



Discrete event simulation of point processes: A computational complexity analysis on sparse graphs

Cyrille Mascart, Alexandre Muzy, Patricia Reynaud-Bouret

► To cite this version:

Cyrille Mascart, Alexandre Muzy, Patricia Reynaud-Bouret. Discrete event simulation of point processes: A computational complexity analysis on sparse graphs. 2020. hal-02455422v1

HAL Id: hal-02455422

<https://hal.science/hal-02455422v1>

Preprint submitted on 26 Jan 2020 (v1), last revised 4 Mar 2021 (v2)

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.

Discrete event simulation of point processes: Computational complexity analysis on sparse graphs

CYRILLE MASCART, Université Côte d'Azur

ALEXANDRE MUZY, Laboratoire I3S, Université Côte d'Azur, France and CNRS

PATRICIA REYNAUD-BOURET, Laboratoire Jean-Alexandre Dieudonné, Université Côte d'Azur, 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 [10], with the activity tracking algorithm [18] 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** → **Discrete mathematics**; *Statistical software*; *Mathematical software performance*;

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. 2020. Discrete event simulation of point processes: Computational complexity analysis on sparse graphs. *ACM Trans. Algor.* 1, 1, Article 1 (January 2020), 28 pages.

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2020 Association for Computing Machinery.

Manuscript submitted to ACM

Manuscript submitted to ACM

CONTENTS

Abstract	1
Contents	2
1 Introduction	3
2 Set-up	4
2.1 Mathematical framework	4
2.2 Simulation of univariate processes	6
2.3 Discrete event version of classical multivariate algorithm for point processes	7
3 Specific discrete event data structures and operations	8
4 Local graph algorithm for point processes	11
4.1 Local independence graph	11
4.2 Local-graph algorithm	12
5 Hawkes evaluation	13
5.1 Notations and data structures	13
5.2 Algorithm for the transformation method for piecewise constant intensities	14
5.3 Full scan and local graph algorithms	15
5.4 Complexities of both algorithms	16
6 Numerical experiments	19
6.1 Hardware and software	19
6.2 Statistical analysis	19
6.3 Performance	24
7 Conclusion	26
References	27

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, 7, 10, 19, 25, 27]).

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 children birth dates 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 $j = 1, \dots, M$ and the joint process $N = N_1 \cup \dots \cup N_M$. 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 [24]. 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 [28], 10^8 for rats [15], 10^6 for cockroaches), each cell receives approximately informations from 10^3 to 10^4 presynaptic (upstream) neurons [22]. 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 [10], 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.

Simulation of point processes have a long history that dates back to Doob in the 40's [11] for Markov processes. In the 70's, Gillespie [13] popularized the method for a particular application: chemical reactions. At the same time, Lewis and Shedler [16] proposed a thinning algorithm for simulating univariate inhomogeneous Poisson processes (this can also be viewed as a rejection method). Few years later, Ogata [20] 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 [23]

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) [9], 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 [26] 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 [29]. In its principles, discrete event modelling and simulation seems 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 [17] integrating more and more formally stochastic simulation concepts [30]. We hope to show here that both domains can take advantage from each other. Especially, whereas discrete event simulation algorithms have been developed considering independently the components (nodes) of a system, a new algorithm for activity tracking simulation [18] have been proposed to track activity (events from active nodes to children). The activity tracking algorithm was 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 simulate 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 [20] is explained. 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 what is a local independence graph [10], 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 [24]. 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 set A , $N(A)$ is the number of points that lie in A . It is often associated to a corresponding counting process $(N_t)_{t \in \mathbb{R}_+}$.

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 by

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

for infinitesimal dt . 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 $N = N_1 \cup \dots \cup N_M$ and for each $t_k \in N$, there exists in fact only one $j = j_k$ such that $t_k \in N_j$. This is the mark j_k associated to the k -th point of N . We are given the set of intensities of each of the N_j , $\lambda_j(t)$ 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 $(v_i)_{i=1, \dots, M}$. In this case, all λ_i are constant and not even random and for all i ,

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

We see in this expression, that the intensities do not depend on time, or on the previous occurrences. This is why one often refer to such dynamics as “memory loss”. 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 v_i , independently from anything else. To unify frameworks, this exponential variable might also be seen as $-\log(U)/v_i$, with U a uniform variable on $[0, 1]$.

- **Inhomogeneous Poisson processes** with time-dependent rates $(f_i)_{i=1, \dots, M}$. 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

$$\int_{t_k^i}^{t_{k+1}^i} f_j(s) \, ds = -\log(U)$$

- **Linear multivariate Hawkes process** with spontaneous parameter $(v_j)_{j=1, \dots, M}$ and non negative interaction functions $(h_{j \rightarrow i})_{i, j=1, \dots, M}$ on \mathbb{R}_+ . This process has intensity

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

This process is used for many excitatory systems, especially the ones modelling the spiking activity of neurons [24]. It can be interpreted in this sense, informally: to a homogeneous Poisson process of rate v_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 $h_{j \rightarrow i}(\delta)$, 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 $h_{j \rightarrow i}$'s are exponentially decreasing [9], 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 [5] or [4] for more mathematical insight) states that if a point process N has a conditional intensity $\lambda(t)$, and if

$$\forall t, \Lambda(t) = \int_0^t \lambda(s) \, ds,$$

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 $k \in \mathbb{N}_*$,

$$\int_{t_k}^{t_{k+1}} \lambda(s) \, ds = -\log(U) \quad (2)$$

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 $(t_k, +\infty)$ if no other point occurs is doable.

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 [16], it has been generalized to general processes by Ogata [20]. One can find a complete proof in [8].

Algorithm 1 Thinning algorithm

```

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

```

2.3 Discrete event version of classical multivariate algorithm for point processes

To simulate multivariate processes, Ogata's algorithm [20] is usually used but only applies to thinning procedures and constant intensities for λ^* . Based on a discrete event scheduling strategy, this algorithm can be generalised and simplified as proposed in Algorithm 2. This algorithm is called full scan because the intensities of every node of the graph need to be scanned and updated at each time stamp t_k . The main steps of this new 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

- 1: $t_0 \leftarrow 0$
 - 2: **while** $t_k < T$ **do**
 - Step a/ 3: **Compute intensity sums** $\sum_{j=1}^i \lambda_j(t) = \bar{\lambda}_i(t)$, for $i \in \{1, \dots, M\}$ on $t \in (t_k, +\infty)$
 - Step b/ 4: **Get** by simulation t_{k+1} as the **next point** of a univariate point process of intensity $\bar{\lambda}_M(t)$
 - Step c/ 5: **Select the associated node** i_{k+1} such that $\frac{\bar{\lambda}_{i-1}(t_{k+1})}{\bar{\lambda}_M(t_{k+1})} < V \leq \frac{\bar{\lambda}_i(t_{k+1})}{\bar{\lambda}_M(t_{k+1})}$, with $V \sim \mathcal{U}[0, 1]$ and $\bar{\lambda}_0 = 0$
 - Step d/ 6: **Update intensities** λ_j on $(t_{k+1}, +\infty)$, $\forall j \in \{1, \dots, M\}$
 - 7: $k \leftarrow k + 1$
 - 8: **return** points (t_1, \dots, t_{k-1}) and associated nodes (i_1, \dots, i_{k-1})
-

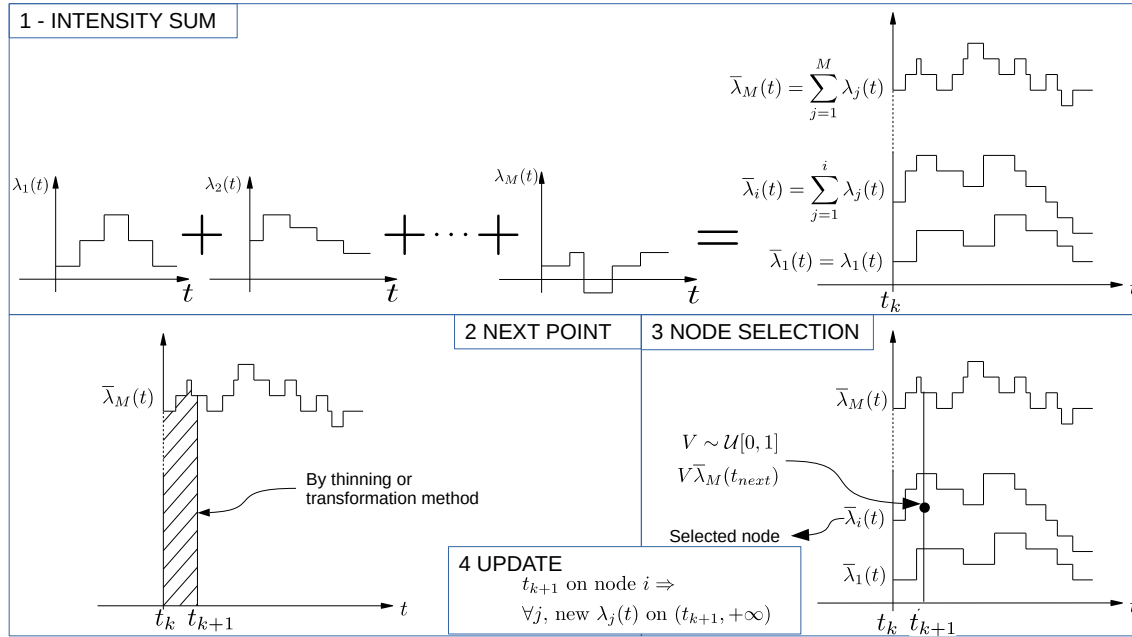


Fig. 1. Steps of the full scan algorithm for point processes. The intensities are piecewise constant (cf. Section 5).

Original Ogata's algorithm was using 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 an ordered set of events, $ev_i = (t_i, v_i)$, 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 $|Q|$. As shown in Figure 2,

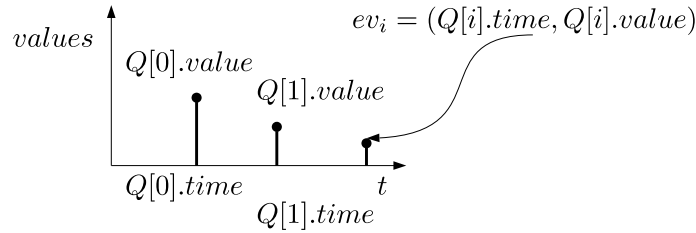


Fig. 2. Example of events in scheduler Q . For an event ev_i in the scheduler, the value is accessed by $Q[i].value$ and the time is accessed by $Q[i].time$, with $i = 0$ the first event index.

In both full scan and local graph algorithms schedulers are used. They are implemented using a self-balancing binary tree. The operation costs of self-balancing binary trees is bounded by the logarithm (\log_2) of the number of elements in the set. Any element of the scheduler can be accessed linearly. Depending on the programming language used it may not be the case. However, this is the case when using C++ programming language, which is the one used here.

Basic scheduler operations consist of:

Insert operation of an event (cf. Figure 3): $Q \oplus (t, v)$, which has complexity $O(\log_2(|Q|))$ to find the place of the event.

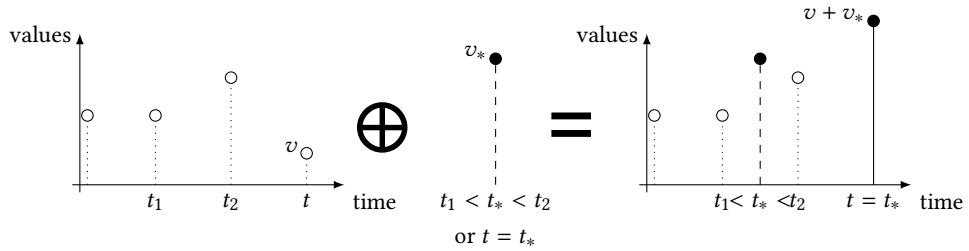


Fig. 3. A graphical example of the insertion of a new event (t_*, v_*) inside a scheduler. There are two cases: if the event time is unmatched in the set of event times already present in the scheduler (case $t_1 < t_* < t_2$), the event is just inserted in the right place; otherwise (case $t = t_*$) 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): $Q \ominus (t, v)$, which has complexity $O(\log_2(|Q|))$ to find the event.

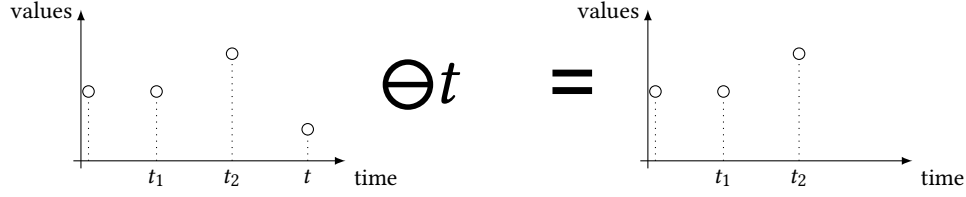


Fig. 4. A graphical example of the removal of an event given its time t .

Remove first operation over the scheduler Q^* , which removes the first event $ev_1 = (Q[0].time, Q[0].value)$ from the scheduler, the second event becoming the first. Operation Q^* has complexity $O(1)$.

Prune operation of the scheduler Q^t (cf. Figure 5): The operation Q^t has complexity $O(\log_2(|Q|))$ (no suppression cost).

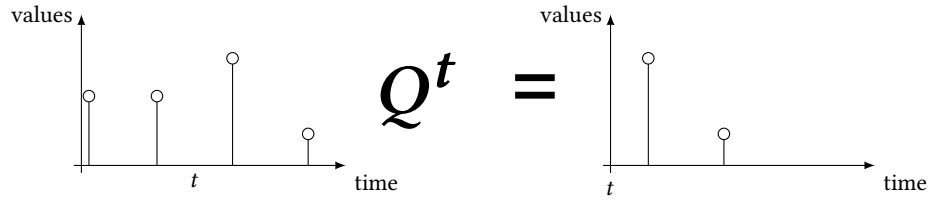


Fig. 5. A graphical example of the prune operation : Here all the points with a time less or equal to t (actually the first two points of this scheduler) are removed.

Upper and lower bound operations (cf. Figure 6): Upper bound operation: $\lceil t \rceil_Q$ and lower bound operation $\lfloor t \rfloor_Q$, which have complexity $O(\log_2(|Q|))$.

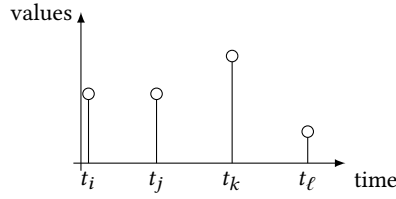


Fig. 6. Upper/lower bound operations on an example of scheduler Q . For $t_k < t < t_\ell$ the upper bound operation consists of $\lceil t \rceil_Q = \ell$. The lower bound operation consists of $\lfloor t \rfloor_Q = k$.

Shift operation of the scheduler $Q_{\rightarrow t}$ (cf. Figure 7): The operation $Q_{\rightarrow t}$ has complexity $O(|Q|)$

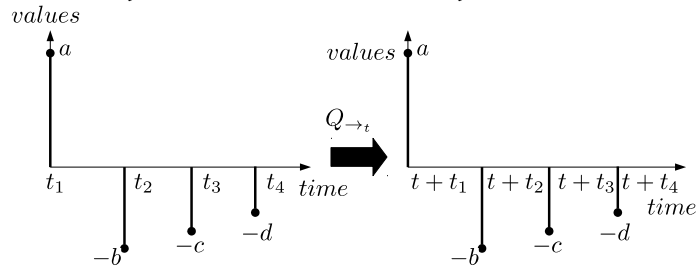


Fig. 7. The shift operation $Q_{\rightarrow t}$.

A scheduler can encode piecewise constant functions on $[t, +\infty)$ by having its first event.time at t with value the value of the piecewise constant function h at t , the other event.time t_k corresponds to the other breaks of h and the other event.value at time t_k correspond to $h(t_{k+1}) - h(t_k)$.

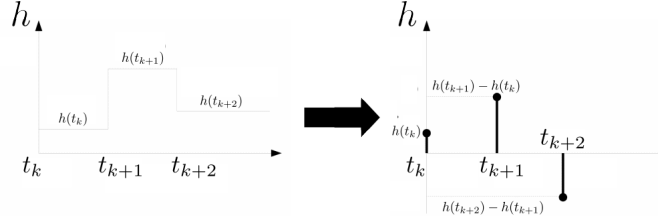


Fig. 8. Optimized encoding of a piecewise constant function h (left) as a list of breakpoints (right).

Note in particular that with this encoding of a piecewise constant function, the union of two schedulers correspond to the addition of two piecewise constant functions.

Union operation over two schedulers $Q \cup Q'$ (cf. Figure 9): The operation $Q \cup Q'$ has complexity $\mathcal{O}(\min(|Q|, |Q'|) \log_2(\max(|Q|, |Q'|)))$.

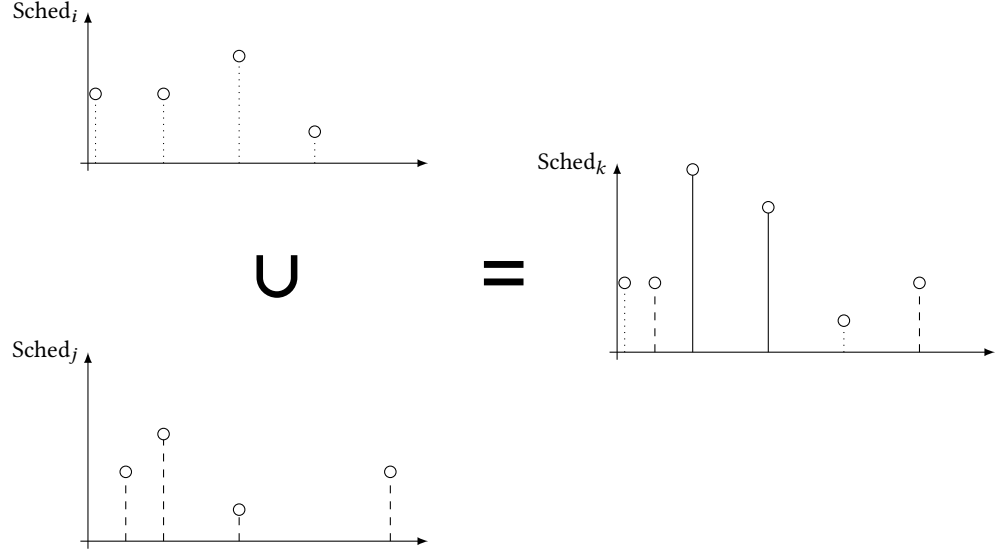
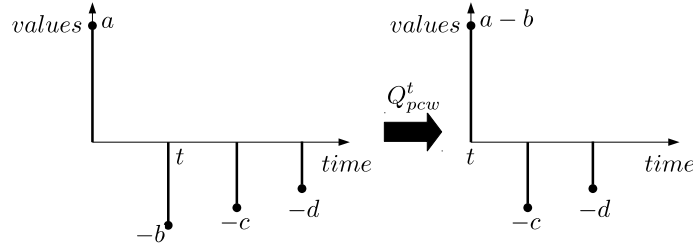


Fig. 9. A graphical example of the union of two schedulers. The events of Sched_i are represented as dotted, while the events of Sched_j are dashed. In the merged scheduler Sched_k , the values of events at the same time in both schedulers i and j are summed and the resulting event is represented with a continuous line.

Piece-wise prune operation of the scheduler Q_{pcw}^t (cf. Figure 10): The operation Q_{pcw}^t has complexity $\mathcal{O}(\log_2(|Q|) + \lceil t \rceil_Q)$. The piecewise prune operation is the prune operation Q^t but in addition at time t , there is an event with value, the sum of all the event values up to time t .

Fig. 10. Piece-wise shift operation Q_{pcw}^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 [10]. For a given multivariate point process $(N_j), j = 1, \dots, M$, the corresponding local independence graph is a directed graph between the nodes $j = 1, \dots, M$ (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 $N = N_1 \cup \dots \cup N_M$ 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 $I \subset \{1, \dots, M\}$, \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 $j \rightarrow i$ means that the apparition of a point at time t on i is independent from $\mathcal{F}_{t-}^{\{j\}}$ conditionally to $\mathcal{F}_{t-}^{\{j\}^c}$, where $\{j\}^c = \{1, \dots, M\} \setminus \{j\}$.

So this means that for every time t , the intensity $\lambda_i(t)$ of N_i with respect to $(\mathcal{F}_t)_{t \geq 0}$ 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

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

Therefore it means that the intensity $\lambda_i(t)$ at time t of N_i with respect to $(\mathcal{F}_t)_{t \geq 0}$ 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 $j \notin ch(i) \cup \{i\}$.

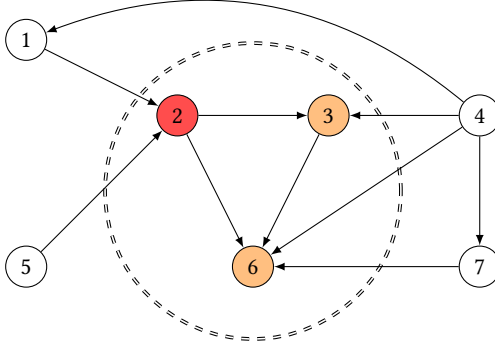


Fig. 11. Example of local independence graph. With this graph, $ch(2) = \{3, 6\}$ and $pa(2) = \{1, 5\}$. As indicated by the difference of colour, a point with mark 2 shall impact the point generation only for $\{2\} \cup \{3, 6\}$.

4.2 Local-graph algorithm

The children of each node is stored in a simple one-dimension array, whose indexes are the node indexes and elements are a vector of the children indexes of the node. So accessing a node simply costs $\mathcal{O}(1)$.

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 $ev_i = (t_{next}^i, i)$, where t_{next}^i 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. More details about the algorithm steps are provided in Section 5, which presents 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 [18].

```

1:  $t_0 \leftarrow 0$ 
2:  $I \leftarrow \{1, \dots, M\}$ 
3: while  $t_k < T$  do
Step a/ 4:   Compute the next possible points  $t_{next}^i$  for each  $i \in I$  based on intensity  $\lambda_i$  on  $(t_k, +\infty)$ 
Step b/ 5:   Update  $P$  with each next possible point  $t_{next}^i$  for each  $i \in I$ 
Step c/ 6:   Get next selected point  $t_{k+1} \leftarrow \min\{t_{next}^i\}$  and  $i$  the associated node, updating  $Q \leftarrow Q^*$ 
Step d/ 7:   Find corresponding children and update  $I \leftarrow ch(i) \cup \{i\}$ 
Step e/ 8:   Update intensities  $\lambda_j(t)$  for each node  $j \in I$  on  $(t_{k+1}, +\infty)$ 
9:    $k \leftarrow k + 1$ 
10: return  $(t_1, \dots, t_{k-1})$  points and associated nodes  $(i_1, \dots, i_{k-1})$ 

```

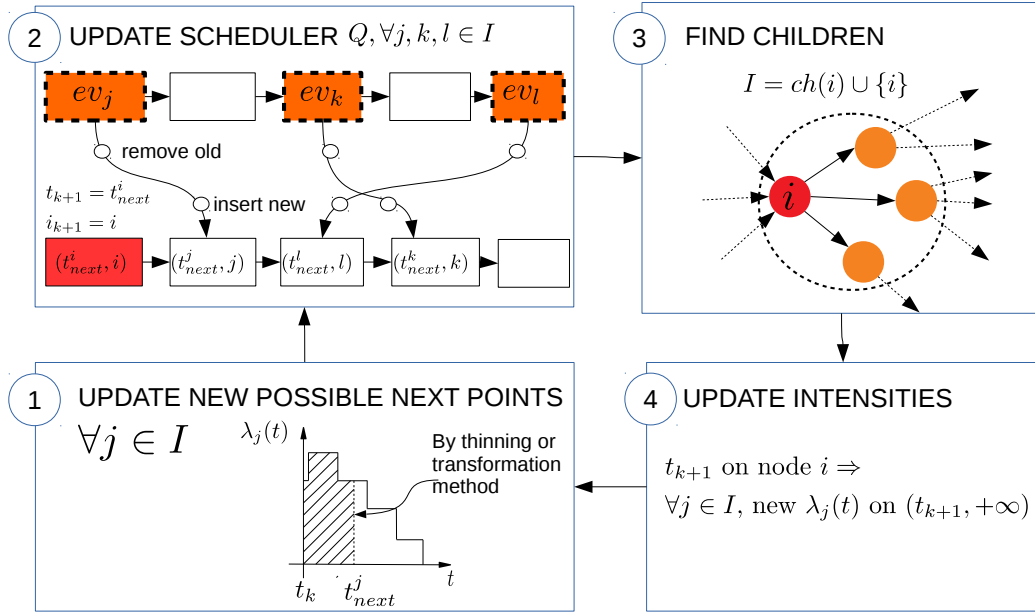


Fig. 12. Steps of the local graph algorithm for the simulation of point processes. As for figure 1, the intensities are piecewise constant.

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 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 Notations and data structures

Data structure oriented notations consist of:

- P : A scheduler of next point events $ev_i = (t_{next}^i, i)$, where t_{next}^i is a possible next point associated to node i .
- $L[i](\leftrightarrow \lambda_i)$: is the scheduler of discrete event intensities corresponding to the piece-wise constant intensity of node i , with intensity events $ev_k = (t_k, \delta_k)$, and $\delta_k = \lambda_i(t_k) - \lambda_i(t_{k-1})$ the intensity difference at time t_k (cf. Figure 13). The length of the scheduler is $L_t^i = \text{length}(L[i])$ when $L[i][0].time = t$.

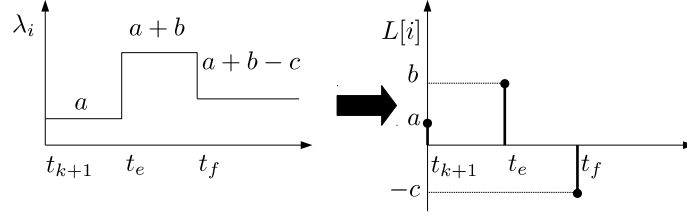


Fig. 13. Discrete event mapping of a piece-wise constant intensity trajectory.

- $h[j][i] (\leftrightarrow h_{j \rightarrow i})$: is the scheduler of discrete event interactions corresponding to the piece-wise constant interaction $h_{j \rightarrow i}$ (with support included in $[0, S]$) from node j to node i and interaction events $ev_k = (t_k, \delta_k)$, with $\delta_k = h_{j \rightarrow i}(t_k) - h_{j \rightarrow i}(t_{k-1})$ the interaction difference at time t_k (cf. Figure 14). The maximum number of events in $h[j][i]$ is noted $A \geq \text{length}(h[j][i])$.

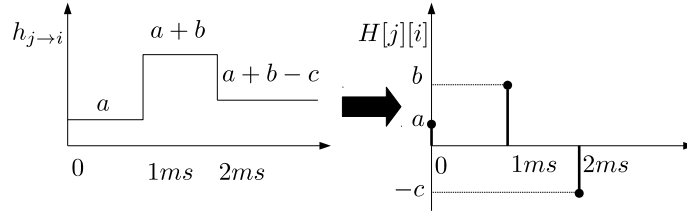


Fig. 14. Discrete event mapping of a piece-wise constant interaction trajectory.

- $\bar{L} = L[1] \cup \dots \cup L[M] (\leftrightarrow \bar{L} \leftrightarrow \bar{\lambda}(t) = \sum_{i=1}^M \lambda_i(t))$: is a scheduler storing the union of all intensity events of all $L[i]$, from node 1 to node M (cf. Figure 9). This is the sum of all piece-wise constant trajectories.
- $\bar{L}[i] = L[1] \cup \dots \cup L[i] (\leftrightarrow \bar{L}[i] \leftrightarrow \bar{\lambda}_i(t) = \sum_{j=1}^i \lambda_j(t))$: is a scheduler storing the intermediate union of intensity events of nodes 1 to node i . Obviously, $\bar{L}[M] = \bar{L}$.

5.2 Algorithm for the transformation method for piecewise constant intensities

Algorithm 4 presents the basic transformation method for piecewise constant intensities.

Algorithm 4 Function `GETNEXT(Q)` with Q a scheduler storing the events corresponding to a piece-wise constant intensity trajectory (cf. Figure 13).

```

1: function GETNEXT(Q)
2:    $V \sim \mathcal{U}[0, 1]$ 
3:    $integral \leftarrow 0$ 
4:    $t_{next} \leftarrow Q[0].time$ 
5:    $k \leftarrow 0$ 
6:    $val \leftarrow Q[0].value$ 
7:   repeat
8:      $val \leftarrow val + Q[k].value$ 
9:      $integral \leftarrow integral + (Q[k+1].time - Q[k].time) \times val$ 
10:     $k \leftarrow k + 1$ 
11:  until ( $integral > -\log(V)$  or  $k = \text{size}(Q)$ )
12:  if  $integral \leq -\log(V)$  then
13:     $val \leftarrow val + Q[k].value$ 
    return  $t_{next} = Q[k].time - \frac{integral + \log(V)}{val}$ 

```

The complexity of the `GETNEXT(Q)` operation is $\mathcal{O}(|Q|)$.

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 Algo2. We splitted step d to lower the complexity.

Algorithm 5 Full scan algorithm for Hawkes processes

```

1:  $t_0 \leftarrow 0$ 
2: while  $t_k < T$  do
3:    $\bar{L}[1] \leftarrow L[1]$ 
4:   for all  $j \in \{2, \dots, M\}$  do
5:      $\bar{L}[j] \leftarrow \bar{L}[j-1] \cup L[j]$ 
6:    $t_{k+1} \leftarrow \text{GETNEXT}(\bar{L}(M))$ 
7:   for all  $j \in \{1, \dots, M\}$  do
8:      $\bar{L}[j] \leftarrow \bar{L}[j]_{pcw}^{t_{k+1}}$ 
9:     compute  $\ell[j] = \bar{\lambda}_j(t_{k+1})$  by  $\ell[1] \leftarrow L[1][0].value$ 
10:     $\ell[0] \leftarrow L[0][0].value$ 
11:    for all  $j \in \{1, \dots, M\}$  do
12:       $\ell[j] \leftarrow \ell[j-1] + L[j][0].value$ 
13:      Select the associated node  $i_{k+1}$  such that  $\frac{\ell[j]}{\ell[M]} < V \leq \frac{\ell[j-1]}{\ell[M]}$ , with  $V \sim \mathbb{U}[0, 1]$ 
14:      for all  $j \in \{1, \dots, M\}$  do
15:        Update intensities  $L[j] \leftarrow L[j] \cup h[i_{k+1}][j]_{\rightarrow t_{k+1}}$ 
16:   $k \leftarrow k + 1$ 
17: return points  $(t_1, \dots, t_{k-1})$  and associated nodes  $(i_1, \dots, i_{k-1})$ 

```

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

Algorithm 6 Local graph algorithm for Hawkes processes

```

1:  $I \leftarrow \{1, \dots, M\}$ 
2: while  $t_k < T$  do
3:   Compute the next point  $t_{next}^i \leftarrow \text{GETNEXT}(L[i])$  of each  $i \in I$ 
4:    $P \leftarrow P^* \oplus (t_{next}^i, i)$  of each  $i \in I$ 
5:    $t_{k+1} = P[0].time$ 
6:    $i_{k+1} = P[0].value$ 
7:   Find corresponding children and update  $I \leftarrow Ch(i_{k+1}) \cup \{i_{k+1}\}$ 
8:   for all  $i \in Ch(i_{k+1})$  do
9:      $L[i] \leftarrow L[i]_{pcw}^{t_{k+1}} \cup h[i_{k+1}][i]_{\rightarrow t_{k+1}}$ 
10:    if  $i \notin Ch(i_{k+1})$  then
11:       $L[i] \leftarrow L[i]_{pcw}^{t_{k+1}} \cup h[i_{k+1}][i]_{\rightarrow t_{k+1}}$ 
12:     $k \leftarrow k + 1$ 
13: return  $(t_1, \dots, t_{k-1})$  points and associated nodes  $(i_1, \dots, i_{k-1})$ 

```

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 node 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 $m = (m_1, \dots, m_M)^T$ (see [14]):

$$m = (I_M - H)^{-1}v, \quad (3)$$

with $v = (v_1, \dots, v_M)^T$, I_M the identity matrix of size M and $H = (\int_0^{+\infty} h_{j \rightarrow i}(x) dx)_{i,j=1,\dots,M}$. Note that the non explosivity of a linear Hawkes process is equivalent to a spectral radius of H strictly less than 1. In this case, the non explosive Hawkes process, with no points before time 0, has always less points than the stationary version. Therefore,

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

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 $R = (\mathbf{1}_{h_{j \rightarrow i} \neq 0})$.

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

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

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 $h_{j \rightarrow i}$.

Therefore we can compute the order of magnitude of L_t^i 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 $AS = O(1)$,

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

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

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

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

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

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

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

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

$$\mathcal{O}(T|m|_1(M + |Rm|_1) \log(M + |Rm|_1)). \quad (6)$$

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 $|m|_1 = \mathcal{O}(1)$ (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 $\mathcal{O}(TM \log(M))$. 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

$$\mathcal{O}(TM^2 d \log(dM)).$$

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

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

The complexity of step a/ is

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

The complexity of step b/ is

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

Steps c/ and d/ have complexity $\mathcal{O}(1)$.

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

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

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

The main point is that node j is appearing in average only Tm_j times during the simulation, which leads us to a global complexity of

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

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

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

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

Thanks to the efficiency of the proposed algorithms, the implementations were prototyped on a regular laptop with 4GB of RAM and an Intel I3 processor as well as part of the statistical analyses. The main simulations have been performed on 5 nodes of a Symmetric MultiProcessing (SMP), i.e., share memory, computer¹. Each of this computational nodes has up to 20 physical cores (2×10), 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 (2×10), 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). We used the implementation of the Mersenne-Twister random generator². No other external libraries were used for the simulator, which is compiled using gcc 2.7. The plots and statistical analyses were obtained using using R software (v3.6), part of it using the UnitEvent package (v0.0.5)³.

6.2 Statistical analysis

We generated an Erdős-Renyii network of 100 nodes with connection probability $p = 1/100$, that is fixed for the rest of the statistical analysis. When an edge $j \rightarrow i$ is in the graph, we associate it to an interaction function $t \mapsto h_{j \rightarrow i}(t) =$

¹The documentation can be found on <https://hpc.isima.fr/doku.php>.

²Accessed on 12/19/2019, on <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html>

³Source accessible at <https://sourcesup.renater.fr/projects/uepackage>. Last access on 12/19/2019.

$5 \cdot \mathbf{1}_{t \in [0, 0.02]}$. The spontaneous parameters v_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 [21], 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 [6]), 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. Then Ogata advertised to perform as follow 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 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 15 and 16). 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.

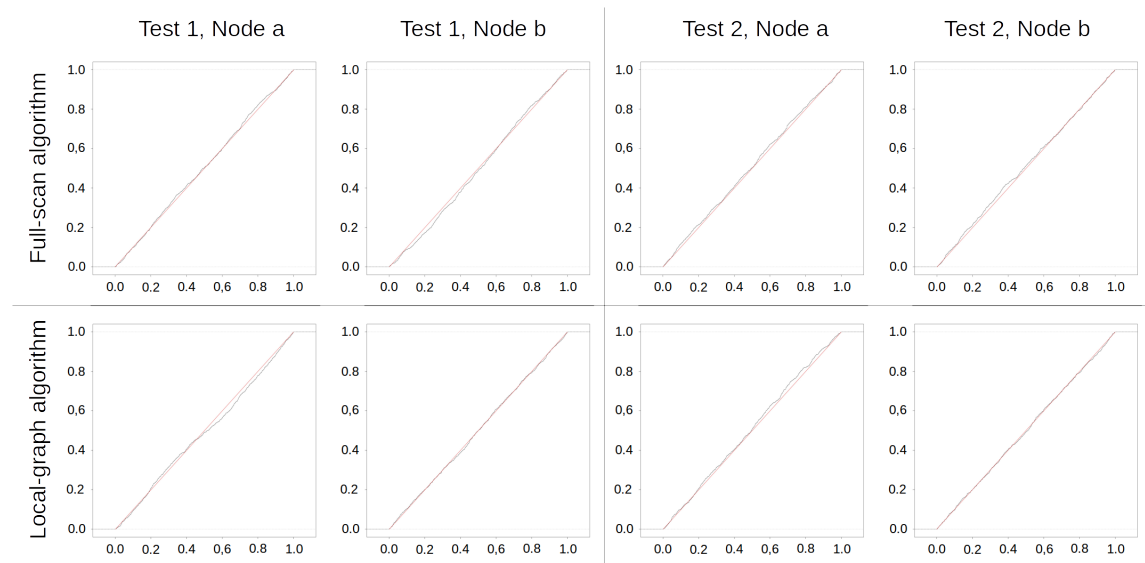


Fig. 15. Cumulative distribution functions of the p-values of Test 1 and 2. In columns the test and node, in rows the algorithms (full scan then local graph)

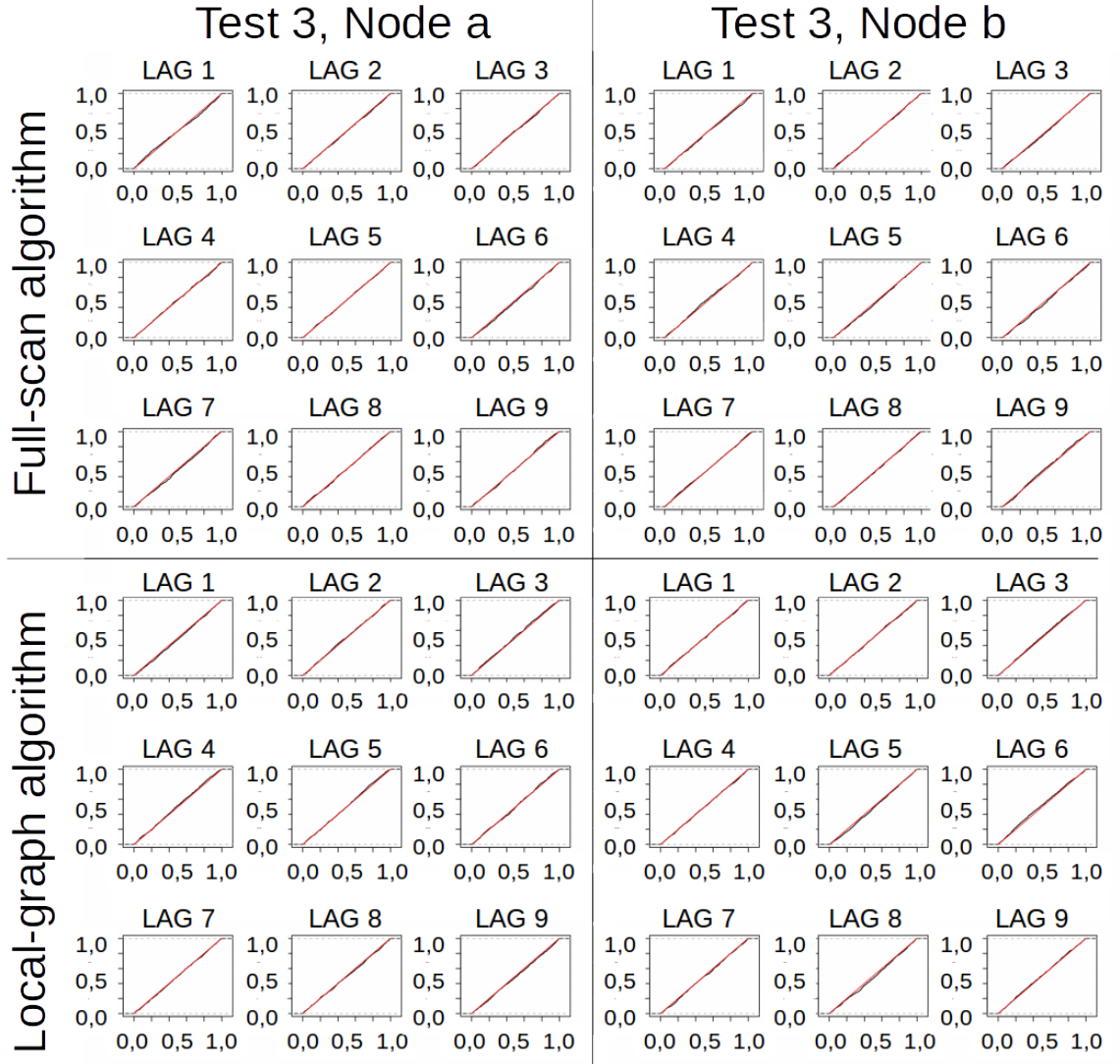


Fig. 16. Cumulative distribution functions of the p-values of Test 3. In columns the node, in rows the algorithm (full scan then local graph)

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 17 and 18 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.

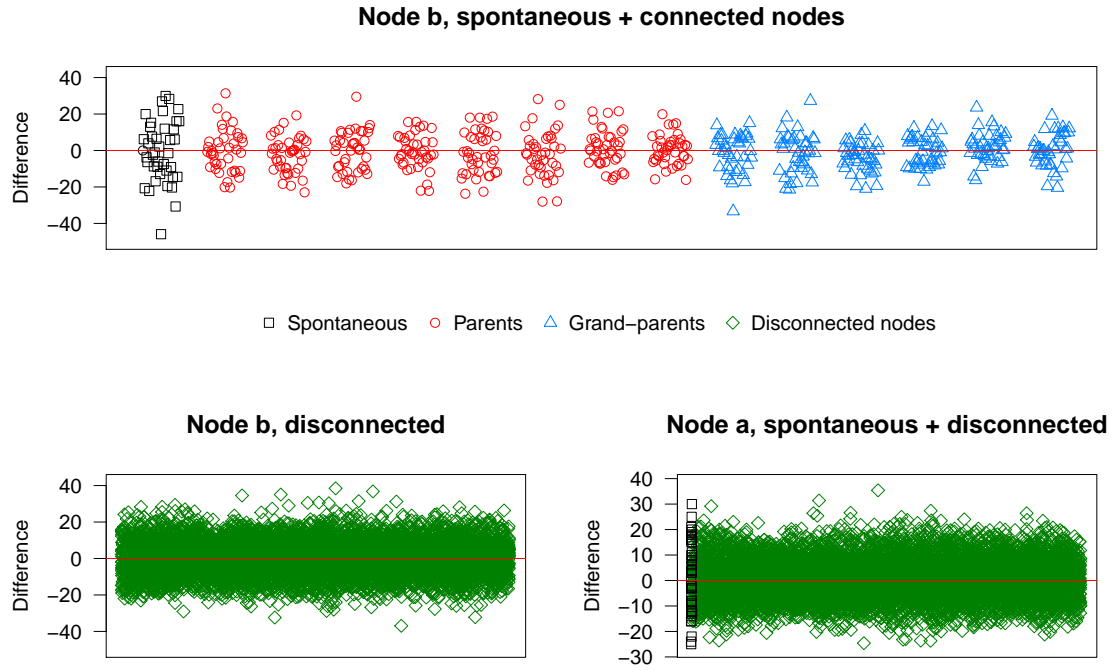


Fig. 17. Full scan algorithm: verifying the Martingale property for Nodes a and b. The black points represent the X^1 (spontaneous), then for the nodes connected to Node b, X^{2j} and X^{2j+1} are displayed in red. In blue are the X^k and X^{k+1} (still for Node b) from Node b's grand-parents to Node b's parents. Finally the two green scatter plots show the Nodes not disconnected from b and a respectively.

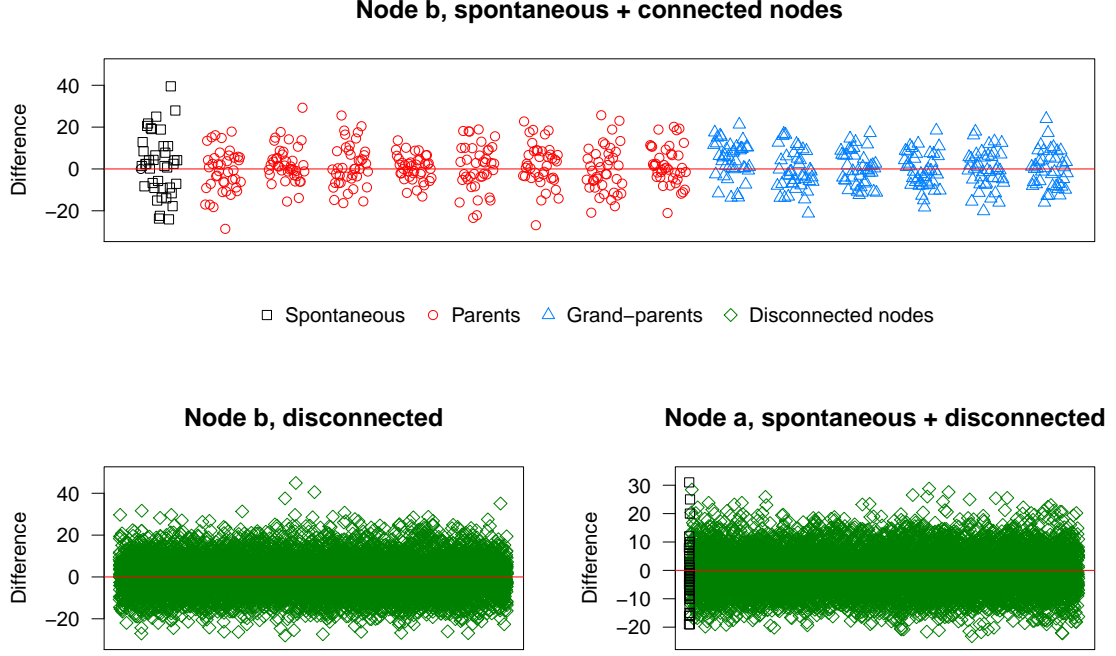


Fig. 18. Local-graph algorithm: verifying the Martingale property for Nodes a and b. The black points represent the X^1 (spontaneous), then for the nodes connected to Node b, X^{2j} and X^{2j+1} are displayed in red. In blue are the X^k and X^{k+1} (still for Node b) from Node b's grand-parents to Node b's parents. Finally the two green scatter plots show the Nodes not disconnected from b and a respectively.

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:

- Erdős-Renyii: a topology model where each edge has a probability p of being present or absent, independently of the other edges. We took p in $\{0, \frac{1}{M}, \frac{2}{M}, \dots, \frac{(\ln(M)-1)}{M}\}$ 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 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 Erdős-Renyii by block. In our setting the nodes are parted 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	$\begin{pmatrix} \frac{M}{2} & \frac{M}{2} \end{pmatrix}$	$\begin{pmatrix} \frac{M}{2} & \frac{M}{2} \end{pmatrix}$	$\begin{pmatrix} \ln M & M - \ln M \end{pmatrix}$
Probability matrices	$\begin{pmatrix} \frac{2}{M} \ln \frac{M}{2} & 0 \\ 0 & \frac{2}{M} \ln \frac{M}{2} \end{pmatrix}$	$\begin{pmatrix} 0 & \frac{2}{M} \ln \frac{M}{2} \\ \frac{2}{M} \ln \frac{M}{2} & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & \frac{\ln(\lfloor \ln M \rfloor)}{\lfloor \ln M \rfloor} \\ \frac{\ln(M - \lfloor \ln M \rfloor)}{(M - \lfloor \ln M \rfloor)} & 0 \end{pmatrix}$

Each graph was generated using a different pseudorandom generator seed. Each existing edge $j \rightarrow i$ is associated with an interaction function $t \mapsto h_{j \rightarrow i}(t) = 5 \cdot \mathbf{1}_{t \in [0, 0.02]}$. 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 $m = (10, \dots, 10)$ and compute the v_i 's by $v = (I_M - H)m$. It may happen that some of the v_i 's become negative. These graphs should be also discarded. Because of the parameter values, especially the interaction functions, no graph was discarded here. A total of $2890 = 1770 + 280 + 840$ (Erdős-Renyii + Cascade + Stochastic-Bloc) graphs was obtained with $M = \{10, 20, \dots, 100\} \cup \{150, 200, \dots, 500\} \cup \{600, 1100, \dots, 5100\}$ for the local-graph algorithm, and $M = \{10, 20, \dots, 100\} \cup \{150, 200, \dots, 500\}$ 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.

Figure 19 shows that the theoretical complexities of both full scan and local graph algorithms are equivalent to their actual execution times.

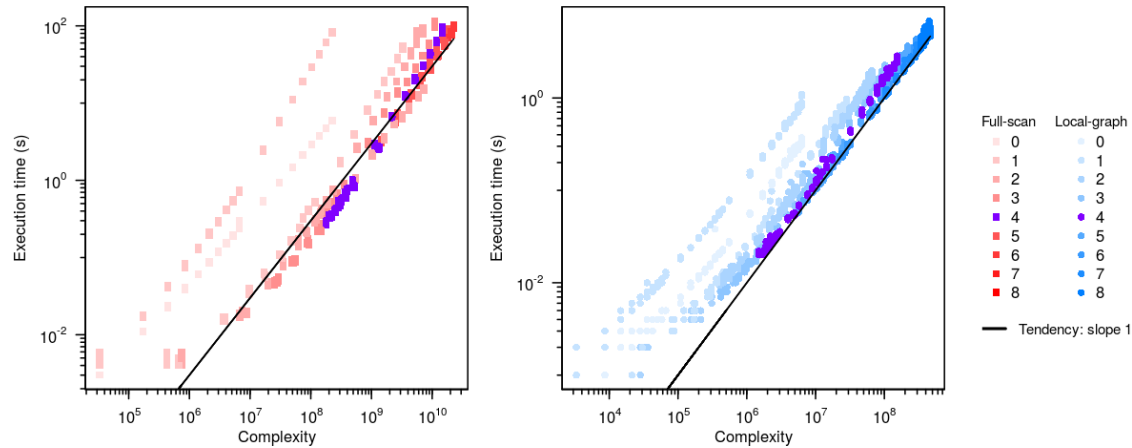


Fig. 19. Three topologies together: the execution time (vertical axis, log-scaled) and the theoretical complexity (horizontal axis, log-scaled), for the full scan algorithm (red squares, left part) and the local-graph algorithm (blue circles, right part). A line of slope 1 is displayed in black, on both 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 $M = \{10, 20, \dots, 100\} \cup \{150, 200, \dots, 500\}$ for the full-scan algorithm and $M = \{10, 20, \dots, 100\} \cup \{150, 200, \dots, 500\} \cup \{600, 1100, \dots, 5100\}$ for the local graph algorithm.

Figure 20 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.

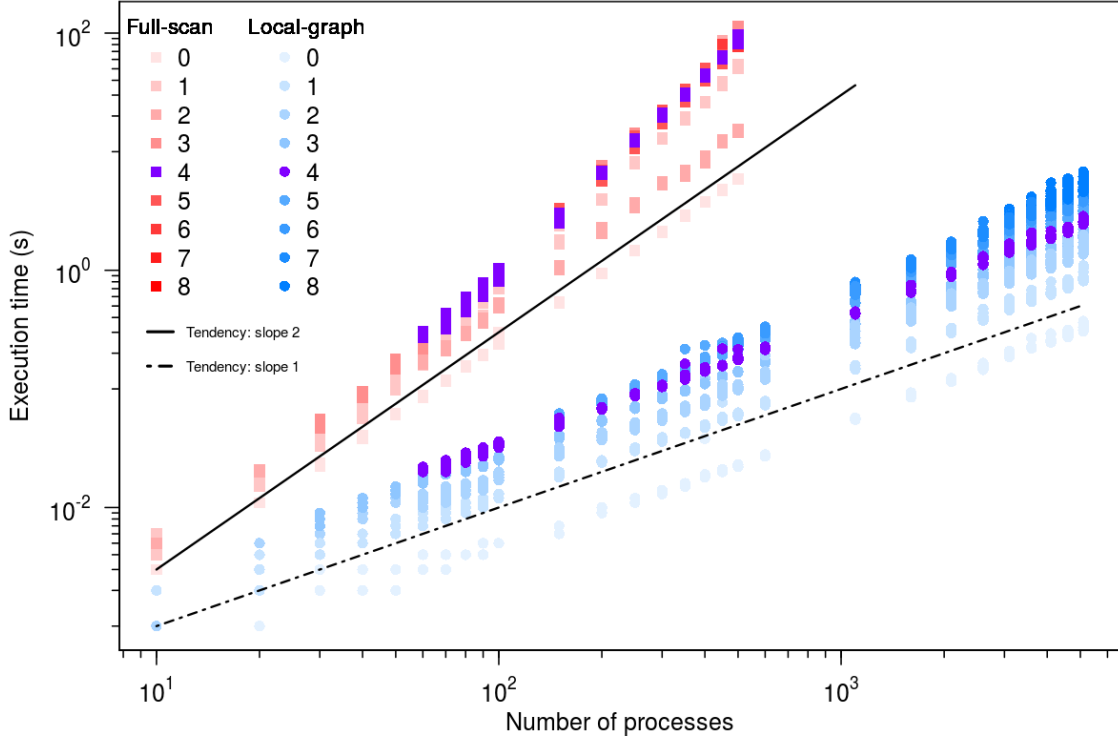


Fig. 20. Three topologies together: the execution time (vertical axis, log-scaled) and the theoretical complexity (horizontal axis, log-scaled), for the full scan algorithm (red squares, left part) and the local-graph algorithm (blue circles, right part). A line of slope 2 is displayed in black, on the 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 $M = \{10, 20, \dots, 100\} \cup \{150, 200, \dots, 500\}$ for the full-scan algorithm and $M = \{10, 20, \dots, 100\} \cup \{150, 200, \dots, 500\} \cup \{600, 1100, \dots, 5100\}$ for the local graph algorithm.

7 CONCLUSION

We presented two new discrete event simulations for point processes: one being already an optimization of usual Ogata's algorithm [20], called the full scan algorithm, another one, called 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 [10]). The computational complexity reduction of the local graph algorithm 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. 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 [22]. 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. [12]. In the brain, last estimations consist of 86 billions of neurons [28], 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, H       Toussaint, William Guyot-L       and Boris Lonjon, for their valuable help.

This work was supported by the French government, through the UCA^{Jedi} and 3IA C       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 Universit   C      .

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-C      , 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. Br      . 1981. *Point Processes and queues: martingale dynamics*. Springer.
- [5] 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.
- [6] 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.
- [7] J.H. Cha and M. Finkelstein. 2018. *Point Processes for Reliability Analysis*. Springer.
- [8] J. Chevallier, M.J. C      , 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.
- [9] A. Dassios and H. Zhao. 2013. Exact simulation of Hawkes process with exponentially decaying intensity. *Electronic Communications in Probability* 18, 62 (2013).
- [10] V. Didelez. 2008. Graphical models of markes point processes based on local independence. *J.R. Statist. Soc. B* 70, 1 (2008), 245–264.
- [11] J.L. Doob. 1945. Markoff chains – Denumerable case. *Trans. Amer. Math. Soc.* 58, 3 (1945), 455–473.
- [12] W. Gerstner and W.M. Kistler. 2002. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press.
- [13] D.T. Gillespie. 1977. Exact Stochastic Simulation of Coupled Chemical Reactions. *he Journal of Physical Chemistry* 81, 25 (1977), 2340–2361.
- [14] A.G. Hawkes. 1971. Spectra of Some Self-Exciting and Mutually Exciting Point Processes. *Biometrika* 58, 1 (1971), 83–90. <http://www.jstor.org/stable/2334319>
- [15] 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>

- [16] P.A.W. Lewis and G.S. Shedler. 1978. Simulation of nonhomogeneous Poisson processes. Technical Report. Naval Postgraduate School, Monterey, California.
- [17] M.D. Mesarovic and Y. Takahara. 1975. General systems theory: mathematical foundations. Vol. 113. Academic press.
- [18] 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>
- [19] 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.
- [20] Y. Ogata. 1981. On Lewis' simulation method for point processes. IEEE Transaction on Information Theory 27, 1 (1981), 23–31.
- [21] Y. Ogata. 1985. Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes. J. Amer. Statist. Assoc. 83, 401 (1985), 9–27.
- [22] 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](https://doi.org/10.1016/S0531-5565(02)00151-1) Proceedings of the 6th International Symposium on the Neurobiology and Neuroendocrinology of Aging.
- [23] E.A.J.F. Peters and G. de With. 2012. Rejection-free MonteCarlo sampling for general potentials. Physical Review E 85, 026703 (2012).
- [24] 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.
- [25] P. Reynaud-Bouret and S. Schbath. 2010. Adaptive estimation for Hawkes processes; application to genome analysis. Annals of Statistics 38, 5 (2010), 2781–2822.
- [26] K.D. Tocher. 1967. PLUS/GPS III Specification. Technical Report. United Steel Companies Ltd, Department of Operational Research, Sheffield.
- [27] 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.
- [28] 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.
- [29] B.P. Zeigler. 1976. Theory of Modelling and Simulation. John Wiley. <https://books.google.fr/books?id=M-ZQAAAAAAAJ>
- [30] B.P. Zeigler, A. Muzy, and E. Kofman. 2018. Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations. Academic Press.