probability $1 - \frac{\lambda^{(i)}_{x^{(i)}x^{(i)}}}{\lambda^{(i)}}$;
  - $n := n + 1$;
- until the end of the simulation.

The Appendix contains a discussion (mostly extracted from [15]) on algorithms to perform efficient discrete choices with preassigned probabilities among a large number of possibilities.

Comparing the performance of **Markov-3**, **Markov-4** and **Markov-5** is generally very hard since they are model dependent: number of components, size of each component, size of the reachable state space, rate values, etc. We give only here an indication of the relative cost of the basic steps of the simulations, namely *random* calls, *log* calls, discrete choices and insertion.

To perform fast random choices, it is compulsory to have an efficient *random* generator. The generator we use is FSU-ULTRA developed by researchers at Florida State University[7] [9].

The table below gives the time spent in each function call on an *Ultra Sparc 2* with 512 Megabytes of memory. In Table 1 *dis(n)* indicates a discrete choice from a set of n possibilities using the **Discrete-1** method (see Appendix) and *ins(n)* denotes the insertion of an element in a sorted list of n items.

TABLE 1. Clock increment in microseconds.

add	random	log(random)
0.05	0.23	0.7

TABLE 2. Choice calls in microseconds.

dis(10)	dis(100)	dis(1000)	dis(10000)	ins(10)	ins(100)	ins(1000)	ins(10000)
0.56	0.8	1.03	1.36	1.42	1.43	1.48	12.7

From Table 1, it can be seen that a constant clock time increment is about 10 times more efficient than the random time increment using a logarithm. From Table 2 we see that the calls for random discrete choices are more efficient than corresponding insertion calls for the same data size. Indeed, in our approach, an insertion in a list of 10 000 items should be typically replaced by two discrete choices on subsets of 10 and 1000 items and the speed up obtained by this approach is about 10.

[7]It may be obtained by contacting Arif Zaman (`arif@stat.fsu.edu`) or George Marsaglia (`geo@stat.fsu.edu`).

3. Using events and modularity

One of the issues in simulating the λ-HM chain is the computation of λ. Indeed, if the chain is very large (*e.g.*, a vector of more than 10^4 components) an exact computation in the general case is impossible as it involves the explicit computation of all the diagonal entries of the matrix. Thus, we need methods to compute a clock rate value which is a reasonable approximation to the optimal clock. The method developed here is based on the concept of event in Markov chains (rather than transitions) and the modularity of the description. The concept of event is important for SANs, but has also been proposed in previous work [5], for Petri nets and in the event list simulation.

3.1. Markov chain simulation using events

Consider a continuous-time Markov chain $X = (X_t)$, $t \in \mathbb{R}$, with finite state space S and transition matrix Q.

We are interested in one strongly connected reachable set R, on which the chain is typically ergodic. The dynamics of this chain on R is completely defined by a set of independent events noted E. We assume that event e has a constant rate λ_e. An event is said to be *fireable* in a state, if and only if it may occur in this state (this terminology is standard in the context of Petri nets). In modelling, events are easily identified. We believe that, in most systems, the number of events is much smaller than the number of transitions. For instance, in a Jackson network, the events are the arrivals and departures. An arrival leads to several transitions, as many as there are possible states in which this event may occur.

In state x, a fireable event e may lead to multiple destinations according to routing probabilities. Denote them $p_{e,xy}$, for all possible destinations y from state x. These probabilities induce a discrete probability distributions on the set of possible destinations, possibly degenerating to a single destination. Denote by E_x the set of fireable events in state x. This set E_x is such that:

- for all x, y such that $\lambda_{xy} \neq 0$, there exists a single $e \in E_x$ such that $\lambda_{xy} = \lambda_e \times p_{e,xy}$;
- for all x, for all e fireable in x, $\sum_{y \in R} p_{e,xy} = 1$, and
- $\lambda_{xx} = \sum_{y \in R} \lambda_{xy} = \sum_{e \in E_x} \lambda_e$.

Proposition. *$h = \sum_{e \in E} \lambda_e$ is a possible uniformization clock rate and is the optimal one if there exist a reachable state in the chain X where all the events are fireable.*

Proof. If there exists a reachable state for the chain X where all the events are fireable, all events compete (independent and exponentially distributed random variables), and the rate at which the first event occurs is precisely $h = \sum_{e \in E} \lambda_e$. If such a state does not exist, $\lambda_{xx} = \sum_{e \in E_x} \lambda_e$ is always strictly less than λ. In state x, the rate of dummy transitions for this h-HM chain is $\lambda - \sum_{e \in E_x} \lambda_e$ and corresponds to the cumulative rate of non fireable events in x. The simulation of the h-HM chain is as follows:

Markov-6

- $n := 0$; Initial state is x;
- Repeat
  - choose the event to fire e with probability $\frac{\lambda_e}{\lambda}$;
  - if e is fireable in state x, choose a destination y with probability $p_{e,xy}$ then $x := y$ else dummy transition;
  - $n := n + 1$;
- until the end of the simulation.

This algorithm is a modification of the algorithm **Markov-2** by decomposing the discrete choice of the transition into two levels:

- for an actual transition, $x \neq y$, the choice of an event is followed by the choice of the routing probability, and according to the definitions given above, $\lambda_{xy} = \lambda_e \times p_{e,xy}$;
- the dummy transition in state x occurs when choosing an event which is not fireable in x. The probability of a dummy transition in x is $1 - \frac{\sum_{e \in E_x} \lambda_e}{\lambda} = 1 - \lambda_{xx} = \frac{\sum_{e \notin E_x} \lambda_e}{\lambda}$.

In this algorithm the part "else dummy transition" may be omitted (and will be in future versions) as it corresponds to a "null" instruction (no change of state).

3.2. Hierarchy of events

It is always possible to classify events into a hierarchy of subsets. To simplify the demonstration, we propose a classification with one level. Let $(B_1, \ldots, B_K)$ be a partition of E. The algorithm becomes:

Markov-7

- $n := 0$; Initial state is x;
- Repeat
  - choose the set of events to fire B_k with probability $\frac{\sum_{e \in B_k} \lambda_e}{\lambda}$;
  - choose the event to fire e with probability $\frac{\lambda_e}{\sum_{e \in B_k} \lambda_e}$;
  - if e fireable in state x, choose a destination y with probability $p_{e,xy}$ then $x := y$;
  - $n := n + 1$;
- until the end of the simulation.

Markov-7 obviously simulates the same Markov chain as **Markov-6** with an extra level of hierarchy in the choices. As an example, consider a Jackson network with N queues (external arrivals to queue i with rate λ_i and service rate μ_i). The optimal clock is $\lambda = \sum_{j=1}^{N} (\lambda_i + \mu_i)$ and the sets B_k can be:

- the events with the same rate. This requires a classification of the rates and is easy when the networks is built by duplication of basic elements. In this case, the first discrete choice is that of the class and the second one is uniform, thus less expensive;

- the events (external arrival and end of service) of a queue. The first choice is among N possibilities with respective probabilities $\left(\frac{\lambda_i+\mu_i}{\lambda}\right),\ i=1\ldots N$. The second one is a Bernoulli choice between arrival or departure, with probabilities $\frac{\lambda_i}{\lambda_i+\mu_i}, \frac{\mu_i}{\lambda_i+\mu_i}$;
- all the events of the same type (arrival, end of service). The first choice determines whether it is an arrival or a departure with probabilities $\left(\frac{\sum_{i=1}^N \lambda_i}{\lambda}, \frac{\sum_{i=1}^N \mu_i}{\lambda}\right)$. The second one selects the queue with probabilities $\left(\frac{\lambda_i}{\sum_{i=1}^N \lambda_i}\right),\ i=1\ldots N$ if an arrival has been chosen by the first choice, and $\left(\frac{\mu_i}{\sum_{i=1}^N \lambda_i}\right),\ i=1\ldots N$ otherwise.

The actual choice of a hierarchy is governed by the values of the rates, which determine which choices will lead most frequently to fast discrete choice algorithms.

3.3. Functional event rates

Assume now that the rate of an event is not constant, but varies according to the state in which it takes place (functional rates is another term for state dependent rates). In a queuing network, this is the case for a queue with multiple server, for a server with a processor sharing discipline, etc. It is a frequently arising situation. For any event e, denote by $\lambda_e^{\max}$ the maximum value of the rate function. The rate of e in state x can always be written as $\lambda_e^{\max} \times f_e(x)$. Then $f_e(x)$ is a positive function with maximum value 1 and can be interpreted as a probability.

In this context and without extensive analysis of the diagonal of Q, a possible clock is $h = \sum_{e\in E}\ \lambda_e^{\max}$. This clock is optimal if and only if there is a state in which all events are fireable and at a maximal rate. This condition is very strong and might rarely be reached in practice. Considering the transition matrix of the h-HM Markov chain, the values $f_e(x)$ and $1-f_e(x)$ appear to be routing probabilities: with probability $f_e(x)$ an actual transition occurs and with probability $1-f_e(x)$ a dummy transition occurs. The simulation is:

Markov-8

- $n := 0$; Initial state is x;
- Repeat
  - choose the event to fire e with probability $\frac{\lambda_e}{\lambda}$;
  - if e fireable in state x, then with probability $f_e(x)$ choose a destination y with probability $p_{e,xy}$, then $x := y$;
  - $n := n+1$;
- until the end of the simulation.

In this algorithm, a transition is fired to state $y \neq x$ with probability $\frac{\lambda_e f_e(x) p_{e,xy}}{\lambda}$. A dummy transition is fired if e is not fireable in x or if the Bernoulli trial with

probability $f_e(x)$ is not successful, so with probability $\frac{\sum_{e \notin E_x} \lambda_e}{\lambda} + \frac{\sum_{e \in E_x} \lambda_e (1 - f_e(x))}{\lambda}$ $= 1 - \frac{\sum_{e \in E_x} \lambda_e f_e(x)}{\lambda}$. Therefore this algorithm simulates the Markov chain with transition matrix $I + \frac{Q}{\lambda}$.

This last algorithm concludes the use of events for the simulation of uniformized Markov chains. The next section studies further improvements due to modularity in the context of SANs.

3.4. SAN presentation

A Stochastic Automata Network is a formalism which facilitates the modular description of vector Markov chains using the concept of event. A SAN is a collection of components called stochastic automata. There are basically two ways in which stochastic automata interact [11]: the rate at which a transition may occur in one automaton may be a *function* of the state of other automata. Such transitions are called *functional*[8] transitions. A transition in one automaton may *force* a transition to occur in one or more other automata. We refer to such transitions collectively under the name of *synchronized* transitions. Synchronized transitions may also be functional. In any given automaton, transitions that are not synchronized transitions are said to be *local* transitions [1, 10]. Transitions are associated to events that are local or synchronizing events. Events may be associated with several transitions.

Consider a SAN with N automata. It is an N-component Markov chain whose components are not independent due to functional transition rates and synchronizing events. A local state for automaton i is denoted by $x^{(i)}$ and the global state of the SAN is a vector $x = (x^{(1)}, \ldots, x^{(N)})$. A vector $(x^{(1)}, \ldots, x^{(i)})$, with $i \leq N$ is called a partial state. The respective sets of states are denoted by $S^{(i)}$ and $S = S^{(1)} \times \ldots \times S^{(N)}$. The reachable state space of the SAN is denoted by R.

In a SAN, transitions are described on each automaton. A transition is labeled with the events names and rates which may cause this transition and a routing probability indicating the possibility of different transitions from the same initial local state and for the same event. These routing probabilities may also be functional and are denoted by $p_{e,x^{(i)}y^{(i)}}(x)$. The events are defined as follows:

- a local event is associated with the transitions of only one automaton;
- a synchronizing event is associated with the transitions of more than one automaton. Actually, each synchronizing event e is associated with a set of concerned automata O_e. The event is fired if and only if all the automata in O_e are ready to procede for this synchronized transition, and effectively do it. To generalize the notation, if e is local to i, $O_e = \{i\}$. Within O_e an automaton is arbitrarily elected to be the "master" of the synchronizing event. It can be the automaton which models the triggering component for

[8] We use the term functional instead of the expression "state-dependent" in order to insist on the the fact that these rates are dependent on the state of *other automata*. In queuing theory, a state dependent rate of a queue depends on the state of the queue itself.

this event (if there is a single triggering component, which is not always the case);
- all these events are independent and have functional transition rates in the form $f_e(x)\lambda_e^{\max}$ in state x. Denote by E the set of all events.

Let us extend the use of the terminology *fireable*:

- an event e is fireable in local state $x^{(i)}$ if there exists a transition labeled with e from $x^{(i)}$ in automaton i. If e is a synchronizing event and fireable on a local state, this does not imply that it is -actually-fireable on the global state (see further);
- an event e is fireable in the partial state $(x^{(1)}, \ldots, x^{(i)})$ if for all automata $i \in O_e \bigcap [1, i]$, there exists a transition labeled with e from $x^{(i)}$;
- an event is fireable in the global state x if for all automata $i \in O_e$, there exists a transition labeled with e from $x^{(i)}$. If e is fireable in the global state $x = (x^{(1)}, \ldots, x^{(N)})$, it is fireable on each $x^{(i)}$ in O_e.

According to these definitions, denote by E_x the set of all fireable events in state x, x being a global, partial or local state.

Let us consider a Jackson network: the local events are the external arrivals to a queue, the end of service leading to a departure or a feed-forward routing. The synchronizing events correspond to transitions of one customer from one queue to another.

The issue now is to benefit from the SAN structure to compute a good approximation of the clock and propose an efficient hierarchy of discrete choices.

3.5. Simulation of a SAN

The first option in simulating a SAN, is to directly apply the general framework of **Markov-8**: the clock is $h_1 = \sum_{e \in E} \lambda_e^{\max}$ and it is obviously greater than the optimal clock λ as it is the rate at which all the events of the model compete at their maximum rate.

SAN-1

- $n := 0$; Initial state is $x = x^{(1)}, \ldots, x^{(N)}$;
- Repeat
  - choose the event to fire e in E with probability $\frac{\lambda_e^{\max}}{h_1}$;
  - if e fireable in state x, with probability $f_e(x)$ choose a destination y with probability $\Pi_{i \in 0_e} p_{e,x^{(i)}y^{(i)}}(x)$ then $x := y$;
  - $n := n + 1$;
- until the end of the simulation.

In this simulation

- an actual transition from x to y is fired with probability $\frac{1}{h_1}\lambda_e^{\max} f_e(x) \times \Pi_{i \in 0_e} p_{e,x^{(i)}y^{(i)}}(x)$;
- a dummy transition is fired (as in **Markov-8**) with rate $\frac{\sum_{e \notin E_x} \lambda_e^{\max}}{h_1} + \frac{\sum_{e \in E_x} \lambda_e^{\max}(1 - f_e(x))}{h_1}$.

This simulates a Markov chain with transition matrix $I + \frac{1}{h_1}Q$. The problem here is that h_1 may be much too large and this simulation may produce too many dummy transitions.

A possible improvement is to define a so-called local clock $\lambda^{(i)}$ for each automaton as if they were independent. For this denote, for each automaton:

- $\bar{E}_{x^{(i)}} \subset E_{x^{(i)}}$ the set of all local events, fireable in local state $x^{(i)}$ plus the synchronizing events fireable in local state $x^{(i)}$ if i is the master. Denote $\lambda_{x^{(i)}} = \sum_{e \in \bar{E}_{x^{(i)}}} \lambda_e^{\max}$;
- $\lambda^{(i)} = \max_{x^{(i)} \in S^{(i)}} (\lambda_{x^{(i)}})$. $\lambda^{(i)}$ can be computed from the transition matrix of automaton i;
- finally, set $h_2 = \sum_{i=1}^{N} \lambda^{(i)}$. h_2 is the sum of some maximal rates without redundancy: local events are counted once on each automaton and synchronizing events once in their master automaton.

Proposition. *With the definitions given above, $h_2 = \sum_{i=1}^{N} \lambda^{(i)}$ is a possible uniformization clock rate.*

Proof. we have to show that h_2 is greater than λ to be eligible for being a clock. As λ is the maximum of all diagonal coefficient λ_{xx} of Q (x being a global state), it is sufficient to prove that for all x, $\lambda_{xx} \leq h_2$.

Consider a global state x. The set of fireable events in $x = (x^{(1)}, \ldots, x^{(N)})$, E_x, is included in $\bigcup_{i=1}^{N} \bar{E}_{x^{(i)}}$. This can easily be seen from the definitions (Sect. 3.4) since a synchronizing event fireable in x is fireable in $x^{(i)}$ if i is the master of the transition, and not the opposite. Thus

$$\lambda_{xx} = \sum_{e \in E_x} f_e(x) \lambda_e^{\max} \leq \sum_{e \in E_x} \lambda_e^{\max} \leq \sum_{i=1}^{N} \sum_{e \in \bar{E}_{x^{(i)}}} \lambda_e^{\max} \leq \sum_{i=1}^{N} \lambda^{(i)} = h_2.$$

This concludes the proof of $\lambda \leq h_2$. It can be seen that $h_2 \leq h_1$ as h_2 is the sum of some maximum rates (without redundancy) and h_1 is the sum of all maximum rates. This clock allows us to define a hierarchical choice: the automaton is chosen, and then the event as shown in the next simulation, **SAN-2**:

SAN-2

- $n := 0$; Initial state is $x = x^{(1)}, \ldots, x^{(N)}$;
- Repeat
  1. choose the automaton i with probability $\frac{\lambda^{(i)}}{h_2}$;
  2. choose $e \in \bar{E}_{x^{(i)}}$ with probability $\frac{\lambda_e^{\max}}{\lambda^{(i)}}$ or goto 4 with probability $1 - \frac{\sum_{e \in \bar{E}_{x^{(i)}}} \lambda_e^{\max}}{\lambda^{(i)}}$;
  3. if e fireable in state x, then with probability $f_e(x)$ choose a destination y with probability $\Pi_{i \in 0_e} p_{e,x^{(i)}y^{(i)}}(x)$ then $x := y$;
  4. $n := n + 1$;
- until the end of the simulation.

This simulates a Markov chain with transition matrix $I + \frac{1}{h_2}Q$ as

- an actual transition from x to y is fired with probability $\frac{1}{h_2}\lambda_e^{\max} f_e(x) \times \Pi_{i \in 0_e} p_{e,x^{(i)}y^{(i)}}(x)$;
- a dummy transition is fired with rate $\frac{\sum_{e \notin E_x} \lambda_e^{\max}}{h_2} + \frac{\sum_{e \in E_x} \lambda_e^{\max}(1-f_e(x))}{h_2}$.

Note that **SAN-2** is not obtained from **SAN-1** by a simple partition of the set of events E. **SAN-2** takes into account the structure of the SAN. More precisely, which events may compete (and which do not compete and thus do not increase the clock) is an important criterion which is partially exploited in **SAN-2**.

As it is impossible to assess the performance of such a method in general, due to the impact of the model characteristics, the next section describes an example of road traffic simulation to demonstrate the effectiveness of this approach.

4. Road traffic analysis

4.1. The model and its simulation

The objectives of road traffic simulation are traffic light regulation, impact analysis for new infrastructures, traffic jam avoidance, users information, etc. Different types of simulations are possible: microscopic simulations in which cars and their routes are modeled individually, mesoscopic simulation in which cars are modeled individually but interactions can be modeled with a higher level of abstraction, and finally macroscopic simulations which handles subsets of vehicles and flows [7, 8, 12–14]. The type of simulation we present is mesoscopic in the sense that the state of the roads are numbers of vehicles per road section and the movements between sections are done unit per unit (as in queues) and not by flows. Interactions are modeled by state dependent rates and probabilities.

In the model we present, the basic element (an automaton) is a road section: a road section is the set of lanes going in the same direction between two separating elements (traffic lights, signs, crossing, sensor, or user defined separation). We do not model lane changing, but this can be done within the SAN framework, by having one automaton per lane.

An urban traffic area is described by a set of road sections. A road section is characterized by its name, length, width, number of lanes, speed limit, allowed vehicle types (cars, trucks, bus, bikes, etc.), traffic lights or priorities, possible output sections, presence of bus stops, pedestrian crossing, lateral parking, traffic sensors... The traffic in this area is characterized by a set of input and output points, input flows, an origin/destination (O/D) matrix for these points and the initial state of the area. The input of all this data requires a dedicated interface [13].

The modelling process gives the following result (i and j denotes sections and t a vehicle type):

- a road section is modelled by an automaton, whose state is the number of vehicles per type (cars, buses, etc.). The events are the vehicle inputs in the

area and the vehicle exits from each section (per type). A road section is a queue with a state dependent service rate, and synchronizing constraints;

- a road section is characterized by a capacity (computed from the length and width), one throughput function per vehicle type, routing probabilities $p_{ij,t}$ per vehicle type (for choosing their next section) and a signalization function (see further);
- the throughput function is pre-computed from static parameters as the number of lanes, speed limit, presence of bus stops, pedestrian crossing, lateral parking (all kinds of elements that can increase or decrease the throughput), and is a function of dynamic parameters such as the number of vehicles in the section or in the section just ahead, which we shall call, the *up-ahead* section. This function has a concave profile and gives the different throughput values of the section as the number of vehicles increase: first a linear increase, then a stagnation, finally a decreasing throughput as the number of vehicles becomes too large and provokes congestion. Computing this profile is a subject of research in itself and is not within the scope of this paper. This throughput function has the format $\lambda_i^{\max} f_{e_{i,t}}(x)$, i being the section and $e_{i,t}$ the event "exit from section i of a type t vehicle. This maximum throughput $\lambda_i^{\max}$ is common to all types of vehicles in the same section (even if it is never reached for some vehicle types). The sections are classified according to their maximum throughput $\lambda_i^{\max}$ (which is not directly connected to the speed limit). Thus a section *class* is characterized by a maximum rate (or throughput);
- the routing probabilities are pre-computed from the origin/destination matrix, using shortest path algorithms with road ponderations and some randomization. They are also possibly functions of up-ahead section occupancy;
- the signalization function is a time dependent function with value 0 or 1 in case of a traffic light, and a probabilistic $r_{ij,t}$ function depending on the sections occupancy in case of priority rules. This signalization function depends on the chosen up-ahead section j.

We do not go into details of covered parking, input/outputs, sensors, which are included in the model. They are not mentioned in the algorithm sketched below to simplify the presentation.

Without going through a formal (and tedious) description, we shall assume that the set of sections of an urban area is a SAN with a Markovian behavior under the assumptions that residence times of vehicles in sections are exponentially distributed and that inputs are Poisson. If the model has traffic lights or time dependent inputs or O/D matrix, the Markov process is not homogeneous and is an extension of the general framework presented in this paper[9].

A possible (there exist many alternatives and it is impossible to enumerate them) simulation is the following:

[9]In this case, the process is governed by rates with a cyclic behavior (such as traffic lights). An homogeneous chain is obtained by considering the state of the area only at the starting points of each cycle.

Traffic

- $n := 0$; Initialize;
- Repeat
  - choose the section class;
  - choose the section within the class;
  - if there are vehicles in the section
    - choose the vehicle type;
    - choose the up-ahead section;
    - if e fireable then do it;
  - $n := n + 1$;
- until the end of the simulation.

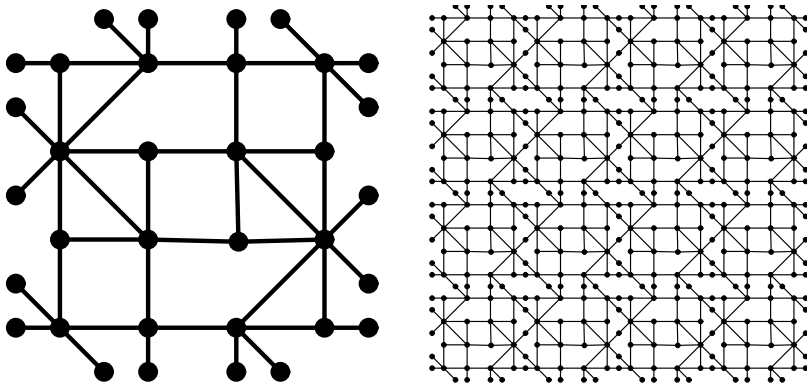
This algorithm follows the **SAN-2** structure. Items 1 and 2 decompose the choice of the automaton into two levels and while 3 and 4 decompose the choice of the event into two levels. These events are mainly synchronizing events as they synchronize the output and input sections. The condition "fireable" is related to capacity limits, traffic lights, actual rate, and priority rules. The clock is the sum of the maximum throughput or rates of all the sections (including input sections), which are the local clocks for each section. Let us examine the various choices:

1. **choice of the section class**: if there are K classes containing n_k, $k \in [1, K]$ sections, each characterized by a maximum throughput $\lambda_k^{\max}$, then this choice is discrete with distribution $\left(\frac{n_k \lambda_k^{\max}}{\lambda}\right)$, and the clock of the simulation is $\lambda = \sum_{k\in[1,K]} n_k \lambda_k^{\max}$. It is implemented with a **Discrete-1** algorithm (see Appendix) and a binary search as the probabilities of the choices are very different due to the factors n_k and the high number of classes;
2. **choice of the section within the class**: the choice of the section within the class is done by using a simple uniform choice on n_k possibilities since the sections have the same maximum rate. Once the section i is chosen, the event set $E_{x^{(i)}}$ corresponds to a vehicle exit from section i and the routing probabilities are defined by the vehicle type and the up-ahead section;
3. **choice of the vehicle type**: as we mentioned before, all the vehicle types have the same maximum rate in one section. So if T denotes the number of vehicle types, and x_t the current number of vehicles of each type t in the section, then this choice is a discrete choice with distribution $\left(\frac{x_t}{\sum_{t\in[1,T]} x_t}\right)$. Note that this distribution has to be recomputed at each step since the values x_t vary in time. It is implemented with the **Discrete-1** algorithm and a linear search as the probabilities are very different due to the higher proportion of cars and the number of vehicle types is small;
4. **choice of the up-ahead section**: this choice is done according to the routing probabilities of traffic. These routing probabilities depend on the vehicle type only. It is implemented with the **Discrete-1** algorithm and a linear search as the number of possible destinations is small;
5. **firing conditions**: if the selected event e is "a vehicle of type t is going from section i to section j", the event is not fireable if the up-ahead section

cannot contain this extra vehicle or if the corresponding traffic light is red (a traffic light value is associated with each possible up-ahead section). If these conditions enable the transition, slowing down conditions are due to actual traffic rates and possible priority rules. The actual rate is in the form $f_{e_{i,t}}(x)\lambda_i^{\max}$ and the transition is accepted with probability $f_{e_{i,t}}(x)$. If the up-ahead section j is chosen, priority is modelled by a probability $r_{ij,t}(x)$ which takes into account the number of vehicles in the conflicting sections. The transition takes place with probability $r_{ij,t}(x)$ and is not allowed otherwise.

4.2. Experimentations results

To illustrate the performance of this approach, we built an artificial city area as depicted in Figure 1a. Each line in this model represents 2 street sections, each section represents a road sense. This area has 88 sections of equal length, width (which gives a capacity of 25 cars) and speed limit. It has two traffic lights and the rest is governed by right-hand-side priorities. It has two vehicle types (bus and cars) and no dedicated lanes. It has 18 input and output points. The input rate is constant. The routing probabilities are constant and uniform on the number of possibilities at each crossing. The throughput function is a discretized convex function of the section load with 10 different values. The priority function is a decreasing function of the "on the right" section load with 4 different values. The initial state of the city is 16 cars per section.



(a) (b)

Figure 1. Basic city model with 88 sections and a replication with 1408 sections.

This small city area is then used to built larger areas by connecting inputs to outputs as depicted in Figure 1. Basically for our experiment we chose areas

with 88, 352, 1408, 5632, 22 528 and 90 112 sections (each being 4 times greater than the preceding one).

In the rest of the section, we study the impact of the model size on different measures: clock step (Fig. 2a), memory occupancy (Fig. 2b), number of simulation iterations per CPU time (Fig. 3a), simulation speed (simulated time *versus* CPU time, Fig. 3b). Table 3 shows the values that are used to plot the following curves and represent simulation runs for 4000 seconds of simulated time.

TABLE 3. General table.

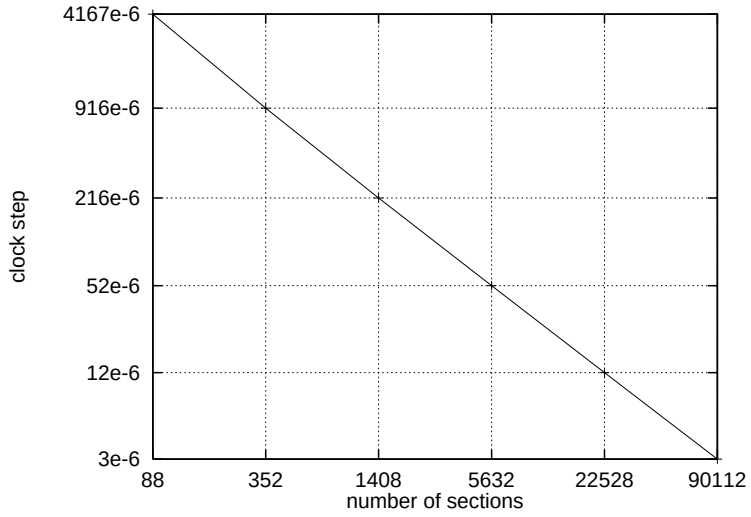
size	mem.(Kbytes)	clock step	CPU time (seconds)	number of iterations
88	1264	4167e-6	3.74	960 000
352	1376	916e-6	17.38	4 370 000
1408	1832	216e-6	75.73	18 500 000
5632	3696	52e-6	387.65	76 100 000
22528	11 264	12e-6	1664.48	308 000 000
90112	40 960	3e-6	7352.96	1 245 000 000

In Figure 2a, we see that the clock step (basic increment) is a linear decreasing function of the model size. This result is a logical consequence of the clock computation algorithm (see **Traffic** algorithm): we have 3 section classes (inputs, outputs and inside sections) and as the model size is bigger, the increase of inside sections dominates. The clock step is roughly a linear function of the number of inside sections.

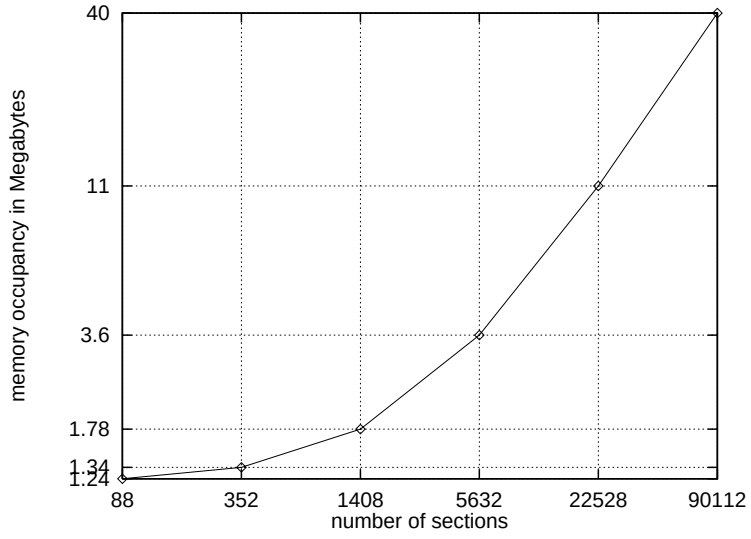
In Figure 2b, we show the amount of memory used to run the simulations: an area of 90 000 sections requires 40 Mgbytes of memory. These values depend on the chosen area (number of sections, traffic lights, vehicle types, etc.).

By comparing the area with 88 sections and the area with 90112 sections, we observe that, when the area is roughly 1000 times bigger, the number of iterations per second increases by a factor of 2 (see Fig. 3a). In a larger town, each iteration leading to an actual vehicle move is expensive, but the hierarchy of choices reduces this increasing cost to a minimum. Note that there are more dummy transitions in a large town and these dummy transitions are cheaper.

In Figure 3b, we see that the simulation speed (number of simulated time units per CPU time units) is a decreasing function; when it is goes below the value 1, the simulation is slower that real time. For our experiment, this limit is reached with about 40 000 sections. Note that this simulation speed decreases when the model size increases because each iteration is more costly (Fig. 3a) and because the clock step decreases (Fig. 2a).



(a)

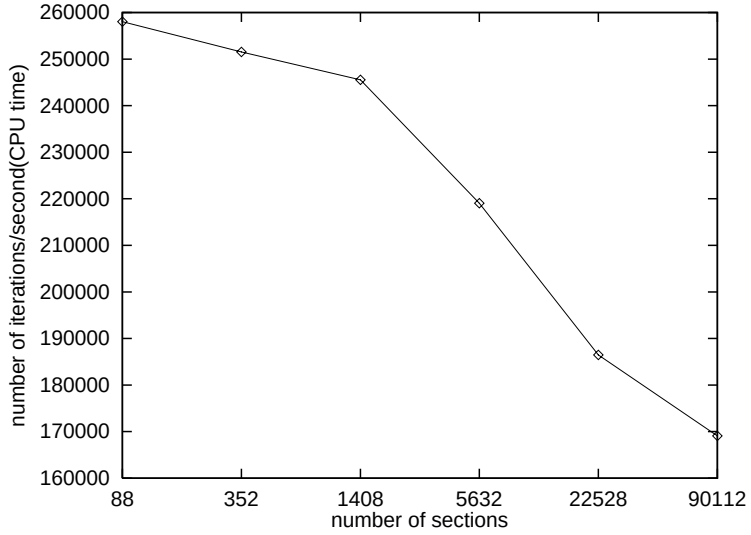


(b)

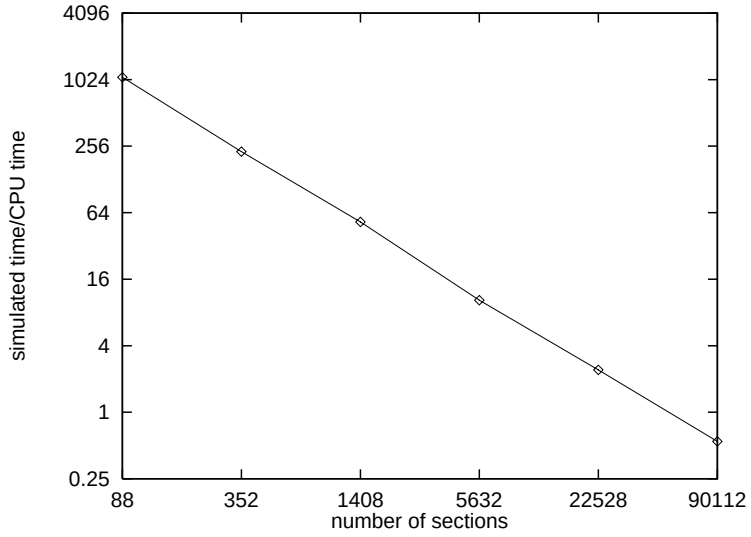
FIGURE 2. Clock step *versus* model size (a) and memory occupancy *versus* model size (b).

5. Further work

This work on simulation using uniformization, events and modularity of SAN models has shown that this approach may be very efficient. From these first results,



(a)



(b)

FIGURE 3. Number of iterations per second of CPU time *versus* model size (a) and simulated speed (simulated time *versus* CPU time) *versus* model size (b).

improvements can be achieved by studying various alternatives for choose the clock and implementing discrete random choices. Here, only two possibilities were given,

but many others can be suggested. Comparing them and automatically choosing the best algorithm for a particular model will be the object of future research. Parallelization is also an issue of interest in order to increase the size of the model and the relative speed.

Appendix: Fast discrete choices

In this section, Z denotes a discrete random variable on the finite integer state space $[1, N]$, with distribution $(p_1, \ldots, p_N)$. Denote by $F_i = \sum_{j=1}^{i} p_j$ the cumulative distribution function. The basic algorithm for generating a random variable with this distribution is the following (*random* denotes the call to a random generator returning a real value uniformly chosen in $[0, 1]$).

Discrete-1

- $i := 1$; $R := random$;
- Repeat while $R < F_i$
  - $i := i + 1$;
- $Z := i$.

As the number of tests in the previous algorithm is i with probability p_i, the best ranking of the values is $p_1 \geq p_2 \geq \ldots \geq p_N$. Typically, this method is efficient if a small number of probabilities p_i has a sum close to 1. This sequential search may be replaced by a binary search if the sequence is long.

Another method for generating the same random variable is called the rejection method and is depicted in **Discrete-2**.

Discrete-2

- Repeat
  - $i := random[1, N]$;
- Until $random < \frac{p_i}{p_1}$;
- $Z := i$.

where $random[1, N]$ returns an integer uniformly chosen in $[1, N]$ and p_1 is the largest value of the sequence $(p_1, \ldots, p_N)$. *random* is another call to a random generator returning a real number in $[0, 1]$.

The proof is as follows, if t is the number of turns in the loop:

$$Prob(Z = i) = \sum_{t=1}^{\infty} \frac{p_i}{Np_1} \left(1 - \frac{1}{Np_1}\right)^{t-1} = \frac{p_i}{Np_1} \times Np_1 = p_i$$

since $\frac{p_i}{Np_1}$ is the probability of exiting a turn in the loop with value i and $1 - \frac{1}{Np_1} = 1 - \sum_{i=1}^{N} \frac{p_i}{Np_1}$ is the probability of not exiting the loop.

This is called a rejection algorithm since it can be interpreted as such: the first random choice proposes a value i as a result, but this proposition can be

rejected with probability $1 - \frac{p_i}{p_1}$. Note that the proposition 1 is never rejected. This algorithm is efficient if the values p_i are not very different. Note also that it is not very sensitive to the value of N.

These two methods (**Discrete-1** and **Discrete-2**) can be combined into a decomposition method. The framework is as follows: let $(B_1, \ldots, B_K)$ be a partition of the integer interval $[1, N]$, $q_m = \sum_{i \in B_m} p_i$ and for $i \in B_m$ $r_i = \frac{p_i}{q_m}$. The algorithm is as follows:

Discrete-3

- choose m according to the distribution $(q_1, \ldots, q_K)$;
- choose $i \in B_m$ according to the Bernoulli $r_i = \frac{p_i}{q_m}$ for $i \in B_m$;
- $Z := i$.

This decomposition can be used with an arbitrary depth of decomposition, with the objective of having an efficient discrete choice algorithm at each level, keeping in mind that the fastest discrete choice is always the one with equal probabilities.

Acknowledgements. Thanks to Guillaume Etievent and his skills for implementing our graphical interfaces.

References

[1] K. Atif, *Modélisation du Parallélisme et de la Synchronisation.* Ph.D. Thesis, Institut National Polytechnique de Grenoble (1992).

[2] S. Donatelli, Superposed stochastic automata: A class of stochastic petri nets with parallel solution and distributed state space. *J. Performance Evaluation* **18** (1993) 21-36.

[3] P. Fernandes, B. Plateau and W.J. Stewart, Efficient descriptor-vector multiplications in stochastic automata networks. *J. ACM* **45** (1998) 381-414.

[4] P. Fernandes, B. Plateau and W.J. Stewart, Optimizing tensor product computations in stochastic automata networks. *RAIRO: Oper. Res.* **32** (1998) 325-351.

[5] W.K. Grassmann, Finding transient solutions in Markovian event systems through randomization, in *1st International Workshop on the Numerical Solution of Markov Chains*, edited by W. Stewart. North Carolina State University, NC, U.S.A. (1990) 357-372.

[6] P. Kemper, Numerical analysis of superposed gspns. *IEEE Trans. Software Engrg.* **22** (1996).

[7] I. Kosonen, *HUTSIM – Simulation tool for traffic signal control planning.* Ph.D. Thesis, Helsinki University of Technology, Department of Eletrical and Communications Engineering, Finland (1996).

[8] R. Liu, *Dracula microscopic traffic simulation*, ITS Working Paper 431. University of Leeds (1994).

[9] G. Marsaglia and A. Zaman, A new class of random number generators. *J. Appl. Probab.* **1** (1991) 462-480.

[10] B. Plateau, On the stochastic structure of parallelism and synchronization models for distributed algorithms, in *ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*. Austin, Texas, U.S.A. (1985).

[11] B. Plateau and K. Atif, Stochastic automata network for modeling parallel systems. *IEEE Trans. Software Engrg.* **17** (1991) 1093-1108.

[12] T. Saito, K. Yasui, S. Fuji and S. Itakura, Development of microscopic simulation model for traffic network (micstram ii) and traffic flow simulator for evaluation of traffic signal control (tras-tsc), in *2nd World Congress on Intelligent Transport Systems*, Vol. IV, Yokohama (1995) 1920-1925.
[13] P.L. Toint, Transportation modelling and emerging technologies, *Tech. Rep. 93/23, Transportation Research Group*. Department of Mathematics, Facultés Universitaires Notre-Dame de la Paix, Belgium (1993).
[14] Q. Yang and H.N. Koutsopoulos, A microscopic traffic simulator for evaluation of dynamics traffic management systems. *Transportation Res. Part C* **4** (1996) 113-129.
[15] B. Ycart, *Simulation de modèles markoviens*. Cours DESS d'Ingénierie Mathématique, Université Joseph Fourier, Grenoble, France (1997).
`ftp://ftp.imag.fr/pub/MAI/simarrk.ps.gz`