

Clones: CLOsed queueing Networks Exact Sampling

Anne Bouillard, Ana Bušić, Christelle Rovetta

▶ **To cite this version:**

Anne Bouillard, Ana Bušić, Christelle Rovetta. Clones: CLOsed queueing Networks Exact Sampling. ValueTools, Dec 2014, Bratislava, Slovakia. hal-01095314

HAL Id: hal-01095314
https://hal.science/hal-01095314

Submitted on 15 Dec 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Clones: CLOsed queueing Networks Exact Sampling

Anne Bouillard
ENS Paris
45 rue d'Ulm
75005 Paris, France
anne.bouillard@ens.fr

Ana Bušić
Inria
23 avenue d'Italie
75013 Paris, France
ana.busic@inria.fr

Christelle Rovetta
Inria
23 avenue d'Italie
75013 Paris, France
christelle.rovetta@inria.fr

ABSTRACT

We present `Clones`, a Matlab toolbox for exact sampling from the stationary distribution of a closed queueing network with finite capacities. This toolbox is based on recent results using a compact representation of sets of states that enables exact sampling from the stationary distribution without considering all initial conditions in the coupling from the past (CFTP) scheme. This representation reduces the complexity of the one-step transition in the CFTP algorithm to $O(KM^2)$, where K is the number of queues and M the total number of customers; while the cardinality of the state space is exponential in the number of queues. In this paper, we focus on the algorithmic and implementation issues. We propose a new representation, that leads to one-step transition complexity of the CFTP algorithm that is in $O(KM)$. We provide a detailed description of our matrix-based implementation. The toolbox can be downloaded at `http://www.di.ens.fr/~rovetta/Clones`.

Keywords

queueing networks, simulation, coupling from the past

1. INTRODUCTION

Closed queueing networks are largely used in various application domains due to their modeling simplicity and product-form stationary distribution in the unlimited capacity case [5]. This structure is no longer guaranteed when the queues have a finite capacity, due to the effect of *blocking*. There are different types of blocking, see [8] for a detailed discussion. According to the terminology introduced in that survey, our blocking model is RS-RD (*repetitive service – random destination*). Under this blocking policy, the stationary distribution has a product form only if the routing is reversible. When the stationary distribution does not have a product-form, the exact analysis may not be computationally tractable, and we may turn to approximations [2] or simulation. The main difficulty of the Markov chain Monte Carlo approach is the stopping criterion. Usual tools are the asymptotic variance in the Central Limit Theorem or mixing times [1, 7], unfortunately there is no generic technique for the non-reversible Markov chain case.

In the 1990's, Propp and Wilson introduced a method for sampling a random variable according to the stationary distribution of a finite ergodic Markov chain [9]: the coupling from the past (CFTP) algorithm. The CFTP algorithm automatically detects and stops when the sample has the correct distribution. The main drawback of the original CFTP is that it considers a coupling from all initial conditions. In the case of closed queueing networks the cardinality of the state space is exponential in the number of queues, which is untractable.

When the network has a product-form distribution, Kijima and Matsui [6] proposed a perfect sampling algorithm with overall complexity $O(K^3 \ln(KM))$. However, their method strongly relies on the product form representation of the stationary distribution and it cannot be applied to the general case of queues with finite capacity. In a recent paper by Bouillard and al. [3], a new representation of the sets of states has been proposed. This representation is used to derive a bounding chain for the CFTP algorithm for closed queueing networks, that enables exact sampling from the stationary distribution without considering all initial conditions in the CFTP. This method is far more general, as it does not rely on the product-form property.

The complexity of one-step transition of the CFTP algorithm in [3] is $O(KM^2)$. In this paper, we derive a new algorithm, based on even more compact representation. The complexity of one-step transition of this new CFTP algorithm is $O(KM)$. Note that this is the same as the complexity of the computation of the normalizing constant in the product-form case using Buzen's algorithm [4]. The overall complexity of our algorithm depends however also on the coupling time of the chain, so it becomes of a practical interest when the network does not have a product form. The main focus of this paper is on algorithmic and implementation issues. We give a detailed description of the matrix-based representations used in our Matlab toolbox `Clones`.

The paper is organized as follows. In Section 2 we describe the model and give a short overview of results in [3]. We then introduce a new *gap-free diagram* representation and a new CFTP algorithm with one-step transition complexity of $O(KM)$. In Section 3 we present our Matlab toolbox `Clones`

and discuss its functionalities and implementation issues. Section 4 is devoted to numerical results. Final remarks and conclusions are contained in Section 5.

2. BACKGROUND

2.1 Model

Consider a closed network of K $./M/1/C$ queues with M customers in total. Each queue $k \in \{1, \dots, K\} =: Q$ has a finite capacity $C_k \leq M$ and a service rate ν_k.

After a customer has been served in queue i, it is directed to queue j with probability $p_{i,j}$, independently of the current state and past evolution of the network. If queue j is full, then the customer remains in queue i and has to be served again. Its new destination will be chosen independently from j. We assume that the routing matrix $P = (p_{i,j})_{i,j \in Q}$ is stochastic and irreducible (that is, the network of queues is strongly connected). We set $R = \{(i,j) \mid p_{i,j} > 0\}$.

The state space is denoted by $\mathcal{S}$: it is the set of elements $x = (x_1, x_2, \dots, x_K) \in \mathbb{N}^K$ such that $\sum_{k=1}^{K} x_k = M$ and $\forall k \in Q,\ 0 \leq x_k \leq C_k$. We know that $|\mathcal{S}| \leq \binom{K+M-1}{K-1}$, which grows exponentially with K.

For $(i,j) \in R$, $t_{i,j} : \mathcal{S} \to \mathcal{S}$ is the function that describes the routing of a customers from queue i to queue j:

$$t_{i,j}(x) = x + (e_j - e_i)\mathbf{1}_{x_i > 0 \text{ and } x_j < C_j}$$

where e_i is the vector that has all its coefficients equal to 0 except the i-th, equal to 1 and $\mathbf{1}_A$ is the characteristic function of A. This network can be described by an ergodic Markov chain: at each step, a pair (i,j) is randomly chosen, with probability $\frac{\nu_i p_{i,j}}{\sum_{k \in K} \nu_k}$. Our objective is to sample the stationary distribution with perfect sampling techniques (as in [9]). Algorithm 1 (PSS) is the direct application of the perfect sampling algorithm to this model. It produces a sample of the stationary distribution in finite expected time.

Algorithm 1: Perfect sampling using sets of state (PSS)

Data: $(U_{-n} = (i_{-n}, j_{-n}))_{n \in \mathbb{N}}$ an i.i.d sequence of r.v
Result: $x \in \mathcal{S}$

```
begin
    n ← 1;
    t ← t_{U_{-1}};
    while |t(S)| ≠ 1 do
        n ← 2n;
        t ← t_{U_{-1}} ∘ ··· ∘ t_{U_{-n}};
    return x, the unique element of {t(S)}.
```

This algorithm has a complexity at least linear in $|\mathcal{S}|$, which grows exponentially with K. In [3] a new and compact representation of the state space has been proposed, for which sets of states have a strucured representation, polynomial in K and M. The main idea is to represent states as paths in a directed graph called *diagram*. In the next subsection, we give a summary of results in [3].

2.2 Diagram

Let $D = (N, A)$ be a directed graph where $N = \{0, \dots, K\} \times \{0, \dots, M\}$ and $g : \mathcal{S} \to \mathcal{P}(N^2)$ denote the function which associates a set of arcs to each state:

$$g(x) = \bigcup_{i=1}^{K} \Big\{ \big((i-1, \sum_{k=1}^{i-1} x_k), (i, \sum_{k=1}^{i} x_k)\big) \Big\}.$$

These arcs form a path from node $(0,0)$ to (K,M). The directed graph $D = (N, A)$ is called *diagram* if there exists $S \subseteq \mathcal{S}$ such that $A = g(S) := \bigcup_{s \in \mathcal{S}} g(s)$.

The *complete diagram* is $\mathcal{D} = (N, \mathcal{A})$ where $\mathcal{A} = g(\mathcal{S})$, the image of the state space. An example of such a diagram is given in Figure 1. Note that $|A| \leq |\mathcal{A}| \leq \frac{K(M+2)(M+1)}{2}$.

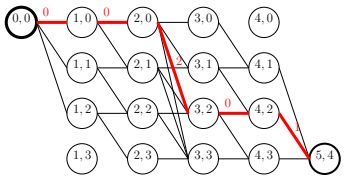


Figure 1: Complete diagram with $K = 5$, $M = 3$ and $C = (2,1,3,1,2)$. The set of bold (and red) arcs is the image of state $x = (0,0,2,0,1)$.

Let $a = \big((k-1, m), (k, m')\big) \in A$. The value of a is $v(a) = m' - m$ and represents the number of customers in queue k. Graphically, the value of an arc corresponds to its slope. Note that $0 \leq v(a) \leq C_k$.

Function g can be used to define the transformation from a set of states to a diagram and conversely: for a diagram $D = (N, A)$ and a set $S \subseteq \mathcal{S}$, we have

$$\phi(S) = (N, g(S)) \qquad \text{and} \qquad \psi(D) = g^{-1}(A).$$

We can now define the transformation $T_{i,j}$ corresponding to the service of a customer from queue i to queue j: for $(i,j) \in R$,

$$T_{i,j}(D) = \phi \circ t_{i,j} \circ \psi(D).$$

The transformation $T_{i,j}$ can be performed directly without having to compute ψ, $t_{i,j}$ and ϕ. It will be shown in Section 3 that $T_{i,j}(D)$ can be computed in time polynomial in $O(KM^2)$ (while computing $t_{i,j}$ for all states, as in PSS, has exponential complexity).

THEOREM 1. *PSD algortihm terminates in finite expected time and produces an exact sample from the stationary distribution.*

2.3 Gap-free diagram

In this subsection, we present a sub-class of diagrams that can be represented even more compactly. A diagram $D = (N, A)$ is called *gap-free* if $\forall k \in Q$, $\forall s \leq C_k$, $\forall m_1 < m_2 <$

$m_3 \leq M$, $\forall m \leq M$, $\forall s_1 < s_2 < s_3 \leq C_k$,

- $\big((k-1,m_1),(k,m_1+s)\big), \big((k-1,m_3),(k,m_3+s)\big) \in A$
  $\Rightarrow \big((k-1,m_2),(k,m_2+s)\big) \in A$;
- $\big((k-1,m),(k,m+s_1)\big), \big((k-1,m),(k,m+s_3)\big) \in A$
  $\Rightarrow \big((k-1,m),(k,m+s_2)\big) \in A$;
- $\big((k-1,m-s_3),(k,m)\big), \big((k-1,m-s_1),(k,m)\big) \in A$
  $\Rightarrow \big((k-1,m-s_2),(k,m)\big) \in A$.

The second and third implications state that the values of arcs from (to) a node are contiguous. It can be easily seen that the complete diagram is gap-free and that gap-free diagrams have a more compact representation: it suffices to know for each value s and each queue k what are the minimal and maximal arc (from the first implication). Consequently, it can be represented by less than $2K(M+1)$ arcs.

In most cases, if D is gap-free, $T_{i,j}(D)$ is also gap-free. But it is not true in general, as shown on Figure 8. For a diagram D, we denote D^{gf} the smallest gap-free diagram (for arc inclusion) that contains D. This construction is illustrated on Figure 2. From this transformation, we can define a new transition function on gap-free diagrams: for $(i,j) \in R$, $T^{\text{gf}}_{i,j}(D) = [T_{i,j}(D)]^{\text{gf}}$.

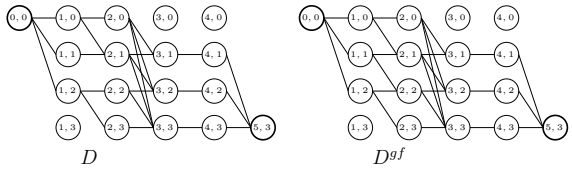


Figure 2: Transformation of a diagram with a gap into a gap-free diagram. Left: D has a gap (no arc $((1,1),(2,2))$; Right: D^{gf} is constructed by adding this arc to D.

The perfect sampling algorithm can be adapted to gap-free diagrams. It suffices to replace every occurrences of T in Algorithm 2 by T^{gf}. This is PSF algortihm.

Algorithm 2: Perfect sampling using diagrams (PSD)

Data: $(U_{-n} = (i_{-n}, j_{-n}))_{n\in\mathbb{N}}$ an i.i.d sequence of r.v
Result: $x \in \mathcal{S}$

```
begin
    n ← 1;
    T ← T_{U_{-1}};
    while |ψ(T(D))| ≠ 1 do
        n ← 2n;
        T ← T_{U_{-1}} ∘ ··· ∘ T_{U_{-n}};
    return x, the unique element of {ψ(T(D))}.
```

THEOREM 2. *PSF algorithm terminates in finite expected time and produces a sample distributed according to the stationary distribution.*

PROOF SKETCH. The algorithm terminates in finite expected time for the same reason as Algorithm 2: the coupling sequence exhibited in [3] produces gap-free diagrams only.

Let $(U_n)_{n=1}^N \in R^N$. As $T_{U_1} \circ \cdots \circ T_{U_N}(\mathcal{D}) \subseteq T^{\text{gf}}_{U_1} \circ \cdots \circ T^{\text{gf}}_{U_N}(\mathcal{D})$, and as diagrams representing one state are gap-free, we are ensured that the result of the algorithm is distributed as the stationary distribution. □

3. TOOLBOX IMPLEMENTATION

This section is devoted to the implementation of the transition function of diagrams. As we will see, it is convenient to represent diagrams with matrices. Then a natural choice for implementation is Matlab, as it handles matrix computations (specially when they are sparse) very well. The toolbox `Clones` is a Matlab toolbox that implements perfect sampling algorithms using different state space representations: state space, diagrams and gap-free diagrams. It is available at `http://www.di.ens.fr/~rovetta/Clones`. At the same address there is also a detailed documentation on how to use the Toolbox. In this section, we first introduce the matrix representation of diagrams, then present the transition algorithms and finally describe some additional functionalities of the toolbox.

3.1 Matrix representations

3.1.1 Representation of diagrams

Let $D = (N, A)$ be a diagram and $k \in Q$. In the toolbox, diagrams are implemented with incidence matrices. More precisely, $\mathtt{D_k}$ is a Boolean matrix of size $(M+1) \times (M+1)$ representing the k-th column of D:

$$\mathtt{D_k}(m, m') = \mathbf{1}_{((k-1,m),(k,m'))\in A}.$$

Because $a \in A$ implies that $0 \leq v(a) = m - m' \leq C_k$, matrices $\mathtt{D_k}$ are Boolean upper-triangular and $\mathtt{D_k}(m,m') = 0$ if $m' - m > C_k$. Figure 3 gives an example.

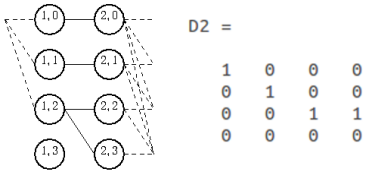


Figure 3: Matrix representation $\mathtt{D}_2$ of the second column of D.

Let $i, j \in Q$. We will denote by $\mathtt{D_{i,j}}$ the matrix of size $(M+1) \times (|i-j|+1)(M+1)$ that is the concatenation of matrices $\mathtt{D_i}, \ldots, \mathtt{D_j}$:

$$\mathtt{D_{i,j}} = \begin{cases} \mathtt{D_i}, \mathtt{D_{i+1}}, \ldots, \mathtt{D_j} & \text{if } i < j \\ \mathtt{D_j}, \ldots, \mathtt{D_{i-1}}, \mathtt{D_i} & \text{if } i > j \\ \mathtt{D_i} & \text{if } i = j. \end{cases}$$

A diagram $D = (N, A)$ is represented by the matrix $\mathtt{D} = \mathtt{D_{1,K}}$. Figure 4 shows the matrix representation $\mathtt{D}$ of a complete diagram with $K = 5$, $M = 3$ and $C = (2, 1, 3, 1, 2)$. It is composed of 5 Boolean matrices of size 4×4. Note that due to the shape of a diagram, only the first line of

$\mathtt{D}_1$ is non-null and only the last column of $\mathtt{D}_K$ is non-null. The computation of this matrix is implemented by function `Clones_CompleteD(C,M)`.

```
D =

  1  1  1  0 | 1  1  0  0 | 1  1  1  1 | 0  1  0  0 | 0  0  0  0
  0  0  0  0 | 0  1  1  0 | 0  1  1  1 | 0  1  1  0 | 0  0  0  1
  0  0  0  0 | 0  0  1  1 | 0  0  1  1 | 0  0  1  1 | 0  0  0  1
  0  0  0  0 | 0  0  0  0 | 0  0  0  1 | 0  0  0  1 | 0  0  0  1
```

Figure 4: Diagram representation of the complete diagram in Figure 1.

3.1.2 *Representation of gap-free diagrams*

Gap-free diagrams can be more efficiently implemented. Because of the implication $\forall k \in Q,\ \forall s \leq C_k,\ \forall m_1 < m_2 < m_3 \leq M,\ \big((k-1,m_1),(k,m_1+s)\big), \big((k-1,m_3),(k,m_3+s)\big) \in A \Rightarrow \big((k-1,m_2),(k,m_2+s)\big) \in A$, each column can be represented by a $2 \times (C_k+1)$ matrix $\mathtt{F}_k$:

- $\mathtt{F}_k(1,s) = \min(m,\ \big((k-1,m),(k,m+s)\big) \in A)$
- $\mathtt{F}_k(2,s) = \max(m,\ \big((k-1,m),(k,m+s)\big) \in A)$
- with the convention that $\mathtt{F}_k(1,s) = \mathtt{F}_k(2,s) = -1$ if $\{m,\ \big((k-1,m),(k,m+s)\big) \in A\} = \emptyset$.

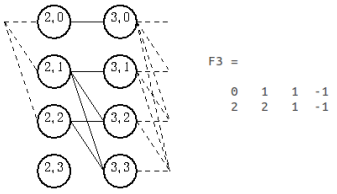


Figure 5: Gap-free representation of a column.

The gap-free representation of the diagram of Figure 1 is given in Figure 5.

Using the notations defined for $\mathtt{D}$, a gap-free diagram is represented by matrix $\mathtt{F} = \mathtt{F}_{1,K}$ of size $2 \times (\sum_{k=1}^{K} C_k + K)$. Figure 6 shows $\mathtt{F}$, the gap-free representation of the complete diagram of Figure 1. It is a 2×14-matrix.

```
F =

  0  0  0 | 0  0 | 0  0  0  0 | 1  0 | 3  2  1
  0  0  0 | 2  2 | 3  2  1  0 | 3  2 | 3  2  1
```

Figure 6: Gap-free representation of the complete diagram in Figure 1.

Function `Clones_CompleteF(C,M)` computes the gap-free representation of a complete diagram and `Clones_FtoD(F,C,M)` returns the diagram matrix which corresponds to the gap-free matrix $\mathtt{F}$.

3.2 Diagram transition algorithms

In this section, we present the main algorithm of the `Clones` toolbox: the transition on diagrams. First, we need to find some classes of paths in the diagram: those that will remain unchanged by the transition and those that will evolve. In the whole section we denote by $\mathtt{D}$ the diagram matrix representation of diagram $D = (N, A)$ and describe the transition $T_{i,j}(D)$.

3.2.1 *Finding paths in a diagram*

To compute $T_{i,j}(D)$, we need to know what are the paths of D that will remain unchanged because queue i is empty or queue j is full, and what are the paths that will be modified. These paths can be determine only by the knowledge of their arc value at the i-th and j-th columns. So, we need to define matrices $\mathtt{D}^{\mathtt{V}}_{\mathtt{i}\to\mathtt{j}}$ that represents the paths of D from columns i to j whose arcs at column i have value in V. It is constructed as follows.

Let $V \subseteq \{0,1,\ldots,C_i\}$ be a set of values, $\mathtt{D}^{\mathtt{V}}_{\mathtt{i}}$ is the matrix that represents arcs in the column i with values in V:

$$\mathtt{D}^{\mathtt{V}}_{\mathtt{i}}(m,m') = \mathtt{D}_{\mathtt{i}}(m,m')\mathbf{1}_{m'-m \in V}.$$

Paths from column i to column j generated by arcs with value in V in column i are represented by the matrix $\mathtt{D}^{\mathtt{V}}_{\mathtt{i}\to\mathtt{j}} = [Y_1, Y_2, \ldots, Y_{|i-j|+1}]$. The matrices Y_k are computed as:

Case	Y_1	Y_k	$Y_{\lvert i-j\rvert+1}$
$i < j$	$\mathtt{D}^{\mathtt{V}}_{\mathtt{i}}$	$\mathtt{diag}(\mathbb{1}_{M+1}Y_{k-1})\mathtt{D}_{\mathtt{k}}$	
$i > j$	$\mathtt{D}_{\mathtt{k}}\mathtt{diag}(Y_{k+1}\mathbb{1}^t_{M+1})$		$\mathtt{D}^{\mathtt{V}}_{\mathtt{j}}$

where $\forall v \in \{0,1\}^N$, $\mathtt{diag}(v)(m,m') = v(m)\mathbf{1}_{m=m'}$.

Let $\mathbb{1}_M$ be the identity matrix of size M. Note that as $\mathtt{diag}(v)$ is a diagonal Boolean matrix, computing $\mathtt{diag}(\mathbb{1}_{M+1}Y_{k-1})\mathtt{D}_{\mathtt{k}}$ does not require matrix multiplication. It boils down to copying the lines of $\mathtt{D}_{\mathtt{k}}$ corresponding to non-null elements of the diagonal of $\mathtt{diag}(\mathbb{1}_{M+1}Y_{k-1})$. The most complex operation is to compute $\mathbb{1}_{M+1}Y_{k-1}$, which can be done in $O(M^2)$. As $|j-i|+1 \leq K$, the complexity of finding those path is in $O(KM^2)$. Figure 7 depicts $\mathtt{D}^{\{2\}}_{3\to5}$ on the diagram.

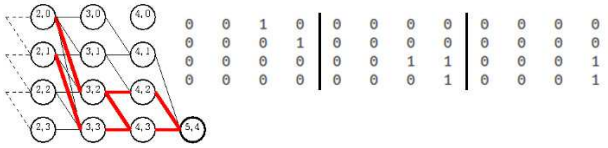


Figure 7: Diagram D and $\mathtt{D}^{\{2\}}_{3\to5}$ in bold (red) paths. It corresponds to paths from column 3 to column 5 generated by arcs of values 2.

3.2.2 *Transition algorithm*

We are now ready to present the transition algorithm. Let D be a diagram and $(i,j) \in R$. We explain how to compute $T_{i,j}(D)$. First let us explain the transformation on diagram D (see Figure 8). We begin by computing three sets of arcs (not necessarily disjoint), depending on whether they will be modified by the transition:

- $\mathcal{E}mpty$ is the set of arcs that belong to paths with arc value 0 at column i. Those paths correspond to states where the queue i is empty.
- $\mathcal{F}ull$ is the set of arcs that belong to paths with arc value C_j at column j. Those paths correspond to states where the queue j is full.
- $\mathcal{T}ransit$ is the set of arcs that belong to paths with value $< C_j$ at column j and > 0 at column i. Those paths correspond to states that are modified by $t_{i,j}$.

Second, arcs in $\mathcal{T}ransit$ are modified in accordance to $t_{i,j}$:

- In column i, value of arcs are decreased by one (keeping the same source).
- In column $k \in \{\min(i,j)+1, \ldots, \max(i,j)-1\}$, arcs are shifted up or down depending on the sign of $i-j$.
- In column j, slopes of arcs are increases by one (keeping the same destination).

If we call $\mathcal{T}ransit'$ this new set, $T_{i,j}(D) = (N, \mathcal{E}mpty \cup \mathcal{F}ull \cup \mathcal{T}ransit')$.

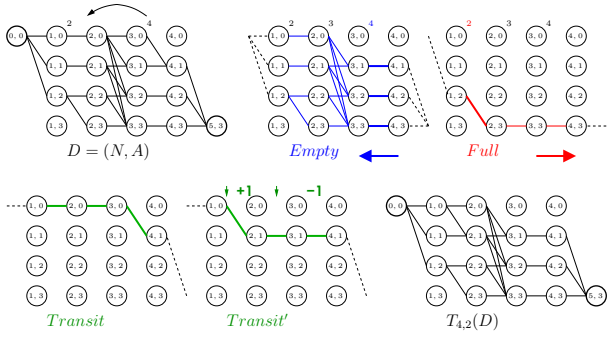


Figure 8: Steps for computing $T_{4,2}(D)$. Diagram D is obtained from the complete diagram by some transition; $D = T_{4,5} \circ T_{4,5} \circ T_{2,1}(\mathcal{D})$.

The transformation using matrix operations is given in Algorithm 3, where the Boolean operators AND and OR are component-wise and where

$$[\leftarrow X]_{m,m'} = X_{m,m'+1}\mathbf{1}_{m'<M}, \quad [\rightarrow X]_{m,m'} = X_{m,m'-1}\mathbf{1}_{m'>0},$$
$$[\uparrow X]_{m,m'} = X_{m+1,m'}\mathbf{1}_{m<M}, \quad [\downarrow X]_{m,m'} = X_{m-1,m'}\mathbf{1}_{m>0},$$
$$[\nwarrow X] = [\leftarrow[\uparrow X]], \quad [\searrow X] = [\rightarrow[\downarrow X]].$$

The algorithm is performed on $|i-j|+1$ matrices. As $|i-j|+1 \leq K$, and all the matrix operations can be done in $O(M^2)$, Algorithm 3 requires $O(KM^2)$ elementary operations. It is implemented by fonction `Clones_T(D, C, [i, j])` in the toolbox `Clones`.

To compute transitions on gap-free diagrams, the steps of the algorithm are the same. The only difference is that there is no need to compute *all* the arcs in $\mathcal{E}mpty$, $\mathcal{F}ull$ and $\mathcal{T}ransit$, but only the extremal ones in each column, which

Algorithm 3: Transition algorithm

Data: D the matrix representation of a diagram D, $(i,j) \in R$
Result: D′ the matrix representation of $T_{i,j}(D)$.

```
begin
    D^Empty_{i,j} ← D^{0}_{i→j};
    D^Full_{i,j} ← D^{C_j}_{j→i};
    D^Transit_{i,j} ← D^{1,...,C_i}_{i→j} AND D^{0,...,C_j−1}_{j→i};
    if i < j then
        D′_i^Transit ← [←D_i^Transit];
        for k from i+1 to j−1 do D′_k^Transit ← [↖D_k^Transit]
        D′_j^Transit ← [↑D_j^Transit];
    else
        D′_i^Transit ← [↓D_i^Transit];
        for k from i+1 to j−1 do D′_k^Transit ← [↘D_k^Transit]
        D′_j^Transit ← [→D_j^Transit];
    D′_{i,j} ← D^Empty_{i,j} OR D^Full_{j,i} OR D^Transit_{i,j};
    return D′ = [D_{1,min(i,j)−1}, D′_{i,j}, D_{max(i,j)+1,K}].
```

can be computed directly from matrix `F`. As a consequence, the transformation can be done in $O(KM)$. This function is implemented in `Clones_gfT(D, C, M, [i, j])`.

3.3 Additional functionalities of Clones

The conversion of a diagram D into a set of states, $\psi(D)$, is performed by `Clones_Psi(D)`. Function `Clones_CardPsi(D)` returns the cardinal of $\psi(D)$. Function `Clones_StateSpace(C, M)` uses `Clones_Psi(D)` to compute the entire state space.

Diagrams can be plotted with `Clones_plotD(D, C, mode)`; `mode` allows to write parameters or $|\psi(D)|$ in the graphic (Figure 9). It is also possible to have diagram animations.

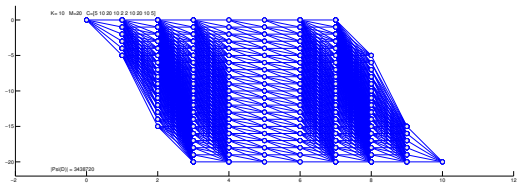


Figure 9: Plot of a complete diagram.

Object-oriented programming can also be used with `Clones`. Three classes have been defined to handle transitions and perfect sampling using respectively sets of states, diagrams and gap-free diagrams. Let `R` be the routing matrix. The functions for perfect sampling with sets of states, diagram and gap-free diagrams are respectively `Clones_PSS(C, M, R, N)`, `Clones_PSD(C, M, R, N)` and `Clones_PSF(C, M, R, N)`. They produce N samples.

4. PERFECT SAMPLING WITH Clones

In this section we are interested in the running time of the perfect sampling algorithms. Experiments have been performed on a desktop machine.

We chose a small number of queues ($K = 5$) in order to compare running time of Algorithms PSD and PSF against PSS (that has an exponential complexity). Each queue has a capacity $M/2$ and the routing matrix is fixed and randomly chosen for the whole experimentation. For each sample, the same sequence of random variables $(U_n)_n$ is used for the three algorithms. Thus they produce the same sample. For each experiment, we produce 100 samples and are interested in the mean running time.

Figure 10 compares the running times for the three algorithms with $M \in \{2, 4, \ldots, 60\}$. It can be observed that the running time of Algorithm 1 grows exponentially fast, and the two other algorithms (with diagrams and gap-free diagrams) terminate much faster.

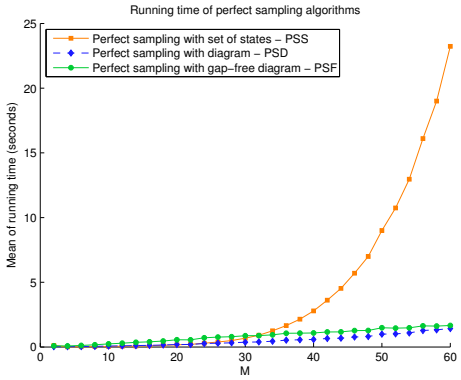


Figure 10: Mean running time - 100 simulations

Figure 11 compares the perfect sampling algorithms using respectively diagrams and gap-free diagrams when the number of customers is $M = \{5, 10, \ldots, 200\}$. We can see that for values from $M = 5$ to 60, the algorithm with diagrams is slightly faster than the one in gap-free diagrams, whereas for higher values of M, using gap-free diagrams is much more efficient than using diagrams. The latter result was expected, as the algorithmic complexity of one step transition is in $O(KM)$ for gap-free diagrams instead of $O(KM^2)$ for diagrams. The running time when $M \leq 60$ is more surprising. We believe that the main reason is that Matlab is optimized for matrix operations. Specially, when it comes to sparse matrices. On the other hand, our implementation of gap-free diagrams cannot benefit from this optimization, and there is room for improvement.

5. CONCLUSION

In this paper, we presented the toolbox `Clones`, that performs efficient coupling from the past of closed queueing networks. This toolbox enables the comparison between three different algorithms (CFTP with sets of states, diagrams and gap-free diagrams). Experimentally, PSD and PSF algorithms are very efficient compared to PSS algorithm. For small values of M, the CFTP with diagram implementation terminates faster than with gap-free diagrams. Future work will focus on the improvement of the gap-free one-step transition implementation.

Further investigation also include the study on the coupling time of the CFTP algorithms. Indeed, the coupling time of CFTP with diagrams is larger than that of PSS algorithm. Experimentally, the difference between the coupling times of the three implementations is rather small, but obtaining a theoretical bound would prove the efficiency of our approach. Last, we will also explore possible extensions to other classes of closed queueing networks.

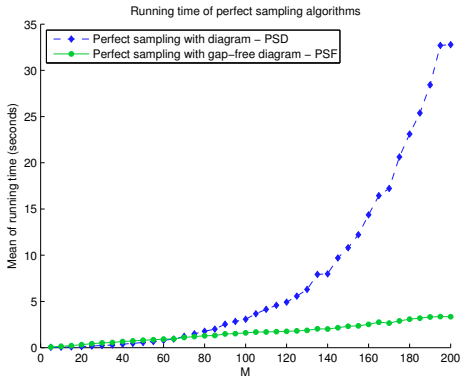


Figure 11: Mean running time - 100 simulations

6. ACKNOWLEDGMENTS

This research is supported by the French National Research Agency grant ANR-12-MONU-0019.

7. REFERENCES

[1] S. Asmussen and P. W. Glynn. *Stochastic Simulation: Algorithms and Analysis*, volume 57 of *Stochastic Modelling and Applied Probability*. Springer-Verlag, New York, 2007.

[2] S. Balsamo. Queueing networks with blocking: Analysis, solution algorithms and properties. In D. D. Kouvatsos, editor, *Network Performance Engineering*, volume 5233 of *LNCS*, pages 233–257. Springer Berlin Heidelberg, 2011.

[3] A. Bouillard, A. Busic, and C. Rovetta. Perfect sampling for closed queueing networks. *Performance Evaluation*, 79(0):146–159, 2014. Special Issue: Performance 2014.

[4] J. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Comm. ACM*, 16:527–531, 1973.

[5] W. Gordon and G. Newel. Closed queueing systems with exponential servers. *Oper. Res.*, 15,2:254–265, 1967.

[6] S. Kijima and T. Matsui. Randomized approximation scheme and perfect sampler for closed Jackson networks with multiple servers. *Annals of Operations Research*, 162(1):35–55, 2008.

[7] D. Levin, Y. Peres, and E. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2009.

[8] R. O. Onvural. Survey of closed queueing networks with blocking. *ACM Comput. Surv.*, 22(2):83–121, 1990.

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