Adv. Appl. Prob. **40,** 897–917 (2008)
Printed in Northern Ireland

PERFECT SAMPLING METHODS FOR RANDOM FORESTS

HONGSHENG DAI,* *Lancaster University*

Abstract

A weighted graph G is a pair $(V, \mathcal{E})$ containing vertex set V and edge set $\mathcal{E}$, where each edge $e \in \mathcal{E}$ is associated with a weight W_e. A subgraph of G is a forest if it has no cycles. All forests on the graph G form a probability space, where the probability of each forest is proportional to the product of the weights of its edges. This paper aims to simulate forests exactly from the target distribution. Methods based on coupling from the past (CFTP) and rejection sampling are presented. Comparisons of these methods are given theoretically and via simulation.

Keywords: Coupling from the past; MCMC; perfect sampling; rejection sampling; trees and forests

2000 Mathematics Subject Classification: Primary 65C05; 65C50
Secondary 05C80

1. Introduction

A weighted graph G is a pair $(V, \mathcal{E})$ containing vertex set V and edge set $\mathcal{E}$, where each edge $e \in \mathcal{E}$ is associated with a weight W_e. A subgraph of G is a forest, denoted by F, if it contains all vertices and has no cycles. A forest is a tree, denoted by T, if all vertices are connected. Figure 1 shows examples of a graph, a tree, and a forest.

Two probability spaces, the forest space and tree space, are particularly important. The forest space is formed by all forests in the graph G, with forest probability distribution

$$\mathrm{P}(F) \propto h(F) = \prod_{e \in F} W_e. \tag{1}$$

The tree space is defined similarly, with tree probability distribution

$$\mathrm{P}(T) \propto h(T) = \prod_{e \in T} W_e. \tag{2}$$

The forest space and tree space on graph G are denoted by $\mathcal{F}(G)$ and $\mathcal{T}(G)$, respectively. We use $\mathcal{W}_{\mathcal{F}(G)} := \sum_{F \in \mathcal{F}(G)} \prod_{e \in F} W_e$ and $\mathcal{W}_{\mathcal{T}(G)} := \sum_{T \in \mathcal{T}(G)} \prod_{e \in T} W_e$ to denote the normalising constants for distributions (1) and (2), respectively.

We are interested in the characteristics of the forest or tree distribution. For example, in graphical models, where vertices denote random variables and edges denote the conditional correlation of two variables given all other variables, we are interested in the edge inclusion

Received 7 February 2008; revision received 24 June 2008.
* Postal address: Department of Mathematics and Statistics, Fylde College, Lancaster University, Lancaster LA1 4YF, UK. Email address: h.dai@lancaster.ac.uk

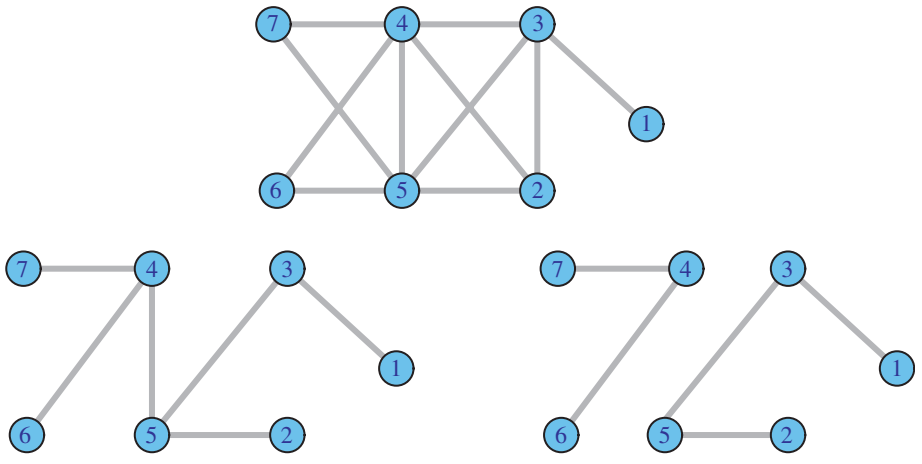

FIGURE 1: *Top*: a graph with seven vertices; *bottom-left*: a tree on the graph; *bottom-right*: a forest on the graph.

probability, $\mathrm{P}(e \in F)$, or the expected number of connected components in a forest. These quantities are nontrivial to calculate explicitly. Markov chain Monte Carlo (MCMC) methods, however, can be used to simulate forests or trees approximately from the above distributions and then, based on the sampled realisations, estimates can be obtained. A potential weakness of MCMC methods is that the simulated trajectory of a Markov chain will depend on its initial state. A common practical recommendation is to ignore the early stages, the so-called *burn-in* phase, before collecting realisations of the state of the chain. In practice, judgements about convergence are often made by visual inspection of the realised chain or the application of simple rules of thumb. Concerns about the *quality* of the sampled realisations of the simulated Markov chains have motivated the search for Monte Carlo methods that can be guaranteed to provide samples from the target distribution. This is usually referred to as *perfect sampling*.

Sampling a tree exactly from distribution (2) has been well studied in [1], [2], [4], [8], and [14]. However, sampling forests from distribution (1) has received little attention. Although distributions (1) and (2) are similar, perfect sampling for forests is much harder than perfect sampling for trees. This is because rescaling the edge weights, W_e, does not change the probability of a tree, since each spanning tree has the same number of edges, but it does change the probability of a forest. For example, the simple graph $G = (V, \mathcal{E})$, where $V = \{1, 2, 3\}$ and $\mathcal{E} = \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$, induces seven spanning forests. If each edge has weight $W_e = 1$ then each forest has probability $\mathrm{P}(F) = \frac{1}{7}$. But, if we rescale the weights by a constant factor $\alpha \neq 1$, that is, $W_e = \alpha$, then $\mathrm{P}(F) = \alpha^2/(1 + 3\alpha + 3\alpha^2)$ if F has two edges, and $\mathrm{P}(F) = \alpha/(1 + 3\alpha + 3\alpha^2)$ if F has one edge.

This paper aims to develop methods to draw samples exactly from distribution (1). We propose new sampling methods based on *coupling from the past* and rejection sampling. The paper is organised as follows. In Section 2 we introduce the notation and basic definitions. Then in Section 3 we develop coupling-from-the-past methods for sampling forests. In Section 4 we present rejection sampling methods. Comparisons of the two methods are provided in Section 5. Discussions are given in Section 6.

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

2. Preliminaries

An undirected *graph* G is a pair $(V, \mathcal{E})$ consisting of a set of vertices $V = \{1, \ldots, p\}$ and a set of edges $\mathcal{E}$ that join pairs of vertices. The edge $e \in \mathcal{E}$ that joins the vertices i and j is denoted by $e = \{i, j\}$. Vertices i and j are *neighbours* if $e = \{i, j\} \in \mathcal{E}$. A graph $G_1 = (V_1, \mathcal{E}_1)$ is called a *subgraph* of $G_2 = (V_2, \mathcal{E}_2)$ if $V_1 \subseteq V_2$ and $\mathcal{E}_1 \subseteq \mathcal{E}_2$. The *degree* of a vertex is the number of vertices to which it is joined. A leaf in a tree or forest is a vertex of degree 0 or 1. The *boundary* $\mathrm{bd}(A)$ of a subset A is the subset of vertices in $V \setminus A$ that are neighbours of vertices in A.

Let A be a subset of V. Then A is said to *induce* a subgraph $G_A = (A, \mathcal{E}_A)$, where $\mathcal{E}_A$ is the subset of $\mathcal{E}$ consisting of edges that join vertices in A. The *size* of a subgraph G_A is equal to the number of vertices in A.

A *simple path* between two vertices i and j is a sequence $i = \alpha_1, \ldots, \alpha_n = j$ of distinct vertices such that $\{\alpha_{k-1}, \alpha_k\} \in \mathcal{E}$ for all $k = 2, \ldots, n$. A closed path $\{\alpha_1, \alpha_2, \ldots, \alpha_n, \alpha_1\}$ is called a *cycle*. Vertices i and j are *connected* if there is a path between them. A graph is said to be connected if all pairs of vertices are connected. A graph $G = (V, \mathcal{E})$ is said to have *components* $G_{A_1}, \ldots, G_{A_k}$ if each subgraph G_{A_i} is connected, $A_1, \ldots, A_k$ is a partition of V, and $\mathrm{bd}(A_i) = \varnothing$, $i = 1, \ldots, k$.

A subgraph of G is a forest if it contains all vertices and has no cycles. A forest is a tree if all vertices are connected. Forests and trees equipped with probability distributions (1) and (2) are called *random forests* and *random trees*. Sampling a random forest or a random tree from its distribution can be rephrased as sampling it from the weighted graph G.

3. Coupling from the past

3.1. Introduction of coupling from the past

Coupling from the past (CFTP) was introduced in the landmark paper of Propp and Wilson [13], which showed how to provide perfect samples from the limiting distribution of a Markov chain. The idea is to run all possible Markov chains simultaneously from the past until all chains coalesce into a single chain, then keep running the coalesced chain and collect a sample at time 0.

Let $\{X_t\}$ be an ergodic Markov chain with state space $\mathcal{X} = \{1, \ldots, n\}$, where the probability of going from i to j is p_{ij} and the stationary distribution is π. Suppose that we design an update function $\phi(\cdot, U)$, which satisfies $\mathrm{P}(\phi(i, U) = j) = p_{ij}$, where ϕ is a deterministic function and U is a random variable. To simulate the next state Y of the Markov chain, currently in state i, we draw a random variable U and let $Y = \phi(i, U)$. Let $f_t(i) = \phi(i, U_t)$, and define the composition

$$F_{t_1}^{t_2} = f_{t_2-1} \circ f_{t_2-2} \circ \cdots \circ f_{t_1+1} \circ f_{t_1} \quad \text{for } t_1 < t_2.$$

The coalescent idea of CFTP is that if $F_{-M}^{0}(\mathcal{X})$ has only one element denoted by X_0^* then the unique element X_0^* is sampled from π.

Lemma 1. (From [13].) *Assume that, with probability* 1, *there exists a time* $t = -T$, *the backward coupling time, such that chains starting from any state in* $\mathcal{X} = \{1, \ldots, n\}$ *at time* $t = -T$, *and with the same sequence* $\{U_t,\ t = -T, \ldots, -1\}$, *arrive at the same state* X_0^*. *Then it must follow that* X_0^*, *defined with probability* 1, *comes from* π.

If we run an ergodic Markov chain from time $t = -\infty$ and with the sequence $\{U_t,\ t = -T, \ldots, -1\}$ after $-T$, the Markov chain will arrive at X_0^*. Then X_0^* comes exactly from π,

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

since it is collected at time 0 and the Markov chain started from $-\infty$. Therefore, a CFTP algorithm is that if (i) $F^0_{-M}(\cdot)$ is a single point then output $F^0_{-M}(\cdot)$, otherwise (ii) let $M = 2M$ and carry out step (i) again. Propp and Wilson [13] showed that the computational cost of the algorithm can be reduced if there is a partial order for the state space $\mathcal{X}$, which is preserved by the update function ϕ. This is called monotone CFTP. Although monotone CFTP is easy to perform, the requirement of monotonicity is very restrictive and finding a partial order preserved by the Markov chains is a nontrivial task in many cases. An alternative improvement is CFTP with bounding chains, such as that in [9] and [12]. If the bounding chains, which bound all the Markov chains, coalesce then all Markov chains coalesce. Thus, if only several bounding chains are required, the efficiency of the CFTP algorithm can be improved significantly.

3.2. MCMC algorithms

To use CFTP, we need to build an MCMC sampling algorithm. There is some previous work on constructing MCMC samplers for more general graphs. For example, Jones [10] constructed two samplers:

(a) Metropolis–Hastings where an edge to be updated is selected randomly, and

(b) Metropolis–Hastings where the choice of deleting or adding an edge is made first;

an edge is then selected at random either in the current graph or not, as appropriate. Propp and Wilson [14] provided a Markov chain for moving among directed trees. It updates the Markov chain by randomly selecting an edge not in the current directed tree and then deleting an existing edge.

We will start by describing the MCMC sampler (a), as above, i.e. to pick the edge e uniformly at random and then, with an appropriate probability, decide whether or not it should be added or deleted in the forest.

We need to introduce some notation before giving the MCMC algorithm. We use $\{i, j\}$ to denote an undirected edge and (i, j) to denote a directed edge (an arrow), $i \to j$. Given a weighted graph $G = (V, \mathcal{E})$ with weight $W_{i,j}$ for edge $e = \{i, j\}$, we use $(i, k) \sim \text{Unif}\{\mathcal{E}_d\}$ to denote that the directed edge (i, k) is drawn by selecting an edge e uniformly from the edge set $\mathcal{E}$ and the direction $i \to k$ is randomly assigned. We use $i \leftrightarrow k|G$ and $i \nleftrightarrow k|G$ to denote that vertices i and k are connected and not connected in graph G, respectively.

The following algorithm provides a possible way of running a forest Markov chain.

Algorithm 1. (Forest Markov chain algorithm.)

UpdateForest$((i,k), U, F_t, F_{t+1})$
Inputs: $(i,k) \sim \text{Unif}\{\mathcal{E}_d\}$, $U \sim \text{Unif}[0,1]$, and F_t; output: F_{t+1}.
If $e = \{i,k\} \in F_t$ or $i \nleftrightarrow k|F_t$ 01
    If $U \le W_{i,k}/(1 + W_{i,k})$ 02
        $F_{t+1} = F_t \cup e$ 03
    Else 04
        $F_{t+1} = F_t \setminus e$ 05
Else 06
    Find the path from i to k, $i \to j \ne k \to \cdots \to k$, in F_t 07
    If $U \le W_{i,k}/(W_{i,j} + W_{i,k})$ 08
        $F_{t+1} = F_t \cup \{i,k\} \setminus \{i,j\}$ 09
    Else 10
        $F_{t+1} = F_t$ 11

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

The algorithm samples an edge (i, k) uniformly from the edge set V and then decides whether (i, k) should be added to the current forest F_t or removed from F_t. If (i, k) is in F_t or vertices i and k are not connected in F_t then, by deleting $\{i, k\}$ from F_t or adding $\{i, k\}$ to F_t, the new graph is still a forest. In such cases we update the forest Markov chain according to lines 01 to 05. If i and k are connected in the current forest then, by adding $\{i, k\}$ to F_t, the new graph will have a cycle. In this case we have to remove an edge from the cycle to guarantee that the new graph is a forest. The edge to be removed is either $\{i, k\}$ or some other edge $\{i, j\}$, which is randomly chosen. If $\{i, k\}$ is removed then the chain does not move (line 11); if $\{i, j\}$ is removed then $F_{t+1} = F_t \cup \{i, k\} \setminus \{i, j\}$ (line 09). The Markov chain generated by repeating Algorithm 1 is ergodic and we have the following result.

Proposition 1. *The stationary distribution of the Markov chain generated by repeating Algorithm 1 is* $\mathrm{P}(F)$ *given in (1).*

Proof. Algorithm 1 gives us a probability transition matrix $R(F_0, F_1)$. Since

$$\frac{\mathrm{P}(F_1)R(F_1, F_0)}{\mathrm{P}(F_0)R(F_0, F_1)} = 1,$$

the Markov chain has stationary distribution (1) by detailed balance.

3.3. CFTP with a lower chain

In this subsection we present a CFTP method where one bounding chain is involved.

A forest F is uniquely determined by its edge set $\mathcal{E}(F)$. Therefore, we use $\{\mathcal{E}(F_t)\}$ to denote the state of a forest Markov chain at time t. CFTP involves running individual Markov chains simultaneously, starting from each possible initial forest. The chains are coupled so that when two trajectories coincide they coalesce and continue as a single chain. The chains therefore reduce in number as time goes on. At time t, let $\mathcal{F}_t$ be the set of forests that remain and let $\cap\mathcal{E}(F_t) = \bigcap_{F\in\mathcal{F}_t} \mathcal{E}(F)$ be the set of edges that are common to these remaining forests. Let $\mathcal{L}_t$ be a subset of $\cap\mathcal{E}(F_t)$.

Suppose that we could run all the Markov chains starting from $-\infty$ and have a sequence $\{\mathcal{L}_t\}$, as described above. We call $\{\mathcal{L}_t\}$ the lower chain. Suppose that we record the graphical structure of $\mathcal{L}_t$ at each step. If $\mathcal{L}_t$ becomes a spanning tree at time $-\tau$ then all the forest Markov chains will have the same structure as $\mathcal{L}_t$. This means that the forest Markov chains coalesce at time $-\tau$.

From the above observation we see that we should try to find updating rules for the chain $\{\mathcal{L}_t\}$ such that if $\mathcal{L}_t \subset \cap\mathcal{E}(F_t)$ then $\mathcal{L}_{t+1} \subset \cap\mathcal{E}(F_{t+1})$. Algorithm 2, below, provides an updating approach for the lower chain $\mathcal{L}_t$ which guarantees that this condition is satisfied. Note that the following notation is used in Algorithm 2. We use A and C to denote the set of neighbours of i in G and $\mathcal{L}_t$, respectively. We use $\{i, A\}$ to denote the edge set of i in the graph G, and we use $\{i, C\}$ to denote the edge set of i in $\mathcal{L}_t$. We define $W_{i,0} = 1$.

Algorithm 2. (An updating approach for the lower chain.)

```
LowerCFTP((i, k), U, L_t, L_{t+1})
# Inputs: (i, k) ~ Unif{E_d}, U ~ Unif[0, 1], and L_t; output: L_{t+1}.
Find vertex sets A and C                                              01
# Here A is the boundary of vertex i in G and C is the boundary
# of i in L_t.
If {i, k} ∈ L_t                                                       02
   If U ≤ W_{i,k}/(1 + W_{i,k}) then L_{t+1} = L_t ∪ e                03
```

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

```
    Else then $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus e$                                   04
Else if $i \nleftrightarrow k | \mathcal{L}_t$                                                  05
    Find the path $i \to j(\neq k) \to \cdots \to k \in \mathcal{L}_t$                          06
    If $U \le W_{i,k}/(W_{i,j} + W_{i,k})$ then $\mathcal{L}_{t+1} = \mathcal{L}_t \cup \{\{i,k\}\} \setminus \{\{i,j\}\}$   07
    Else then $\mathcal{L}_{t+1} = \mathcal{L}_t$                                               08
Else                                                                                            09
    If $U \le \min_{j \in A \cup \{0\}} W_{i,k}/(W_{i,j} + W_{i,k})$                            10
        $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i, C\} \cup \{\{i,k\}\}$                 11
    Else                                                                                        12
        Find $D = \{j | j \in C, j \ne k, U \le W_{i,k}/(W_{i,k} + W_{i,j})\}$                  13
        $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i, D\}$                                  14
```

The following proposition proves that $\mathcal{L}_t$ is always a subset of $\cap\mathcal{E}(F_t)$.

Proposition 2. *With Algorithm 2, if $\mathcal{L}_t \subset \cap\mathcal{E}(F_t)$ then $\mathcal{L}_{t+1} \subset \cap\mathcal{E}(F_{t+1})$.*

Proof. See Appendix A.

Here we only provide a simple explanation for the algorithm. In Algorithm 2, lines 02 to 08 update the lower chain according to the updating rules of the MCMC algorithm (Algorithm 1). They guarantee Proposition 2 since only a subtree in $\mathcal{L}_t$ is involved and this subtree must be in any F, $F \in \mathcal{F}_t$.

When $i \nleftrightarrow k | \mathcal{L}_t$, lines 10 to 14 of Algorithm 2 will be performed. The edge $\{i,k\}$ will be added in $\mathcal{L}_t$, if it should be added in all forests, and some edges are removed from $\mathcal{L}_t$ to guarantee Proposition 2.

Finally, we use Algorithm 3, below, to sample a forest from the target distribution.

Algorithm 3. (CFTP with a lower chain.)

```
FLAG $= 0$, $t = -M$ and $\mathcal{L}_t = \varnothing$                                          01
Generate $U_t \sim \mathrm{Unif}[0,1]$ and $(i,k)_t \sim \mathrm{Unif}\{\mathcal{E}_d\}: t = -\infty, \ldots, -1$   02
Repeat FLAG$= 1$ and $t = 0$                                                                    03
    If FLAG$= 0$                                                                                04
        LowerCFTP$((i,k)_t, U_t, \mathcal{L}_t, \mathcal{L}_{t+1})$                             05
    Else                                                                                        06
        UpdateForest$((i,k)_t, U_t, \mathcal{L}_t, \mathcal{L}_{t+1})$                          07
    If FLAG$= 0$ and $\mathcal{L}_{t+1}$ is a spanning tree                                     08
        FLAG$= 1$                                                                               09
    $t = t + 1$                                                                                 10
    If FLAG$= 0$ and $t = 0$                                                                    11
        $M = 2M$, $t = -M$ and $\mathcal{L}_t = \varnothing$                                    12
```

Proposition 3. *Algorithm 3 returns a forest with probability distribution (1).*

Proof. This follows since all the Markov chains coalesce.

The running time of this CFTP algorithm is at least equal to the waiting time that the lower chain becomes a tree (see the complexity analysis in later sections). Therefore, when the graph is not strongly connected (different parts of the graph are connected via edges with very small weights), coupling takes a long time. To improve the coupling, we add an upper chain to the algorithm, as described in the next subsection.

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

3.4. CFTP with bounding chains

We follow the notation of Section 3.3 and we define, at time t, $\mathcal{U}_t$ to be a superset of the union of the edges in all the edges in the remaining forests, that is,

$$\mathcal{U}_t \supset \cup\mathcal{E}(F_t) = \bigcup_{F\in\mathcal{F}_t} \mathcal{E}(F).$$

The sequence $\{\mathcal{U}_t\}_{-\infty}^{0}$ can be viewed as an upper chain. We run $\{\mathcal{L}_t\}$ and $\{\mathcal{U}_t\}$ simultaneously. When $\mathcal{L}_t = \mathcal{U}_t$, all the forest Markov chains are squeezed into $\mathcal{L}_t$. This means coalescence. Therefore, we only need to set up the updating rules for $\mathcal{L}_t$ and $\mathcal{U}_t$ such that, for any F_t, at any time t, we have $\mathcal{L}_t \subset \mathcal{E}(F_t) \subset \mathcal{U}_t$. Algorithm 4 and Algorithm 5, below, together provide us with a way of doing this.

Algorithm 4. (An updating approach for the bounding chains.)

LowerUpperCFTP$((i,k), U, \mathcal{L}_t, \mathcal{U}_t, \mathcal{L}_{t+1}, \mathcal{U}_{t+1})$
Inputs: $(i,k) \sim \text{Unif}\{\mathcal{E}_d\}$, $U \sim \text{Unif}[0,1]$, $\mathcal{L}_t$, and $\mathcal{U}_t$;
outputs: $\mathcal{L}_{t+1}$ and $\mathcal{U}_{t+1}$.
If $\{i,k\} \in \mathcal{L}_t$ 01
  If $U \le W_{i,k}/(1+W_{i,k})$ 02
    $\mathcal{L}_{t+1} = \mathcal{L}_t$, $\mathcal{U}_{t+1} = \mathcal{U}_t$ 03
  Else then $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i,k\}$, $\mathcal{U}_{t+1} = \mathcal{U}_t \setminus \{i,k\}$ 04
Else if $i \leftrightarrow k | \mathcal{L}_t$ 05
  Find the path $i \to j(\neq k) \to \cdots \to k \in \mathcal{L}_t$ 06
  If $U \le W_{i,k}/(W_{i,j}+W_{i,k})$ 07
    $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i,j\} \cup \{i,k\}$, $\mathcal{U}_{t+1} = \mathcal{U}_t \setminus \{i,j\} \cup \{i,k\}$ 08
  Else then $\mathcal{L}_{t+1} = \mathcal{L}_t$, $\mathcal{U}_{t+1} = \mathcal{U}_t$ 09
Else 10
  UPDATE$((i,k), U, \mathcal{L}_t, \mathcal{U}_t, \mathcal{L}_{t+1}, \mathcal{U}_{t+1})$ 11

Algorithm 5. (The subroutine of Algorithm 4.)

UPDATE $((i,k), U, \mathcal{L}_t, \mathcal{U}_t, \mathcal{L}_{t+1}, \mathcal{U}_{t+1})$
Inputs: $(i,k) \sim \text{Unif}\{\mathcal{E}_d\}$, $U \sim \text{Unif}[0,1]$, $\mathcal{L}_t$, and $\mathcal{U}_t$;
Outputs: $\mathcal{L}_{t+1}$ and $\mathcal{U}_{t+1}$.
Find the set $B = \{j | j \neq i, i \to j \to \cdots \to k \in \mathcal{U}_t\}$ 01
If $B = \phi$ or $B = \{k\}$ 02
  If $U \le W_{ik}/(W_{ik}+1)$ 03
    $\mathcal{U}_{t+1} = \mathcal{U}_t \cup \{i,k\}$ and $\mathcal{L}_{t+1} = \mathcal{L}_t \cup \{i,k\}$ 04
  Else then $\mathcal{U}_{t+1} = \mathcal{U}_t \setminus \{i,k\}$ and $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i,k\}$ 05
Else 06
  Find 07
    $C = \{j | j \neq k, j \neq i, \{i,j\} \in \mathcal{L}_t\}$
    $D = \{j | j \in C, j \neq k, U \le W_{ik}/(W_{ik}+W_{ij})\}$
    $A = B \cup \{0\} \setminus \{k\}$
  If $U \le \min_{j\in A}\{W_{i,k}/(W_{i,j}+W_{i,k})\}$ 08
    Find $E = \{l | l \neq k$ and $k \leftrightarrow l | \mathcal{L}_t\}$ 09
    $\mathcal{L}_{t+1} = (\mathcal{L}_t \setminus \{i,C\}) \cup \{i,k\}$ and $\mathcal{U}_{t+1} = (\mathcal{U}_t \setminus \{i,E\}) \cup \{i,k\}$ 10
  Else if $U \ge \max_{j\in A}\{W_{i,k}/(W_{i,j}+W_{i,k})\}$ 11
    $\mathcal{U}_{t+1} = \mathcal{U}_t \setminus \{i,k\}$ and $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i,k\}$ 12
  Else then $\mathcal{U}_{t+1} = \mathcal{U}_t \cup \{i,k\}$ and $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i,D\}$ 13

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

In Algorithm 5, $B = \phi$ or $B = \{k\}$ means that $i \nleftrightarrow k | \mathcal{U}_t$ or i and k are connected only via edge $\{i, k\}$. The vertex sets C and D are defined similarly to those in Algorithm 2.

The following proposition proves that $\mathcal{L}_t$ and $\mathcal{U}_t$ governed by Algorithm 4 and Algorithm 5 bound all the edge sets, $\mathcal{E}(F)$, for $F \in \mathcal{F}_t$.

Proposition 4. *Given $\mathcal{U}_t \supset \cup\mathcal{E}(F_t)$ and $\cap\mathcal{E}(F_t) \supset \mathcal{L}_t$, Algorithm 4 and Algorithm 5 guarantee that $\mathcal{U}_{t+1} \supset \cup\mathcal{E}(F_{t+1})$ and $\cap\mathcal{E}(F_{t+1}) \supset \mathcal{L}_{t+1}$.*

Proof. See Appendix B.

Here we only provide a simple explanation for the algorithms. In Algorithm 4, lines 01 to 09 update $\mathcal{L}_t$ and $\mathcal{U}_t$ according to the MCMC algorithm (Algorithm 1). When $i \nleftrightarrow k | \mathcal{L}_t$ (line 10), it will call the subroutine **UPDATE**, given by Algorithm 5. In Algorithm 5, lines 02 to 05 update $\mathcal{U}_t$ and $\mathcal{L}_t$ according to the MCMC algorithm and they guarantee Proposition 4 since i and k can be connected only through the edge $\{i, k\}$. Lines 08 to 10 of Algorithm 5 add edge $\{i, k\}$ to $\mathcal{L}_t$ and $\mathcal{U}_t$, delete all edges where possible from $\mathcal{L}_t$, and delete all edges $\{i, l\}$, which conflict with $i \leftrightarrow k \leftrightarrow l | \mathcal{U}_t$, from $\mathcal{U}_t$. Lines 11 to 12 delete $\{i, k\}$ from both $\mathcal{L}_t$ and $\mathcal{U}_t$. Line 13 adds $\{i, k\}$ to $\mathcal{U}_t$ and deletes $\{i, k\}$ from $\mathcal{L}_t$.

Finally, we have Algorithm 6, below, the CFTP algorithm with two bounding chains, which returns a forest from the target distribution.

Algorithm 6. (CFTP with bounding chains.)

```
FLAG = 0, t = −M, U_t = E(G) and L_t = ∅                                  01
Generate U_t ~ Unif[0, 1] and (i, k)_t ~ Unif{E_d}: t = −∞, ..., 0        02
Repeat until FLAG= 1 and t = 0                                            03
    If FLAG= 0                                                            04
        LowerUpperCFTP((i, k)_t, U_t, L_t, U_t, L_{t+1}, U_{t+1})         05
    Else                                                                  06
        UpdateForest((i, k)_t, U_t, L_t, L_{t+1})                         07
    If FLAG= 0, and L_{t+1} = U_{t+1} or L_{t+1} is a spanning tree       08
        FLAG= 1                                                           09
    t = t + 1                                                             10
    If FLAG= 0 and t = 0                                                  11
        M = 2M, t = −M, U_t = E(G) and L_t = ∅                            12
```

Proposition 5. *Algorithm 6 returns a forest with probability distribution (1).*

Proof. This follows since, when the upper chain and the lower chain become the same, all the Markov chains coalesce.

Algorithm 6 is more efficient than Algorithm 3. This is because we have two bounding chains: the upper chain and the lower chain. The upper chain will remove the edges which have tiny weights. Then the graph is divided into several parts. Thus, the large graph will be divided into several small graphs which have edges with large weights. The lower chain will deal with these small graphs. So, if the upper chain and lower chain work together, the coalescence will be achieved rapidly.

Note that finding the vertex sets A, B, C, D, and E in Algorithm 4 is not difficult. Vertex sets C or D can be found in polynomial time, since it is equivalent to finding some neighbour set of i in $\mathcal{L}_t$. Vertex set E is the connectivity set of k in $\mathcal{L}_t$. It needs at most time complexity $O(p^2)$, the complexity of visiting all edges. Finding vertex set B needs a running time of at most $pO(\omega)$, where ω is the complexity of finding a path from i to k in $\mathcal{U}_t$. So B can be found

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

in polynomial time since finding a path from i to k is easier than finding the connectivity set of i. It then follows that A can be found in polynomial time.

4. Rejection sampling methods

4.1. Rejection sampler with Bernoulli sampling

In this subsection we present a naive sampling algorithm.

Algorithm 7. (Bernoulli rejection sampler.)

```
# Inputs: G and W_e, e ∈ ℰ = {e_1, ..., e_m}; output: F.
g = ∅                                                          01
Repeat i = 1, ··· , m                                          02
    U ~ Unif[0, 1]                                             03
    If U ≤ W_{e_i}/(1 + W_{e_i})                               04
        g = g ∪ e_i                                            05
        If g is not a forest                                   06
          Return to line 02 and g = ∅                          07
F = g                                                          08
```

Algorithm 7 samples edges one by one according to the probability $W_e/(1 + W_e)$. The result graph is accepted if it is a forest, otherwise it is rejected. It can easily be proved that Algorithm 7 returns a forest from the target distribution. The algorithm will be efficient if the graph G is a sparse graph and no cycle in G is made of heavily weighted edges. If there is a cycle in which all edge weights are large then the Bernoulli sampler is likely to return a graph g with this cycle and g will be rejected.

4.2. Rejection method based on tree sampling

Given a graph G, we may add a new vertex 0 and edges $\{0, i\}$, $i = 1, \ldots, p$, to the graph and obtain $\tilde{G}$. Then we can sample a random tree $\tilde{T}$ from $\tilde{G}$ by using any existing tree sampling methods given in [1], [4], [8], or [14]. Then delete the edges of vertex 0 from $\tilde{T}$ and obtain a forest F^*.

Proposition 6. *If a forest F^* is sampled by the above approach then F^* has probability $\hat{\mathrm{P}}(F^*)$ given by*

$$\hat{\mathrm{P}}(F^*) \propto \hat{h}(F^*) = c(F^*) \prod_{e \in \mathcal{E}(F^*)} W_e,$$

where

$$c(F^*) = \prod_{i=1}^{k} s_i,$$

k is the number of components in F^*, and s_i is the size of the ith component.

Proof. If we delete vertex 0 and its edges in $\tilde{T}$, sampled from $\tilde{G}$, then we obtain a forest F^*. Each F^* may correspond to several different trees, $\tilde{T}$. This is illustrated in Figure 2. Let $\mathcal{A} = \mathcal{A}(F^*) = \{\tilde{T} : \text{ such that } F \text{ can be derived from } \tilde{T}\}$. Therefore,

$$\hat{\mathrm{P}}(F^*) = \frac{\sum_{\tilde{T} \in \mathcal{A}} \prod_{e \in \mathcal{E}(\tilde{T})} W_e}{\sum_{\tilde{T} \in \mathcal{T}(\tilde{G})} \prod_{e \in \mathcal{E}(\tilde{T})} W_e}.$$

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

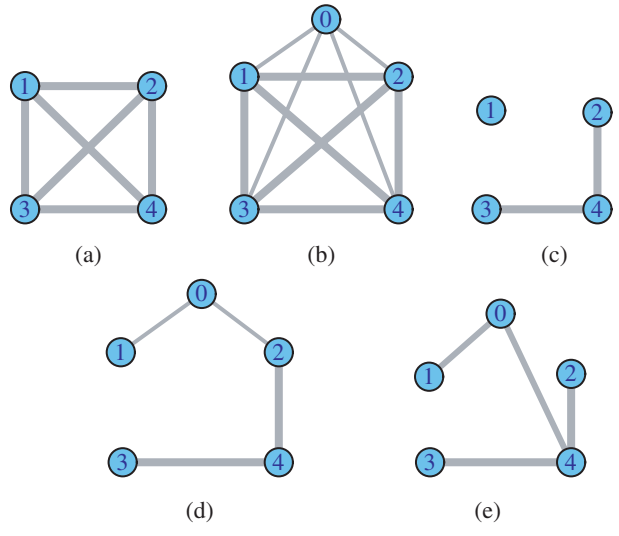

FIGURE 2: Rejection sampler. (a) Original graph G; (b) graph $\tilde{G}$; (c) the sampled forest; (d) and (e) two trees sampled from $\tilde{G}$.

Obviously, all the $\tilde{T}$s in $\mathcal{A}$ have the same probability. Then it follows that

$$\hat{\mathrm{P}}(F^*) = \frac{c(F^*)\prod_{e\in\mathcal{E}(F^*)} W_e}{\sum_{F^*\in\mathcal{F}(G)} c(F^*)\prod_{e\in\mathcal{E}(F^*)} W_e},$$

where $c(F^*)$ denotes the number of trees that correspond to the same forest F^*. We may also write $\hat{\mathrm{P}}(F^*) \propto \hat{h}(F^*) = c(F^*)\prod_{e\in\mathcal{E}(F^*)} W_e$.

Suppose that F^* has k components and that the ith component has size s_i. Then the ith component of F^* is a subtree with s_i vertices. This subtree can be derived from s_i different subtrees in $\tilde{G}$ by deleting vertex 0 and its edges. Therefore, the total number of $\tilde{T}$s in $\mathcal{A}$ is $c(F^*) = \prod_{i=1}^{k} s_i$.

We have $h(F) \le \hat{h}(F)$, according to $c(F) \ge 1$. We can then use rejection sampling to sample F from $\mathrm{P}(F)$, since we can sample from $\hat{h}(F)$, by using Proposition 6. Therefore, summarising the above, we have the following algorithm which returns a forest with probability distribution given by (1).

Algorithm 8. (Rejection sampler based on tree sampling.)

```
# Inputs: G and W_e, e ∈ E; output: F.
Sample a tree T from G̃ and U from Unif[0, 1]                    01
Delete vertex 0 and its edges in T̃ and achieve an undirected
forest F*                                                       02
# F* has the probability P̂(F*).
If U ≤ P(F*)/P̂(F*) then accept and output F*;                   03
else reject F* and go back to step 01.                          04
```

The acceptance probability of the algorithm is $\mathcal{W}_{\mathcal{F}(G)}/\mathcal{W}_{\mathcal{T}(\tilde{G})}$. Therefore, the efficiency of the algorithm depends on the graph structure. In Section 5 we will provide a detailed complexity analysis for the algorithm.

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

There are various tree sampling algorithms. The methods in [4] and [8] are based on the matrix tree theorem (see Appendix D) and matrix determinant calculation, therefore, they are called determinant-based tree sampling algorithms. The methods in [1] and [14] are based on running random walks on the graphs, therefore, they are called random-walk tree sampling algorithms. According to different tree sampling methods, we categorise rejection sampling based on tree sampling (Algorithm 8) as determinant rejection sampling and random-walk rejection sampling.

5. Complexity analysis for special graphs

All existing tree sampling algorithms [1], [4], [8], [14] are polynomial-time algorithms with complexity $O(p^3)$ (determinant-based tree sampling algorithms) or $O(p\log(p))$ (random-walk tree sampling algorithms). However, sampling forests is more complex than sampling trees. When we sample a random forest from a weighted graph, analysing the complexity of the algorithms in Sections 3 and 4 is nontrivial. But it is possible to derive complexity results for simple graphs, such as complete graphs and square lattices with weight 1 for each edge.

Definition 1. A square lattice $L_n = (V, \mathcal{E})$ is a graph having vertex set

$$V = \{1, \ldots, n\} \times \{1, \ldots, n\},$$

with the two vertices (i, j) and (i', j') being adjacent if $|i - i'| + |j - j'| = 1$.

5.1. Complexity analysis of CFTP algorithms

When we sample a forest from a uniformly weighted square lattice with Algorithm 3, we can show that the running time of the algorithm grows exponentially, as the number of vertices increases. Algorithm 2 can be simplified to Algorithm 9, below, for a lattice graph where each edge has weight 1.

Algorithm 9. (A lower chain for a square lattice.)

LowerUniformCFTP$((i,k), U, \mathcal{L}_t, \mathcal{L}_{t+1})$
Inputs: $(i,k) \sim \text{Unif}\{\mathcal{E}_d\}$, $U \sim \text{Unif}[0,1]$, and $\mathcal{L}_t$; output: $\mathcal{L}_{t+1}$.
If $\{i,k\} \in \mathcal{L}_t$
  If $U \le 0.5$ then $\mathcal{L}_{t+1} = \mathcal{L}_t$
  Else then $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i,k\}$
Else if $i \leftrightarrow k | \mathcal{L}_t$
  Find the path $i \to j \ne k \to \cdots \to k \in \mathcal{L}_t$
  If $U \le 0.5$ then $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i,j\} \cup \{i,k\}$
  Else $\mathcal{L}_{t+1} = \mathcal{L}_t$
Else
  If $U \le 0.5$ then $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i,C\} \cup \{i,k\}$
  Else $\mathcal{L}_{t+1} = \mathcal{L}_t$

Proposition 7. *Given that G is a uniformly weighted square lattice, if we start $\mathcal{L}_t$ from the empty forest and update it with Algorithm 9, then it takes an exponential time to become a spanning tree.*

Proof. See Appendix C.

When we sample from a square lattice, Algorithm 6 is more efficient than Algorithm 3, since Algorithm 6 uses two bounding chains and the amount of coalescent time is always less than the amount of time required by CFTP with a lower chain. But it is nontrivial to analyse

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

theoretically the complexity of Algorithm 6 for sampling from a square lattice. We therefore leave this as future research work.

5.2. Complexity analysis for Algorithm 8

Algorithm 8 is a polynomial-time algorithm when sampling from a complete graph with weight 1 on each edge, but it is an exponential-time algorithm when sampling from a square lattice.

Proposition 8. *The running time of Algorithm 8 for the complete graph with weight* 1 *on each edge is* $O((p+1)\gamma)$*, where* γ *is the running time of sampling a tree from the graph* $\tilde{G}$*.*

Proof. For a complete graph with weight 1 on each edge, the normalising constant $\mathcal{W}_{\mathcal{F}(G)}$ or $\mathcal{W}_{\mathcal{T}(G)}$ is the number of forests or trees in G. Therefore, the running time of the rejection sampler (Algorithm 8) is the running time of sampling $\tilde{T}$ from $\tilde{G}$ ($\gamma = O((p+1)\log(p+1))$ for random-walk tree sampling methods and $\gamma = O((p+1)^3)$ for determinant tree sampling methods) divided by the acceptance probability, $\mathcal{W}_{\mathcal{F}(G)}/\mathcal{W}_{\mathcal{T}(\tilde{G})}$.

It is well known that the number of trees in a complete graph is p^{p-2}. According to [15], we know that in a complete graph $\mathcal{W}_{\mathcal{F}(G)}/\mathcal{W}_{\mathcal{T}(G)}$, the ratio of the number of forests and the number of spanning trees, is approximately $\sqrt{\mathrm{e}}$ when p is large. We then have

$$\frac{\mathcal{W}_{\mathcal{F}(G)}}{\mathcal{W}_{\mathcal{T}(\tilde{G})}} \approx \frac{\sqrt{\mathrm{e}}\,p^{p-2}}{(p+1)^{p-1}} = \frac{1/(p+1)}{\sqrt{\mathrm{e}}} \quad \text{for large } p.$$

Therefore, Algorithm 8 is a polynomial-time algorithm when sampling from complete graphs.

Proposition 9. *Algorithm 8 is an exponential-time algorithm when we sample a forest from a uniformly weighted square lattice, i.e. a square lattice with* $W_e = 1$ *for each edge.*

Proof. From Lemmas 2 and 3, we have

$$\frac{\mathcal{W}_{\mathcal{T}(\tilde{G}_n)}}{\mathcal{W}_{\mathcal{F}(G_n)}} \geq \frac{4.2655^{(n-1)(n-2)}}{3.741\,01^{n^2}},$$

which increases exponentially with n^2. This means that the acceptance probability decreases to 0 exponentially with n^2. Therefore, Algorithm 8 needs an exponential running time when sampling from a square lattice.

Lemma 2. (From [3].) *Let* G_n *be an* $n \times n$ *square lattice* L_n*. Then*

$$3.644\,97 \leq \lim_{n\to\infty} [\mathcal{W}_{\mathcal{F}(G_n)}]^{1/n^2} \leq 3.741\,01.$$

Lemma 3. *Let* $\tilde{G}_n$ *be the graph derived from the* $n \times n$ *square lattice by Algorithm 8. The number of spanning trees in* $\tilde{G}_n$ *satisfies* $\mathcal{W}_{\mathcal{T}(\tilde{G}_n)} > 4.2655^{(n-1)(n-2)}$.

Proof. See Appendix D.

5.3. Method comparisons via simulation

We compare the running times of various forest sampling algorithms for uniformly weighted complete graphs and lattices. We consider Bernoulli rejection sampling, CFTP with bounding chains, and random-walk rejection sampling. We ignore CFTP with a lower chain and determinant rejection sampling since, for complete graphs and lattices, they are always less efficient than CFTP with bounding chains and random-walk rejection sampling, respectively.

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

TABLE 1: Running time (in seconds) comparisons for forest sampling on a square lattice, where the last row gives the mean number of trees required for each forest in random-walk rejection sampling.

Number of nodes	16	25	36
Bernoulli rejection sampling	<1	1	3
CFTP	14	66	234
Random-walk rejection sampling	8	149	4120
Mean number of trees	55	443	4590

TABLE 2: Running time (in seconds) comparisons for sampling on uniformly weighted complete graphs, where the last row gives the mean number of trees required for each forest in random-walk rejection sampling.

Number of nodes	15	25	35
Bernoulli rejection sampling	–	–	–
CFTP	1.2×10^3	2.8×10^4	7.9×10^5
Random-walk rejection sampling	2.5	10	36
Mean number of trees	19	32	50

The results are shown in Table 1 and Table 2. For sampling from the square lattices, random-walk rejection sampling performs very poorly because the acceptance probability decreases to 0 exponentially; CFTP performs much better than random-walk rejection sampling, but worse than Bernoulli rejection sampling.

Now we consider sampling forests from complete graphs where each edge has weight 1. Random-walk rejection sampling uses polynomial running time; for example, for a graph with 35 nodes, it only needs 36 seconds to obtain 1000 forests. Bernoulli rejection sampling continues indefinitely and fails to obtain an output in realistic time, and CFTP takes a long time to couple, because the upper chain, started from the whole graph, and the lower chain, started from the empty graph, are far from each other.

Bernoulli rejection sampling is the best method for sampling from a lattice. But if the lattice has loops made of heavily weighted edges, Bernoulli rejection sampling will be very inefficient. For example, for the graph given in Figure 3, Bernoulli rejection sampling takes 30 minutes and random-walk rejection sampling takes about 90 minutes, while CFTP only needs 6 minutes. In this case CFTP is the best.

We conclude that perfect sampling for random forests is more difficult than perfect sampling for random trees. Among the methods introduced in this paper, no method is uniformly the best. For complete graphs where each edge has weight 1, Algorithm 8 is very efficient and we recommend this algorithm for almost uniformly weighted graphs. The algorithm also works well for some heterogeneously weighted graphs, but the acceptance probability depends on the underlying graphical structure.

CFTP algorithms are very inefficient for uniformly weighted complete graphs, but for square lattices with loops made of heavily weighted edges, we recommend using the CFTP with

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

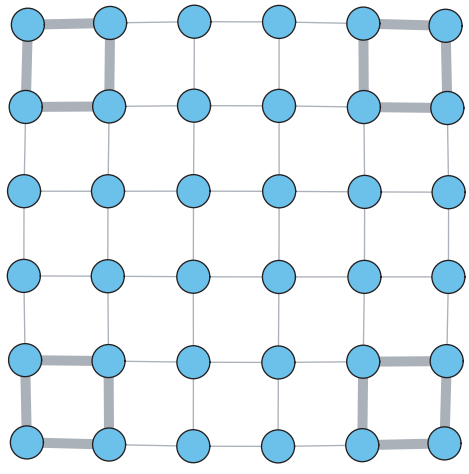

FIGURE 3: Lattice with unequal weights. The thin lines have weight 1 and the thick lines have weight 20.

bounding chains algorithm (Algorithm 6). For sparse graphs without heavily weighted loops, we recommend using Bernoulli rejection sampling (Algorithm 7).

6. Discussion

It is a very challenging problem to simulate forests from a graph. This paper provides practical solutions. When sampling from square lattices, we provide the complexity of CFTP with a lower chain (Algorithm 2). But, for the same problem, the complexity of CFTP with bounding chains (Algorithm 6) is still unknown. This is left as future research work.

The proposed methods can be applied to Gaussian graphical models where we are interested in the selection of a conditional correlation structure for multivariate random variables. The correlation structure is usually represented as a graph where the vertices denote the random variables and the edges denote the conditional correlation of two variables given all other variables. We may assume that the unknown graph structure is a forest. With some standard Bayesian approaches [6], [7], [10], the posterior distribution of the forests has the form of (1). We can use the presented perfect sampling methods introduced in Sections 3 and 4 to sample realisations from the posterior and then make inference on the characteristics of the forest posterior distribution. More application results can be found in [5].

Appendix A. Proof of Proposition 2

Algorithm 2 updates the lower chain $\mathcal{L}_t$ through three cases:

1. the randomly selected edge, $\{i, k\} \in \mathcal{L}_t$,
2. $\{i, k\} \notin \mathcal{L}_t$ and $i \leftrightarrow k | \mathcal{L}_t$,
3. $i \nleftrightarrow k | \mathcal{L}_t$.

We now prove that, under each case, Proposition 2 is true, i.e. if $\mathcal{L}_t \subset \cap\mathcal{E}(F_t)$ then $\mathcal{L}_{t+1} \subset \cap\mathcal{E}(F_{t+1})$.

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

Case 1. If $\{i,k\} \in \mathcal{L}_t$ then $\{i,k\} \in \mathcal{E}(F_t)$ for any $F_t \in \mathcal{F}_t$. According to Algorithms 1 and 2, if $U \le W_{i,k}/(1+W_{i,k})$ then $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t),\ F_t \in \mathcal{F}_t$, and $\mathcal{L}_{t+1} = \mathcal{L}_t$. So we have

$$\mathcal{L}_{t+1} = \mathcal{L}_t \subset \cap\mathcal{E}(F_t) = \cap\mathcal{E}(F_{t+1}).$$

On the other hand, we also know that if $U > W_{i,k}/(1+W_{i,k})$ then $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \setminus \{i,k\}$ and $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i,k\}$. So

$$\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i,k\} \ \subset [\cap\mathcal{E}(F_t)] \setminus \{i,k\} = \cap[\mathcal{E}(F_t) \setminus \{i,k\}] = \cap\mathcal{E}(F_{t+1}).$$

Case 2. $\{i,k\} \notin \mathcal{L}_t$ and $i \to j \to \cdots \to k \in \mathcal{E}(F_t),\ j \ne k, F_t \in \mathcal{F}_t$. According to Algorithm 1, if $U \le W_{i,k}/(W_{i,j}+W_{i,k})$ then $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \setminus \{i,j\} \cup \{i,k\}$; otherwise $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t)$. Furthermore, according to lines 05 to 08 of Algorithm 2, if $U \le W_{i,k}/(W_{i,j}+W_{i,k})$ then $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i,j\} \cup \{i,k\}$; otherwise $\mathcal{L}_{t+1} = \mathcal{L}_t$. Obviously, if $U \le W_{i,k}/(W_{i,j}+W_{i,k})$ and $\mathcal{L}_t \subset \cap\mathcal{E}(F_t)$ then $\mathcal{L}_{t+1} \subset \cap\mathcal{E}(F_{t+1})$.

Case 3. If $i \nleftrightarrow k | \mathcal{L}_t$ then vertices i and k are in different subtrees of $\mathcal{L}_t$. So there may exist different paths from i to k in different forest Markov chains, say $i \to j_{F_t} \to \cdots \to k \in \mathcal{E}(F_t),\ F_t \in \mathcal{F}_t$. Note that if i and k are not connected in F_t or they are neighbours, we define $j_{F_t} = 0$.

Remember the definition of A in Algorithm 2. If $U \le \min_{j \in A \cup \{0\}} W_{i,k}/(W_{i,j}+W_{i,k})$ then $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \setminus \{i, j_{F_t}\} \cup \{i,k\}$, according to the updating rules of the forest Markov chain. Line 11 of Algorithm 2 also tells us that $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i,C\} \cup \{i,k\} = \mathcal{L}_t \setminus \{i,A\} \cup \{i,k\}$. So

$$\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i,A\} \cup \{i,k\} \subset \cap(\mathcal{E}(F_t) \setminus \{i,j_{F_t}\} \cup \{i,k\}) = \cap\mathcal{E}(F_{t+1}).$$

If $U > \min_{j \in A \cup \{0\}} W_{i,k}/(W_{i,j}+W_{i,k})$, we have to introduce extra notation to simplify the proof. For simplicity, we use F_t^* to denote F_t if $j_{F_t} = 0$. When $j_{F_t} \ne 0$, if $U > W_{i,k}/(W_{i,k}+W_{i,j_{F_t}})$, we use $\tilde{F}_t$ to denote F_t; otherwise we use $\bar{F}_t$. With this notation, we divide all the forests F_t into three groups: F_t^*, $\tilde{F}_t$, and $\bar{F}_t$. According to $D = \{j | j \in C,\ j \ne k,\ U \le W_{i,k}/(W_{i,k}+W_{i,j})\}$ in line 13 of Algorithm 2, we have $\bigcup_{\bar{F}_t}\{i, j_{\bar{F}_t}\} \cup \{i,k\} \supset \{i,D\}$. Therefore,

$$\begin{aligned}
\mathcal{L}_{t+1} &= \mathcal{L}_t \setminus \{i,D\} \\
&= \mathcal{L}_t \setminus \left[\bigcup_{\bar{F}_t}\{i, j_{\bar{F}_t}\} \cup \{i,k\}\right] \\
&\subset \left[\bigcap_{\tilde{F}_t}\mathcal{E}(\tilde{F}_t)\right] \cap \left[\bigcap_{F_t^*}\mathcal{E}(F_t^*)\right] \cap \left[\bigcap_{\bar{F}_t}\mathcal{E}(\bar{F}_t)\right] \setminus \left[\bigcup_{\bar{F}_t}\{i, j_{\bar{F}_t}\} \cup \{i,k\}\right] \\
&\subset \left[\bigcap_{\tilde{F}_t}\mathcal{E}(\tilde{F}_t)\right] \cap \left[\bigcap_{F_t^*}\mathcal{E}(F^*{}_t) \setminus \{i,k\}\right] \cap \left[\bigcap_{\bar{F}_t}\mathcal{E}(\bar{F}_t) \setminus \bigcup_{\bar{F}_t}\{i, j_{\bar{F}_t}\}\right] \\
&= \left[\bigcap_{\tilde{F}_t}\mathcal{E}(\tilde{F}_t)\right] \cap \left[\bigcap_{F_t^*}\mathcal{E}(F^*{}_t) \setminus \{i,k\}\right] \cap \left[\bigcap_{\bar{F}_t}(\mathcal{E}(\bar{F}_t) \setminus \{i, j_{\bar{F}_t}\})\right] \\
&\subset \left[\bigcap_{\tilde{F}_t}\mathcal{E}(\tilde{F}_{t+1})\right] \cap \left[\bigcap_{F_t^*}\mathcal{E}(F_{t+1}^*)\right] \cap \left[\bigcap_{\bar{F}_t}\mathcal{E}(\bar{F}_{t+1})\right] \qquad (3) \\
&= \cap\mathcal{E}(F_{t+1}).
\end{aligned}$$

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

Note that we should prove that the relation '$\subset$' in (3) is correct. According to Algorithm 1 (a) if $j_{F_t} \neq 0$ then $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \setminus \{i, j_{F_t}\} \cup \{i, k\}$ given $U \leq W_{i,k}/(W_{i,k} + W_{i,j_{F_t}})$, which implies that $\mathcal{E}(\bar{F}_t) \setminus \{i, j_{\bar{F}_t}\} \subset \mathcal{E}(\bar{F}_{t+1})$, (b) if $j_{F_t} \neq 0$ then $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t)$ given $U > W_{i,k}/(W_{i,k} + W_{i,j_{F_t}})$, which implies that $\mathcal{E}(\tilde{F}_{t+1}) = \mathcal{E}(\tilde{F}_t)$. On the other hand, if $j_{F_t} = 0$ then $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \cup \{i, k\}$ given $U \leq W_{i,k}/(W_{i,k} + 1)$ and $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \setminus \{i, k\}$ given $U > W_{i,k}/(W_{i,k} + 1)$. So (c) if $j_{F_t} = 0$ then $\mathcal{E}(F_t) \setminus \{i, k\} \subset \mathcal{E}(F_{t+1})$.

Therefore, the relation '$\subset$' in (3) is correct.

Appendix B. Proof of Proposition 4

Algorithm 4 updates the upper chain and lower chain through three cases:

1. the randomly selected edge, $\{i, k\} \in \mathcal{L}_t$,
2. $\{i, k\} \notin \mathcal{L}_t$ and $i \leftrightarrow k|\mathcal{L}_t$,
3. $i \nleftrightarrow k|\mathcal{L}_t$.

Under cases 1 and 2, the proposition follows from the proof of Proposition 2. In case 3 Algorithm 4 calls Algorithm 5. So we only need to prove that Algorithm 5 guarantees the proposition under $i \nleftrightarrow k|\mathcal{L}_t$.

Algorithm 5 updates the upper and lower chains through two cases: (a) $B = \phi$ or $B = \{k\}$, (b) $B \neq \phi$ and $B \neq \{k\}$.

If $B = \phi$ or $B = \{k\}$ then i and k are either neighbours or not connected in F_t, $F_t \in \mathcal{F}_t$. So given $U \leq W_{ik}/(W_{ik} + 1)$ or not, the upper chain and the lower chain should be updated in the same way. In this case the proposition follows from the proof of Proposition 2.

If $B \neq \phi$ and $B \neq \{k\}$ then i and k may be connected through different paths. If $U \leq \min_{j \in A}\{W_{i,k}/(W_{i,j} + W_{i,k})\}$ then we can prove that $\mathcal{L}_{t+1} \subset \cap\mathcal{E}(F_{t+1})$, as in the proof of Proposition 2. Remember the definition of the vertex set E in Algorithm 5, which is the connectivity set of k in $\mathcal{L}_t$. Therefore, E will also be part of the connectivity set of k in $\mathcal{L}_{t+1}$. In $\mathcal{L}_{t+1}$, i and k are neighbours. Obviously, in $\mathcal{L}_{t+1}$ there is a path between i and any vertex in E via k. So, if $j \in E$ then $\{i, j\} \notin \mathcal{E}(F_{t+1})$, and, furthermore, $\{i, E\} \notin \cup\mathcal{E}(F_{t+1})$. In addition, given $U \leq \min_{j \in A}\{W_{i,k}/(W_{i,j} + W_{i,k})\}$, we have $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \setminus \{i, j_{F_t}\} \cup \{i, k\} \subset \mathcal{E}(F_t) \cup \{i, k\}$. Therefore,

$$\begin{aligned}
\mathcal{U}_{t+1} &= [\mathcal{U}_t \setminus \{i, E\}] \cup \{i, k\} \\
&\supset [\cup\mathcal{E}(F_t) \cup \{i, k\}] \setminus \{i, E\} \\
&\supset \cup\mathcal{E}(F_{t+1}) \setminus \{i, E\} \\
&= \cup\mathcal{E}(F_{t+1}).
\end{aligned}$$

When $U \geq \max_{j \in A}\{W_{i,k}/(W_{i,j} + W_{i,k})\}$, $\{i, k\}$ will be deleted from all F_t. We have $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \setminus \{i, k\}$. So

$$\begin{aligned}
\mathcal{U}_{t+1} &= \mathcal{U}_t \setminus \{i, k\} \supset \cup\mathcal{E}(F_t) \setminus \{i, k\} \\
&= \cup\mathcal{E}(F_{t+1}) \\
&\supset \cap\mathcal{E}(F_{t+1}) \\
&= \cap[\mathcal{E}(F_t) \setminus \{i, k\}] \\
&\supset \mathcal{L}_t \setminus \{i, k\} \\
&= \mathcal{L}_{t+1}.
\end{aligned}$$

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

If $\min_{j\in A}\{W_{i,k}/(W_{i,j}+W_{i,k})\} \le U \le \max_{j\in A}\{W_{i,k}/(W_{i,j}+W_{i,k})\}$ then, for some $F_t \in \mathcal{F}_t$, $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t)$, for some F_t, $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \setminus \{i,k\}$, and for the other F_t, $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \setminus \{i, j_{F_t}\} \cup \{i,k\}$. Obviously, we have

$$\mathcal{U}_{t+1} = \mathcal{U}_t \cup \{i,k\} \supset \cup\mathcal{E}(F_t) \cup \{i,k\} \supset \cup\mathcal{E}(F_{t+1}).$$

We can prove that $\cap\mathcal{E}(F_{t+1}) \supset \mathcal{L}_{t+1}$, by arguments similar to the proof of Proposition 2. Therefore, the proposition is proved.

Appendix C. Proof of Proposition 7

Before proving this proposition, we introduce some necessary definitions and notation. Let $G_n = (V, \mathcal{E})$ be the $n \times n$ square lattice, let S_t be the largest subtree in $\mathcal{L}_t$, and let $|S_t|$ be the number of edges of S_t. Let $G_n \setminus S_t$ be the graph induced by $V \setminus V_{S_t}$, the vertices in G_n not in S_t. Let $\xi_t = n^2 - 1 - |S_t|$. Note that $\xi_t = 0$ implies that $\xi_{t+1} = 0$. Assume that $1 > \alpha \ge \frac{1}{2}$. Let $\sigma_t = \{\mathcal{L}_t : \alpha(n^2+1) < |S_t| < n^2 - 1\}$. We define the edge sets

$$\begin{aligned}
A_1 &= \{(i,j) \mid i, j \in G_n \setminus S_t\},\\
A_2 &= \{(i,j) \mid i, j \in S_t\} \cap \{(i,j) \mid (i,j) \notin S_t\},\\
A_3 &= \{(i,j) \mid (i,j) \in S_t\},\\
A_4 &= \{(i,j) \mid i \in G_n \setminus S_t\} \cap \{(i,j) \mid j \in S_t\},\\
A_5 &= \{(i,j) \mid i \in S_t\} \cap \{(i,j) \mid j \in G_n \setminus S_t\}.
\end{aligned}$$

As when running the lower chain in Algorithm 9, we randomly choose a directed edge $e = (i,j)$ in each step and then update $\mathcal{L}_t$. Obviously, each set A_i, $i \in \{1,\dots,5\}$, consists of all the possible choices of $e = (i,j)$. For simplicity, we use A_i to denote the event $e \in A_i$. Let $\mathrm{P}(A_i \mid \sigma_t) = \alpha_i$, $i = 1,\dots,5$. Note that α_i depends on σ_t and $\sum_{i=1}^5 \alpha_i = 1$.

Examples of the five kinds of events $A_1,\dots,A_5$ are provided in Figure 4. Figure 4(a) is an example of event A_1, where vertices i and j do not belong to the largest subtree S_t. Figure 4(b) is an example of event A_2, where both i and j belong to S_t, but the edge $\{i,j\}$ does not belong to S_t. In Figure 4(c), $\{i,j\}$ belongs to S_t, therefore it is event A_3. Figure 4(d) is event A_4, where j belongs to S_t but i does not. Figure 4(e) is event A_5.

The following lemma is also necessary for proving Proposition 7.

Lemma 4. *With the previous definitions and notation,* $\alpha_4 = \alpha_5$ *and* $\alpha_3 \ge \frac{1}{3}(\alpha_4 + \alpha_5)$.

Proof. Let $|V_{S_t}|$ be the number of vertices in S_t. Therefore, given a large value of n and $|V_{S_t}| > \alpha n^2$, S_t has at least three vertices which are not leaves (vertices with degree 1), since a lattice has maximum vertex degree 4. Then we can find three vertices which have at most two neighbours not in S_t, since it has at most four neighbours, of which at least two are in S_t. Similarly, each of the other vertices has at most three neighbours not in S_t.

We have $\alpha_4 = \alpha_5$ and $\alpha_4 + \alpha_5$ is the probability that (i,j) has one end in S_t and the other end not in S_t. Then, given σ_t and a large value of n, we have $\alpha_4 + \alpha_5 \le 3(|V_{S_t}| - 3)/|\mathcal{E}| + (2)(3)/|\mathcal{E}| = 3(|V_{S_t}| - 1)/|\mathcal{E}|$, where $|\mathcal{E}|$ is the number of edges in the square lattice.

We also have $\alpha_3 = |S_t|/|\mathcal{E}| = (|V_{S_t}| - 1)/|\mathcal{E}|$, so that $\alpha_3 \ge \frac{1}{3}(\alpha_4 + \alpha_5)$.

Proof of Proposition 7. Assume that we run $\mathcal{L}_t$ from time 0 to ∞. The coalescence time is $T = \min\{t;\ |S_t| = n^2 - 1\} = \min\{t;\ \xi_t = 0\}$. Note that $\xi_0 = n^2 - 1$. For some $\delta > 1$, we

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

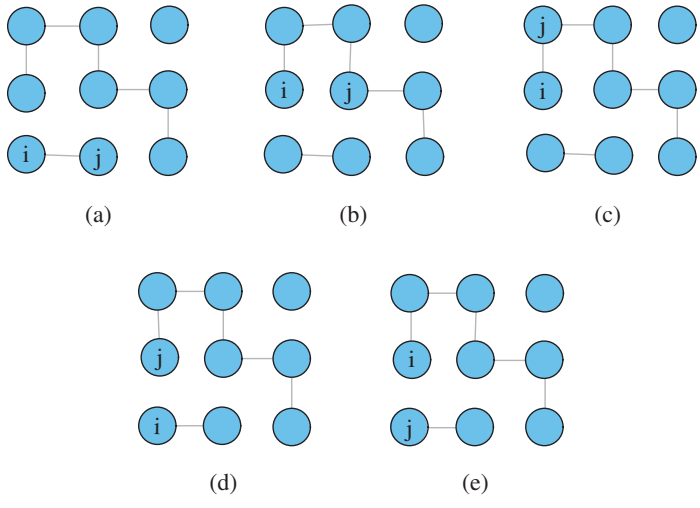

FIGURE 4: The events $\{e \in A_i\}$, $i = 1, \ldots, 5$. (a) A_1, (b) A_2, (c) A_3, (d) A_4, and (e) A_5.

immediately attain the following inequality:

$$\begin{aligned} \mathrm{P}(T \le t \mid \xi_0) &= \mathrm{P}(\xi_t = 0 \mid \xi_0) \\ &= \mathrm{P}(\delta^{-\xi_t} = 1 \mid \xi_0) \\ &\le \mathrm{E}(\delta^{-\xi_t} \mid \xi_0). \end{aligned}$$

Given $A_1 = \{(i, j) \mid i, j \in G_n \setminus S_t\}$, we have $\mathrm{P}(|S_{t+1}| = |S_t| \mid A_1, \sigma_t) = 1$, since, according to Algorithm 9, given $e \in A_1$, the choice of e has no effect on $|S_{t+1}|$. So $\mathrm{E}_{A_1}(\delta^{|S_{t+1}|-|S_t|} \mid \sigma_t) = \alpha_1$.

Similarly, given $A_2 = \{(i, j) \mid i, j \in S_t\} \cap \{(i, j) \mid (i, j) \notin S_t\}$, we have $\mathrm{P}(|S_{t+1}| = |S_t| \mid A_2, \sigma_t) = 1$. So $\mathrm{E}_{A_2}(\delta^{|S_{t+1}|-|S_t|} \mid \sigma_t) = \alpha_2$.

Given $A_3 = \{(i, j) \mid (i, j) \in S_t\}$, then

$$\mathrm{P}(|S_{t+1}| = |S_t| \mid A_3, \sigma_t) = \mathrm{P}(|S_{t+1}| - |S_t| \le -1 \mid A_3, \sigma_t) = \tfrac{1}{2}.$$

This is because, when $(i, j) \in S_t$, the edge (i, j) has probability $\frac{1}{2}$ of being removed from S_t and probability $\frac{1}{2}$ of not being removed. If (i, j) is removed then the number of edges of the largest subtree in $\mathcal{L}_t$ will decrease by at least 1. Thus, $\mathrm{E}_{A_3}(\delta^{|S_{t+1}|-|S_t|} \mid \sigma_t) \le \alpha_3(\frac{1}{2} + \frac{1}{2}\delta^{-1})$.

Recall that $A_4 = \{(i, j) \mid i \in G_n \setminus S_t\} \cap \{(i, j) \mid j \in S_t\}$. In this case, edge (i, j) has probability $\frac{1}{2}$ of being added to S_t and probability $\frac{1}{2}$ of not being added. Then

$$\mathrm{P}(|S_{t+1}| = |S_t| \mid A_4, \sigma_t) = \mathrm{P}(|S_{t+1}| = |S_t| + 1 \mid A_4, \sigma_t) = \tfrac{1}{2}.$$

So $\mathrm{E}_{A_4}(\delta^{|S_{t+1}|-|S_t|} \mid \sigma_t) = \alpha_4(\frac{1}{2} + \frac{1}{2}\delta)$.

Remember that $A_5 = \{(i, j) \mid i \in S_t\} \cap \{(i, j) | j \in G_n \setminus S_t\}$. Similar to the case of A_3, we have $\mathrm{P}(|S_{t+1}| = |S_t| \mid A_5, \sigma_t) = \frac{1}{2}$ and $\mathrm{P}(|S_{t+1}| - |S_t| \le -1 \mid A_5, \sigma_t) = \frac{1}{2}$. Therefore, $\mathrm{E}_{A_5}(\delta^{|S_{t+1}|-|S_t|} \mid \sigma_t) \le \alpha_5(\frac{1}{2} + \frac{1}{2}\delta^{-1})$.

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

With all the previous illustration and Lemma 4, given $\delta \le \frac{5}{3}$, we have

$$\begin{aligned}
&\mathrm{E}(\delta^{|S_{t+1}|-|S_t|} \mid \alpha(n^2+1) < |S_t| < n^2-1) \\
&\quad = \sum_{i=1}^{5} \mathrm{E}_{A_i}(\delta^{|S_{t+1}|-|S_t|} \mid \sigma_t) \\
&\quad \le \alpha_1 + \alpha_2 + \alpha_3\left(\tfrac{1}{2} + \tfrac{1}{2}\delta^{-1}\right) + \alpha_4\left(\tfrac{1}{2} + \tfrac{1}{2}\delta\right) + \alpha_5\left(\tfrac{1}{2} + \tfrac{1}{2}\delta^{-1}\right) \\
&\quad = 1 + \tfrac{1}{2}((\alpha_3 + \alpha_5)(\delta^{-1} - 1) + \alpha_4(\delta - 1)) \\
&\quad \le 1 + \tfrac{1}{2}\left(\left(\tfrac{5}{3}\alpha_5\right)(\delta^{-1} - 1) + \alpha_5(\delta - 1)\right) \\
&\quad \le 1. \qquad (4)
\end{aligned}$$

From (4) we have

$$\begin{aligned}
&\mathrm{E}(\delta^{-\xi_{t+1}} \mid \xi_t) \\
&\quad \le \mathrm{E}(\delta^{-\xi_{t+1}} \mid 0 < \xi_t \le (1-\alpha)(n^2-1) - 2\alpha) \\
&\qquad + \mathrm{E}(\delta^{-\xi_{t+1}} \mid (1-\alpha)(n^2-1) - 2\alpha < \xi_t < n^2 - 1) \\
&\quad \le \delta^{-\xi_t}\,\mathrm{E}(\delta^{-(\xi_{t+1}-\xi_t)} \mid 0 < \xi_t \le (1-\alpha)(n^2-1) - 2\alpha) + \delta^{-[(1-\alpha)(n^2-1)-2\alpha]} \\
&\quad = \delta^{-\xi_t}\,\mathrm{E}(\delta^{|S_{t+1}|-|S_t|} \mid \alpha(n^2+1) < |F_t| < (n^2-1)) + \delta^{-[(1-\alpha)(n-1)-2\alpha]} \\
&\quad \le \delta^{-\xi_t} + \delta^{-[(1-\alpha)(n-1)-2\alpha]}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathrm{P}(T \le t \mid \xi_0) &\le \mathrm{E}(\delta^{-\xi_t} \mid \xi_0) \\
&= \mathrm{E}(\mathrm{E}(\delta^{-\xi_t} \mid \xi_{t-1}) \mid \xi_0) \\
&\le \mathrm{E}(\delta^{-\xi_{t-1}} \mid \xi_0) + \delta^{-[(1-\alpha)(n^2-1)-2\alpha]} \\
&\le \delta^{-\xi_0} + t\delta^{-[(1-\alpha)(n^2-1)-2\alpha]}.
\end{aligned}$$

Therefore, given $t_1 = \lfloor (1 - \delta^{-\xi_0})\delta^{[(1-\alpha)(n^2-1)-2\alpha]} \rfloor$,

$$\begin{aligned}
\mathrm{E}(T \mid \xi_0) &= \sum_{t \ge 0} \mathrm{P}(T \ge t \mid \xi_0) \\
&\ge \sum_{t \ge 0} \max\{0, 1 - \delta^{-\xi_0} - t\delta^{-[(1-\alpha)(n^2-1)-2\alpha]}\} \\
&= \sum_{0 \le t \le t_1} (1 - \delta^{-\xi_0} - t\delta^{-[(1-\alpha)(n^2-1)-2\alpha]}) \\
&= O(t_1).
\end{aligned}$$

This means that we need exponential coupling time, since t_1 increases exponentially with n^2. The largest value of δ available is $\frac{5}{3}$. Given $\delta = \frac{5}{3}$ and $\alpha = \frac{1}{2}$, we have $t_1 = O((\frac{5}{3})^{(n^2-3)/2})$. This means that the expected running time is at least $O((\frac{5}{3})^{(n^2-3)/2})$.

Appendix D. Matrix tree theorem and the proof of Lemma 3

Lemma 5. (Matrix tree theorem.) *The normalising constant $\mathcal{W}_{\mathcal{T}}$ is equal to any cofactor of the weighted degree matrix of G minus the weighted adjacency matrix of G.*

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

The weighted degree matrix of G is a diagonal matrix with the ith diagonal entry equal to $\sum_{j\neq i} W_{ij}$, where the summation is for all neighbours of i. The weighted adjacency matrix of G is $\boldsymbol{A} = (a_{ij})$, where $a_{ii} = 0$ and $a_{ij} = W_{ij}$, $i \neq j$, for all i, j. See, for example, [11] for a proof of Lemma 5.

Proof of Lemma 3. Assume that G_n is an $n \times n$ uniformly weighted square lattice and that $\tilde{G}_n$ is the graph formed by adding an extra vertex 0 and extra edges $\{i, 0\}$ into G_n.

According to the matrix tree theorem, we have

$$\mathcal{W}_{\mathcal{T}(\tilde{G}_n)} = \det \begin{pmatrix} \boldsymbol{A}-\boldsymbol{I} & -\boldsymbol{I} & 0 & \cdots & 0 & 0 \\ -\boldsymbol{I} & \boldsymbol{A} & -\boldsymbol{I} & \cdots & 0 & 0 \\ 0 & -\boldsymbol{I} & \boldsymbol{A} & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & \cdots & \boldsymbol{A} & -\boldsymbol{I} \\ 0 & 0 & 0 & \cdots & -\boldsymbol{I} & \boldsymbol{A}-\boldsymbol{I} \end{pmatrix}, \tag{5}$$

which is a determinant of an $n^2 \times n^2$ partitioned tridiagonal matrix. In (5), $\boldsymbol{I}$ is the $n \times n$ identity matrix and $\boldsymbol{A}$ is an $n \times n$ tridiagonal matrix given by

$$\boldsymbol{A} = \begin{pmatrix} 4 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 5 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 5 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 5 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 4 \end{pmatrix}.$$

Let J_n be the determinant of an $n \times n$ tridiagonal matrix with 4.5 in the diagonal entries and -1 in the subdiagonal and superdiagonal entries. We have the recursive formula $J_n = 4.5J_{n-1} - J_{n-2}$. It is easy to have $J_n = \tilde{\alpha}\tilde{a}_1^n + \tilde{\beta}\tilde{a}_2^n$, where

$$\tilde{\alpha} = \frac{36.5 + 9\sqrt{16.25}}{32.5 + 9\sqrt{16.25}} > 0 \quad \text{and} \quad \tilde{\beta} = \frac{36.5 - 9\sqrt{16.25}}{32.5 - 9\sqrt{16.25}} < 0.$$

Therefore,

$$\left| \boldsymbol{A} - \frac{\boldsymbol{I}}{2} \right| \geq 3.5^2 J_{n-2} - 7J_{n-3} + J_{n-4} \geq 10.5J_{n-2} \geq \tilde{a}_1^{n-1}, \tag{6}$$

since $J_{n-1}/J_n \leq 2/(4.5 + \sqrt{16.25}) \leq 0.25$ and $J_n \geq \tilde{a}_1^n$.

From

$$\det \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} = \det(\boldsymbol{A}) \det(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}),$$

we have $\mathcal{W}_{\mathcal{T}(\tilde{G}_n)} = \prod_{i=1}^n \det(\tilde{\boldsymbol{A}}_i)$, where $\tilde{\boldsymbol{A}}_i = \boldsymbol{A} - \tilde{\boldsymbol{A}}_{i-1}^{-1}$, $i = 1, \ldots, n-1$, $\tilde{\boldsymbol{A}}_0 = \boldsymbol{I}$, and $\tilde{\boldsymbol{A}}_n = \boldsymbol{A} - \boldsymbol{I} - \tilde{\boldsymbol{A}}_{n-1}^{-1}$.

Denote the eigenvalues of $\boldsymbol{A}$ by λ_l for $l = 1, \ldots, n$. Obviously, for all l, $\lambda_l > 3$. Thus, we have $\det(\tilde{\boldsymbol{A}}_2) = \det(\boldsymbol{A} - (\boldsymbol{A} - \boldsymbol{I})^{-1}) = \prod_{l=1}^n (\lambda_l - (\lambda_l - 1)^{-1})$, which is larger than $\prod_{l=1}^n (\lambda_l - \frac{1}{2}) = \det(\boldsymbol{A} - \boldsymbol{I}/2)$. Similarly, for all $i = 1, \ldots, n-1$, $\det(\tilde{\boldsymbol{A}}_i) > \det(\boldsymbol{A} - \boldsymbol{I}/2)$.

https://doi.org/10.1239/aap/1222868191 Published online by Cambridge University Press

With the results $\det(\tilde{\boldsymbol{A}}_1) > 1$, $\det(\tilde{\boldsymbol{A}}_n) > 1$, and (6), we have

$$\begin{aligned}\mathcal{W}_{\mathcal{T}(\tilde{G}_n)} &= \prod_{i=1}^{n} \det(\tilde{\boldsymbol{A}}_i) \\ &> \det\left(\boldsymbol{A} - \frac{\boldsymbol{I}}{2}\right)^{n-2} \\ &> \left(\frac{4.5 + \sqrt{16.25}}{2}\right)^{(n-1)(n-2)} \\ &> 4.2655^{(n-1)(n-2)}.\end{aligned}$$

References

[1] Aldous, D. J. (1990). The random walk construction of uniform spanning trees and uniform labelled trees. *SIAM J. Discrete Math.* **3,** 450–465.

[2] Broder, A. (1989). Generating random spanning trees. *In Proc. 30th IEEE Symp. Foundations of Computer Science*, IEEE, New York, pp. 442–447.

[3] Calkin, N., Merino, C., Noble, S. and Noy, M. (2003). Improved bounds for the number of forests and acyclic orientations in the square lattice. *Electron. J. Combinatorics* **10,** R4.

[4] Colbourn, C. J., Day, R. R. J. and Nel, L. D. (1989). Unranking and ranking spanning trees of a graph. *J. Algorithms* **10,** 271–286.

[5] Dai, H. (2007). Perfect simulation methods for Bayesian applications. Doctoral Thesis, University of Oxford.

[6] Dawid, A. P. and Lauritzen, S. L. (1993). Hyper Markov laws in the statistical analysis of decomposable graphical models. *Ann. Statist.* **21,** 1272–1317.

[7] Giudici, P. (1996). Learning in graphical Gaussian models. In *Bayesian Statistics 5*, Oxford University Press, pp. 621–628.

[8] Guenoche, A. (1983). Random spanning tree. *J. Algorithms* **4,** 214–220.

[9] Huber, M. (2004). Perfect sampling using bounding chains. *Ann. Appl. Prob.* **14,** 734–753.

[10] Jones, B. *et al.* (2005). Experiments in stochastic computation for high-dimensional graphical models. *Statist. Sci.* **4,** 388–400.

[11] Jones, B. D., Pittel, B. G. and Verducci, J. S. (1999). Tree and forest weights and their application to nonuniform random graphs. *Ann. Appl. Prob.* **9,** 197–215.

[12] Møller, J. (1999). Perfect simulation of conditionally specified models. *J. R. Statist. Soc. B* **61,** 251–264.

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

[14] Propp, J. G. and Wilson, D. B. (1998). How to get an exact sample from a generic Markov chain and sample a random spanning tree from a directed graph, both within the cover time. *J. Algorithms* **27,** 170–217.

[15] Takács, L. (1990). On the number of distinct forests. *SIAM J. Discrete Math.* **3,** 574–581.