Efficient Simulation of Sparse Graphs of Point Processes

Cyrille Mascart, David Hill, Alexandre Muzy, Patricia Reynaud-Bouret

Efficient Simulation of Sparse Graphs of Point Processes

CYRILLE MASCART, Université Côte d'Azur, France
ALEXANDRE MUZY, Université Côte d'Azur, CNRS, I3S, France
PATRICIA REYNAUD-BOURET, Université Côte d'Azur, CNRS, LJAD, France

We derive new discrete event simulation algorithms for marked time point processes. The main idea is to couple a special structure, namely the associated local independence graph, as defined by Didelez [12], with the activity tracking algorithm [22] for achieving high performance asynchronous simulations. With respect to classical algorithm, this allows reducing drastically the computational complexity, especially when the graph is sparse.

CCS Concepts: • **Mathematics of computing** → **Stochastic processes**; • **Computing methodologies** → **Discrete-event simulation**; • **General and reference** → *Performance*; • **Applied computing** → Bioinformatics.

Additional Key Words and Phrases: Point processes, discrete event simulation, Hawkes point processes, computational complexity, local independent graphs

ACM Reference Format:
Cyrille Mascart, Alexandre Muzy, and Patricia Reynaud-Bouret. 2021. Efficient Simulation of Sparse Graphs of Point Processes . *ACM Trans. Model. Comput. Simul.* 1, 1 (March 2021), 28 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' addresses: Cyrille Mascart, Université Côte d'Azur, Laboratoire I3S, 2400 route des Lucioles, Biot, 06000, France, mascart@i3s.unice.fr; Alexandre MuzyUniversité Côte d'Azur, CNRS, I3S, France, alexandre.muzy@univ-cotedazur.fr; Patricia Reynaud-BouretUniversité Côte d'Azur, CNRS, LJAD, France, patricia.reynaud-bouret@univ-cotedazur.fr.

Contents

1 INTRODUCTION

Point processes in time are stochastic objects that model efficiently event occurrences. The variety of applications is huge: from medical data applications (time of death or illnesses) to social sciences (dates of crimes, weddings, etc), from seismology (earthquake occurrences) to micro-finance (actions of selling or buying a certain assets), from genomics (gene positions on the DNA strand) to reliability analysis (breakdowns of complex systems) (see e.g. [1, 8, 12, 24, 30, 32]).

Most of the time, point processes are multivariate, in the sense that either several processes are considered at the same time, or in the sense that one process regroups together all the events of the different processes and marks them by their type. A typical example consists in considering either two processes, one counting the wedding events of a given person and one counting the birth dates of children of the same person. One can see this as a marked process which regroups all the possible dates of birth or weddings independently and on each event one marks it by its type, here wedding or birth.

In the sequel, we denote the individual process N_j, the set of all events corresponding to type j, for $j = 1, ..., M$ and the joint process $N = N_1 \cup .. \cup N_m$. In this multivariate or marked case, the individual processes are usually globally dependent, the apparition of one event or point on a given type influencing the apparition of other points for the other types and the simulation of the whole system cannot be easily parallelized.

This is especially true in neuroscience [29]. Let us detail a bit more this set up which is a benchmark example here. Neurons are excitable electric cells that are linked together inside a huge network (10^{11} for humans [33], 10^8 for rats [18], 10^6 for coackroaches), each cell receives inputs from approximately 10^3 to 10^4 presynaptic (upstream) neurons [27]. Depending on its excitation, the neuron might then produce an action potential also called spike, information which is propagated to postsynaptic (downstream) neurons.

From a stochastic point of view, one might then see the spike trains emitted by a given neuron as an individual point process which in fact is embedded in a multivariate point process with M, the total number of neurons as the total number of types. The size of the network requires then very well adapted simulation schemes that may use the relative sparseness of the network with respect to the global size of the network.

To do so, we use the mathematical notion of local independence graph for marked point processes due to Didelez [12], which is detailed in Section 4 and which informally corresponds to the real neuronal network. In this sense, in the sequel we call marks, processes and type the nodes of the graph. The only strong assumption that is used is the time asynchrony hypothesis, (i.e. points or events of different mark or types, meanings points or events appearing in different nodes, cannot occur at the exact same time) together with the fact that all processes have a conditional intensity [4].

Simulation of point processes has a long history that dates back to Doob in the 40's [13] for Markov processes. In the 70's, Gillespie [15] popularized the method for a particular application: chemical reactions. At the same time, Lewis and Shedler [19] proposed a thinning algorithm for simulating univariate inhomogeneous Poisson processes (this can also be viewed as a rejection method). Few years later, Ogata [25] produced a hybrid algorithm able to simulate multivariate point processes in the general case even if they are not markovian, including both a choice of the next point by thinning and a choice of the node to activate thanks to Gillespie principle. This method is still up to now the benchmark for simulating such processes, and is for instance used in recent packages such as `ppstat` in `R` (2012). It has been rediscovered many times in various cases, most of the time as a Generalized Gillespie method (see for instance [2]).

When the number of types or nodes is huge, this method can quickly become inefficient in terms of computational times. Many people have found shortcuts, especially in Markovian settings. For instance, Peters and de With [28]

proposed a new simulation scheme exploiting a network of interaction for particular physical applications. In [3], the authors reformulated this algorithm in a more mathematical way for a particular case of Piecewise Deterministic Markov Processes. People have even exploited very particular structures such as Hawkes processes, with exponential interactions (special case which leads to Markovian intensities) [11], to be able to simulate huge networks, as in the `Python` package `tick` (2017).

In the mean time, the technique of discrete event simulation first appeared in the mid-1950s [31] and was used to simulate the components (machines) of a system changing state only at discrete "events". This technique has then been formalised in the mid-1970s [34]. Discrete event modelling and simulation seem very close to point process models (dealing with events, directed graphs, continuous time, etc.). Against all expectations, as far as we know, there is no direct use of any discrete event simulation algorithm for point processes. Maybe, the sophistication of these algorithms being of the same order than the mathematical technicality of point processes, prevented any direct application. Besides, the continuous nature of the conditional intensity associated to a point process with respect to the discreteness of event-based simulations could make appear the two domains as separated whereas discrete event theory is a computational specification of mathematical (continuous) systems theory [21] integrating more and more formally stochastic simulation concepts [35]. We hope to show here that both domains can take advantage from each other, by introducing new discrete-events models that act as generators of point processes. Especially, whereas discrete event simulation algorithms have been developed considering independently the components (nodes) of a system, a new algorithm for activity tracking simulation [22] have been proposed to track activity (events from active nodes to children). The activity tracking algorithm is used here and proved to be the right tool for both simplifying usual discrete event algorithms (which are difficult to relate to usual point process algorithms) and efficiently simulating point processes.

Our aim is to derive a new simulation algorithm, which generalises the algorithm of [3] to general multivariate point processes that are not necessarily Markovian, by exploiting the underlying network between the types, which is here a local independence graph. In Section 2, the main mathematical background and notations are provided and the classical multivariate algorithm due to Ogata [25] is explained. A simplified version in discrete event terms and called full scan algorithm is proposed. In Section 3, discrete event data structure and operations specific to point processes are designed. In Section 4, after recalling the notion of local independence graph [12], a new local graph algorithm is presented. In Section 5, we evaluate the computational complexities of both algorithms on Hawkes processes with piecewise constant interactions, which model easily neuronal spike trains [29]. We show that in this case, for sparse graphs, new local graph algorithm clearly outperforms the classical Ogata's algorithm in its discrete event version.

2 SET-UP

2.1 Mathematical framework

A (univariate) point process N in $\mathbb{R}_+$ is a random countable set of points of $\mathbb{R}_+$. For any subset A of $\mathbb{R}_+$, $N(A)$ is the number of points that lie in A.

As real random variables might be defined by their density with respect to Lebesgue measure, if it exists, a point process is characterised by its conditional intensity with respect to a given filtration or history $(\mathcal{F}_t, t \geq 0)$. For the mathematical details, we refer the reader to [4]. Informally, the filtration or history at time $t-$, $\mathcal{F}_{t-}$, contains all the information that is needed to simulate the next point of N, when one is just before time t. It usually includes as

generators, all the points $T \in N$ such that $T < t$ in particular. The (conditional) intensity of the point process N is then informally defined [4] by

$$\lambda(t) = \lim_{dt \to 0} \frac{1}{dt} \mathbb{P}\left(\text{there is a point of } N \text{ in } [t, t+dt] \middle| \mathcal{F}_{t-}\right),$$

for infinitesimal dt, where $\mathbb{P}\left(\text{there is a point of } N \text{ in } [t, t+dt] \middle| \mathcal{F}_{t-}\right)$ is the probability that a point appears in the interval $[t, t+dt]$ given what happened strictly before t in the history. This is a random process which, at time t, may depend in particular on all the past occurrences of the process itself, that is the $T < t$.

A multivariate point process can be seen as a collection of M different point processes N_j. With the time asynchrony hypothesis, one can also consider equivalently the univariate joint point process $N = N_1 \cup ... \cup N_M$ and say that for each point t of N there is one and only one subprocess j such that $t \in N_j$. This j is then called the mark of point t, or the node associated to t.

We are given the set of intensities of each of the N_j, $t \mapsto \lambda_j(t)$, with respect to a common filtration $(\mathcal{F}_t, t \geq 0)$. Note that $\mathcal{F}_{t-}$ includes as generators, all the points $T \in N$, the joint process such that $T < t$ as well as their respective marks.

Examples. Let us give just few basic examples

- **Homogeneous Poisson processes** with rates $(\nu_i)_{i=1,...,M}$. In this case, all λ_i are constant and not even random and for all i,

  $$\lambda_i(t) = \nu_i.$$

  We see in this expression, that the intensities do not depend neither on time, nor on the previous occurrences. This is why one often refers to such dynamics as "memoryless". Since these processes do not interact, one can of course simulate each N_i in parallel if need be. In this case, for each of them, it is sufficient to simulate the time elapsed until the next point, by an exponential variable of parameter ν_i, independently from anything else. To unify frameworks, this exponential variable might also be seen as $-\log(U)/\nu_i$, with U a uniform variable on $[0, 1]$.
- **Inhomogeneous Poisson processes** with time-dependent rates $(f_i)_{i=1,...,M}$. In this case, the λ_i's are not necessarily constant and but they are still non random and for all i,

  $$\lambda_i(t) = f_i(t).$$

  Once again parallelization is possible, and for each individual process N_i and given point t_k^i, one finds the next point t_{k+1}^i by solving

  $$\int_{t_k^i}^{t_{k+1}^i} f_j(s) \ \mathrm{d}s = -\log(U)$$
- **Linear multivariate Hawkes process** with spontaneous parameter $(\nu_j)_{j=1,...,M}$ and non negative interaction functions $(h_{j\to i})_{i,j=1,...,M}$ on $\mathbb{R}_+$. This process has intensity

  $$\lambda_i(t) = \nu_i + \sum_{j=1}^{M} \sum_{T \in N_j, T<t} h_{j\to i}(t - T). \tag{1}$$

  This process is used for many excitatory systems, especially the ones modelling the spiking activity of neurons [29]. It can be interpreted in this sense, informally: to a homogeneous Poisson process of rate ν_i, which models the spontaneous activity of the neuron i, one adds extra-points coming from the interactions. Typically a point T

of mark (neuron) j adds a term $h_{j\to i}(\delta)$, after delay δ to the intensity of N_i making the apparition of a new point at time $t = T + \delta$ more likely. In this sense there is an excitation of j on i. Here we see a prototypical example of global dependence between the marks. Each new point for each mark depends on all the points that have appeared before, with all the possible marks, preventing a brute force parallelization of the simulation. Except when the $h_{j\to i}$'s are exponentially decreasing [11], this process is clearly not Markovian. It is for this kind of general process that one needs efficient simulation algorithms.

2.2 Simulation of univariate processes

The time-rescaling theorem (see [6] or [4] for more mathematical insight) states that if a point process N has a conditional intensity $\lambda(t)$, and if

$$\forall t, \Lambda(t) = \int_0^t \lambda(s) \ \mathrm{d}s,$$

then $\mathcal{N} = \{\Lambda(T), T \in N\}$ is a Poisson process of rate 1. This is why, even for general point processes, it is always possible to find, by iteration the next point of N by solving recursively, for all $k \in \mathbb{N}^*$ the set of positive natural numbers,

$$\int_{t_k}^{t_{k+1}} \lambda(s) \ \mathrm{d}s = -\log(U) \tag{2}$$

initializing the method with $t_0 = 0$.

Of course, to be able to mathematically solve this easily, one needs to be able at time t_k to compute $\lambda(t)$ on $(t_k, +\infty)$ if no other point occurs. This in particular happens if the filtration $\mathcal{F}_t$ is reduced to the filtration generated by the points themselves and this is what we will assume here. Of course all algorithms discussed here can easily be adapted to richer filtrations, as long as the computation of $\lambda(t)$ on $(t_k, +\infty)$ can be carried out (if no other point occurs).

In this situation, two cases might happen, each of them leading to a different algorithm:

Transformation method: The function $\lambda(t)$ on $(t_k, +\infty)$ (and if no other point occurs) has an easily computable primitive function with inverse $\Lambda^{-1}(t)$. Then (2) reduces to

$$t_{k+1} = \Lambda^{-1}(-\log(U) + \Lambda(t_k)).$$

Thinning method: It applies if the previous computation is not possible or easy but one can still compute $\lambda^*(t) \geq \lambda(t)$ such that $\lambda^*(t)$ has all the desired properties of the transformation method (typically $\lambda^*(t)$ is constant, with constant that might depend on the t_ℓ for $\ell \leq k$). Then the algorithm does as follows to compute a possible next point (cf. Algorithm 1). If thinning for Poisson processes is due to [19], it has been generalized to general processes by Ogata [25]. One can find a complete proof in [9].

Algorithm 1 Thinning algorithm

initialize $t_0^* \leftarrow t_k$
repeat
  Generate next point t^* after t_0^* of a point process with intensity function λ^* by the Transformation method.
  Generate $U \sim \mathcal{U}[0,1]$
  if $U > \lambda(t^*)/\lambda^*(t^*)$ **then** # Rejection
    $t_0^* \leftarrow t^*$
until $U \leq \lambda(t^*)/\lambda^*(t^*)$
return $t_{k+1} \leftarrow t^*$

2.3 Discrete event version of classical multivariate algorithm for point processes

To simulate multivariate processes, Ogata [25] made an hybridation between two different notions : the thinning algorithm presented above and the attribution of marks. Indeed, since all processes N_j are communicating with each other, Ogata's idea for the attribution of marks is to generate the next point of the aggregated process N and, then, to decide for its mark.

In the present article, we choose to discard the thinning part, which can be added to all the algorithms that we derive here. The attribution part of Ogata's multivariate algorithm [25], is referred in the sequel as full scan, because the intensities of all nodes of the graph need to be scanned and updated at each time stamp t_k. Once the next point of N is decided, the basic idea for the attribution of marks is to attribute them at random, the distribution taking into account the relative value of the intensity of each subprocess. The main steps of this algorithm are presented in Figure 1 for a visual representation of the method. More details about the algorithm steps are provided through the Hawkes application in Section 5.

Algorithm 2 Full scan multivariate algorithm modified from [25]

$t_0 \leftarrow 0$
while $t_k < T$ **do**
Step a/ **Compute intensity sums** $\sum_{j=1}^{i} \lambda_j(t) = \overline{\lambda}_i(t)$, for $i \in \{1, ..., M\}$ on $t \in (t_k, +\infty)$
Step b/ **Get** by simulation t_{k+1} as the **next point** of a univariate point process of intensity $\overline{\lambda}_M(t)$
Step c/ **Select the unique possible node** i_{k+1}, such that $\frac{\overline{\lambda}_{i_{k+1}-1}(t_{k+1})}{\overline{\lambda}_M(t_{k+1})} < V \leq \frac{\overline{\lambda}_{i_{k+1}}(t_{k+1})}{\overline{\lambda}_M(t_{k+1})}$, with $V \sim \mathcal{U}[0,1]$ and $\overline{\lambda}_0 = 0$
Step d/ **Update intensities** λ_j on $(t_{k+1}, +\infty), \forall j \in \{1, ..., M\}$
  $k \leftarrow k + 1$
return points $(t_1, ..., t_{k-1})$ and associated nodes $(i_1, ..., i_{k-1})$

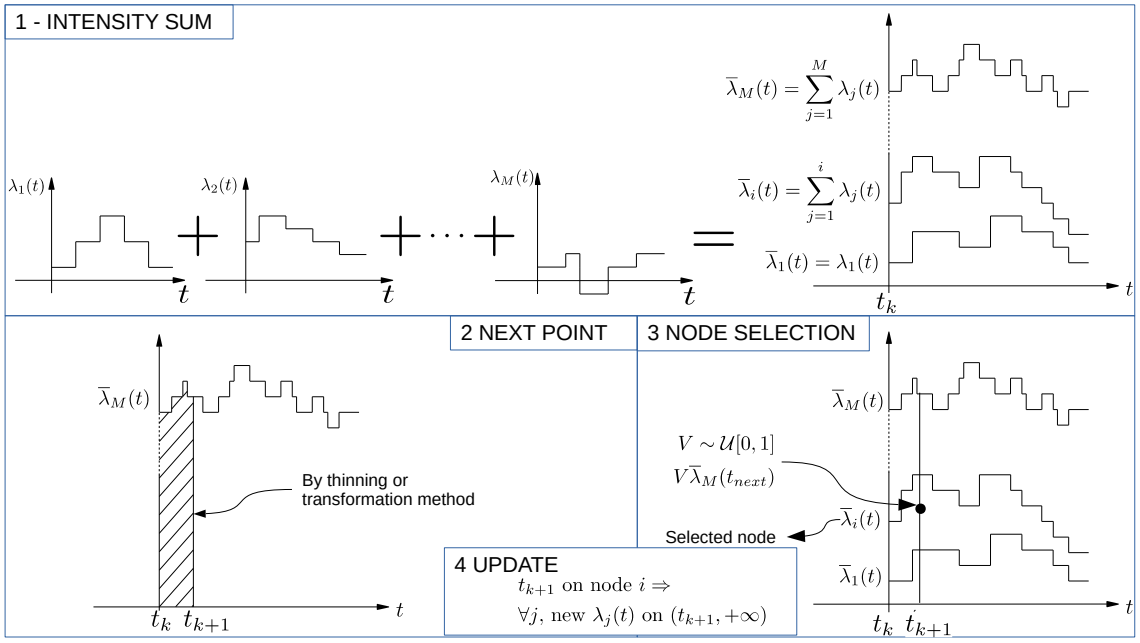

Fig. 1. Steps of the full scan algorithm for point processes. The intensities are piecewise constant (cf. Section 5).

Original Ogata's algorithm uses thinning at step 4 of Algorithm 2. However the complexity of a thinning step is difficult to evaluate because it depends on both the complexity of the upper-bounding function λ^* and how far this function is from λ, which influences how much time the thinning algorithm rejects. Therefore for a clear evaluation of the complexity, we focused on simulations where the transformation method is doable, typically when the intensities are piecewise constant.

3 SPECIFIC DISCRETE EVENT DATA STRUCTURES AND OPERATIONS

Before introducing our new algorithm, we present a particular structure, which is very important for discrete events algorithm : the scheduler.

A scheduler Q is a data structure, which can be represented as an ordered set of events, and which is provided with a set of operations for ensuring the correct order of the events. The events are noted $ev_i = (t_i, v_i)$, where t_i is the event time and v_i is the event value. The events in the scheduler are increasingly ordered in time, i.e., $ev_i, ev_j \in Q$, $ev_i < ev_j \iff t_i < t_j$. The length of the scheduler is noted $|Q|$.

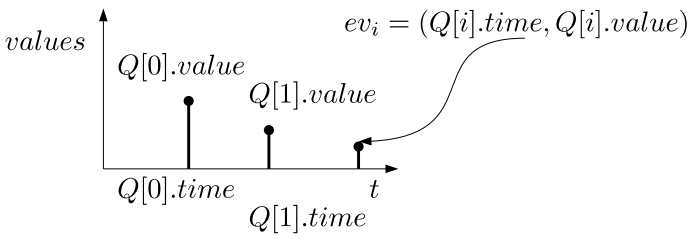

Fig. 2. Example of events in scheduler Q. For an event ev_i in the scheduler, the value is accessed by $Q[i].value$ and the time is accessed by $Q[i].time$, with $i = 0$ the first event index.

In both full scan and local graph algorithms schedulers are used. They are usually implemented using one of the many kinds of self-balancing binary tree. The choice of structure (AVL, red-black, etc.) depends on corresponding operation complexity. Here we choose the red-black self balancing tree, which exhibits the best performance with respect to other usual self-balancing trees [17].

To make a scheduler, the red-black tree is equipped with a set of classical (**insert**, **remove**, **upper bound** and **lower bound**) and non-classical operations. All these operations, except stated otherwise, have time complexity bounded by the logarithm $\log_2$ of the number of elements in the set. In addition it is supposed that any element of the scheduler can be accessed with a constant time. This may not be the case depending on the language used for the implementation. However, it is true for C++ language, which was used in our implementation.

3.1 Basic scheduler operations

We describe and illustrate the set of operations completing the red-black tree to form the scheduler data structure needed for our algorithms. Any operation on a scheduler Q directly modifies it. Also, to simplify the operations, it is supposed that all events stored in a scheduler Q are unique.

Access element operation $(t^i, v^i)_{i\in\{0,\dots,|Q|-1\}}$ is the list of events stored in scheduler Q and sorted by ascending values of t. Then operation $Q[i]$ returns event (t^i, v^i) with constant time complexity $\mathcal{O}(1)$.

Insert operation of an event $(t, v) \in \mathbb{R}_+ \times \mathbb{R}$ (cf. Figure 3): $Q \oplus (t, v)$ inserts the event in the tree. This operation has a total maximum complexity of $O(log_2(|Q|))$, to find the place of the event and rebalance the tree.

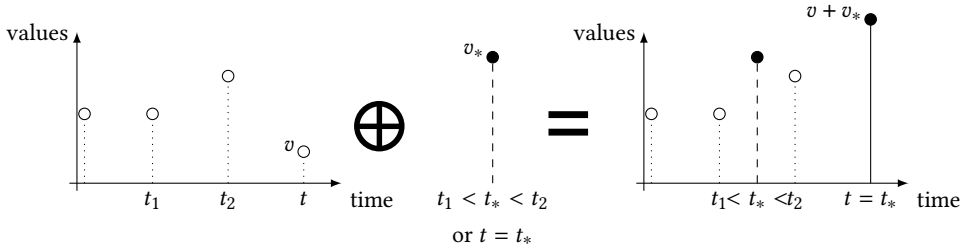

Fig. 3. A graphical example of the insertion of a new event (t_*, v_*) inside a scheduler. There are two cases: if the event time is unmatched in the set of event times already present in the scheduler (case $t_1 < t_* < t_2$), the event is just inserted in the right place; otherwise (case $t = t_*$) the event in the scheduler with the same time has its value increased by the value v_* of the new event.

Remove operation of an event (cf. Figure 4): $Q \ominus (t, v)$, removes the event (t, v) from the tree. This operation has a maximum complexity of $\mathcal{O}\left(\log_2(|Q|)\right)$, to find the place of the event and rebalance the tree. If the index i of the element is known, the remove operation can be executed in constant time.

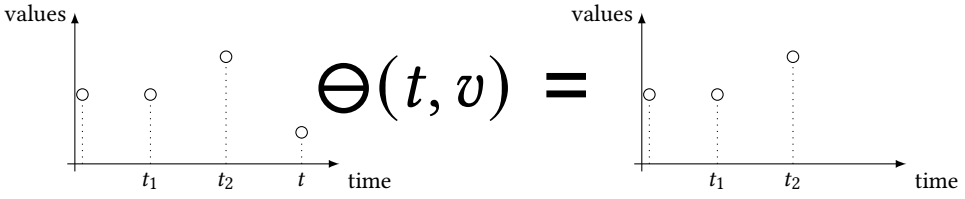

Fig. 4. A graphical example of the removal of an event given its time t.

Remove first operation Q^* over the scheduler Q, which removes the first event $Q[0] = (t^0, v^0)$ from the scheduler, the second event becoming the first. Operation $\cdot^*$ has complexity $O(1)$.

Prune operation of the scheduler Q^t (cf. Figure 5) removes all events (t^*, v^*) from $|Q|$ whose time value $t^* \leq t$. The operation Q^t has complexity $O\left(log_2(|Q|) + |\text{number of events until time } t|\right)$.

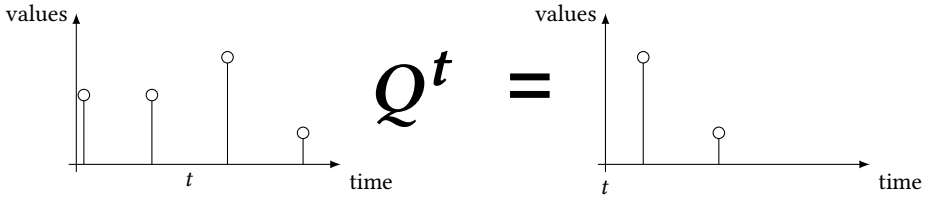

Fig. 5. A graphical example of the prune operation : Here all the points with a time less or equal to t (here the first two points of this scheduler) are removed. The time complexity of the operation is $\mathcal{O}(\log_2(4) + 2)$ (scheduler of size $|Q| = 4$ and 2 events removed).

Upper and lower bound operations (cf. Figure 6): Upper bound operation: $\lceil t \rceil_Q$ and lower bound operation $\lfloor t \rfloor_Q$, return the smaller event (t^*, v^*) verifying respectively $t^* > t$ and $t^* \geq t$. In both cases the time complexity of the operation is $O(log_2(|Q|))$.

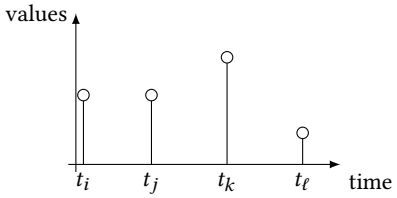

Fig. 6. Upper/lower bound operations on an example of scheduler Q. For $t_k < t < t_\ell$ the upper bound operation consists of $\lceil t \rceil_Q = \ell$. The lower bound operation consists of $\lfloor t \rfloor_Q = k$.

3.2 Using a scheduler to represent piecewise constant functions

Let $h\colon \mathbb{R}^* \to \mathbb{R}, t \mapsto h(t)$ be a piecewise constant function with finite support $S \subseteq \mathbb{R}^*$. A scheduler can encode piecewise constant functions on S. There are two possible methods to achieve this:

(1) Let each discontinuity $(t, h(t))$ be its own event in the scheduler Q representing the function h (see top part of Figure 7).
(2) Represent the discontinuity not as a new value $(t, h(t))$ but as an increment, that is a difference between the new value of h after the discontinuity and the value of h just before: $(t, h(t) - h(t_-))$ (see bottom part of Figure 7).

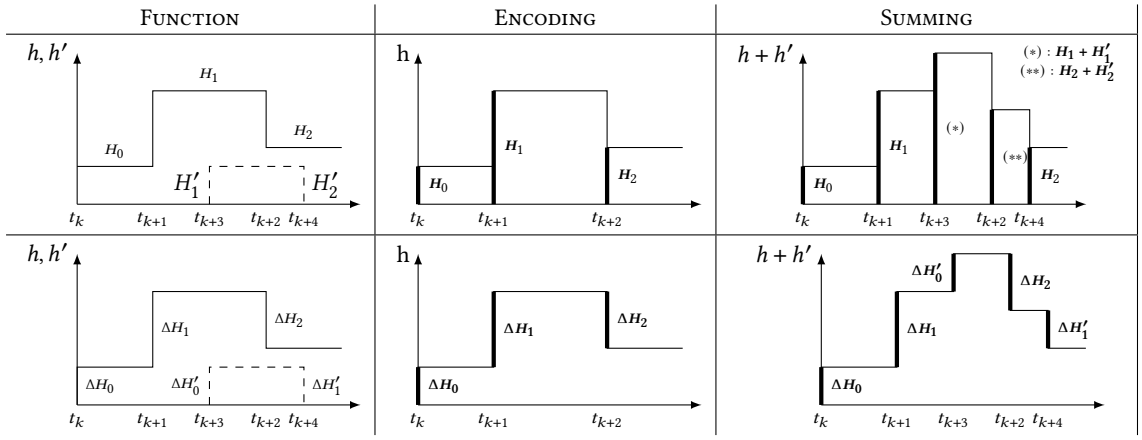

Fig. 7. Example of a straightforward encoding of a piecewise constant function (first line) and an optimised one (second line). We use the notation $H_n = h(t_{k+n}), n \in \mathbb{N}$ for concision. The Function column represents the piecewise constant function h to encode (plain lines), as well as another function h' (dashed lines, see column Summing). Column Encoding represents in bold the events $(t_n, h(t_n))$ encoded by a scheduler representing the function h. Column Summing represents in bold the values encoded by a scheduler now representing the function $h + h'$.

The first representation is more straightforward, and is very efficient when the function is stored only for lookups: the function can then be stored in an array and a dichotomic search algorithm be used to locate any event in logarithmic time.

However, if discontinuities must be introduced or removed, for instance by adding another piecewise constant function to h, then some of the events stored in the scheduler may need to be changed or moved in the scheduler, thus creating an undesirable overhead. For instance in the first line of figure 7, a piecewise function h (first column) is represented and encoded (second column) with the straightforward method by a scheduler. Last column shows the new encoding representing a sum. The discontinuity at t_{k+2} had to be modified because of the introduction of discontinuities at t_{k+3} and t_{k+4}. For a function with many more discontinuities, or when summing two piecewise functions of similar size, the operation would necessitate to modify a large part of the already represented points.

The solution we proposed is to store not the values $h(t)$ of the function h at a time t where a discontinuity happens, but the variation $\Delta h_t = h(t) - h(t_-)$ of the value of h during the discontinuity. An example is represented on the second line of Figure 7.

Let us now present two different operations for piecewise contant functions with this encoding:

Piecewise sum, which is in fact a union operation over two schedulers Q, Q' encoding two piecewise constant functions (cf. Figure 8): The operation $Q \cup Q'$ has complexity $O(min(|Q|, |Q'|)log_2(max(|Q|, |Q'|)))$, since the smallest scheduler is inserted into the largest scheduler.

Piecewise prune operation of the scheduler Q^t_{pcw} (cf. Figure 9): removes from scheduler Q all the events (t^*, v^*) with $t^* \leq t$, while accumulating all the values v^* up to time t. At the end of the operation the scheduler begins with an event of the form $(t, \sum v^*)$. The operation Q^t_{pcw} has complexity $O(log_2(|Q|) + |\text{number of events until time } t|)$.

Shift operation of the scheduler $Q_{\rightarrow_t}$ (cf. Figure 10): shifts all points by t. The operation $Q_{\rightarrow_t}$ has complexity $O(|Q|)$.

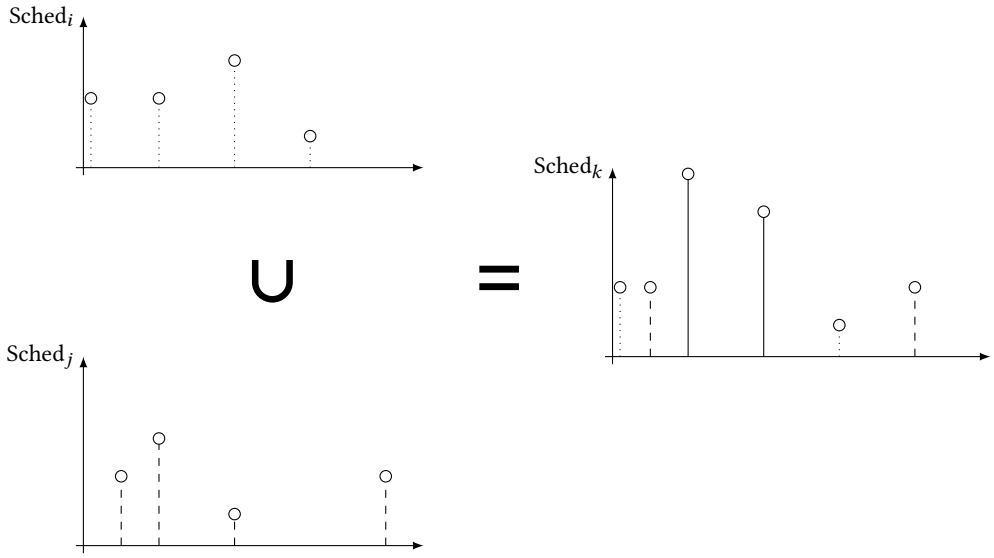

Fig. 8. A graphical example of the union of two schedulers. The events of Sched$_i$ are represented as dotted, while the events of Sched$_j$ are dashed. In the merged scheduler Sched$_k$, the values of events at the same time in both schedulers i and j are summed and the resulting event is represented with a continuous line.

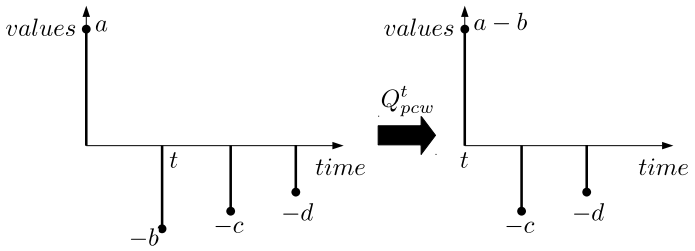

Fig. 9. Example of piecewise prune operation Q^t_{pcw}.

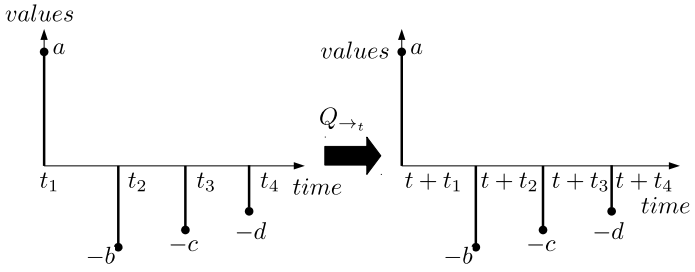

Fig. 10. The shift operation $Q_{\to_t}$

4 LOCAL GRAPH ALGORITHM FOR POINT PROCESSES

4.1 Local independence graph

Local independence graphs are fully presented in a sound mathematical form in [12]. For a given multivariate point process N_j, $j = 1, ..., M$, the corresponding local independence graph is a directed graph on the nodes $j = 1, ..., M$ (see for instance Figure 11). We assume for sake of simplicity that the filtration is reduced to the internal history, that is $\mathcal{F}_t$ is generated only by the $T < t$ in $N = N_1 \cup ... \cup N_M$ and their associated mark or node.

To explain more fully what a local independence graph means, we need to define rougher filtration. For a subset $I \subset \{1, ..., M\}$, $\mathcal{F}_t^I$ is the filtration generated by the $T < t$ in $\cup_{i \in I} N_i$ and their associated node.

In a local independence graph, the absence of edge $j \to i$ means that the apparition of a point at time t in N_i is independent from $\mathcal{F}_{t-}^{\{j\}}$ conditionally to $\mathcal{F}_{t-}^{\{j\}^c}$, where $\{j\}^c = \{1, ..., M\} \setminus \{j\}$.

So this means that for every time t, the intensity $\lambda_i(t)$ of N_i does not depend directly on the positions of the points of N_j strictly before t.

This extends directly to the notion of parents and children in the graph. For a given node i, one defines

$$pa(i) = \{j, j \to i \text{ is in the graph}\} \text{ and } ch(i) = \{j, i \to j \text{ is in the graph}\}.$$

Therefore it means that the intensity $\lambda_i(t)$ at time t of N_i in fact only depends on the points of N_j for $j \in pa(i)$ strictly before t.

Conversely, a point on N_i directly impacts the occurrence of points for N_j for $j \in ch(i)$. Note that in any case, it also impacts the next point of N_i because even for a Poisson process without memory one needs by the transformation method to know t_k for finding t_{k+1}. However, it will not have any direct impact on the future points of N_j for $j \notin ch(i) \cup \{i\}$.

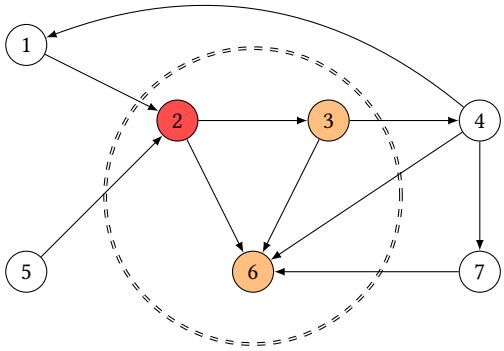

Fig. 11. Example of local independence graph. With this graph, $ch(2) = \{3, 6\}$ and $pa(2) = \{1, 5\}$. As indicated by the difference of colour, a point with mark 2 shall impact the point generation only for $\{2\} \cup \{3, 6\}$.

For instance, for linear Hawkes process (cf. Equation 1), $ch(i) = \{j / h_{i \to j} \neq 0\}$.

4.2 Local-graph algorithm

The children of each node are stored in a simple one dimensional array, whose indexes are the node indexes, and whose cells contain vectors of the children indexes. So accessing a node simply costs $O(1)$.

Because of the interpretation of $I = ch(i) \cup \{i\}$ of a given node i given above in the local independence graph, it means that in fact, after having simulated t_k with mark/node i_k in the joint process, we know that only the next points of N_j for $j \in I$ have to be modified.

At simulation level, discrete events are used to track activity nodes associated to selected points (time stamps) to their children. Discrete events are stored into a scheduler Q of events $ev_i = (t^i_{next}, i)$, where t^i_{next} is the possible next point associated to node i.

The local graph algorithm for point processes is described in Algorithm 3. A visual representation is presented in Figure 12.

Let us detail a bit more each step.

- At Step a/, we decide for the next possible point of each of the subprocesses N_i by either using the transformation method or the thinning algorithm (Algorithm 1).
- At Step b/, for each $i \in I$ we remove the old event associated to i and insert the one computed at Step a/.
- At Step c/, we retrieve the minimum of all possible times to find out which subprocess i is actually generating a new point.
- At Step d/, we look for the children of i and update I.
- Step e/ depends hugely on the type of process at hand. The apparition of a new point has clear impact on the intensity but this depends on the formula. For instance if the intensity is given by Equation (1), we need to shift intensities and update sums (see Section 5 for more details).

More details about the algorithm steps are provided in Section 5, which presents in full details the application of the algorithm to the Hawkes case.

Algorithm 3 Local graph algorithm for the simulation of point processes: Application of the simulation activity tracking algorithm [22].

$k \leftarrow 0, t_k \leftarrow 0$
$I \leftarrow \{1, ..., M\}$
while $t_k < T$ **do**
Step a/ **Compute the next possible points** t^i_{next} for each $i \in I$ based on intensity λ_i on $(t_k, +\infty)$
Step b/ **Update** Q with each next possible point t^i_{next} for each $i \in I$
Step c/ **Get next selected point** $t_{k+1} \leftarrow min\{t^i_{next}\}$ and i the **associated node**, updating $Q \leftarrow Q^*$
Step d/ **Find the children of** i **and update** $I \leftarrow ch(i) \cup \{i\}$
Step e/ **Update intensities** $\lambda_j(t)$ for each node $j \in I$ on $(t_{k+1}, +\infty)$
$k \leftarrow k + 1$
return $(t_1, ..., t_{k-1})$ points and associated nodes $(i_1, ..., i_{k-1})$

5 HAWKES EVALUATION

We want to evaluate the complexity of the previous algorithms, but this of course depends on the computational complexity of the conditional intensities associated to each point process. Previous general algorithms for simulating point processes are applied here to non explosive Hawkes processes with piecewise constant interactions with finite support (see Equation (1)). In this situation, note that the λ_i's become piecewise constant, so that the complexity for calculating such intensities or updating them will be linked to the number of breakpoints of the corresponding piecewise constant function. Moreover with piecewise constant intensities, one can apply the transformation method directly, so

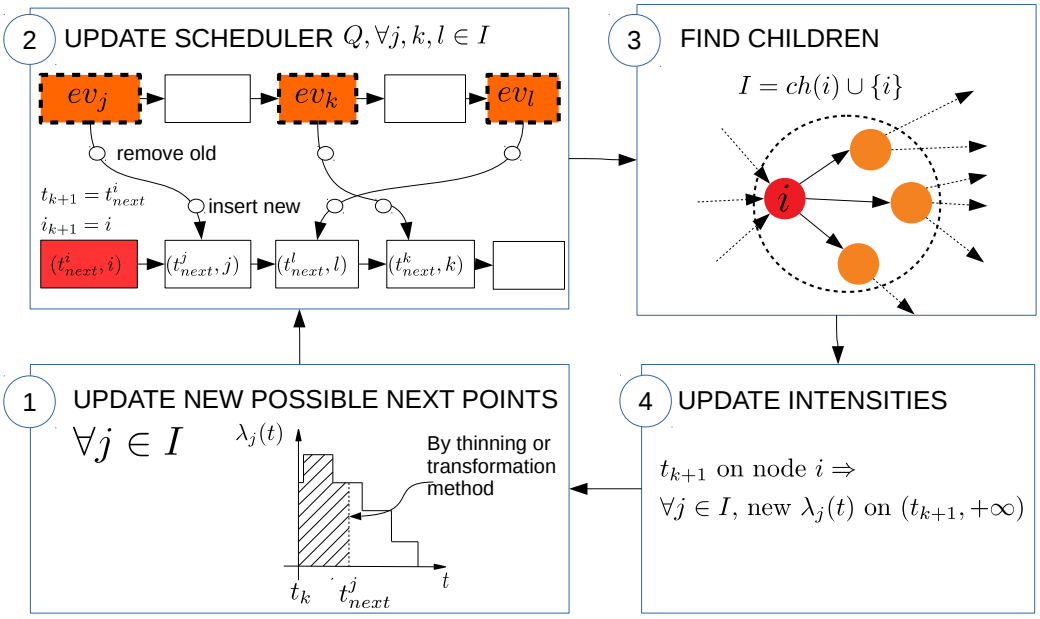

Fig. 12. Steps of the local graph algorithm for the simulation of point processes. As for figure 1, the intensities are piecewise constant.

we do not evaluate the complexity of the thinning /rejection step. The general algorithms are specified at data structure level in order to detail the computational complexity of each algorithmic step.

5.1 Notation and data structures

In the following, $\leftrightarrow$ means "corresponds to the mathematical notation". Data structure notation consists of:

- Q: A scheduler of next point events $ev_i = (t^i_{next}, i)$, where t^i_{next} is a possible next point associated to node i.
- $L[i] (\leftrightarrow \lambda_i)$: is the scheduler corresponding to the piecewise constant intensity of node i, The length of the scheduler is $L^i_t = length(L[i])$ when $L[i][0].time = t$.
- $h[j][i] (\leftrightarrow h_{j \to i})$: is the scheduler corresponding to the piecewise constant interaction $h_{j \to i}$ (with support included in $[0, S]$) from node j to node i. The maximum number of events in $h[j][i]$ is noted $A \geq length(h[j][i])$.
- $\overline{L} = L[1] \cup ... \cup L[M]$ $(\leftrightarrow \overline{\lambda}(t) = \sum_{i=1}^{M} \lambda_i(t))$: is a scheduler storing the piecewise sum of all $L[i]$, from node 1 to node M (cf. Figure 8).
- $\overline{L}[i] = L[1] \cup ... \cup L[i]$ $(\leftrightarrow \overline{\lambda}_i(t) = \sum_{j=1}^{i} \lambda_j(t))$: is a scheduler storing the partial piecewise sum of the intensities from node 1 to node i. Obviously, $\overline{L}[M] = \overline{L}$.

5.2 Algorithm for the transformation method for piecewise constant intensities

Algorithm 4 presents the basic transformation method for piecewise constant conditional intensity functions, here represented by the scheduler Q (cf. Equation 2).

The complexity of the GETTNEXT(Q) operation is $O(|Q|)$.

Algorithm 4 Function GETTNEXT(Q) with Q a scheduler storing the events corresponding to a piecewise constant intensity trajectory.

function GETTNEXT(Q)
  $V \sim \mathcal{U}[0,1]$
  $integral \leftarrow 0$
  $t_{next} \leftarrow Q[0].time$
  $k \leftarrow 0$
  $val \leftarrow 0$
  repeat
    $val \leftarrow val + Q[k].value$
    $integral \leftarrow integral + (Q[k+1].time - Q[k].time) \times val$
    $k \leftarrow k + 1$
  until ($integral > -log(V)$ or $k = size(Q) - 1$)
  if $integral \leq -log(V)$ **then**
    $val \leftarrow val + Q[k].value$
  return $t_{next} \leftarrow Q[k].time - \frac{integral+log(V)}{val}$ // $t_{next} \leftarrow +\infty$ if $val = 0$

5.3 Full scan and local graph algorithms

Algorithm 5 is the application of Algorithm 2 to Hawkes processes. The mention to a, b c, d refers to the steps in Algorithm 2. We split step d to lower the complexity. The value T is an arbitrary stopping time, the condition $t_k < T$ ending the main loop in both Algorithm 5 and Algorithm 6 can be changed by whatever condition is deemed more appropriate by the modeller.

Algorithm 5 Full scan algorithm for Hawkes processes

$t_0 \leftarrow 0$
$k \leftarrow 0$
while $t_k < T$ **do**
  Step a/
  $\overline{L}[1] \leftarrow L[1]$
  for all $j \in \{2, ..., M\}$ **do**
    Prune intensities $\overline{L}[j] \leftarrow \overline{L}[j-1] \cup L[j]$
  Step b/
  $t_{k+1} \leftarrow$ GETTNEXT($\overline{L}[M]$)
  Step d1/
  for all $j \in \{1, ..., M\}$ **do**
    $L[j] \leftarrow L[j]^{t_{k+1}}_{pcw}$
  Step c/
  $\ell[0] \leftarrow 0$
  for all $j \in \{1, ..., M\}$ **do**
    compute $\ell[j] = \overline{\lambda}_j(t_{k+1})$ **by** $\ell[j] \leftarrow \ell[j-1] + L[j][0].value$
  Select the associated node i_{k+1} **as the only** j **such that** $\frac{\ell[j-1]}{\ell[M]} < V \leq \frac{\ell[j]}{\ell[M]}$ **for** $V \sim \mathcal{U}[0,1]$
  Step d2/
  for all $j \in \{1, ..., M\}$ **do**
    Update intensities $L[j] \leftarrow L[j] \cup h[i_{k+1}][j]_{\rightarrow t_{k+1}}$
  $k \leftarrow k + 1$
return points $(t_1, ..., t_{k-1})$ **and associated nodes** $(i_1, ..., i_{k-1})$

Algorithm 6 is the application of Algorithm 3 to Hawkes processes.

Algorithm 6 Local graph algorithm for Hawkes processes

$I \leftarrow \{1, ..., M\}, k \leftarrow 0$
while $t_k < T$ **do**
  Step a/ **Compute the next point** $t^i_{next} \leftarrow \text{GETTNEXT}(L[i])$ of each $i \in I$
  Step b/ $Q \leftarrow Q^* \oplus (t^i_{next}, i)$ of each $i \in I$
  Step c/ $t_{k+1} = Q[0].time$
  Step c/ $i_{k+1} = Q[0].value$
  Step d/ **Find children of** i **and update** $I \leftarrow ch(i_{k+1}) \cup \{i_{k+1}\}$
  Step e/ **for all** $i \in ch(i_{k+1})$ **do**
    $L[i] \leftarrow L[i]^{t_{k+1}}_{pcw} \cup h[i_{k+1}][i]_{\rightarrow t_{k+1}}$
  Step e/ **if** $i \notin ch(i_{k+1})$ **then**
    $L[i] \leftarrow L[i]^{t_{k+1}}_{pcw} \cup h[i_{k+1}][i]_{\rightarrow t_{k+1}}$
  $k \leftarrow k + 1$
return $(t_1, ..., t_{k-1})$ points and associated nodes $(i_1, ..., i_{k-1})$

5.4 Complexities of both algorithms

If A (the number of breakpoints to describe the interaction functions $h_{j\rightarrow i}$) and S (the support of the $h_{j\rightarrow i}$'s) are true constants, assumed to be of order 1 in the sequel, the size of the different schedulers that are used in the previous algorithms are most of the time random and changing step after step. They depend in particular on the number of points of N_j appearing in the interaction range that is $N_j([t-S, t))$. To evaluate further the order of such a random quantity, we know that a stationary Hawkes process has a mean intensity vector $m = (m_1, ..., m_M)^T$ (see [10]):

$$m = (I_M - H)^{-1} \nu, \tag{3}$$

with $\nu = (\nu_1, ..., \nu_M)^T$, I_M the identity matrix of size M and $H = (\int_0^{+\infty} h_{j\rightarrow i}(x)\ \mathrm{d}x)_{i,j=1,...,M}$. Note that the linear Hawkes process is explosive when the spectral radius of H is strictly larger than 1 (we refer the reader to [5] and [20] for demonstrations even in the non-linear cases). When the spectral radius is strictly less than 1, the non explosive Hawkes process, with no points before time 0, has always less points than the stationary version. Therefore,

$$\mathbb{E}(N_j([t-S, t))) \leq m_j S \tag{4}$$

with m_j given by Equation (3).

Moreover, the local independence graph for Hawkes process is completely equivalent to the graph with edge $j \rightarrow i$ if and only if $h_{j\rightarrow i}$ is non zero. The corresponding adjacency matrix is denoted $R = (\mathbf{1}_{\int h_{j\rightarrow i} \neq 0})$.

At time t, the scheduler $L[i]$ describes the piecewise constant conditional intensity $\lambda_i(.)$ on $[t, +\infty)$ in absence of new points after t. The number of breakpoints of $L[i]$ is denoted L^i_t. But (1) can be rewritten as

$$\lambda_i(t) = \nu_i + \sum_{j \in pa(i)} \sum_{T \in N_j, T \in [t-S, t)} h_{j\rightarrow i}(t - T)$$

So we can first note that L_t^i and therefore its expectation $\mathcal{L}_i = \mathbb{E}(L_t^i)$ are always larger than 1 because the scheduler $L[i]$ is at least of size 1. Moreover this piecewise constant function has potential breakpoints at all $T+a$, for $T \in N_j, T \in [t-S,t)$, and a breakpoints of $h_{j\to i}$.
Therefore we can compute the order of magnitude of L_t^i, which is the length of the scheduler associated to $\lambda_i(t)$, by

$$L_t^i = O\left(1 + A \sum_{j \in pa(i)} N_j([t-S,t)\right)$$

where O means that there exists an absolute positive constant C such that

$$L_t^i \leq C\left(1 + A \sum_{j \in pa(i)} N_j([t-S,t)\right).$$

In expectation, this gives, thanks to (4) and since $AS = O(1)$,

$$\mathcal{L} = O\left(1 + Rm\right) = O\left(1 + R(I_M - H)^{-1}\nu\right) \tag{5}$$

with $\mathcal{L} = (\mathcal{L}_i)_{i=1,\dots,M}$, the notation O being understood coordinate by coordinate.

Now we can evaluate the (mean) complexity of both algorithms, replacing L_t^i by $\mathcal{L}_i$ thanks to the respective complexities of each operation on the schedulers, see Section 3.

Full-Scan Algorithm. Step a/ has a complexity of

$$O\left(\sum_{j=1}^{M} \mathcal{L}_i \log(\sum_{i=1}^{j} \mathcal{L}_i)\right) = O\left(|\mathcal{L}|_1 \log|\mathcal{L}|_1\right)$$

with $|\mathcal{L}|_1 = \mathcal{L}_1 + \dots + \mathcal{L}_M \geq M$ Step b/ has complexity $O\left(|\mathcal{L}|_1\right)$, as well as Step d1/. Step c/ has complexity $O(M) \leq O\left(|\mathcal{L}|_1\right)$. Step d2/ has complexity

$$O\left(\sum_{j=1}^{M} (A\log(\mathcal{L}_i) + A + \mathcal{L}_i)\right) = O\left(|\mathcal{L}|_1 \log|\mathcal{L}|_1\right)$$

So globally one iteration of the full scan algorithm has a complexity of the order

$$O\left(|\mathcal{L}|_1 \log|\mathcal{L}|_1\right) = O\left((M + |Rm|_1)\log(M + |Rm|_1)\right).$$

Therefore since the mean total number of iterations of this algorithm is also the mean total number of points produced on $[0,T]$, that is $T|m|_1$, the full-scan algorithm should have the following mean complexity

$$O\left(T|m|_1(M + |Rm|_1)\log(M + |Rm|_1)\right). \tag{6}$$

As expected, the complexity is linear with the duration T of the simulation. Moreover this complexity heavily depends on the whole set of parameters (type of graph, strength of the interaction functions etc), because in particular these parameters affect the number of points that have to be produced. So for very unbalanced networks where $|m|_1 = O(1)$ (if for instance only one node in the whole network is clearly active and the others almost silent), the complexity seems to be of order $O(TM\log(M))$. But these very unbalanced networks are not the most usual. Let us look now at more balanced networks. Let us assume that all the m_j's are roughly the same and are of order 1 (no really small m_j) and that

the number of parents of a given node is bounded by d, this give us a complexity of

$$O\left(TM^2 d\log(dM)\right).$$

So up to the log factor, if the network is sparse but balanced, the complexity is quadratic in the number of nodes of the network. If the network is a full complete graph, the complexity is cubic in M.

Local graph algorithm. As before we need to evaluate first the complexity of one iteration of the algorithm. But because I is chosen at step d/ and the size of I impacts the complexity of steps a/b/ and e/, we choose to evaluate the complexity of an iteration which starts with e/ and then does a/b/ c/ and d/, so that that until d/ the set I is the same. If the node $i_{k+1} = j$, then the complexity of step e/ is

$$O\left(\mathcal{L}_j + \sum_{i\in ch(j)} (\mathcal{L}_i + A + A\log(\mathcal{L}_i))\right) = O\left(\mathcal{L}_j + \sum_{i\in ch(j)} (\mathcal{L}_i + \log(\mathcal{L}_i))\right).$$

The complexity of step a/ is

$$O\left(\mathcal{L}_j + \sum_{i\in ch(j)} \mathcal{L}_i\right).$$

The complexity of step b/ is

$$O\left(\log(M) + \sum_{i\in ch(j)} \log(M)\right).$$

Steps c/ and d/ have complexity $O(1)$.

So for one iteration "e/a/b/c/d/" after a point on node j, the complexity is

$$O\left(\mathcal{L}_j + \log(M) + [R'(\mathcal{L} + \log(M)\mathbf{1})]_j\right),$$

with R' the transpose of R and $\mathbf{1}$ the vector of size M full of ones.

The main point is that a point in N_j is appearing in average at most only Tm_j times during the simulation since m_j is the mean intensity of N_j, which leads us to a global complexity of

$$O\left(Tm'\mathcal{L} + T\log(M)|m|_1 + Tm'R'[\mathcal{L} + \log(M)\mathbf{1}]\right) = TO\left(m'Rm + \log(M)|m|_1 + m'R'Rm + \log(M)|Rm|_1\right). \quad (7)$$

As before this is linear in the duration of the simulation T and depends heavily on the parameters. But the complexity is much lower. Indeed, for very unbalanced networks where only one node is really active, the complexity logarithmic in M. For balanced networks where the m_j's are roughly the same and if the number of children of a given node, as well as the number of parents is bounded by d, then we get a complexity of

$$O\left(TMd[d + \log(M)]\right),$$

For sparse balanced graphs, we therefore get a complexity which is linear in M up to logarithmic factors. The gain is clear with respect to the full scan algorithm. For complete graphs, we also get a cubic complexity in terms of M, as the full scan algorithm but without logarithmic factors.

So at least theoretically speaking, it seems that the local graph algorithm is always a better choice than the full-scan algorithm, with a clear decrease of complexity from quadratic to linear in the number of nodes for balanced sparse graphs.

6 NUMERICAL EXPERIMENTS

This section is devoted to two main problems: statistically proving that both algorithm (full scan and local graph) indeed simulate a Hawkes process and asserting that the local graph algorithm clearly outperforms the full scan algorithm.

6.1 Hardware and software specifications

The main simulations have been performed on 5 nodes of a Symmetric MultiProcessing (SMP), i.e., shared memory, computer1. Each of this computational nodes has up to 20 physical cores (210), 25 MB of cache memory and 62.5 GB of RAM. The processors are Intel(R) Xeon(R)CPU E5-2670 (v0 and v2) at 2.60 GHz. The statistical analysis required more RAM, so we used another type of node, which has 770GB of RAM, 25MB of cache memory, 20 physical cores (210), each processor being an Intel(R) Xeon(R)CPU E5-2687W v3 at 3.10GB. The algorithms were implemented in C++ programming language (2011 version). No other external libraries were used for the simulator, which is compiled using gcc 4.7. The plots and statistical analyses were obtained using using R software (v3.6), part of it using the UnitEvent package (v0.0.5).

6.2 Statistical analysis

We generated an Erdös-Rényi network of 100 nodes with connection probability $p = 1/100$, that is fixed for the rest of the statistical analysis. When an edge $j \to i$ is in the graph, we associate it to an interaction function $t \mapsto h_{j\to i}(t) = 5 \cdot \mathbb{1}_{t\in[0,0.02]}$. The spontaneous parameters ν_i are all fixed to 10. Out of this multivariate Hawkes process, we focus on two nodes a and b. The node a is fully disconnected, meaning the corresponding process should be an homogeneous Poisson process of rate 10. The node b is the one with the largest number of parents (4 parents).

Time transformation. In [26], Ogata derives methodological benchmarks to assess if the data are obeying a point process with a given intensity, and in particular Hawkes processes. This is based on the time-rescaling theorem (see for instance [7]), which says that if λ_s is the conditional intensity of the point process N and if $\Lambda(t) = \int_0^t \lambda_s ds$, then the points $\tilde{N} = \{\Lambda(T), T \in N\}$ form an homogeneous Poisson process of rate 1. Ogata proposed the following method to test that a given point process has intensity given by λ_s

- Apply the time-rescaling transformation. This leads to a point process $\tilde{N}$.
- Test that the consecutive delays between points of $\tilde{N}$ obeys an exponential distribution of rate 1, for instance by Kolmogorov-Smirnov test (**Test 1**)
- Test that the points of $\tilde{N}$ themselves are uniformly distributed, for instance by Kolmogorov-Smirnov test (**Test 2**).
- Test that the delays between points of $\tilde{N}$ are independent, for instance by checking that the autocorrelation between delays with a certain lag are null (**Tests 3**). We performed them up to lag 9.

We simulated the multivariate Hawkes process on $[0, T]$ with $T = 150$ and we applied the previous tests to node a and node b.

Table 1. Table of the p-values of a Kolmogorov-Smirnov test (for uniformity) applied to the p-values obtained with tests 1, 2 and 3 for 1000 independent simulations of the same Hawkes point processes (with the same underlying graph).

	FULL-SCAN		LOCAL-GRAPH	
	Node a	Node b	Node a	Node b
Test 1	0.5384525	0.1491268	0.0594925	0.86789804
Test 2	0.6008973	0.2462138	0.1819709	0.99025263
Test 3 with lag 1	0.1602718	0.1781804	0.4385096	0.92162419
Test 3 with lag 2	0.7498109	0.9038829	0.6954876	0.90558993
Test 3 with lag 3	0.5604420	0.7220130	0.4144515	0.77140051
Test 3 with lag 4	0.7003987	0.1838913	0.4367523	0.83833821
Test 3 with lag 5	0.9960351	0.4009543	0.3740874	0.14749913
Test 3 with lag 6	0.1883506	0.1246654	0.4387684	0.12202262
Test 3 with lag 7	0.1259022	0.8588754	0.9114556	0.47030751
Test 3 with lag 8	0.8848928	0.9720601	0.5200698	0.03765871
Test 3 with lag 9	0.2278844	0.3880436	0.5042846	0.92768290

If we have simulated indeed the correct Hawkes processes for the processes associated to node a and b, the p-values should be uniform. So we performed 1000 simulations of the same Hawkes process (with the same underlying graph) but with different pseudorandom generator seeds for the simulation of the points themselves. We can visually check that they are indeed uniform by seeing diagonals for their cumulative distribution functions (see Figures 13 and 14). In order to confirm this qualitative result with a more quantitative one, the p-values for the three tests 1, 2 and 3 are independently tested for uniformity with another Kolmogorov-Smirnov test. The resulting p-values are displayed in Table 1.

Martingale properties. Another very important property of the Hawkes process is that $t \mapsto N_t - \Lambda_t$ is a martingale and this property remains true if we integrate with respect to a predictable process. So for each node a or b, we can compute

$$X^k = \int_0^T \psi_t^k (dN_t - d\Lambda_t),$$

for $\psi_t^1 = 1$ or $\psi_t^{2j} = N_j([t-0.02, t))$ or $\psi_t^{2j+1} = N_j([t-0.04, t-0.02))$. If the martingales properties are true, then the variable X^k for each k should be centered around 0. We also expect eventually different behaviors, when $k = 1$, which corresponds to the spontaneous part or when $k = 2j$ or $2j+1$ for a node j which is connected to the node of interest or disconnected from the node of interest. We simulated the network 40 times on $[0, T]$ with $T = 20$ and reported the X^k. We see on Figure 15 and 16 that the variables X^k are indeed centered in both cases as expected. So we can conclude that both algorithms indeed simulate the given Hawkes process.

6.3 Performance

We want to assess the performances of both algorithm in the main interesting case: sparse balanced networks. To do so, we took three different topologies of graphs:

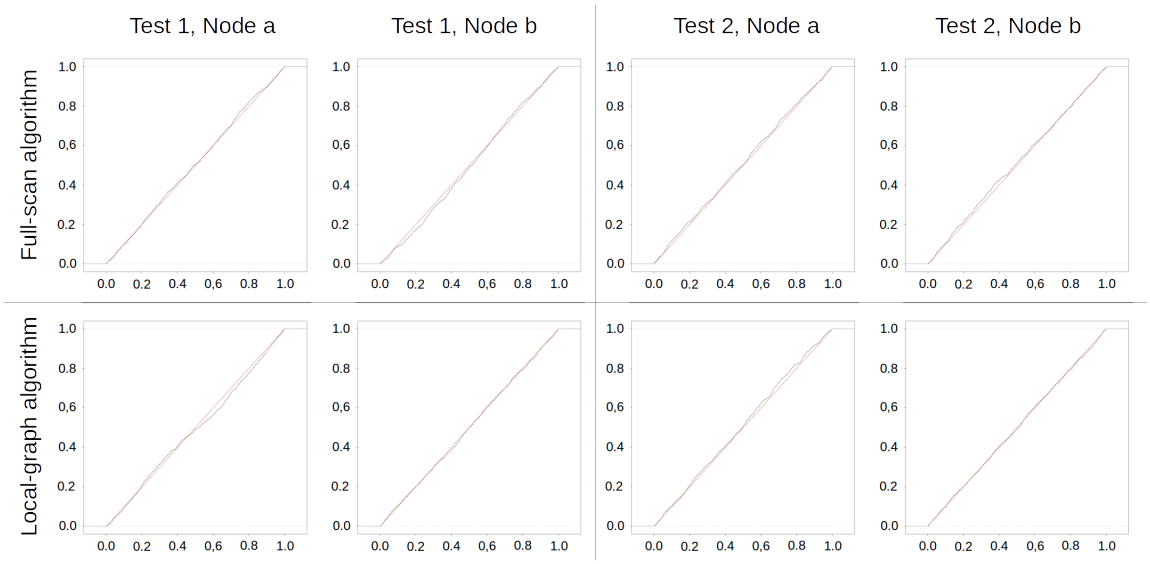

Fig. 13. Cumulative distribution functions of the p-values of Test 1 and 2. In columns the test and node, in rows the algorithms (full scan then local graph)

- Erdös-Rényi: a topology model where each edge has a probability p of being present or absent, independently of the other edges. We took p in $\{0, \frac{1}{M}, \frac{2}{M}, \ldots, \frac{(In(M)-1)}{M}\}$ for M the number of nodes. These choices for p ensure a sparse graph with roughly speaking $d = pM$ parents and children for each node.
- Cascade: a classical topology model where each node has exactly one parent and one child (except for two nodes, start and end, that have respectively no parent and one child, and one parent and zero child), and there are no cycles in the network. There is only one graph per number M.
- Stochastic-Block: it is an Erdös-Rényi by block. In our setting the nodes are partitioned in two blocks and the matrix gives the probability of inter-block connection of intra-block connection (see Table 2).

Table 2. The three bloc sizes vectors (first line) and the three probability matrices (second line) used for the simulations.

Block sizes	$\left(\frac{M}{2} \quad \frac{M}{2}\right)$	$\left(\frac{M}{2} \quad \frac{M}{2}\right)$	$\left(\ln M \quad M - \ln M\right)$
Probability matrices	$\begin{pmatrix} \frac{2}{M}\ln\frac{M}{2} & 0 \\ 0 & \frac{2}{M}\ln\frac{M}{2} \end{pmatrix}$	$\begin{pmatrix} 0 & \frac{2}{M}\ln\frac{M}{2} \\ \frac{2}{M}\ln\frac{M}{2} & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & \frac{\ln(\lceil \ln M \rceil)}{\lceil \ln M \rceil} \\ \frac{\ln(M-\lceil \ln M \rceil)}{(M-\lceil \ln M \rceil)} & 0 \end{pmatrix}$

Each graph was generated using a different pseudorandom generator seed. Each existing edge $j \to i$ is associated with an interaction function $t \mapsto h_{j\to i}(t) = 5 \cdot \mathbb{1}_{t\in[0,0.02]}$. We computed the largest eigen-value of the corresponding matrix H. If it is larger than 1, this graph should be discarded. To force the balance of the network, we decided to take $m = (10, ..., 10)$ and compute the ν_i's by $\nu = (I_M - H)m$. It may happen that some of the ν_i's become negative. These graphs should be also discarded, as by construction the conditional intensity $\lambda(t)$ of a Hawkes point process must remain positive at all time, and so it is the case for the background intensity ν_i (see [4] for reference). Because of the parameter values, especially the interaction functions, no graph was discarded here. A total of $2890 = 1770+280+840$ (Erdös-Rényi +

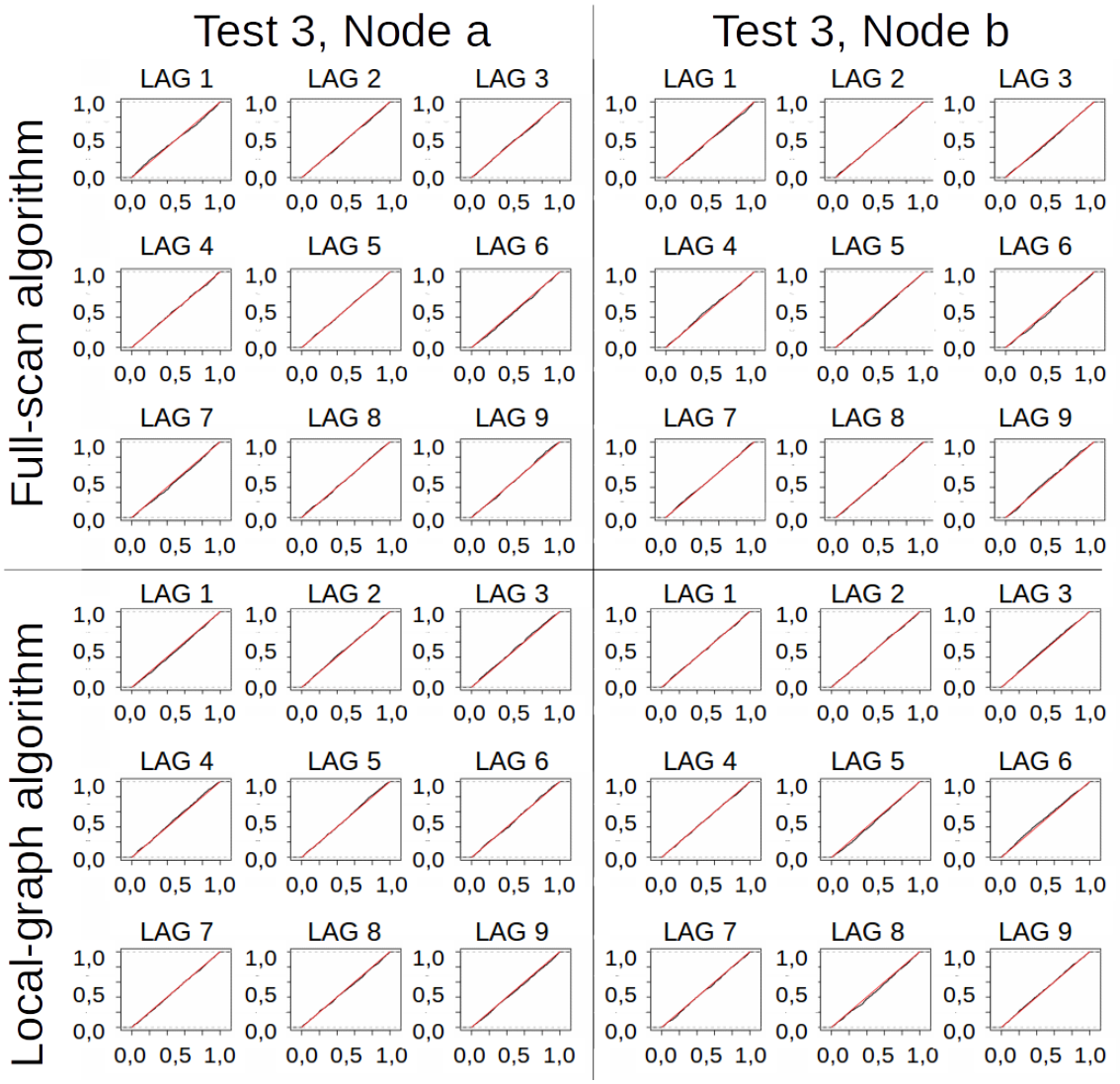

Fig. 14. Cumulative distribution functions of the p-values of Test 3. In columns the node, in rows the algorithm (full scan then local graph)

Cascade + Stochastic-Block) graphs was obtained with $M = \{10, 20, \ldots, 100\} \cup \{150, 200, \ldots, 500\} \cup \{600, 1100, \ldots, 5100\}$ for the local-graph algorithm, and $M = \{10, 20, \ldots, 100\} \cup \{150, 200, \ldots, 500\}$ for the full-scan algorithm. Once the parameters of the Hawkes process are fixed, we simulated 10 times each process on $[0, T]$ with $T = 10$, each simulation with a different generator seed.

Figure17 shows that the theoretical complexities of both full scan and local graph algorithms are equivalent to their actual execution times.

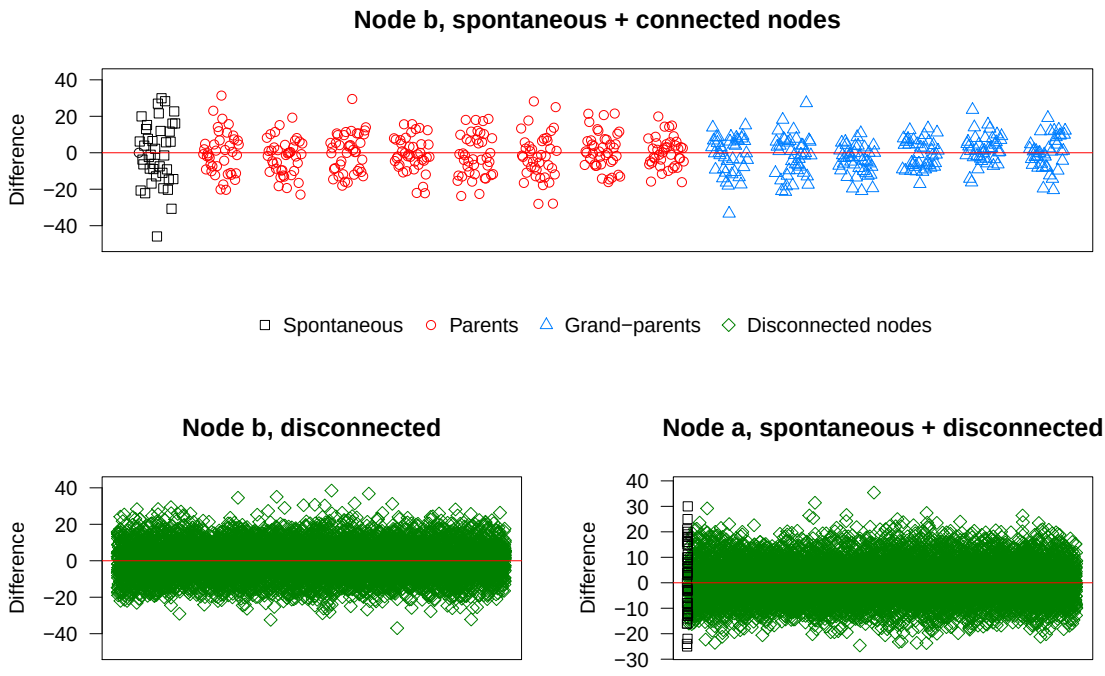

Fig. 15. Full scan algorithm: verifying the Martingale property for Nodes a and b. The black points represent the X^1 (spontaneous), then for the nodes connected to Node b, X^{2j} and X^{2j+1} are displayed in red. In blue are the X^k and X^{k+1} (still for Node b) from Node b's grand-parents to Node b's parents. Finally the two green scatter plots show the Nodes not disconnected from b and a respectively.

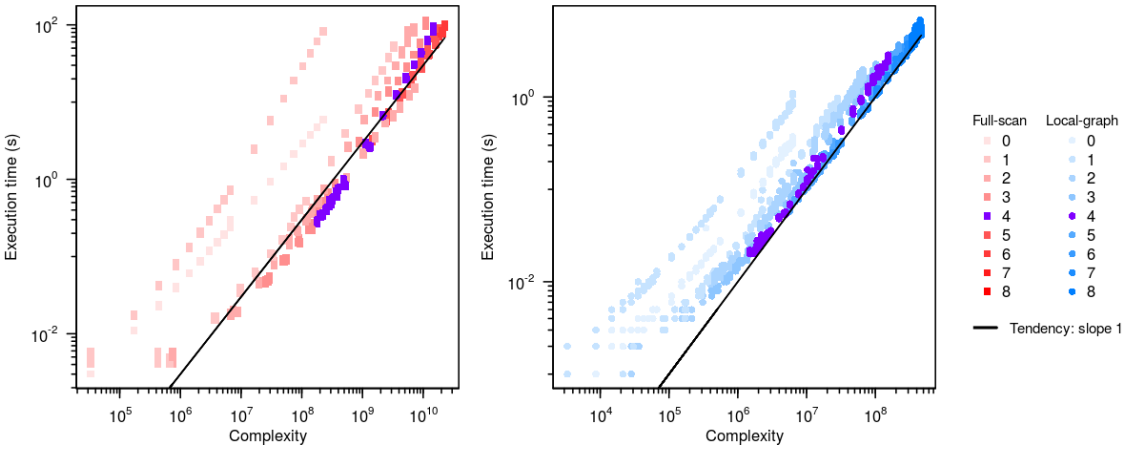

Fig. 17. Three topologies together: the execution time (vertical axis, log-scaled) and the theoretical complexity (horizontal axis, log-scaled), for the full scan algorithm (red squares, left part) and the local-graph algorithm (blue circles, right part). A line of slope 1 is displayed in black, on both scatter plots, showing the equivalence. The colour gradient represents similar values of the mean number of connections per process. A particular value (mean of 4 children per process) is emphasised with a violet tone. The number of nodes is $M = \{10, 20, \ldots, 100\} \cup \{150, 200, \ldots, 500\}$ for the full-scan algorithm and $M = \{10, 20, \ldots, 100\} \cup \{150, 200, \ldots, 500\} \cup \{600, 1100, \ldots, 5100\}$ for the local graph algorithm.

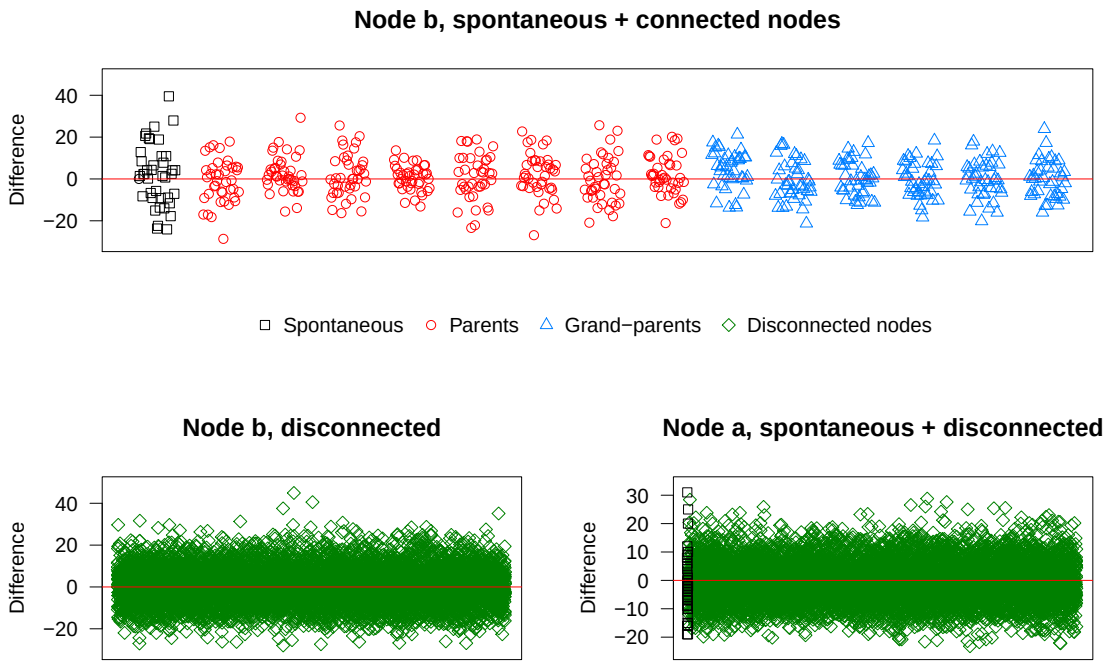

Fig. 16. Local-graph algorithm: verifying the Martingale property for Nodes a and b. The black points represent the X^1 (spontaneous), then for the nodes connected to Node b, X^{2j} and X^{2j+1} are displayed in red. In blue are the X^k and X^{k+1} (still for Node b) from Node b's grand-parents to Node b's parents. Finally the two green scatter plots show the Nodes not disconnected from b and a respectively.

Figure 18 shows that the execution time is quadratic for the full scan algorithm and linear behaviour for the local graph algorithm. For example, when the local graph algorithm is executed in less than 10s for more than 5000 nodes, the execution of the full scan algorithm takes about 100s for 500 nodes. The local graph algorithm clearly outperforms the full scan algorithm.

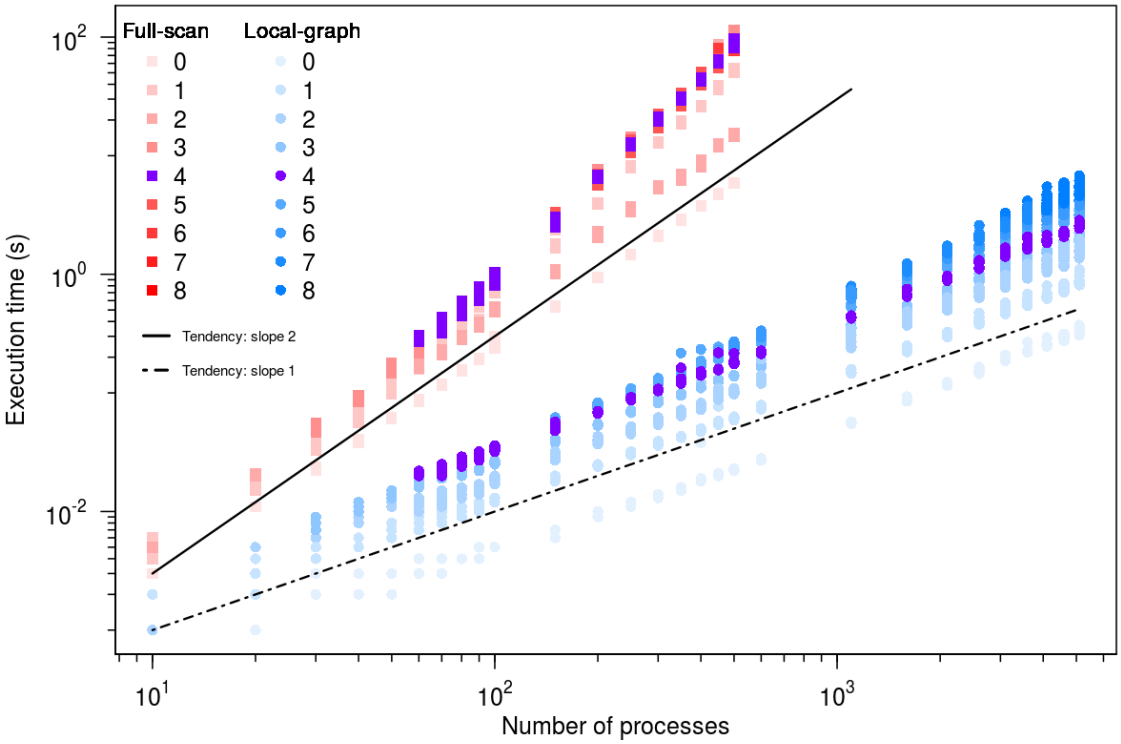

Fig. 18. Three topologies together: the execution time (vertical axis, log-scaled) and the theoretical complexity (horizontal axis, log-scaled), for the full scan algorithm (red squares, left part) and the local-graph algorithm (blue circles, right part). A line of slope 2 is displayed in black, on the scatter plot for the full scan algorithm, and a line of slope 1 for the local graph algorithm. The colour gradient represents similar values of mean numbers of connections per process. A particular value (mean of 4 children per process) is emphasised with a violet tone. The number of nodes is $M = \{10, 20, \ldots, 100\} \cup \{150, 200, \ldots, 500\}$ for the full-scan algorithm and $M = \{10, 20, \ldots, 100\} \cup \{150, 200, \ldots, 500\} \cup \{600, 1100, \ldots, 5100\}$ for the local graph algorithm.

7 CONCLUSION

We presented a new discrete event simulation for point processes: the local graph algorithm, aiming at tracking only the nodes changing state in the network, only updating their children (based on the local independence graph hypothesis [12]). The computational complexity reduction of the local graph algorithm with respect to the full scan algorithm (an adaptation of Ogata's algorithm [25]) is from M^2 to M. Although there was no simulation algorithm able to simulate large point process networks, the local graph algorithm now opens new perspectives for simulating such networks. Especially, based on the local graph generation, an interesting perspective concerns the memory reduction. Instead of statically storing the whole network topology in memory at the beginning of the simulation, only the local graphs corresponding to the children of changing state nodes could be dynamically generated during the simulations [16]. Based on time asynchrony, this corresponds to asynchronous dynamic structure changes in discrete event systems [23]. Generating only local graphs with respect to the whole network graph should allow simulating very large networks. The same complexity reduction order is expected. However, at execution time level, the cost of re-generating the local graphs will have to be taken into account.

The network structure plays a central role in the arguments. While we assume that all processes in the population are of the same type, the connectivity between the processes in the population is not homogeneous. Each process in the population of N nodes receives input from C randomly selected processes in the population. Sparse connectivity means that the ratio $\delta = \frac{C}{N} \ll 1$ is a small number. One can ask if it is this realistic. In the context of the human brain, a typical pyramidal neuron in the cortex receives several thousand synapses from presynaptic neurons while the total number of neurons in the cortex is much higher [27]. Thus globally the cortical connectivity $\frac{C}{N}$ is low. On the other hand, we may concentrate on a single column in visual cortex and define, e.g., all excitatory neurons in that column as one population. We estimate that the number N of neurons in one column is below ten thousand. Each neuron receives a large number of synapses from neurons within the same column. In order to have a connectivity ratio of 0.1, each neuron should have connections to about a thousand other neurons in the same column. [14]. In the brain, last estimations consist of 86 billions of neurons [33], each neuron having around 7′000 connections. Either for the overall brain or for a single column of the visual cortex, the hypothesis of sparse connectivity of the network remains valid. This work thus allows achieving grounded stochastic simulations of the neuronal functional interactions in parts of the human brain.

ACKNOWLEDGEMENT

For the SMP simulations we would like to deeply thank the LIMOS CNRS laboratory, from the University of Clermont-Auvergne, which graciously provided access. In particular we would like to thank their current administrators, Hélène Toussaint, William Guyot-Lénat and Boris Lonjon, for their valuable help.
This work was supported by the French government, through the UCA$^{\text{Jedi}}$ and 3IA Côte d'Azur Investissements d'Avenir managed by the National Research Agency (ANR-15- IDEX-01 and ANR-19-P3IA-0002) and by the interdisciplinary Institute for Modeling in Neuroscience and Cognition (NeuroMod) of the Université Côte d'Azur.

REFERENCES

[1] P.K. Andersen, O. Borgan, R. Gill, and N. Keiding. 1996. Statistical Models Based on Counting Processes. Springer.
[2] M. Barrio, K. Burrage, A. Leier, and T. Tian. 2006. Oscillatory Regulation of hes1: Discrete Stochastic Delay Modelling and Simulation. PLoS Computational Biology 2, 9 (2006), 1017.
[3] A. Bouchard-Côté, S. J. Vollmer, and A. Doucet. 2018. The Bouncy Particle Sampler: A Non-Reversible Rejection Free Markov chain Monte Carlo Method. J. Amer. Statist. Assoc. 113 (2018), 855–867.
[4] P. Brémaud. 1981. Point Processes and Queues. Springer New York. https://books.google.fr/books?id=lo-YZwEACAAJ
[5] Pierre Bremaud and Laurent Massoulie. 1996. Stability of Nonlinear Hawkes Processes. The Annals of Probability 24, 3 (1996), 1563–1588. http://www.jstor.org/stable/2244985
[6] E.N. Brown, R. Barbieri, V. Ventura, R.E. Kass, and L.M. Frank. 2006. The Time-Rescaling Theorem and Its Application to Neural Spike Train Data Analysis. Neural Computation 4, 2 (2006), 325–346.
[7] E. N. Brown, R. Barbieri, V. Ventura, R. E. Kass, and L.M. Frank. 2002. The Time-Rescaling Theorem and Its Application to Neural Spike Train Data Analysis. Neural Computation 14, 2 (2002), 325–346.
[8] J.H. Cha and M. Finkelstein. 2018. Point Processes for Reliability Analysis. Springer.
[9] J. Chevallier, M.J. Cáceres, M. Doumic, and P. Reynaud-Bouret. 2015. Microscopic approach of a time elapsed neural model. Mathematical Models and Methods in Applied Sciences 25, 14 (2015), 2669–2719.
[10] Daryl J Daley and David Vere-Jones. 2003. An introduction to the theory of point processes. Vol. I. Probability and its Applications.
[11] A. Dassios and H. Zhao. 2013. Exact simulation of Hawkes process with exponentially decaying intensity. Electronic Communications in Probability 18, 62 (2013).
[12] V. Didelez. 2008. Graphical models of markes point processes based on local independence. J.R. Statist. Soc. B 70, 1 (2008), 245–264.
[13] J.L. Doob. 1945. Markoff chains – Denumerable case. Trans. Amer. Math. Soc. 58, 3 (1945), 455–473.
[14] W. Gerstner and W.M. Kistler. 2002. Spiking neuron models: Single neurons, populations, plasticity. Cambridge university press.
[15] D.T. Gillespie. 1977. Exact Stochastic Simulation of Coupled Chemical Reactions. he Journal of Physical Chemistry 81, 25 (1977), 2340–2361.
[16] P. Grazieschi, M. Leocata, C. Mascart, J. Chevallier, F. Delarue, and E. Tanré. 2019. Network of interacting neurons with random synaptic weights. ESAIM: ProcS 65 (2019), 445–475. https://doi.org/10.1051/proc/201965445

[17] D.A. Heger. 2004. A Disquisition on the Performance Behavior of Binary Search Tree Data Structures. 5, 5 (2004), 67–75. https://web.archive.org/web/20140327140251/http://www.cepis.org/upgrade/files/full-2004-V.pdf

[18] S. Herculano-Houzel and R. Lent. 2005. Isotropic Fractionator: A Simple, Rapid Method for the Quantification of Total Cell and Neuron Numbers in the Brain. Journal of Neuroscience 25, 10 (2005), 2518–2521. https://doi.org/10.1523/JNEUROSCI.4526-04.2005 arXiv:https://www.jneurosci.org/content/25/10/2518.full.pdf

[19] P.A.W. Lewis and G.S. Shedler. 1978. Simulation of nonhomogeneous Poisson processes. Technical Report. Naval Postgraduate School, Monterey, California.

[20] Iacopo Mastromatteo, Emmanuel Bacry, and Jean-Fran çois Muzy. 2015. Linear processes in high dimensions: Phase space and critical properties. Phys. Rev. E 91 (Apr 2015), 042142. Issue 4. https://doi.org/10.1103/PhysRevE.91.042142

[21] M.D. Mesarovic and Y. Takahara. 1975. General systems theory: mathematical foundations. Vol. 113. Academic press.

[22] A. Muzy. 2019. Exploiting Activity for the Modeling and Simulation of Dynamics and Learning Processes in Hierarchical (Neurocognitive) Systems. Computing in Science Engineering 21, 1 (Jan 2019), 84–93. https://doi.org/10.1109/MCSE.2018.2889235

[23] Alexandre Muzy and Bernard P Zeigler. 2014. Specification of dynamic structure discrete event systems using single point encapsulated control functions. International Journal of Modeling, Simulation, and Scientific Computing 5, 03 (2014), 1450012.

[24] J-F. Muzy, E. Bacry, S. Delattre, and Hoffmann M. 2013. Modelling microstructure noise with mutually exciting point processes. Quantitative Finance 13, 1 (2013), 65–77.

[25] Y. Ogata. 1981. On Lewis' simulation method for point processes. IEEE Transaction on Information Theory 27, 1 (1981), 23–31.

[26] Y. Ogata. 1985. Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes. J. Amer. Statist. Assoc. 83, 401 (1985), 9–27.

[27] B. Pakkenberg, D. Pelvig, L. Marner, M.J. Bundgaard, H.J.G. Gundersen, J.R. Nyengaard, and L. Regeur. 2003. Aging and the human neocortex. Experimental Gerontology 38, 1 (2003), 95 – 99. https://doi.org/10.1016/S0531-5565(02)00151-1 Proceedings of the 6th International Symposium on the Neurobiology and Neuroendocrinology of Aging.

[28] E.A.J.F. Peters and G. de With. 2012. Rejection-free MonteCarlo sampling for general potentials. Physical Review E 85, 026703 (2012).

[29] P. Reynaud-Bouret, V. Rivoirard, and C. Tuleau-Malot. 2013. Inference of functional connectivity in Neurosciences via Hawkes processes. 1st IEEE Global Conference on Signal and Information Processing, Austin, Texas.

[30] P. Reynaud-Bouret and S. Schbath. 2010. Adaptive estimation for Hawkes processes; application to genome analysis. Annals of Statistics 38, 5 (2010), 2781–2822.

[31] K.D. Tocher. 1967. PLUS/GPS III Specification. Technical Report. United Steel Companies Ltd, Department of Operational Research, Sheffield.

[32] D. Vere-Jones and T. Ozaki. 1982. Some examples of statistical estimation applied to earthquake data. Ann. Inst. Statist. Math. 34, B (1982), 189–207.

[33] C.S. von Bartheld, J. Bahney, and S. Herculano-Houzel. 2016. The search for true numbers of neurons and glial cells in the human brain: A review of 150 years of cell counting. Journal of Comparative Neurology 524, 18 (2016), 3865–3895.

[34] B.P. Zeigler. 1976. Theory of Modelling and Simulation. John Wiley. https://books.google.fr/books?id=M-ZQAAAAMAAJ

[35] B.P. Zeigler, A. Muzy, and E Kofman. 2018. Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations. Academic Press.