*Probability, Networks and Algorithms*

# Exact Metropolis-Hastings sampling for marked point processes using a C++ library

M.N.M. van Lieshout, R.S. Stoica

**REPORT PNA-R0403 JUNE 2004**

ISSN 1386-3711

# Exact Metropolis-Hastings sampling for marked point processes using a C++ library

ABSTRACT
This report documents MPPLIB, a C++ library for marked point processes, and illustrates its use by means of a new exact Metropolis--Hastings simulation algorithm.

*2000 Mathematics Subject Classification:* 60G55, 62M40.
*Keywords and Phrases:* Exact simulation, Markov marked point process, Metropolis--Hastings algorithm.
*Note*: Work carried out under project PNA4.3 `Stochastic Geometry'. This research was partially supported by NWO grant `Inference for random sets' (613-03-045).

# Exact Metropolis–Hastings Sampling for Marked Point Processes using a C++ Library

M.N.M. van Lieshout[1], R.S. Stoica[2]

[1] *CWI*

*P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

[2] *Universitat Jaume I*

*Campus Riu Sec, E-12071 Castelló de la Plana, Spain*

ABSTRACT

This report documents MPPLIB, a C++ library for marked point processes, and illustrates its use by means of a new exact Metropolis–Hastings simulation algorithm.

*2000 Mathematics Subject Classification:* 60G55, 62M40.

*Keywords and Phrases:* Exact simulation, Markov marked point process, Metropolis–Hastings algorithm.

*Note:* Work carried out under project PNA4.3 'Stochastic Geometry'. This research was partially supported by NWO grant 'Inference for random sets' (613-03-045).

## 1. Introduction

In [11], we extended some recently proposed exact simulation algorithms to the case of marked point processes. Three families of algorithms were considered: coupling from the past [7, 8, 9] the clan of ancestors technique [3] and the Gibbs sampler. The various algorithms have been tested on several models, including the Widom–Rowlinson mixture model [18], multi-type pairwise interaction processes and the Candy line segment model [17]. A simulation study was carried out in order to analyse the proposed methods in terms of speed of convergence in relation to the model parameters. For general background and motivation, the reader is referred to the prequel [11].

Our conclusions were that for the range of models investigated, the clan of ancestors algorithm using the incompatibility index was the fastest method among the ones analysed in this work, while coupling from the past was applicable to the widest range of parameter values. If one were prepared to approximate by discretisation, a proper choice of Gibbs sampler [5, 6, 13] makes it possible to obtain samples from models that lack monotonicity or have such a high local stability bound as to rule out coupling from the past or clan of ancestor approaches from a practical point of view.

The goal of the present paper is to investigate a simpler version of the Metropolis–Hastings dynamics – based on the generic configuration independent birth and death proposal probabilities [4] – for locally stable point processes than that in [8], to extend the new algorithm to locally stable marked point processes, and to quantify the number of transitions needed to obtain an exact sample as a function of the model parameters, particularly the intensity and range of interaction. A secondary goal is to illustrate MPPLIB, a `C++` library for marked point processes developed at CWI by A.G. Steenbeek and others [16].

The plan of this paper is as follows. We briefly review basic facts on marked point processes in Section 2 and give their MPPLIB class specifications. A Metropolis–Hastings algorithm is developed, and its implementation sketched in Section 3. Finally, a simulation study into the efficiency of the new algorithm is presented in Section 4.

## 2. Marked point processes

### *2.1 Definitions*

Let $K \subset \mathbb{R}^2$ be a compact subset of strictly positive Lebesgue measure $0 < \nu(K) < \infty$ and $M$ a complete separable metric space. A *marked point process* $Y$ with positions in $K$ and marks in $M$ is a point process on $K \times M$ such that the process of unmarked points is (locally) finite [2]. In other words, realisations of $Y$ are of the form $\mathbf{y} = \{(k_1, m_1), \ldots (k_n, m_n)\}$ where $n \in \mathbb{N}_0$, $k_i \in K$ and $m_i \in M$ for all $i = 1, \ldots, n$.

Let $\nu_M$ be a probability measure on the Borel $\sigma-$algebra $\mathcal{B}(M)$. In this paper, we shall restrict attention to marked point processes that are absolutely continuous with respect to the distribution of a Poisson process on $K \times M$ with intensity measure $\nu \times \nu_M$. Thus, under the reference measure, points in a realisation of a unit rate Poisson process on $K$ are given i.i.d. marks distributed according to $\nu_M$.

The *Papangelou conditional intensity* of a marked point process with density $f$ is defined for $(k, m) \in (K \times M) \setminus \mathbf{y}$ as

$$\lambda((k, m); \mathbf{y}) := \frac{f(\mathbf{y} \cup \{(k, m)\})}{f(\mathbf{y})}$$

whenever $f(\mathbf{y}) > 0$ and arbitrarily (say 0) on the null set $\{\mathbf{y} : f(\mathbf{y}) = 0\}$. It may be interpreted as the conditional probability of finding a point at $k$ with mark $m$ conditional on the configuration elsewhere being $\mathbf{y} \setminus \{(k, m)\}$.

Henceforth, we shall assume the following properties to hold:

- the density is *hereditary*, that is, $f(\mathbf{y}) > 0$ implies $f(\mathbf{y}') > 0$ for all $\mathbf{y}' \subseteq \mathbf{y}$;
- the density is *locally stable*, that is the Papangelou conditional intensity is bounded from above by some positive, finite constant $\Lambda$.

It is left to the reader to verify that the models introduced in the next subsections satisfy these assumptions. A density is said to be repulsive if $\lambda((k, m); \mathbf{y})$ is decreasing in its second argument with respect to set inclusion, attractive if it is increasing. Such properties correspond to our intuitive notion of e.g. repulsion since in that case the more marked points there are, the harder it is to introduce yet another one, and the smaller the conditional intensity.

*Widom–Rowlinson mixture model* The Widom–Rowlinson mixture model for penetrable spheres [18] has mark space $M = \{1, 2\}$ and density

$$f(\mathbf{y}) = \alpha \prod_{(k,m) \in \mathbf{y}} \beta_m \prod_{(u,1),(v,2) \in \mathbf{y}} \mathbf{1}\{\| u - v \| > R\} \tag{2.1}$$

with respect to a unit rate Poisson process labelled according to the symmetric Bernoulli distribution. Thus, particles with a different label keep at least a distance $R > 0$ away from each other, $\beta_m > 0$ is an intensity parameter for type $m$ points, and $\alpha = \alpha(\beta_1, \beta_2, R) \in (0, \infty)$ is the normalising constant that ensures that $f$ integrates to 1.

*Multi-type pairwise interaction process* Consider the mark space $M = \{1, \dots, I\}$ for some $I \in \mathbb{N}$ equipped with the uniform probability distribution $\nu_M$. Multi-type pairwise interaction processes [1, 15] are defined by a density $f$ with respect to the dominating Poisson process that is of the form

$$f(\mathbf{y}) = \alpha \prod_{(k,m)\in\mathbf{y}} \beta_m \prod_{(u,i)\neq(v,j)\in\mathbf{y}} \gamma_{ij}(\| u - v \|) \tag{2.2}$$

with the second product ranging over all distinct pairs of marked points. Here $\alpha > 0$ is the normalising constant, the scalars $\beta_m > 0$, $m \in M$, are intensity parameters, and for each pair of labels $i, j \in M$, $\gamma_{ij} : [0, \infty) \to [0, 1]$ is a measurable interaction function. We shall assume that $\gamma_{ij} \equiv \gamma_{ji}$ for all $i, j \in M$.

*Candy model* The Candy model [17, 10] is a line segment process. The segments are characterised by the position of their centre, their length $l \in [l_{\min}, l_{\max}]$ for some $0 < l_{\min} < l_{\max} < \infty$, and orientation $\theta \in [0, \pi)$. The orientation space is equipped with the complete metric $\rho(\theta, \theta') = \min\{|\theta - \theta'|, \pi - |\theta - \theta'|\}$ that identifies 0 and $\pi$. Thus, the Candy model may be seen as a marked point process with marks in $M = [l_{\min}, l_{\max}] \times [0, \pi)$. It has density

$$f(\mathbf{y}) = \alpha\, \beta^{n(\mathbf{y})} \prod_{i=1}^{n(\mathbf{y})} \exp\left[\frac{l_i - l_{\max}}{l_{\max}}\right] \times \gamma_r^{n_r(\mathbf{y})}\, \gamma_o^{n_o(\mathbf{y})} \tag{2.3}$$

with respect to a unit rate Poisson process marked uniformly and independently. The model parameters are $\gamma_r$, $\gamma_o \in (0, 1)$ and $\beta > 0$. The sufficient statistics $n(\mathbf{y})$, $n_r(\mathbf{y})$ and $n_o(\mathbf{y})$ represent the total number of segments in $\mathbf{y}$, the number of pairs of segments crossing at too sharp an angle, and the number of pairs of segments that are disoriented. More formally, for a given $\delta \in (0, \pi/2)$, define the relation $\sim_r$ on $K \times M$ by

$$y = (k, l, \theta) \sim_r y' = (k', l', \theta') \Leftrightarrow \|k - k'\| \leq \max\{l, l'\}/2 \text{ and } \left|\rho(\theta, \theta') - \pi/2\right| > \delta.$$

Then $n_r(\mathbf{y})$ is the number of pairs of different points in $\mathbf{y}$ that are $\sim_r$-related. Moreover, let the influence zone $Z(y)$ of a marked point $y = (k, l, \theta)$ be the union of balls with radius $l/4$ around the endpoints, and define the relation $\sim_o$ on $K \times M$ as follows: $y \sim_o y' \Leftrightarrow \|k - k'\| > \max\{l, l'\}/2$ and either exactly one endpoint of $y$ is a member of $Z(y')$ or exactly one endpoint of $y'$ is a member of $Z(y)$. Then $n_o(\mathbf{y})$ is the number of $\sim_o$ neighbour pairs in $\mathbf{y}$ with the property that $\rho(\theta, \theta') > \tau$. Generalisations of (2.3) may be obtained by distinguishing several types of connection between the segments.

### *2.2 Implementation*

The marked point process models described above – as well as a range of others – have been implemented in `C++`. The object oriented nature of this programming language is well suited to our purposes, as it allows us to define a few general classes for points, marked or classic point processes and samplers, and to introduce derived classes for new models and algorithms when they are needed. The general classes contain (virtual) member functions for the basic operations; the derived classes contain model or algorithm specific variables and functions, and implement the virtual members of their parent class as appropriate.

*Marked point configurations* Points in the plane are captured by class **Point**, whose public members are the x- and y-coordinates. The member function norm returns the (Euclidean) norm, that is the distance to the origin, of the point. Functions that operate on Points include the distance between two points, the basic arithmetic operations subtraction, addition and reflection, as well as Boolean operators that test for (non-) equality. An empty constructor is provided, as well as one that sets the two coordinates.

Marked points, i.e. points to which a label or mark is assigned, are implemented as derived classes from the **Event** class below.

```
class Event {
 public:
   Event( Point p );
   double getX() const;
   double getY() const;
   Point getPosition() const;
   double getDistance( const Event *e ) const;
   virtual void setPosition( Point p );
   virtual void report() const;
};
```

The Event constructor needs an argument of type Point to set the location of the marked point. Member functions return this position (*getPosition*) or its x- and y-coordinates separately (*getX*, *getY*). The function *getDistance* computes the Euclidean distance to another Event, and by default, the virtual functions *setPosition* and *report* operate on the Point only.

The following classes derived from Event implement the labelled points and line segments considered in this paper.

```
class LabelledEvent : public Event {
 public:
   LabelledEvent( Point p, int l );
   int getLabel() const;
   void setLabel( int l );
   void report() const;
};
```

The mark is a type label; its value is returned by *getLabel*, set by the constructor, and changed by *setLabel*.

```
class Segment : public Event {
 public:
   Segment( Point c, double l, double o );
   Segment( Point h, Point t );
   Point getHead() const;
   Point getTail() const;
   double getLength() const;
   double getOrientation() const;
   virtual void setPosition( Point p );
   virtual void setSegment( Point p, double l, double o );
   virtual void report() const;
};
```

The position of a Segment is taken to be its centre. Two real valued marks describe the length and orientation (in between 0 and $\pi$). The mark values can be specified in the constructor; alternatively, the two end points of the segment may be given as arguments. Member functions return the endpoints, length and orientation $\theta$. We use the convention that for the head a multiple of $(\cos\theta, \sin\theta)$ is added to the position, while the same vector is subtracted to obtain the tail. The user may update the marks by calling *setSegment*, a function kept virtual to leave the option open to derive further classes.

The class **Pattern**, derived from the generic class **Vector** (a dynamic array of pointers to deal with configurations of arbitrary length), is used to capture realisations of marked point processes. Its initial capacity is set in the constructor (50 by default). Note the capacity is the currently expected number of pointers in the vector, not the actual length of the array! Indeed, the Pattern constructor creates an empty configuration. Extra capacity is allocated by member functions, such as **Vector.add**, if the length would exceed the capacity.

```
class Pattern: public Vector {
 public:
   Pattern( int init_capacity = 50 );
   Event * getEvent( int i ) const;
   void remove( Event *e );
   Event * random_rmv();
   Event * position_rmv( int i );
   int intersect( Pattern A ) const;
   void destroy();
};
```

The member function *getEvent* retrieves an Event pointer by its current index in the array, *intersect* tests for intersection with another Pattern, that is, it returns true if there is an element belonging to both Patterns. *Destroy* frees the memory taken up by all Events in the current Pattern. We have implemented three options for the removal of Pattern members: *remove* puts the last element of the Vector in the position previously occupied by the Event pointer to be removed. The function *random_rmv* does the same for a uniform choice of the elements, and *position_rmv* deletes the Event pointer at the specified position in the array and replaces it by the last one. Note that since configurations of marked points are unordered sets, it is both allowed and more efficient to move the last pointer rather than shift the array to fill a gap left by a removed Event pointer. Addition is inherited from the parent class Vector.

*Marked point process models* The class for point processes – marked or otherwise – is **Prior**. The distribution is supposed to be given by a hereditary, locally stable density on a planar rectangle with respect to a unit rate Poisson process. The constructor sets the local stability bound $\Lambda$, the width and the height of the rectangle; their values are returned by the functions ubnd, *getWidth* and *getHeight* respectively. The window size may be reset through *setWidth* and *setHeight*.

```
class Prior {
 public:
   Prior( double Lambda, double w, double h);
   virtual ~Prior() { }
   void setWidth( double w );
   void setHeight( double h );
   double getWidth() const;
   double getHeight() const;
   double ubnd() const;
   virtual double cond_intens( const Pattern& p, const Event* e ) const = 0;
   virtual int monotone() const = 0;
   void discretise( double v, int m );
   virtual void report() const = 0;
   virtual double range() const = 0;
   virtual Event *newEvent() const;
   void printInit( char *f1, char *f2, char *f3 );
   void printClose();
   virtual void printParameters( );
   virtual void printStatistics( const Pattern& p );
   virtual void printSample( const Pattern& p );
   int getTotal( const Pattern& p ) const;
};
```

Virtual member functions compute the conditional intensity, and check whether the density represented by the class is monotone (return value 0 for repulsive models, 1 for attractive ones, and 2 otherwise). Since concrete models often assume a particular kind of marked point (e.g. a line segment in case of the Candy model (2.3) to operate on, we need the function *newEvent* to create a suitable Event. In the default case, *newEvent* returns a uniformly distributed point in $K$. The functions *report* and *range* inform the user about the model, its parameters, and interaction distance. A further set of virtual functions handles output files. We considered three such files: one for model parameters (including $\Lambda$), another for summary statistics, and one for samples from the model. In the default case, the latter two functions return the number of marked points in the argument Pattern $p$, which is the return value of *getTotal*, and the positions of marked points in $p$. Output files are opened and closed for writing by *printInit* and *printClose*, respectively.

The description of the Prior class is complete upon mentioning that the Metropolis–Hastings sampler to be developed in this paper works with a partition of the state space $K \times M$. This is implemented by the member function *discretise* which divides the rectangle $K$ in equal cells of area $v$, and the mark space in $m$ bins.

*Multi-type pairwise interaction processes* Repulsive densities of the form

$$f(\mathbf{y}) = \alpha \prod_i \beta(y_i) \prod_{i<j} \gamma(y_i, y_j))$$

with $\gamma \leq 1$ are implemented by the class **MPInteraction** derived from Prior. Note that we do not require that the mark space is finite as in (2.2).

The constructor of MPInteraction has arguments of type double for $\Lambda$, for the range $r$ beyond which the interaction functions $\gamma \equiv 1$, and for the width and height of $K$. A useful summary statistic is the number of pairs $\{y_i, y_j\}$ in a marked point configuration $\mathbf{y}$ that contribute a term less than 1 to the density. This number is returned by the member function `int getInteractingPairs( const Pattern& p ) const;`.

*The Widom–Rowlinson model* We implement (2.1) by the class **WR** derived from MPInteraction. Its parameters $\beta_1$, $\beta_2$ and $R$ are set by the constructor, and can be updated by the function *setParameters*. The constructor also sets the width and height of $K$. Conversely, the intensity values are returned by *getBetaOne* and *getBetaTwo*; the hard core distance $R$ is also the interaction range of the model and as such it is the output value of the Prior member function *range*. Finally, the Events are of type LabelledEvent.

```
class WR : public MPInteraction {
 public:
   WR( double b1, double b2, double r, double w, double h );
   double getBetaOne() const;
   double getBetaTwo() const;
   void setParameters( double b1, double b2, double r );
   void printParameters();
   void printSample( const Pattern& p );
   Event *newEvent() const;
   void report() const;
};
```

*The Candy model* Pairwise interaction line segment processes are implemented by the class **SegmentProcess** derived from MPInteraction, which works with Events of type Segment. Since the mark space $M = [l_{\min}, l_{\max}] \times [0, \pi)$ is a Cartesian product, it is convenient to discretise $M$ by setting the number of bins for the length and orientation separately:
`void markdiscretise( double v, int lbins, int obins );`.
As before $v$ is the cell volume in the partition of $K$. The endpoints of the length interval, $l_{\min}$ and $l_{\max}$, can be obtained by calling the functions `double MinLength() const;` and `double MaxLength() const;`.

The Candy model (2.3) is a derived class as follows.

```
class MPCandy : public SegmentProcess {
 public:
   MPCandy( double w, double h, double l_min, double l_max,
         double tau, double delta,
        double omega_r, double omega_o, double log_beta );
   double getBeta() const;
   double getRepulsionPar() const;
   double getOrientationPar() const;
   void setParameters( double omega_r, double omega_o, double log_beta );
   void setLength( double l_min, double l_max );
   bool nbrRejection( const Segment *e, const Segment *f ) const;
   bool nbrOrientation( const Segment *e, const Segment *f ) const;
   void printStatistics( const Pattern& p );
   void printParameters();
   void getRejectedOrientedSegments( const Pattern& p,
        int *n_r, int *n_o ) const;
   void report() const;
};
```

The Candy model interaction thresholds $\tau$ and $\delta$, as well as the parameters $\gamma_r$, $\gamma_o$ and $\beta$ on the logarithmic scale, are set by the constructor, which also assigns a width and height to $K$, and sets the interval $[l_{\min}, l_{\max}]$. The model parameters may be updated by the function *setParameters*, the length interval by *setLength*. Conversely, *PrintParameters* writes the length, threshold, and model parameters into the output file maintained by the parent class Prior. The intensity value $\beta$ is returned by the member function *getBeta*, whereas $\omega_r = \log \gamma_r$ and $\omega_o = \log \gamma_o$ are obtained by calls to *getRepulsionPar* and *getOrientationPar* respectively. Member function *getRejectedOrientedSegments* returns the values of the sufficient statistics $n_r$ and $n_o$ that serve as summary statistics, in addition to the total number of segments in Pattern $p$. Finally, the Boolean functions *nbrRejection* and *nbrOrientation* test whether their arguments are related under $\sim_r$ respectively $\sim_o$.

## 3. Exact Metropolis–Hastings algorithm

### *3.1 Description*

In recent years, computational statistics has seen the introduction of a range of exact (or perfect) simulation methods following the ground breaking paper by Propp and Wilson [14]. In contrast to Markov Chain Monte Carlo techniques, that need careful burn-in and convergence diagnostics to assess whether the underlying Markov chain has reached its stationary distribution, exact simulation methods are able to determine for themselves during run time if and when equilibrium is reached.

For (unmarked) locally stable point processes, Kendall and Møller [8] proposed a dominated coupling from the past algorithm using discretisation and Metropolis–Hastings dynamics depending on the current configuration. Here we propose an easier alternative with fixed proposal probabilities for each cell and generalise the algorithm to marked point processes.

Recall that a Metropolis–Hastings algorithm is a proposal-acceptance technique. A typical update if the current state is the marked point pattern $\mathbf{y}$ (assumed to have positive density $f(\mathbf{y})$) is to opt for a birth with probability $p_b \in (0,1)$, for a death with the complementary probability $p_d = 1 - p_b$. In case of a birth, a new marked point $\xi$ is sampled from the distribution $\nu \times \nu_M / \nu(K)$, and the proposal is accepted with probability

$$\min\left\{1, \frac{\nu(K)\, \lambda(\xi; \mathbf{y})}{p\,(1 + n(\mathbf{y}))}\right\}$$

where $p = p_b/(1-p_b)$. If a death is proposed, and $\mathbf{y}$ is empty, the state $\mathbf{y}$ remains unchanged. Otherwise, a point $\xi$ is selected uniformly from $\mathbf{y}$ and its deletion accepted with probability

$$\min\left\{1, \frac{p\, n(\mathbf{y}))}{\nu(K)\, \lambda(\xi; \mathbf{y} \setminus \{\xi\})}\right\}.$$

For a proof that the dynamics described above yield an unbiased sample from $f$ in the long run, see for example [4] and the references therein. In order to transform these update dynamics into an exact simulation algorithm, care has to be taken, especially with respect to death transitions. Since coupling from the past is based on the idea of designing a pair of chains, coupled to some dominating one, that maintains the inclusion order and merges eventually [8], it would be convenient if the Hastings ratio for accepting such transitions would be 1, regardless of the current configuration. The ratio is clearly bounded from below by $p/(\nu(K)\,\Lambda)$, but the bound may be smaller than 1. To overcome this problem, following [8], we discretise $K$ and update in a stripwise fashion.

Thus, let $K = \cup_{i=1}^{n_K} K_i$ be a finite partition such that $0 < \nu(K_i) < \infty$ for all cells $K_i$, $i = 1, \dots, n_K$. If the current configuration is $\mathbf{y}$ and the strip to be visited is $K_i \times M$, with probability $p_b^i$ a new marked point $\xi$ is generated with a $\nu$-uniformly distributed location restricted to $K_i$ and mark distribution $\nu_M$; with the complementary probability $p_d^i$, a randomly chosen marked point located in $K_i$ is proposed for deletion – if there is such a point. We assume that $p_b^i = 1 - p_d^i \in (0,1)$ for all $i = 1, \dots, n_K$. Hastings ratios similar to those above are computed to decide whether to accept or not. The procedure is well defined on the set of configurations with strictly positive density, and converges to $f$ (see the next section).

In order to describe the coupling based on these dynamics, let us start with the dominating chain $D$ in which all proposals are accepted. More precisely, the Markov chain visits strips at random with equal probabilities. If strip $K_i \times M$ is being visited, with probability $p_b^i$, a new marked point is generated with location and mark distributed according to the $\nu$-uniform distribution on $K_i$ and to $\nu_M$ respectively; with the complementary probability $p_d^i = 1 - p_b^i$, a marked point is chosen uniformly among those with location in $K_i$ and deleted from $D$ with the proviso that if $K_i \times M$ contains no marked points, nothing is done.

Clearly, the chain is reversible, and patterns in different strips $K_i \times M$ are independent. Provided the birth proposal probability for each strip is strictly positive and less than the proposal probability for a death, $0 < p_b^i < 1/2$, the detailed balance equations for the number of marked points in strip $i$

$$\pi_i(n)\, p_b^i = \pi_i(n+1)\, p_d^i \quad n = 0, 1, 2, \dots$$

have a unique solution

$$\pi_i(n) = \left(\frac{p_b^i}{p_d^i}\right)^n \left(1 - \frac{p_b^i}{p_d^i}\right), \tag{3.1}$$

that is, the number of marked points in $K_i \times M$ has a shifted geometric distribution with success probability $1 - p_b^i/p_d^i$. Given there are $n$ marked points with locations in cell $K_i$, they are i.i.d. with locations following the probability distribution $\nu(\cdot)/\nu(K_i)$ and marks chosen according to $\nu_M$.

Next, turn to the target dynamics. The Hastings ratio for a death transition from $\mathbf{y} \neq \emptyset$ to $\mathbf{y} \setminus \{y_j\}$ is given by

$$\frac{p_i \, n(\mathbf{y} \cap (K_i \times M))}{\nu(K_i) \, \lambda(y_j; \mathbf{y} \setminus \{y_j\})}$$

where $p_i = p_b^i / p_d^i$ is the ratio of birth and death proposal probabilities for the strip $K_i \times M$, $i = 1, \dots, n_K$. Note that if we restrict the Markov chain to the set $\{\mathbf{y} : f(\mathbf{y}) > 0\}$, as $f$ is hereditary by assumption, the Hastings ratio is well-defined. A sufficient condition for it to be bounded from below by 1 is $p_i \geq \Lambda \, \nu(K_i)$ where $\Lambda$ denotes the upper bound on the conditional intensity. Under this condition, proposals to delete a marked point are always accepted.

Let $L \subseteq U$ be two finite marked point patterns, and set

$$\begin{aligned} \alpha_{\min}(U, L, (k,m), i) &:= \min \left\{ \tfrac{\nu(K_i) \, \lambda((k,m); \mathbf{y})}{p_i (1 + n(\mathbf{y} \cap (K_i \times M)))} : L \subseteq \mathbf{y} \subseteq U \right\} \\ \alpha_{\max}(U, L, (k,m), i) &:= \max \left\{ \tfrac{\nu(K_i) \, \lambda((k,m); \mathbf{y})}{p_i (1 + n(\mathbf{y} \cap (K_i \times M)))} : L \subseteq \mathbf{y} \subseteq U \right\} \end{aligned} \tag{3.2}$$

for the bounds on the Hastings ratio for the birth of a point $(k,m)$ in strip $i$ based on marked point patterns sandwiched in between $L$ and $U$.

Based on the ingredients described above, we propose the following algorithm.

**Algorithm 1.** *Let $K$ be partitioned in cells in such a way that $0 < \Lambda \, \nu(K_i) < 1$ and fix $p_b^i$ in such a way that $p_i \in [\Lambda \nu(K_i), 1)$ for all $i = 1, \dots, n_K$. Let $V_t$, $t = -1, -2, \dots,$ be a family of independent, uniformly distributed random variables on $\{1, \dots, n_K\}$ and let $U_t$ be a family of independent, uniformly distributed random variables on $(0,1)$. Initialise $T = 1$, and let $D(0)$ be a realisation of a marked point process with independent (3.1) distributed strip counts, and for which, given a strip contains $n$ points, the locations are scattered $\nu$-uniformly over the cell and marked i.i.d. according to $\nu_M$.*

1. *Extend $D(\cdot)$ backwards to time $-T$ as follows. With probability $p_d^{V_t}$ delete a randomly picked marked point from $K_{V_t} \times M$; otherwise add a marked point $\xi_t$ with $\nu$-uniform location in $K_{V_t}$ and mark distributed according to $\nu_M$ independently of other random variables.*

2. *Generate $L^{-T}(\cdot)$ (lower process) and $U^{-T}(\cdot)$ (upper process) forward in time as follows:*

   - *set $L^{-T}(-T) = \emptyset$ and $U^{-T}(-T) = D(-T)$;*
   - *in case $D(\cdot)$ experiences a backward birth, i.e. $D(t) = D(t+1) \cup \{(k,m)\}$, a point is deleted from $L^{-T}(t)$ and $U^{-T}(t)$ according to the random permutation mechanism of [8];*
   - *if $D(\cdot)$ experiences a backward death, i.e. $D(t) = D(t+1) \setminus \{(k,m)\}$, the marked point $(k,m)$ is added to $L^{-T}(t)$ if $U_t \leq \alpha_{\min}(U^{-T}(t), L^{-T}(t), (k,m), V_t)$ and to $U^{-T}(t)$ if $U_t \leq \alpha_{\max}(U^{-T}(t), L^{-T}(t), (k,m), V_t)$.*

3. *If $U_{-T}(0) = L_{-T}(0)$ stop. Else set $T = 2T$ and repeat.*

*4. Return* $U^{-T}(0)$.

If $f$ is repulsive, (3.2) reduce to $\alpha_{\min}(U, L, (k, m), i) = (\nu(K_i)\,\lambda((k, m); U))/(p_i + p_i\, n(U \cap (K_i \times M)))$ respectively to $\alpha_{\max}(U, L, (k, m), i) = (\nu(K_i)\,\lambda((k, m); L))/(p_i + p_i\, n(L \cap (K_i \times M)))$. In the attractive case, no such reduction is possible in general, but we may use the slightly looser bounds $\alpha_{\min}(U, L, (k, m), i) = (\nu(K_i)\,\lambda((k, m); L))/(p_i + p_i\, n(U \cap (K_i \times M)))$ and $\alpha_{\max}(U, L, (k, m), i) = (\nu(K_i)\,\lambda((k, m); U))/(p_i + p_i\, n(L \cap (K_i \times M)))$.

## *3.2 Proof of correctness*

We need to check the conditions in [8, Theorem 2.1].

In order to do so, first we have to specify the dynamics of the dominating process, and its associated random variables, in greater detail. Apart from the $V_t$ and $U_t$ used in handling the selection of strips and forward births in subprocesses, associate with each $D(t)$ a random permutation $\Sigma(t)$ of its marked points in the strip given by $V_t$. This permutation serves to specify the order in which points may die (in the forward sense), both in $D$ and its subprocesses. Thus, in case of a death proposal at time $t$, if $D(t) \cap (K_{V_t} \times M) = \mathbf{y} = \{y_1, \ldots, y_n\}$ and $\Sigma(t) = (y_{j_1}, \ldots, y_{j_n})$ for some permutation $(j_1, \ldots, j_n)$ of the set $\{1, \ldots, n\}$, the point to die in the dominating process is $y_{j_1}$. The distribution of the $\Sigma$-process is specified conditionally on $D$ and $V$ as follows. If $D(t+1)$ is obtained from $D(t)$ by a death in the strip indexed by $V_t$, assign rank 1 to the marked point that is being removed, and let the ranks of the marked points in $D(t+1)$ with location in the cell indexed by $V_t$ follow a uniformly distributed permutation, independently of all other random variables. Otherwise, let $\Sigma(t)$ be a uniformly distributed permutation of the marked points in $D(t) \cap (K_{V_t} \times M)$. Thus, $\Sigma(t)$ depends on $V_t$, $D(t)$ and $D(t+1)$. The time-reversibility of $D(\cdot)$, the independence of the $U_t$ and $V_t$, and the construction of $\Sigma(\cdot)$ now imply the joint stationarity of $\{(D(t), V_t, \Sigma(t), U_t) : -\infty < t < \infty\}$. Note that conditionally given $D(t) = \mathbf{y} \neq \emptyset$ and $V_t = i$, permutation $\sigma$ of $\{1, \ldots, n(\mathbf{y} \cap (K_i \times M))\}$ has marginal conditional probability

$$\begin{aligned} P(\sigma|\mathbf{y}, i) &= p_b^i \frac{1}{n(\mathbf{y} \cap (K_i \times M))!} + p_d^i \frac{1}{n(\mathbf{y} \cap (K_i \times M))} \frac{1}{(n(\mathbf{y} \cap (K_i \times M)) - 1)!} \\ &= \frac{1}{n(\mathbf{y} \cap (K_i \times M))!}, \end{aligned}$$

i.e. a uniform distribution. Sample paths can be generated as follows. In $D(0)$, cell $i$ contains a shifted geometrically (3.1) distributed number of points (with parameter $p_i$) that are scattered $\nu$-uniformly and marked according to $\nu_M$ independent of each other and of other cells. The random number $U_0$ is uniformly distributed on $(0, 1)$, and $V_0$ is uniformly distributed on the set $\{1, \ldots, n_K\}$, independently of other sources of randomness. Because of its time reversibility, $D$ may easily be extended backwards into the past, say up to $-T$, and forward, say up to $T$. The same is true for the i.i.d. $U_t$ and $V_t$. Conditionally on the paths of $D$ and $V$, $\Sigma(-T), \ldots \Sigma(-1), \Sigma(0), \ldots, \Sigma(T-1)$ are then simulated forward as explained above.

Deaths in subprocesses are handled by the removal of that marked point that is assigned the lowest rank by $\Sigma$. We need to verify that this choice amounts to the deletion of a uniformly distributed point from those in the strip being updated. Thus, let $\Sigma$ be a uniformly distributed permutation of $A = \{y_1, \ldots, y_n\}$, and consider its restriction to the subset $B = \{y_{i_1}, \ldots, y_{i_m}\}$ of $A$. Note that each permutation $\sigma$ of the $n$ marked points has probability $1/n!$, that the order of the $m$ points in $B$ is fixed by $\sigma$, and that there are $(n-m)!$ permutations to fill the remaining ranks that can be chosen in $\binom{n}{n-m}$ different ways consistent with $\sigma$. Hence any permutation of $B$ has probability $1/m!$, as it should. Moreover, a death transition implemented in this way respects the inclusion order: after the deletion of the point with lowest $\Sigma$-rank, the modified $B$ is a subset of the modified $A$. In practice, only those parts of the permutation that are strictly necessary are generated, as in [8, procedure `MHDeath`].

In summary, the upper and lower processes are adapted functionals of the stationary marked dominating process $\{(D(t), V_t, \Sigma(t), U_t) : -\infty < t < \infty\}$. By construction the sandwiching and funnelling properties [8, (2.4)–(2.6)] hold.

Secondly, since the shifted geometric probability distribution assigns positive mass to 0, and the dominating process is in equilibrium, $D$ extended backwards will almost surely reach state $\emptyset$, so that the algorithm will almost surely terminate.

It remains to consider the target process $Y^{-T}(\cdot)$ defined as follows. Set $Y^{-T}(-T) = \emptyset$ and apply the same dynamics as for $U^{-T}(\cdot)$ and $L^{-T}(\cdot)$ except that if $Y^{-T}(t) = \mathbf{y}$ the birth at time $t$ of a marked point $(k, m)$ in the strip indexed by $V_t$ is accepted if

$$U_t \leq \frac{\nu(K_{V_t})\,\lambda((k,m);\mathbf{y})}{p_{V_t}\,(1 + n(\mathbf{y} \cap (K_{V_t} \times M)))}$$

Clearly, $L^{-T}(t) \subseteq Y^{-T}(t) \subseteq U^{-T}(t)$ for all integers $0 \geq t \geq -T$.

The process $Y(t)$ for $t = 0, 1, 2, \ldots$ with $Y(0) = \emptyset$ is defined analogously based on a forward run of the dominating process and its associated random variables. Note that $Y(t)$ exhibits the dynamics of a Metropolis–Hastings Markov process. If the current configuration is $\mathbf{y}$, a strip $K_i \times M$ is chosen uniformly. With probability $p_b^i$ a new marked point $\xi$ is generated in $K_i \times M$ from the probability distribution $\nu \times \nu_M/\nu(K_i)$, and accepted with probability $(\nu(K_i)\,\lambda(\xi;\mathbf{y}))\,/\,(p_i + p_i n(\mathbf{y} \cap K_i \times M)) \leq 1$, where as before $p_i = p_b^i/(1 - p_b^i)$. With the complementary probability $p_d^i$, a randomly chosen marked point located in $K_i$ is proposed for deletion, unless the strip is empty in which case no update occurs. The procedure is well defined on the set of configurations with strictly positive density.

By construction, $f$ is an invariant density for the transition kernel. Since deaths are always accepted, and $p_d^i > 1/2$, for any marked point pattern $\mathbf{y}$, the probability of reaching the empty set is larger than $(2n_K)^{-n(\mathbf{y})} > 0$. Hence the Metropolis–Hastings sampler is irreducible with respect to the Dirac measure $\delta_0$ on $\emptyset$, and consequently with respect to $f$ (cf. [4, Section 3.8]). Moreover, if the current state is $\emptyset$, the probability of staying put is at least $\sum_i p_d^i/n_K > 1/2 \geq (2n_K)^{-1}$. Consequently, sets of the form $\{\mathbf{y} : n(\mathbf{y}) \leq n\}$ (and their measurable subsets) are small with respect to $(2n_K)^{-n}\delta_0$. Since self transitions occur with positive probability, the sampler is aperiodic. Therefore, $Y(t)$ converges to the distribution specified by $f$ weakly and in total variation (note that the convergence holds for $f$-almost all initial states, but recall $Y(0) = \emptyset$, the atom of the limit distribution).

To get rid of the null set, we may use the drift condition [12, Thm 9.1.8]. Define a function $V$ on the non-negative integers by $V(0) = 1$, $V(1) = 2$ and

$$V(n+1) = V(n) + \frac{1}{2\,n_K\,p}\,[V(n) - V(n-1)]$$

for $n = 1, 2, \ldots$. Here $p = \max_i p_b^i / \min_i p_d^i$. Note that $V$ is strictly increasing, hence positive. Consequently, the level sets $\{\mathbf{y} : V(n(\mathbf{y})) \leq c\}$ for $c > 0$ are small sets. Moreover, for any

$\mathbf{y} \neq \emptyset$, we have

$$\mathbb{E}\left[V(n(Y(t+1)))|Y(t)=\mathbf{y}\right] - V(n(\mathbf{y})) =$$

$$\sum_{i=1}^{n_K} \frac{p_b^i}{n_K\,\nu(K_i)} \left[V(n(\mathbf{y})+1) - V(n(\mathbf{y}))\right] \int_{K_i}\int_M \frac{\nu(K_i)\,\lambda((k,m);\mathbf{y})}{p_i\,(1+n(\mathbf{y}\cap(K_i\times M)))}\,d\nu(k)\,d\nu_M(m) +$$

$$\sum_{i=1}^{n_K} \frac{p_d^i}{n_K} \left[\sum_{\xi\in K_i\times M} \frac{V(n(\mathbf{y})-1) - V(n(\mathbf{y}))}{n(\mathbf{y}\cap(K_i\times M))}\right]$$

$$\leq \sum_{i=1}^{n_K} \frac{p_b^i}{n_K}\left[V(n(\mathbf{y})+1) - V(n(\mathbf{y}))\right] - \frac{\min_i p_d^i}{n_K}\left[V(n(\mathbf{y})) - V(n(\mathbf{y})-1)\right]$$

$$\leq \max_i p_b^i \left[V(n(\mathbf{y})+1) - V(n(\mathbf{y}))\right] - \frac{\min_i p_d^i}{n_K}\left[V(n(\mathbf{y})) - V(n(\mathbf{y})-1)\right]$$

$$= -\frac{\min_i p_d^i}{2n_K}\left[V(n(\mathbf{y})) - V(n(\mathbf{y})-1)\right] < 0.$$

Here we use the convention that an empty sum is zero, the assumptions on the values of $p_b^i$ and $\nu(K_i)$, the fact that at least one strip is non-empty, and the definition of $V$. We conclude that the $Y$-chain is Harris recurrent, and the proof is complete.

## *3.3 Implementation*

Most Monte Carlo samplers encountered in the literature are based on a few simple updates: births and deaths of a single marked point. Some also allow for changes in the position or mark. However, as the details vary considerably between algorithms, these are hidden for the user in the non-public parts of the sampler classes.

```
class Sampler {
 public:
   Sampler( Prior *prior, char *name_file );
   virtual ~Sampler();
   virtual void setPrior( Prior *prior );
   Prior *getPrior() const;
   virtual void init( Pattern& D ) = 0;
   virtual void sim( Pattern& D ) = 0;
   virtual void report() const = 0;
   void clean();
   void printTimeJumps();
};
```

The constructor arguments are the name of a file to store output in, and the model to sample from. The model may be reset by the member function *setPrior*, and is retrieved by *getPrior*. The pure virtual member functions *init* and *sim* respectively set the initial state of the algorithm to an appropriate configuration, and run the sampler from there to obtain a realisation from the model. *Report* tells the user which sampler is being used. Since exact samplers often need to keep track of coupled sample paths, a clean up facility is essential and provided by the member function *clean*. Information on the amount of time needed to obtain a sampler as well as the actual number of jumps (in the case of the Metropolis–Hastings algorithm the number of transitions minus those that propose to delete a point from an empty strip) is written into the output file by the member function *PrintTimeJumps*.

In summary, a typical simulation loop looks as follows.

```
Prior *prior = new Model( ... );

char *parfile = "parf";
char *statsfile = "statsf";
char *patfile = "patf";
char *timefile = "timef";
prior->printInit( parfile, statsfile, patfile );

Sampler *sampler = new Algorithm( prior );
Pattern sample;
Pattern D;

for( int i=0; i<100; i++ ) {
   sampler->init( D );
   sample = D;
   sampler->sim( sample );
   prior->printSample( sample );
   sampler->clean();
   D.destroy():
}
prior->printClose();
```

It first creates the model to sample from and a sampler of choice, then opens output files to store the simulation results. For each realisation, an initial marked point configuration is set, the algorithm run and the output reported; memory taken up by the run is freed, and the initial configuration deleted, before a new realisation is generated. Finally, the output files are closed.

For the Metropolis–Hastings algorithm considered in the current paper, the constructor of the class ExactMH derived from Sampler needs two integers for the grid division of the rectangle $K \subseteq \mathbb{R}^2$ and a double for the birth probability. Discretisation of the mark space is model dependent and handled by class Prior. The proposal probabilities and model may be altered by the member function `void setParameters( Prior *p, double pb );`.

## 4. Simulation study

In this section, we present a simulation study to assess the range of applicability and the efficiency of the exact Metropolis–Hastings algorithm. The models we consider are the Widom–Rowlinson model (2.1) the Candy model (2.3), and two multi-type point processes. For the latter, we restrict ourselves to $I = 2$, and between type interaction only (i.e. $\gamma_{ii} \equiv 1$).

Such models are dubbed bivariate pairwise cross interaction models, and implemented by the following class.

```
class MPBiCross : public MPInteraction {
public:
   MPBiCross( double b, double r, double w, double h );
   void printSample( const Pattern& p ) const;
   Event *newEvent() const;
  };
```

We choose

$$\gamma_{12}(t) = \begin{cases} 1-(1-\gamma)\mathbf{1}\{t \le r\} & \textbf{Strauss} \\ 1-1/(1+(t/\sigma)^2)^2\mathbf{1}\{t \le 3\sigma\} & \textbf{Cauchy} \end{cases}$$

for the interaction function. Note that for Cauchy, the range of interaction is $r = 3\sigma$. Both the bivariate Strauss and Cauchy model can be implemented as a derived class from MPBiCross. All that is needed is a constructor for setting the parameters, member functions that can alter the current values of the parameters, and functions for reporting them. The actual implementation of the interaction function is hidden from the user in the private part of the class, as for the Widom–Rowlinson mixture model and the Candy line segment process. All models are sampled on the unit square in the simulation study below.

## *4.1 Choice of mesh size*

Algorithm 1 leaves some freedom in choosing the partition in strips and the birth proposal probabilities. The effect of different choices was examined in the following experiments. We assumed equal birth proposal probabilities $p_b^i \equiv p_b$, and partitioned the unit square $K = [0,1] \times [0,1]$ as an $m \times m$ grid of small squares of equal volume $1/m^2$.

| $p_b$ | Time | Jumps |
|---|---:|---:|
| 0.01 | 34471.94 | 690.80 |
| 0.10 | 4816.90 | 965.64 |
| 0.25 | 4048.90 | 2023.12 |
| 0.45 | 24707.07 | 22233.11 |

Table 1: Average coalescence time and mean number of jumps for simulating 500 samples of a Poisson process with $\beta = 50$, depending on the proposal probabilities.

In the first experiment, the mesh size $m$ was adapted to $p_b$ as $\lceil \sqrt{(}\Lambda/p) \rceil$ where $p = p_b/(1-p_b)$. We simulated a Poisson point process of intensity $\Lambda = 50$ using Algorithm 1 for different values of $p_b$. Note that any independent marking may be applied without affecting the running time of the algorithm. For each $p_b$-value, 500 independent realisations were generated, and the average coalescence time computed. We also recorded the mean number of actual jumps in the dominating chain $D$ by discarding the proposals to delete a point from an empty strip. The results are listed in Table 1.

| Mesh | Time | Jumps |
|---|---|---|
| 25 | 6520.83 | 1304.47 |
| 30 | 9437.18 | 1890.16 |
| 35 | 13828.10 | 2768.14 |
| 40 | 17580.03 | 3518.75 |

Table 2: Average coalescence time and mean number of jumps for simulating 500 samples of a Poisson process with $\beta = 50$, with proposal probability $p_b = 0.1$.

Next, the influence of the mesh size $m$ for fixed $p_b = 0.10$ was investigated (cf. Table 2). As in the previous case, 500 independent samples from a Poisson point process of rate 50 were used to compute the average coalescence time and number of jumps.

Note that for $p_b = 0.10$, $\lceil\sqrt{(\Lambda/p)}\rceil = 22$. Table 2 indicates that increasing $m$ leads to longer coalescence times. Table 1 suggests that a long coalescence time is required for both small and large values of $p_b$, while intermediate values ($p_b = 0.10$ or $0.25$) give good results.

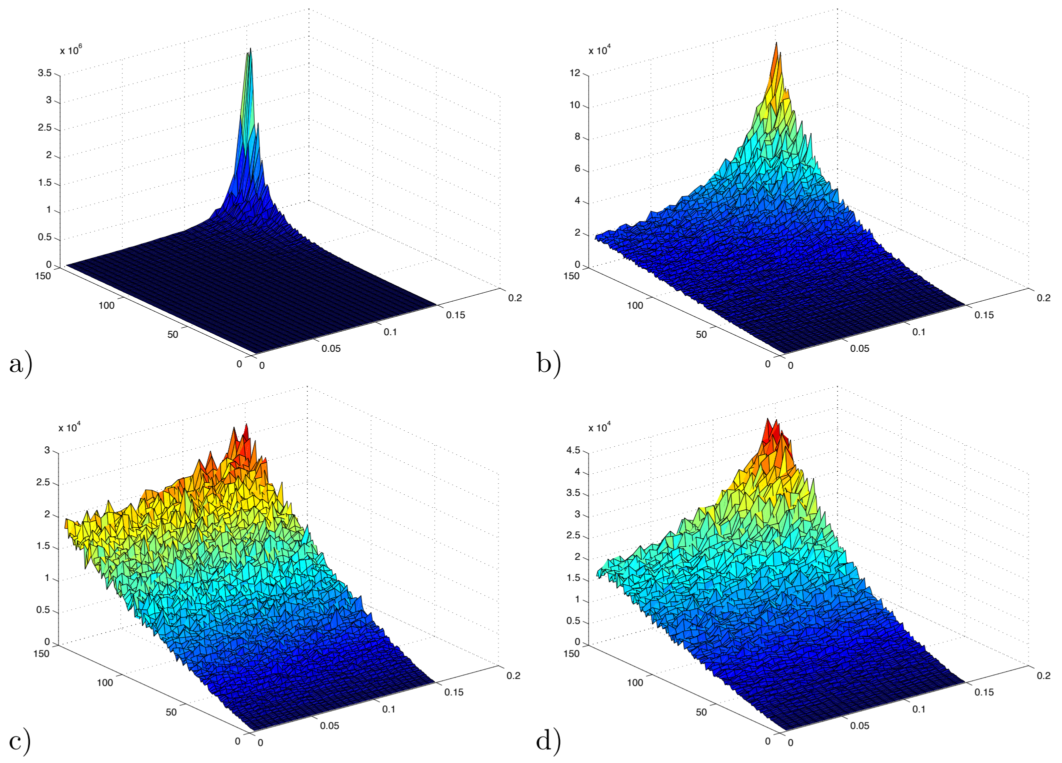

Figure 1: Metropolis–Hastings algorithm a) Widom-Rowlinson b) Strauss c) Cauchy d) Candy.

### 4.2 Coalescence times

In Figure 1 we plot the coalescence time $T$ for the exact Metropolis–Hastings algorithm of Section 3 against intensity and range parameters. It should be noted that the actual number of calculations is larger, as the method is based on successive doubling.

For the Widom–Rowlinson model, the intensity parameter is $\beta_1 = \beta_2$ which ranged between 1 and 150 with steps of 1. The hard core distance between points of different type was taken to be in between 0.005 and 0.15 with steps of 0.005.

For the bivariate Strauss and Cauchy models, the intensity parameter $\beta_1 = \beta_2$ also ranged between 1 and 150 with steps of 1. The range parameter of the Strauss interaction function is $r$, which we allowed to vary between 0.005 and 0.15 with steps of 0.005. We set the strength of interaction equal to $\gamma = 0.5$. For the Cauchy interaction function, the range $r = 3\sigma$. Again, we allowed $r$ to vary between 0.005 and 0.15 with steps of 0.005.

The parameters of the Candy model were chosen as follows. The intensity parameter $\beta$ ranged between 1 and 150 with steps of 1; the orientation and rejection parameters were set to $\gamma_o = \gamma_r = 0.5$, with $\tau = 0.1$ and $\delta = 0.05$. The range of interaction is determined by the length of the segments as $r = 1.25 l_{\max}$. Thus, we assumed the length distribution was concentrated on $l_{\max}$, which we let vary between 0.005 and 0.12 with steps of 0.005.

For each combination of parameters, 25 independent samples were generated by means of Algorithm 1, and the average coalescence time recorded. The birth proposal probability was $p_b = 0.1$, and the unit plane was divided into an $m \times m$ grid of equal volume cells with $m = \lceil \sqrt{(\Lambda/p)} \rceil$.

As expected, Figure 1 shows that the stronger the interaction, the longer it takes to obtain a sample. Indeed, for the same intensity and range parameters, the interaction in the Widom–Rowlinson model is of hard core type, whereas the Strauss interaction function is a positive constant. The Cauchy interaction is more severely repulsive than that of the Strauss model at very short interpoint distances, but less so for most of the range. Finally in the Candy model the maximum strength of interaction is the same as that of the Strauss bivariate interaction process, but only a subset of segments within the interaction range actually contributes to the conditional intensity.

Acknowledgments

The research was carried out partly while the second author was at CWI. It is a pleasure to express our thanks to A.G. Steenbeek for expert programming assistance and to K.K. Berthelsen for helpful comments.

# References

1. A.J. Baddeley and J. Møller. Nearest-neighbour Markov point processes and random sets. *International Statistical Review*, 57:89–121, 1989.
2. D.J. Daley and D. Vere-Jones. *An introduction to the theory of point processes.* New York: Springer Verlag, 1988.
3. P.A. Ferrari, R. Fernández and N.L. Garcia. Perfect simulation for interacting point processes, loss networks and Ising models. *Stochastic Processes and their Applications*, 102:63–88, 2002.
4. C.J. Geyer. Likelihood inference for spatial point processes, In *Stochastic geometry, likelihood and computation*, O. Barndorff–Nielsen, W.S. Kendall and M.N.M. van Lieshout (Eds.) Boca Raton: CRC Press/Chapman and Hall, 1999.
5. O. Häggström and K. Nelander. Exact sampling for anti-monotone systems. *Statistica Neerlandica*, 52:360–380, 1998.
6. O. Häggström and K. Nelander. On exact simulation of Markov random fields using coupling from the past. *Scandinavian Journal of Statistics*, 26:395–411, 1999.
7. W.S. Kendall. Perfect simulation for the area-interaction point process. In *Proceedings of the Symposium on Probability towards the year 2000*, L. Accardi and C. Heyde (Eds.) Berlin: Springer-Verlag, 1998.
8. W.S. Kendall and J. Møller. Perfect simulation using dominating processes on ordered spaces, with application to locally stable point processes. *Advances in Applied Probability (SGSA)*, 32:844–865, 2000.
9. M.N.M. van Lieshout and A.J. Baddeley. Extrapolating and interpolating spatial pattern. In *Spatial cluster modelling*, A.B. Lawson and D.G.T. Denison (Eds.) pp. 61–86. Boca Raton: Chapman and Hall/CRC Press, 2002.
10. M.N.M. van Lieshout and R.S. Stoica. The Candy model revisited: properties and inference. *Statistica Neerlandica*, 57:1–30, 2003.

11. M.N.M. van Lieshout and R.S. Stoica. Perfect simulation for marked point processes. CWI Report PNA-R0306, May 2003.

12. S.P. Meyn and R.L. Tweedie. *Markov chains and stochastic stability.* Springer-Verlag, London, 1993.

13. D.J. Murdoch and P.J. Green. Exact sampling from a continuous state space. *Scandinavian Journal of Statistics*, 25:483–502, 1998.

14. J.G. Propp and D.B. Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 9:223–252, 1996.

15. B.D. Ripley and F.P. Kelly. Markov point processes. *Journal of the London Mathematical Society*, 15:188–192, 1977.

16. A.G. Steenbeek, M.N.M. van Lieshout and R.S. Stoica with contributions from P. Gregori and K.K. Berthelsen. MPPBLIB, a `C++` library for marked point processes. CWI, 2002–2003.

17. R. Stoica, X. Descombes and J. Zerubia. A Gibbs point process for road extraction from remotely sensed images. *International Journal of Computer Vision*, 57:121–136, 2004.

18. B. Widom and J.S. Rowlinson. A new model for the study of liquid-vapor phase transition. *Journal of Chemical Physics*, 52:1670–1684, 1970.