Acceleration of Perfect Sampling by Skipping Events

Furcy Pin
INRIA / ENS
Paris, France
furcy.pin@ens.fr

Ana Bušić
INRIA / ENS
Paris, France
ana.busic@inria.fr

Bruno Gaujal
INRIA Grenoble - Rhône-Alpes
Montbonnot, France
bruno.gaujal@inria.fr

ABSTRACT

This paper presents a new method to speed up perfect sampling of Markov chains by skipping passive events during the simulation. We show that this can be done without altering the distribution of the samples. This technique is particularly efficient for the simulation of Markov chains with different time scales such as queueing networks where certain servers are much faster than others. In such cases, the coupling time of the Markov chain can be arbitrarily large while the runtime of the skipping algorithm remains bounded. This is further illustrated by several experiments that also show the role played by the entropy of the system in the performance of our algorithm.

Categories and Subject Descriptors

I.6 [**Computing Methodologies**]: Simulation and modeling; G.3 [**Probability and Statistics**]: Markov processes

Keywords

Markov chains, perfect sampling, queueing networks.

1. INTRODUCTION

The perfect sampling algorithm for finite Markov chains was introduced in the famed work of Propp and Wilson [10]. The complexity of the algorithm is in $O(S\tau)$ where S is the size of state space and τ is the expected coupling time.

It has been shown that the number of simulated trajectories, S, can be reduced when the Markov chain is monotone by using extreme states [10] or by using bounding chains when the chain is not monotone [7, 3].

As for the coupling time, it may seem that it is impossible to improve on it while keeping the same events giving the construction of the Markov chain. The coupling time is usually difficult to estimate and even to bound, except for some specific Markov chains [5, 1, 8]. Furthermore, there are cases where the coupling time is known to be extremely large, for example when the spectral gap (one minus the size of the second largest eigenvalue of the transition matrix of the chain) is close to zero. This is typically the case when the Markov chain has different time scales: Certain transition events have a very large probability to occur while others have a very small probability. Such chains are very common in queueing theory (where the rate of certain transitions are much higher than the rest) or in social networks (where the connection graph is made of tightly connected components with loose connections between them [6, 9].

In this paper we present a new algorithm for perfect sampling whose complexity is not linear in the coupling time τ but can be arbitrarily smaller. It is based on a partial generation of the sequence of events leading to coalescence. Up to our knowledge, this is the first simulation algorithm of general Markov chains whose runtime can remain bounded while the coupling time of the chain goes to infinity.

Let us now present the main idea of this algorithm more precisely. Perfect simulation as introduced by Propp and Wilson uses a coupling from the past technique. A sequence $u_{n-1} \dots u_1$ of random innovations (or events) is generated backwards: The new random event u_n is added at the head of the sequence, that becomes $u_n u_{n-1} \dots u_1$. This sequence is used to simulate a Markov chain (over a finite state space $\mathcal{S}$): Starting from an initial state X, one can generate a new state obtained by letting the event u_n act on the state, denoted $X \cdot u_n$. By applying the events of the sequence one by one, one gets $X \cdot u_n u_{n-1} \dots u_1 := (((X \cdot u_n) \cdot u_{n-1}) \cdot \dots \cdot u_1)$. The main property of this construction is that if the random innovations are generated according to the transition probabilities of the Markov chain and if the final state $X \cdot u_n u_{n-1} \dots u_1$ is the same for all states $X \in \mathcal{S}$, then this final state is distributed according to the stationary distribution of the Markov chain. This property can be translated into an algorithm for generating samples of the stationary distribution of the Markov chain: For each sequence of events generated according to the transition probabilities the algorithm computes a sample of the stationary distribution.

As mentioned before, in our new perfect sampling algorithm, the sequence of random events is not generated beforehand but only partially at each iteration of the algorithm. We will show that our partial generation does not introduce a bias on the distribution and that our approach can be very useful is certain cases. Indeed, some events may have no effect on the current trajectory. Such passive events, instead of being generated, could simply be skipped. When they occur very often, this leads to a clear gain of memory and computing time. We provide examples where this new algorithm yields a major improvement in the time and memory needed to generate one sample. We give now one simple example, which will be used throughout the paper, to illustrate the motivations of our work.

Example 1. Consider two queues in tandem, Q_1 and Q_2, of capacity $C = 20$ each, with exponential service times of parameters $\mu_1 = \mu_2 = 10$ and receiving customers with a Poisson arrival process with rate λ. Arriving customers join the first queue Q_1, unless it is full in which case they are lost. After being served in Q_1 in a FIFO manner, they go in Q_2 unless it is full, in which case they are lost. When a client in Q_2 is served, it leaves the system. The state of the system is given by the number of customers in each queue. In this Markov chain, there are three types of events: arrival of customers, service in the first queue (implying a transfer in the second queue), and service in the second queue.

An arrival in Q_1 will have no effect on the system when the queue Q_1 is full, since the arriving client will be lost. Thus, when λ is much larger than μ_1 and μ_2, Q_1 will be full most of the time and all the very frequent arrivals will have no effect on the system. One would thus want to ignore these events. Each time Server 1 will execute a service, the next arrival will have an effect on the system again, thus it will be important to ignore events only when they are passive.

Notice that the spectral gap of the transition matrix goes to zero when λ goes to infinity, so that the coupling time also goes to infinity. This is an example where, even if the coupling time goes to infinity, the perfect sampling time will remain bounded using our skipping technique.

The paper is organized as follows. In Section 2 we introduce the notion of a Markov automaton, providing the framework for both the event description of the evolution of a Markov chain and the use of regular language notation. Then, in Section 3 we introduce the idea of event skipping, Section 4 gives a quick reminder of the Envelope Perfect Sampling Algorithm (EPSA) [3], which implements Perfect Sampling by coupling from the past using bounding intervals for coupling detection. In Section 5, we modify this algorithm to apply the event skipping idea to it, and we show its correctness. In the last part we show its efficiency through several experiments.

2. DEFINITIONS AND NOTATIONS

Here is a short introduction to several standard notions, along with some others specific to our work. The partial generation of the sequence of events will be facilitated by seeing this sequence as a *word* and by using *regular expressions.* A discrete event description of a Markov chain can be formally defined as a *Markov automaton.* This also allows for a more compact notation for *bounding interval chains*, used for coalescence detection in perfect sampling.

2.1 Notations on Words

An alphabet A is a set of letters. For example $A = \{a, b, c, \dots, y, z\}$ is the Latin alphabet. A word is a sequence of letters, for example *spaghetti* is a word on the Latin alphabet. We denote by A^* the set of all finite words on the alphabet A, including the empty word ε. We denote by A^ω the set of all infinite words on A.

Words can be naturally concatenated. For instance the concatenation of the words *east* and *wood* gives the word *eastwood.*

A word x is said to be a *factor* of the word $u \in A^*$ if u can be written $u = vxw$ where $v \in A^*$ and $w \in A^*$. For any $i \leq j$, we denote by $u_{i\to j} := u_i u_{i+1} \dots u_j$, and for $i > j$ we define $u_{i\to j} := u_i u_{i-1} \dots u_j$.

A word x is a *prefix* of the word u in A^* if u can be written $u = xv$ with $v \in A^*$ For instance, the prefixes of *saloon* are $\{s, sa, sal, salo, saloo, saloon\}$.

Finally, we say that u is a *subword* of v if the sequence of letters in u appear in that order inside the word v. For instance the word $leaf$ is a subword of $leevancleef$ but not a subword of $vancleef$.

2.2 Regular Expressions

Regular expressions give us a convenient way to write some words or set of words (called *regular languages*).

Definition 1. Regular languages on an alphabet A are inductively defined as follows:

- $\emptyset$ (the empty language) is regular.
- ε (*i.e.* the language $\{\varepsilon\}$) is regular.
- For all letter $a \in A$, the language $\{a\}$ is regular.
- If e and e' are regular languages, then
  - $(e + e') := e \cup e'$ the union of the two languages is regular.
  - $(ee') := \{uu' \mid u \in e, u' \in e'\}$ the concatenation of the two languages is regular.
  - Let e^k denote the concatenation of the language e with itself k times: $e^k := \{u^1 \dots u^k \mid u^1 \in e, \dots, u^k \in e\}$. Then e^k is regular for all k.
  - Let e^* denote the concatenation of the language e with itself any number of times: $e^* := \bigcup_{k\geq 0} e^k$. Then e^* is also regular.

We will also identify any set of letters $B \subset A$ with the regular language $\{b \in B\}$.

2.3 Markov Automaton

It is well known that a Discrete Time Homogeneous Markov Chain (simply Markov chain in the rest of the paper) on a finite state space $\mathcal{S}$ can be generated using a representation with discrete events. Such a representation can be interpreted as the transition of an automaton on a finite alphabet A with the same state space $\mathcal{S}$, and without any initial nor final state (only the transitions are important). Given the adequate probability distribution on the letters on A, drawing a discrete event can be seen as drawing a letter from A. More formally, we introduce a new notion called *Markov Automaton* in the following, which is an automaton without any initial or final states on an alphabet equipped with a probability distribution.

Definition 2. A *Markov Automaton* is a tuple $(\mathcal{S}, A, D, \cdot)$ where $\mathcal{S}$ is the set of states of the automaton, A is an alphabet, D is a probability distribution on that alphabet and $\cdot$ is a right action by the letters of A on the states in $\mathcal{S}$ and is called the *transition function*:

$$\begin{aligned} \cdot \quad : \quad & \mathcal{S} \times A \to \mathcal{S}, \\ & (x, a) \mapsto x \cdot a. \end{aligned}$$

Equivalently, this action can be most naturally defined as (or extended to) a right semigroup action by A^* on $\mathcal{S}$.

$$\begin{aligned} \cdot \quad : \quad & \mathcal{S} \times A^* \to \mathcal{S}, \\ & (x, u) \mapsto x \cdot u := x \cdot u_1 \cdot u_2 \cdot \ldots \cdot u_n \\ & \qquad \text{where } u = u_1 \ldots u_n. \end{aligned}$$

Any Markov automaton $\mathcal{M} := (\mathcal{S}, A, D, \cdot)$ naturally induces a Markov chain with the following construction:

Let $(u_1, \ldots, u_n, \ldots)$ be an infinite sequence of random letters of A *i.i.d.* distributed according to D. More conveniently for the rest of the paper, we will directly consider the infinite random word $u = u_1 \ldots u_n \ldots$ in A^ω and say abusively that it is distributed according to D.

Then for any $x_0 \in \mathcal{S}$, the random process $(X_n := x_0 \cdot u_{1\to n})_{n\in\mathbb{N}}$ is a Markov chain with initial state x_0 and with probability transition matrix P given by

$$\text{for all } x, y \text{ in } \mathcal{S}, \quad P(x, y) = \sum_{\substack{a\in A \\ x\cdot a = y}} D(a). \tag{1}$$

We will say that the Markov chain (X_n) is *generated* by the Markov automaton $\mathcal{A}$. Conversely, for any probability transition matrix P on a finite space state $\mathcal{S}$, it is easy to see that there exists a Markov automaton $\mathcal{A} = (\mathcal{S}, A, D, \cdot)$ such that (1) holds, *i.e.* such that $\mathcal{A}$ generates a Markov chain on S with transition matrix P, but that automaton is not unique in general.

We also introduce the notion of synchronizing words of an automaton, which is well known in automata theory. Informally, a synchronizing word is a word that sends all states of the automaton to the same state.

Definition 3. Let $\mathcal{A} = (\mathcal{S}, A, D, \cdot)$ be a Markov automaton. A word $u \in A^*$ is *synchronizing* for the automaton $\mathcal{A}$ (or u *synchronizes* $\mathcal{A}$) if the set $\{x \cdot u \mid x \in \mathcal{S}\}$ is a singleton.

It is easy to see that if any word u has a factor which is synchronizing, then u is synchronizing as well.

One of the interests of Markov automatons, while the classical transition function representation (with $\varphi(x, a)$ instead of $x\cdot a$) may have sufficed, is the convenience of the notations used in automata theory. Firstly the right action notation allows us to simply denote by $x\cdot ab$ what we should have written $\varphi(\varphi(x, a), b)$, which would become dreadful with more letters. Secondly the simplicity of notations brought by regular expressions will prove to be very handy in the following.

Example 2. We continue Example 1. This queueing system can be modeled using the Markov automaton $\mathcal{A} = (\mathcal{S}, A, D, \cdot)$ with:

- $\mathcal{S} = [0, C] \times [0, C]$ where $[0, C] = \{0, 1, \ldots, C\}$
- $A = \{a, b, c\}$ where a corresponds to an arrival in the system, b to a service by Q_1, and c to a service by Q_2.
- D is given by: $D(a) = \lambda/\Sigma$, $D(b) = \mu_1/\Sigma$ and $D(c) = \mu_2/\Sigma = D(b)$, where $\Sigma = \lambda + \mu_1 + \mu_2$.
- $\cdot$ is given by : for any $x = (x_1, x_2) \in \mathcal{S}$

$$\begin{aligned} x \cdot a &= ((x_1 + 1) \wedge C, x_2) \\ x \cdot b &= \begin{cases} (x_1 - 1, x_2 + 1 \wedge C) & \text{if } x_1 > 0, \\ (x_1, x_2) & \text{else} \end{cases} \\ x \cdot c &= (x_1, (x_2 - 1) \vee 0) \end{aligned}$$

It is straightforward to verify that the Markov chain generated by this automaton will simulate the considered queueing system in discrete time.

2.4 Grand Coupling

Markov automata have a double interest. First, many systems can be modeled with discrete events (*e.g.* queueing networks), and induce naturally a representation of Markov chains with Markov automata. Second, and this is the main interest for us here, an Markov automaton also generates naturally a grand coupling of Markov chains.

Definition 4. Given a Markov chain $\mathcal{M}$ with finite space state $\mathcal{S}$ and transition matrix P, a *grand coupling* of that Markov chain is a family of random processes $\{(X_n(x))_{n\geq 0} \mid x \in \mathcal{S}\}$ such that each $(X_n(x))_{n\geq 0}$ is a Markov chain on $\mathcal{S}$ with transition matrix P starting from x (*i.e.* $X_0(x) = x$).

We say that the grand coupling has *coalesced* at time t if all the processes $(X_n(x))$ are in the same state at time t, or equivalently if the set $\{X_t(x)|x \in \mathcal{S}\}$ is a singleton.

Given a Markov automaton $\mathcal{A} = (\mathcal{S}, A, D, \cdot)$ and an infinite random word u distributed according to D, the family of random processes $\{(X_n(x)) := x \cdot u_{1\to n})_{n\in\mathbb{N}}) \mid x \in \mathcal{S}\}$ is a grand coupling. By construction, a grand coupling given by a Markov automaton always has a property that when a coalescence occurs, the chains stay together from that point on. It is important to notice that this grand coupling has coalesced at time t if and only if the random word $u_{1\to t}$ drawn is a synchronizing word of the automaton $\mathcal{A}$.

2.5 Bounding Interval Chains

We assume now that the space state $\mathcal{S}$ has a lattice order $\preceq$. This can be done with no loss of generality by using the Dedekind-MacNeille completion [4] of the state space. We introduce the following notions.

Definition 5. A *lattice interval* (or simply *interval*) is a set of the form $[m, M] := \{x \in \mathcal{S} \mid m \preceq x \preceq M\}$ for two given states m, M in $\mathcal{S}$. Notice that this interval is nonempty if and only if $m \preceq M$. We denote by $\mathcal{I}$ the set of all nonempty intervals on $\mathcal{S}$.

Definition 6. Given a grand coupling $\{(X_n(x))_{n\geq 0} \mid x \in \mathcal{S}\}$, we will call *bounding interval chain of that grand coupling* any Markov chain $([m_n, M_n])_{n\geq 0}$ of nonempty intervals of $\mathcal{S}$ such that for all x in $\mathcal{S}$ and all $n \geq 0$, $X_n(x) \in [m_n, M_n]$.

In particular, for any bounding interval chain $(m_n, M_n)_{n\geq 0}$, we have $[m_0, M_0] = [\bot, \top] = \mathcal{S}$, where $\bot := \inf \mathcal{S}$ and $\top := \sup \mathcal{S}$.

In the rest of the paper, we assume that we can construct an action $\odot$ by A on $\mathcal{I}$ verifying the two following properties:

($\mathcal{P}_1$) For all interval $[m, M]$, all state $x \in [m, M]$ and all letter a, we have $x \cdot a \in [m, M] \odot a$.

($\mathcal{P}_2$) If $[m, M]$ is an interval, a is a letter and if we denote $[m', M'] := [m, M] \odot a$, then $m = M$ implies $m' = M'$.

Thus, the Markov chain $([m_n, M_n])_{n\in\mathbb{N}}$, starting from the whole space $[m_0, M_0] = [\bot, \top] = \mathcal{S}$, and generated from some infinite random word $u \in A^\omega$ as $[m_n, M_n] = [\bot, \top] \odot u_{1\to n}$ is a bounding interval chain of the grand coupling $\{(X_n(x) := x \cdot u_{1\to n})_{n\in\mathbb{N}}) \mid x \in \mathcal{S}\}$ generated by $\mathcal{A}$ with the same random word u.

We show in [2] how to construct and compute efficiently such an action $\odot$ for a specific type of events which include the ones seen in the example of this paper. We will not give further details about this here.

Bounding interval chains were first introduced in [2]. They can be used to detect when a grand coupling has coalesced without having to compute each Markov chain of the grand coupling. This can be used among other things to perform perfect sampling using coupling from the past as shown in Section 4.2.

3. EVENT SKIPPING

Here, we introduce the concept of event skipping for the forward simulation of one Markov chain. We first give an informal presentation of the skipping method and then state the algorithm.

3.1 Informal Presentation

Let us consider a Markov automaton $\mathcal{A} = (\mathcal{S}, A, D, \cdot)$, generating a Markov chain $(X_n = x_0 \cdot u_{1\to n})_{n\in\mathbb{N}}$ from the random word u.

One can compute X_n inductively by generating the random letters of $u = u_1 \dots u_n \dots$ one by one and updating $X_{n+1} = X_n \cdot u_{n+1}$.

Now, let us assume u is of the form $u = va^k w$ with $v \in A^*$, $a \in A$ and $w \in A^\omega$, such that the state $y := x_0 \cdot v$ is invariant by a: $y \cdot a = y$. In that case, and when k is large, one might want to avoid unnecessary computations of $y \cdot a$ (k times), and even avoid drawing these k letters.

This can be done in the following manner. Let us say we have already generated the word v and computed $y = x_0 \cdot v$. We then draw a new letter according to D which happens to be an a, and by computing $y \cdot a$ notice that is is equal to y. All the consecutive a's we are going to draw will be passive, so we can directly draw the next non-a letter coming in the word with the probability D_a given by

$$\begin{aligned}\forall b \in A, \quad D_a(b) &= \mathbb{P}_D(X = b \mid X \neq a)\\ &= D(b)/(1 - D(a)).\end{aligned}$$

where X is a random letter of A with distribution D.

The word u is then of the form $u = va^k w$ where k is an unknown (and unused) random variable and w is an infinite random word, not drawn yet. Since we know that $x_0 \cdot va = x_0 \cdot v = y$, we have already computed $x_0 \cdot va^k = y$ for free.

Now, assume w is of the form $w = bw'$ for some letter b. We consider two cases:

If $y = x_0 \cdot va^k$ is not invariant by b: $z := y \cdot b \neq y$, then we attained a new state z and we can draw a's again since we do not know if z is invariant by a or not. Thus the next letter drawn will be drawn according to D.

On the other hand, if y is invariant by b: $y \cdot b = y$, we can apply the same reasoning to skip the longest sequence of b which is prefix of w'. But since we also know that y is invariant by both a and b, we can skip the longest prefix of w' in $(a + b)^*$.

Let us assume now that w' is of the form $v'cw''$ where v' is in $(a+b)^*$, *i.e.* v contains only a and b's, c is a letter distinct from a and b, and w'' is an infinite word not generated yet.

Since we know that $y \cdot a = y \cdot b = y$, we have $y \cdot v' = y$, and thus we only need to draw that letter c, which we can obtain by drawing the next non-a and non-b coming in our word with probability $D_{\{a,b\}}$ given by

$$\begin{aligned}\forall c \in A, \quad D_{\{a,b\}}(c) &= \mathbb{P}_D(X = c \mid X \neq a, X \neq b)\\ &= D(c)/(1 - D(a) - D(b)).\end{aligned}$$

If our state y is also invariant by c, we continue the same scheme by deactivating the letter c and drawing the next non-a, b, c letter. If y is not invariant by c, then we reactivate the letters a and b and continue by drawing all the letters with D.

3.2 Algorithm

We derive this into an algorithm for forward simulation of a Markov chain (X_n). We will need the following notations:

Definition 7. Given a Markov automaton $\mathcal{A} = (\mathcal{S}, A, D, \cdot)$ and an alphabet $B \subset A$, we denote by D_B the probability distribution of a random letter of A with distribution D conditionally to the fact that it is not in B. Thus for any letter $a \in A$,

$$D_B(a) = \begin{cases} 0 & \text{if } a \in B \\ D(a)/\left(1 - \sum_{b \in B} D(b)\right) & \text{if } a \in A \setminus B \end{cases}$$

The algorithm consists in updating a *disabling* alphabet B and draw letters with the distribution D_B. We start with B initially empty and $X = x_0$ the initial state of the chain. We then use the following iteration as long as we want to simulate our Markov chain:

- Draw a random letter a in A with distribution D_B:
  - if $X \cdot a = X$, then $B := B \cup \{a\}$
  - else, $X := X \cdot a$ and $B := \emptyset$.

If at some point we have $B = A$, it means that the state x reached is invariant by all events, and is thus an absorbing state. From now on, we will only consider ergodic Markov chains.

We will now apply the same method to perfect sampling of Markov chains using coupling from the past.

4. PERFECT SAMPLING

Consider an ergodic Markov chain $\mathcal{M}$ on a finite state space $\mathcal{S}$, with transition matrix P and stationary distribution π. Perfect sampling is a method to draw in finite time random sample exactly distributed according to the stationary distribution π. This section gives a brief introduction to the perfect sampling algorithm proposed by Propp and Wilson [10] and its extensions using bounding interval chains [3, 2].

4.1 Coupling from the Past

Take $\mathcal{A} = (\mathcal{S}, A, D, \cdot)$ a Markov automaton generating $\mathcal{M}$, and take an infinite random word $u = u_1 \dots u_n \dots$.

We assume that the automaton has a synchronizing word, *i.e.* there exists a word $v = v_1 \dots v_n$ such that the set $\{x \cdot v \mid x \in \mathcal{S}\}$ is a singleton. Then, almost surely, $\widetilde{v} := v_n \dots v_1$ is a factor of u (by the infinite monkey theorem), and thus almost surely, there exists a $N \in \mathbb{N}$ such that $u_{N \to 1}$ is synchronizing.

We define $\tau := \min\{n \mid u_{n \to 1} \text{ is synchronizing}\}$ the *backward coupling time* of the Markov automaton. It is easy to see (from the infinite monkey theorem again) that $\mathbb{E}(\tau) < \infty$. Then, since $u_{\tau \to 1}$ is synchronizing, the random variable $Y_\tau := x \cdot u_{\tau \to 1}$ does not depend on x.

THEOREM 1 (PROPP, WILSON [10]). *The random variable Y_τ is distributed with the stationary distribution π.*

This theorem remains true if we replace the coupling time τ by any stopping time τ' satisfying $\tau' > \tau$ almost surely.

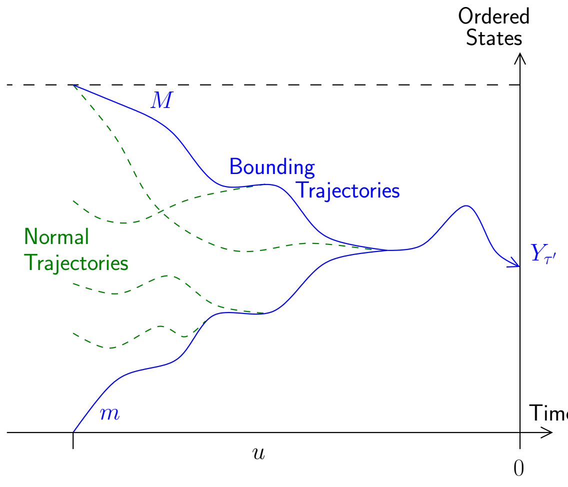

Figure 1: Bounding trajectories provide a perfect sample when they meet at time 0.

4.2 Coupling with Bounding Interval Chains

When the size of the state space $\mathcal{S}$ is large, fully computing the grand coupling of chains starting from each state of $\mathcal{S}$ might be impossible to achieve in practice. For this reason, we will use the bounding interval chains introduced earlier to detect if the grand coupling has coalesced. This is illustrated in Figure 1.

We keep the automaton $\mathcal{A}$ and random word $u \in A^\omega$ introduced previously, and we take an interval action $\odot$ by A on $\mathcal{I}$ verifying the two properties $(\mathcal{P}_1)$ and $(\mathcal{P}_2)$ as seen in Section 2.5. We assume that there exist a word v such that $\mathcal{S} \odot v$ is a singleton (we will say that v reduces $\mathcal{S}$ to a singleton). By $(\mathcal{P}_2)$, any word having v as a factor reduce $\mathcal{S}$ to a singleton. By $(\mathcal{P}_1)$, we also know that any word reducing $\mathcal{S}$ to a singleton synchronizes the automaton $\mathcal{A}$. By taking $\tau' = \min\{n \mid |\mathcal{S} \odot u_{n \to 1}| = 1\}$, we thus have that $\tau' > \tau$ almost surely and $\mathbb{E}(\tau') < \infty$. By Theorem 1, we have [3]:

PROPOSITION 2. *$Y_{\tau'} := x \cdot u_{\tau' \to 1}$ does not depend on x, is almost surely equal to the unique element of $\{\mathcal{S} \odot u_{\tau' \to 1}\}$, and is distributed with the stationary distribution π.*

The following Algorithm 1 draws in finite time a random variable exactly distributed along the stationary distribution π. Informally, the algorithm can be described as follows:

0) Generate a random word u of length 1.
1) If $\mathcal{S} \cdot u$ is not a singleton, then double the length of u by adding a random prefix to it and redo 1), if $\mathcal{S} \cdot u$ is a singleton, return that element.

The reason why we double the length of the word u is that computing $\mathcal{S} \cdot u$ takes a time linear in the length of u, thus adding just one letter to u and recomputing $\mathcal{S} \cdot u$ each time would make the computing time proportional to $\mathbb{E}(\tau'^2)$, while doubling the length of u makes it linear in $\mathbb{E}(\tau')$. This observation was already given in [10] for the monotone case.

Algorithm 1: Perfect Sampling with Bounding Intervals.

```
PSBI($\mathcal{A}$):
Data: $\mathcal{A} = (\mathcal{S}, A, D, \cdot)$ a Markov automaton. Rand($A, D$)
      draws a random letter of $A$ with distribution $D$.
Result: a state distributed along the stationary
        distribution $\pi$.
begin
    let $u := \varepsilon$
    /* $u$ is a word (we use a double chained list in
       practice)                                        */
    let $n := 1$
    while true do
        let $[m, M] := [\bot, \top]$
        let $v := \varepsilon$
        for $i = 1 \dots \lceil n/2 \rceil$ do
            $a :=$ Rand($A, D$)                    // Loop 1
            $v := va$
            $[m, M] := [m, M] \odot a$
        for $i = 1 \dots$ length($u$) do
            $[m, M] := [m, M] \odot u_i$           // Loop 2
        if $m = M$ then
            return $m$
        $u := vu$
        $n := 2n$
end
```

We give Algorithm 1 in a (seemingly) unnecessarily complicated way as this description will be useful to apply the skipping method to accelerate this algorithm. Indeed, it will be important to distinguish between the part where we double the size of the word u by adding to it a new random prefix v of same length (*Loop 1*), and the part where we read the word u previously generated (*Loop 2*).

5. SKIPPING IN PERFECT SAMPLING

We now present our main contribution, Algorithm 2, which improves Algorithm 1 by including the event skipping idea introduced in Section 3.

For convenience in this section, we will say that an event a is *passive* at the current time of the algorithm if $[m, M] \odot a = [m, M]$, and that it is *active* otherwise.

Algorithm 2 will be most effective for systems such as the one in Example 1, where one or multiple events occur much more often than others and are most of the time passive. In that case, one might want to skip such passive events in *Loop 1* of Algorithm 1.

However, these skipped events can be passive for the current interval $[m, M]$ and become active in a further iteration of the algorithm when we read them again in *Loop 2*. Indeed, if $[m', M'] \subset [m, M]$, then $[m, M] \odot a = [m, M]$ does not imply that $[m', M'] \odot a = [m', M']$. Therefore, we need a way to deactivate letters when they are passive in *Loop 1* and reactivate them if they become active again in *Loop 2*. Figure 2 shows an intuitive example where such precaution has to be taken.

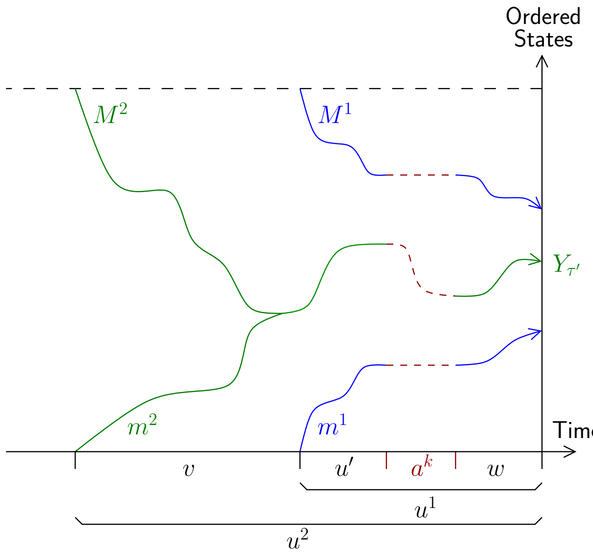

Figure 2: In this illustration, we see that, in the first iteration of the loop, the random word is of the form $u^1 = u'a^k w$, where a is a passive event for the current trajectory $[m^1, M^1]$ and can thus be skipped. But since $|\mathcal{S} \cdot u^1| \neq 1$, the algorithm restarts and adds a new prefix v to u^1 ($u^2 = vu^1$), so that the a^k factor may now become active for the new trajectory $[m^2, M^2]$.

5.1 Algorithm

The main idea of the method is to apply the skipping method as presented above in *Loop 1* and mark passive letters, then use this marking to expand the skipped part of the word when the letters become active.

Formally, if the event alphabet is $A = \{a, b, c, \dots\}$, we introduce a *marked* alphabet $\widehat{A} := \{\widehat{a}, \widehat{b}, \widehat{c}, \dots\}$. We will now code our generated word u with these two alphabets.

THEOREM 3. *Algorithm 2 generates almost surely a random variable distributed along the stationary distribution π.*

Algorithm 2 is decomposed into *Loop 1* and *Loop 2*, given respectively in Algorithms 3 and 4. The rest of the section is devoted to explaining the two loops and proving Theorem 3.

5.1.1 LOOP$_1$

In the first loop, given in Algorithm 3, we generate the new random prefix v and compute the trajectory $[\bot, \top] \odot v$, by marking the passive letters and skipping the repetition of these passive letters. For this we build an alphabet of *disabled* letters (called B^- in the algorithm) and draw letters conditionally to not being in that disabling alphabet. Each time we draw a passive letter a, we mark it ($v := v\widehat{a}$) and add it to this disabling alphabet ($B^- := B^- \cup \{a\}$), such that the repetition of this passive letter will not be drawn as long as the trajectory $[m, M]$ has not moved.

Algorithm 2: Perfect Sampling with Bounding Intervals and Skip.

```
PSBIS($\mathcal{A}$):
Data: $\mathcal{A} = (\mathcal{S}, A, D, \cdot)$ a Markov automaton.
Result: We obtain almost surely a random variable
        distributed along the stationary distribution $\pi$.
begin
    let $u = \varepsilon$    // $u$ is a word on the alphabet $A \cup \widehat{A}$
    let $n = 1$
    while true do
        let $[m, M] := [\bot, \top]$
        $([m, M], v) :=$ LOOP$_1$$([m, M], \lceil n/2 \rceil)$
        $([m, M], u) :=$ LOOP$_2$$([m, M], u)$
        if $m = M$ then
            return $m$
        $u := vu$
        $n := 2n$
end
```

Algorithm 3: LOOP$_1$.

```
LOOP$_1$$([m, M], k)$:
Data: $[m, M]$ interval, $k$ integer. Rand$(A, D)$ draws a
      random letter of $A$ with distribution $D$.
Result: Returns $([m_{out}, M_{out}], v)$, where the word $v$ is of
        length $k$, and $[m_{out}, M_{out}] = [m, M] \odot v$. Passive
        letters in $v$ are marked (except the last letter) and
        the repetitions of these passive letters are skipped.
begin
    let $B^- := \emptyset$    // $B^-$ is the disabling alphabet
    let $v := \varepsilon$
    for $i = 1 \dots k$ do
        let $a =$ Rand$(A, D_{B^-})$
        if $[m, M] = [m, M] \odot a$ & $i \neq k$ then
            // $a$ is passive
            $B^- := B^- \cup \{a\}$ ; $v := v\widehat{a}$
        else
            // $a$ is active
            $B^- := \emptyset$ ; $v := va$
            $[m, M] := [m, M] \odot a$
    return $([m, M], v)$
end
```

When an active letter is drawn, the trajectory moves again, and we do not know anymore which letters are passive or not. Thus the disabling alphabet has to be cleared.

Remark 1. We emphasize that the key ingredient to understand the algorithm (and in particular LOOP$_2$) is that when we draw a passive letter, it is marked and added to the disabling alphabet ($B^- := B^- \cup \{a\}$; $v := v\widehat{a}$), and when we draw an active one, it is not marked and the disabling alphabet is cleared ($B^- := \emptyset$; $v := va$).

For instance, if the output of LOOP$_1$ is $([m, M], v)$ with $v = u\widehat{a}\widehat{b}\widehat{c}w$, this means that letters a, b and c have been drawn but are all passive on the interval $[\bot, \top] \odot u$, thus for any word v' in $u(a + b + c)^*w$ we have $[\bot, \top] \odot v' = [\bot, \top] \odot uw$.

Algorithm 4: LOOP$_2$.

```
LOOP$_2$$([m, M], u)$:
Data: $[m, M]$ interval, $u$ finite word on $A \cup \widehat{A}$.
Rand$(A, D)$ draws a letter of $A$ with distribution $D$.
head$(u)$ and tail$(u)$ give respectively the first letter of $u$
and all the other letters of $u$: If $a =$ head$(u) \in A$ and
$w =$ tail$(u) \in A^*$, then $u = aw$.
Result: Returns $([m_{out}, M_{out}], w)$ where $[m_{out}, M_{out}] =$
        $[m, M] \odot w$ and where $w$ is the word $u$ where we
        re-inserted the repetition of active letters which
        were previously passive (some letters may remain
        passive in $w$, including the re-inserted ones).
begin
    let $B^+ := \emptyset$    // $B^+$ is the enabling alphabet
    let $w := \varepsilon$    // $w$ is a word
    while $u \neq \varepsilon$ do
        if head$(u) \in A$ then
            // head$(u)$ is not marked
            let $a :=$ head$(u)$
            let $[m, M] := [m, M] \odot a$
            $B^+ := \emptyset$ ; $w = wa$
        else
            // head$(u)$ is marked
            let $B^- := \emptyset$ // $B^-$ is a disabling alphabet
            let $\widehat{a} :=$ head$(u)$
            let $a$ be the corresponding non-marked letter
            $B^+ := B^+ \cup \{a\}$
            if $[m, M] = [m, M] \odot a$ then
                // $a$ is still passive
                $B^- := B^- \cup \{a\}$ ; $w := w\widehat{a}$
            else
                // $a$ is now active
                $B^- := \emptyset$ ; $w := wa$
                $[m, M] := [m, M] \odot a$
            while $b :=$ Rand$(A, D_{B^-}) \in B^+$ do
                if $[m, M] = [m, M] \odot b$ then
                    // $b$ is passive
                    $B^- := B^- \cup \{b\}$ ; $w := w\widehat{b}$
                else
                    // $b$ is active
                    $B^- := \emptyset$ ; $w := wb$
                    $[m, M] := [m, M] \odot b$
        $u :=$ tail$(u)$
    return $([m, M], w)$
end
```

5.1.2 Reconstruction

Before explaining the second loop, we show informally how we can, from the marked word u generated by LOOP$_1$, reconstruct the word we should have obtained without skipping any letters. To better illustrate this, we will call *true* word the word that we would have generated without skipping any letter, and *skipped* word, the word obtained by marking passive letters and skipping repetitions, such as in LOOP$_1$.

Let us consider a skipped word $u \in (A \cup \widehat{A})^*$. This word can be decomposed as a sequence of words alternatively in A^* and in $\widehat{A}^*$. We write

$$u = u_1 \widehat{u}_2 u_3 \dots \widehat{u}_{2n} u_{2n+1}$$

where, for all k, $u_{2k+1} \in A^*$ are unmarked factors and $\widehat{u}_{2k} \in \widehat{A}^*$ are marked factors, with $u_1 = \varepsilon$ or $u_{2n+1} = \varepsilon$ if necessary. The corresponding true word must be of the form

$$v = u_1 v_2 u_3 \ldots v_{2n} u_{2n+1},$$

with each v_{2k} in A^* having the unmarked version of $\widehat{u}_{2k}$ as a subword. By construction of u, we see that the marked factors $\widehat{u}_i$ contain each letter of $\widehat{A}$ at most once. Moreover, we can see by construction that if a marked factor is of the form $\widehat{u}_{2k} = \widehat{a}\widehat{b}\widehat{c}\ldots$, then the corresponding v_{2k} will be in $aa^*b(a+b)^*c(a+b+c)^*\ldots$; indeed in LOOP$_1$, each time a marked letter is added to the word u, it is also added to the disabling alphabet, and thus repetitions of that letter are skipped until the disabling alphabet is cleared, which happens when an unmarked letter is added to u. For simplicity, we assume we have $\widehat{u}_{2k} = \widehat{a}\widehat{b}$. Thus, to draw v_{2k}, we need to reinsert random words in a^* first and then in $(a+b)^*$ drawn with the distribution D. To obtain these we can simply draw words in A with the distribution D and keep the longest prefix in a^* and then in $(a+b)^*$. In other words if we want to get a word in $(a+b)^*$, we draw letters in A until we obtain a non-a-nor-b letter. We can thus obtain v from u with the following reconstruction algorithm:

Take $B^+ = \emptyset$ an alphabet (called *enabling* alphabet) and read the letters of u one by one:

- If the current letter is not marked, clear B^+ and read the next letter.
- If the current letter is marked, unmark it, add it to B^+ and insert new letters after the current one by drawing new letters in A according to D as long as they are in B^+. When a letter in $A \setminus B^+$ is drawn, drop it and read the next letter of u.

The enabling alphabet B^+ is in fact a reproduction of the disabling alphabet used to skip letters in the generation of u.

LEMMA 4. *The previous reconstruction algorithm generates a true word where each letter is i.i.d. with distribution D.*

PROOF. (sketch) We can see that the word finally obtained will have the right distribution with the following reasoning: Consider A and $B \subsetneq A$ two alphabets. Any word u has a unique decomposition as $u = vw$ with $v \in B^*$ its longest prefix in B^* and $w \in \varepsilon + (A \setminus B)A^*$. Thus we can draw a word $u \in A^*$ distributed along D by drawing first its longest prefix $v \in B^*$ and then $w \in \varepsilon + (A \setminus B)A^*$. We can obtain v by drawing letters with distribution D one by one and stop at the first letter not in B, dropping that last letter. Assuming $w \neq \varepsilon$ we can draw it by drawing a letter in A conditionally to not being in B, and then draw letters in A normally. By doing so we will obtain a word whose letters are independent and identically distributed along D. Also, v and w can be drawn completely independently, and in particular we can draw w and then later draw v. This is exactly what we do by skipping letters first and reinserting them later. $\square$

5.1.3 LOOP$_2$

The second loop is described in Algorithm 4. In this loop, we read the previously generated word u, apply it to our trajectory $[m, M]$, and reinsert skipped events if they are active again. However, the reconstruction of the true word may not be complete since some reinserted events can become passive again so that they are skipped once more. The second loop will thus combine the reconstruction idea described in the previous paragraph and the skipping method of LOOP$_1$ applied to the reinsertion of letters.

For this, we use two auxiliary alphabets. An *enabling* alphabet B^+ which will be used for the reconstruction, and will reproduce the disabling alphabet used in past iterations of LOOP$_1$ (and LOOP$_2$), and a new *disabling* alphabet B^- which will allow us to skip passive events during the re-insertions.

For instance, let us say that at some iteration of the loop, we have $u = aw$ with $a \in A$ an unmarked letter, and $w \in (A \cup \widehat{A})^*$. Then we just read that letter and update the trajectory $[m, M] := [m, M] \odot a$. If we have instead $u = \widehat{a}w$ with $\widehat{a} \in \widehat{A}$ a marked letter (and a its corresponding unmarked letter), then we add the letter a to the enabling alphabet B^+, and we compute $[m, M] \odot a$. Two distinct cases may then occur.

In the first case, we still have $[m, M] \odot a = [m, M]$. Then a is still passive, we let it marked ($w := w\widehat{a}$) and we add it to the disabling alphabet ($B^- := B^- \cup \{a\}$).

In the other case, a is active again: $[m, M] \odot a \neq [m, M]$. In that case we need to reinsert the skipped repetitions of a. For this we start inserting new letters with distribution D_{B^-} as long as they are in B^+, and we stop at the first drawn letter not in B^+ (this is done in the **while** loop). However some of these inserted letters might be passive as well and we thus skip them. For this we use the same method as before: If the new inserted letter is passive, we mark it and add it to the disabling alphabet B^-, if not we leave it unmarked and clear the disabling set B^-.

5.2 Implementation Issues

The goal of this section is to explain how the algorithms can be efficiently implemented. It also gives some insights on the overhead induced by the skipping method. The words used in these algorithm can be represented using double-chained lists.

The several subroutines used in both algorithms are detailed below:

- The routines `tail` and `head` are easy to implement for double-chained lists with a constant time complexity.
- The computation of the action of one event over an interval: $[m, M] \odot a$ is done using a different function for each event. It has been shown in [2] that for most events in queueing networks, this function can be computed in $O(\log(S))$ elementary operations.
- As for the random generation of events, Rand(A, D_{B^-}), for each new set B^- created by LOOP$_1$ or LOOP$_2$, an alias table is constructed to generate samples under distribution D_{B^-} (see [11] for the alias method for sampling

from an arbitrary discrete probability distribution with a finite number of outcomes). The time needed to construct the alias table is in $O(|A|)$. The generation of one event is done in constant time once the alias table has been built. The alias tables are stored in a hash table so that whenever a set B^- has to be used again, the alias does not need to be reconstructed.

6. NUMERICAL EXPERIMENTS

We first apply our algorithm to Example 1: two queues in tandem with service rate $\mu_1 = \mu_2 = 10$ and arrival rate λ in the first queue.

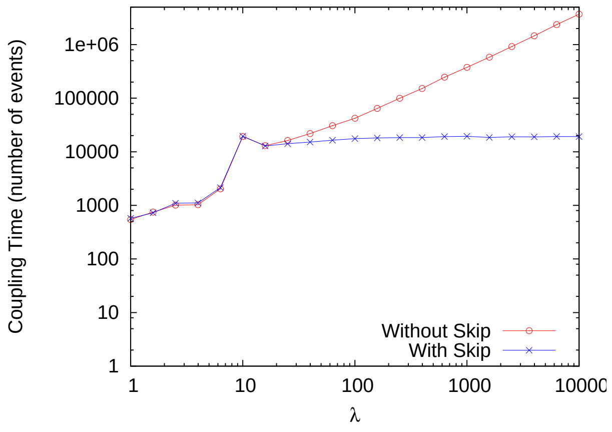

Figure 3: Average coupling time with and without the skipping method on Example 1 (two queues in tandem).

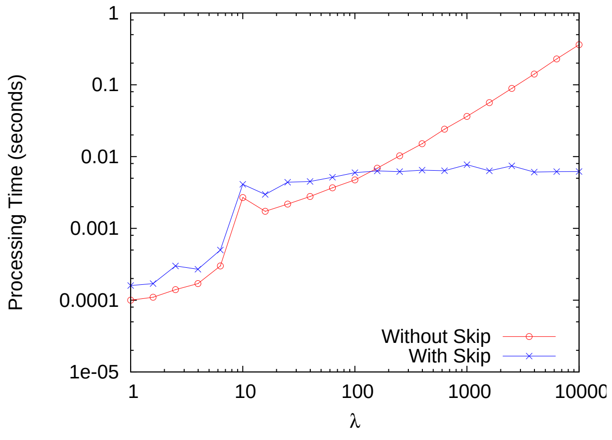

Figure 4: Average processing time with and without the skipping method on Example 1 (two queues in tandem).

Figure 3 shows a comparison of the number of events generated by the algorithm to get one sample, with and without the skipping method, for λ increasing exponentially from 1 to 10000. As one can see, without the skipping method, the log of the number of events is asymptotically linear in $\log \lambda$ while with our method the number of events remains bounded. The small pike at $\lambda = 10$ comes from the fact that at this value the system is in a critical state, which makes the coupling time longer.

Of course, the implementation of the skipping method has a cost. In Figure 4 we compare the average computing time of one sample with and without the skipping method. The skipping method outperforms the classical one when λ is larger than 20 times the service time, with the same asymptotic behavior as for the number of generated events.

Sensitivity to Entropy. Here is a guess on what makes the skipping method efficient: Whenever an event occurs with high probability, many of its occurrences should actually be passive, so that the efficiency of the skipping method should increase when the probabilities of some events are high. To test this conjecture, we computed the ratio of the runtime of the skipping method with the runtime without skipping against the entropy of the event distribution. Indeed, the entropy being a measure of the dispersion of the probabilities, the worst case for the skipping method should be when the entropy is maximal (all the probabilities are equal) and the best case when the entropy goes to 0 (one probability goes to 1).

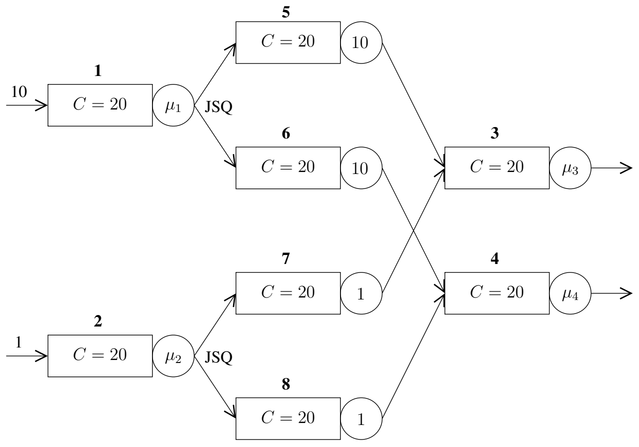

Figure 5: Network used to generate Figure 6.

For that, we consider the queueing network given in Figure 5. Servers 1 and 2 receive clients at rate 10 and 1, respectively. The service rates μ_1, μ_2, μ_3 and μ_4 are kept as parameters while the other services rates are fixed as indicated in the figure. The clients served by Server 1 are sent to Server 5 or 6 according to a *Join the Shortest Queue* (*JSQ*) routing policy, and similarly, between Server 2 and Servers 7 and 8. All queue have capacity $C = 20$. To test the efficiency of our skipping method, we will choose high values for μ_1, μ_2, μ_3 and μ_4, so that the system stays in the stable regime.

Table 1 gives the values taken by μ_1, μ_2, μ_3 and μ_4 up to a permutation, and the corresponding partial entropy $H = -p_1 \log_2 p_1 - \cdots - p_4 \log_2 p_4$ with $p_i = \mu_i/(\mu_1 + \cdots + \mu_4)$. Note that the sum $\mu_1 + \mu_2 + \mu_3 + \mu_4$ remains constant, so that the partial entropy is always equal to the global entropy up a multiplicative constant.

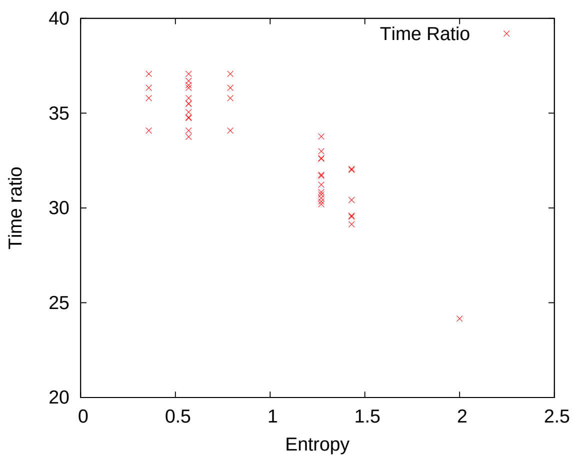

Figure 6: Time ratio in number of event between the algorithm with and without the skipping method, for simulating the network displayed on Figure 5, with the parameters given in Table 1.

Table 1: Several values of service rates and the corresponding entropy, used in Figure 6.

μ_1	μ_2	μ_3	μ_4	H
5500	5500	5500	5500	2
10000	10000	1000	1000	1.43
15000	5000	1000	1000	1.27
19000	1000	1000	1000	0.79
20000	1000	500	500	0.57
21400	200	200	200	0.36

Figure 6 gives the ratio (in terms of the number of events generated) between the algorithm without and with the skipping method against the entropy of the system. The points with the same entropy correspond to all permutations of the values of μ_1, μ_2, μ_3 and μ_4. This ratio is between 20 and 40 and increases when the entropy decreases (at least up to a certain point).

7. CONCLUSION

In this paper, we show that perfect simulation of Markov chains can benefit from an ad-hoc generation of the events driving the simulation: We designed an algorithm where the events are only generated when they have an effect on the current set of trajectories being simulated. Furthermore, we show that the simulation time remains bounded when the rate of some events goes to infinity (implying that the spectral gap of the Markov chain goes to 0).

The next step is to combine the skipping method with other improvements of perfect simulation such as splitting [3] and to improve its implementation by designing a compact and efficient data structure for the partially generated sequence of events.

8. REFERENCES

[1] J. Anselmi and B. Gaujal. Coupling time in markovian queueing networks, 2010. Submitted.

[2] A. Busic, B. Gaujal, and F. Pin. Perfect sampling of Markov chains with piecewise homogeneous events, 2010. Submitted. Preprint arXiv:1012.2910.

[3] A. Busic, B. Gaujal, and J.-M. Vincent. Perfect simulation and non-monotone markovian systems. In *Valuetools'08*, Athens, Grece, 2008.

[4] B. Davey and H. Priestley. *Introduction to lattices and orders*. Cambridge University Press, 1991.

[5] J. Dopper, B. Gaujal, and J.-M. Vincent. Bounds for the coupling time in queueing networks perfect simulation. In *Numerical Solutions for Markov Chains (NSMC'06)*, pages 117–136, Charleston, 2006. The 2006 A.A. Markov Anniversary Meeting (MAM 2006).

[6] M. Granovetter. The Strength of Weak Ties. *The American Journal of Sociology*, 78(6):1360–1380, 1973.

[7] M. Huber. Perfect sampling using bounding chains. *Ann. Appl. Probab.*, 14(2):734–753, 2004.

[8] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov chains and mixing times*. American Mathematical Society, Providence, RI, 2009. With a chapter by James G. Propp and David B. Wilson.

[9] A. Mislove, M. Marcon, K. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proc. of the IMC*, 2007.

[10] J. G. Propp and D. B. Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 9(1-2):223–252, 1996.

[11] A. J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Trans. Math. Softw.*, 3:253–256, September 1977.