

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 graph, forest and tree.

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

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

The tree space is defined similarly, with tree probability distribution

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

* Postal address: Department of Mathematics and Statistics, Fylde College, Lancaster University, Lancaster, LA1 4YF, UK

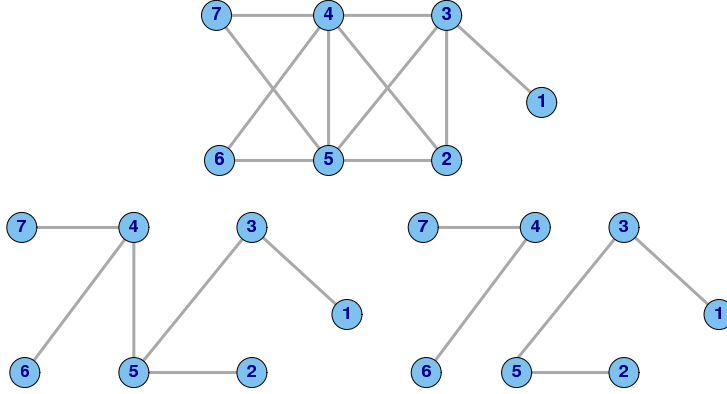


FIGURE 1: Top: a graph with 7 vertices; bottom left: a tree on the graph; bottom right: a forest on the graph.

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 conditional correlation of two variables given all other variables, we are interested in the edge inclusion probability, $P(e \in F)$ or the expected number of connected components in a forest. These quantities are non-trivial 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 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 by [1], [2], [5],

[10] and [17]. However, sampling forests from distribution (1) has had 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 7 spanning forests. If each edge has weight $W_e = 1$, then each forest has probability $P(F) = 1/7$. But if we rescale the weights by a constant factor $\alpha \neq 1$, that is $W_e = \alpha$, then $P(F) = \alpha^2/(1 + 3\alpha + 3\alpha^2)$, if F has two edges, and $P(F) = \alpha/(1 + 3\alpha + 3\alpha^2)$, 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* (CFTP) and rejection sampling respectively. The paper is organised as follows. Section 2 introduces notations and basic definitions. Then in Section 3 we develop CFTP 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.

2. Preliminaries

An undirected *graph* G is a pair (V, \mathcal{E}) consisting of a set of vertices $V = \{1, \dots, p\}$ 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* $\text{bd}(A)$ 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 the number of vertices in A .

A *simple path* between two vertices i and j is a sequence $i = \alpha_1, \dots, \alpha_n = j$ of distinct vertices such that $\{\alpha_{k-1}, \alpha_k\} \in \mathcal{E}$ for all $k = 2, \dots, n$. A closed path $\{\alpha_1, \alpha_2, \dots, \alpha_n, \alpha_1\}$ 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}, \dots, G_{A_k} if each subgraph G_{A_i} is connected, A_1, \dots, A_k is a partition of V and $\text{bd}(A_i) = \emptyset : i = 1, \dots, k$.

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 [16] which showed how to provide perfect samples from the limiting distribution of a Markov chain. The idea is to run all possible Markov chain 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 $\mathcal{X} = \{1, \dots, n\}$, where the probability of going from i to j is p_{ij} and the stationary distribution is π . Suppose we design an update function $\phi(\cdot, U)$, which satisfies $P[\phi(i, U) = j] = p_{ij}$, 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

$$F_{t_1}^{t_2} = f_{t_2-1} \circ f_{t_2-2} \circ \dots \circ f_{t_1+1} \circ f_{t_1}, \quad (3)$$

for $t_1 < t_2$. The coalescent idea of coupling from the past 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 [16]) *Assume that with probability 1 there exists a time $t = -T$, the backward coupling time, such that chains starting from any state in $\mathcal{X} = \{1, \dots, n\}$, at time $t = -T$, and with the same sequence $\{U_t, t = -T, \dots, -1\}$, 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 $\{U_t, t = -T, \dots, -1\}$ after $-T$, the Markov chain will arrive at X_0^* . Then X_0^* comes exactly from π since it is collected at time 0 and the Markov chain started from $-\infty$.

Therefore a CFTP algorithm is that if (i) $F_{-M}^0(\cdot)$ is a single point then output $F_{-M}^0(\cdot)$, otherwise (ii) let $M = 2M$ and carry out step (i) again. [16] showed that the computational cost of the algorithm can be reduced if there is a partial order for the state space \mathcal{X} , that 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 non-trivial task in many cases. An alternative improvement is CFTP with bounding chains, such as that in [11] and [15]. 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. Markov chain Monte Carlo 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 [12] constructs 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. [17] provides 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), $i \rightarrow j$. 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 $i \rightarrow k$ 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 3.1. (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 $U \leq W_{i,k}/(1 + W_{i,k})$	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 , $i \rightarrow j \neq k \rightarrow \dots \rightarrow k$, in F_t	07
If $U \leq W_{i,k}/(W_{i,j} + W_{i,k})$	08
$F_{t+1} = F_t \cup \{i, k\} \setminus \{i, j\}$	09
Else	10
$F_{t+1} = F_t$	11

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 the new graph is a forest. The edge to be removed is either $\{i, k\}$ or some other edge $\{i, j\}$, that 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 3.1 is ergodic and we have the following result.

Proposition 1. *The stationary distribution, of the Markov chain generated by repeating Algorithm 3.1, is $P(F)$ in (1).*

Proof. Algorithm 3.1 gives us a probability transition matrix $R(F_0, F_1)$. Since

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

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

3.3. Coupling from the past with a lower chain

In this section 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 $\cap \mathcal{E}(F_t) = \cap_{F \in \mathcal{F}_t} \mathcal{E}(F)$ 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 3.2 provides an updating approach for the lower chain \mathcal{L}_t that guarantees this condition is satisfied. Note that, the following notation is used in Algorithm 3.2. We use A and C to denote the set of neighbors 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 $\{i, C\}$ to denote the edge set of i in \mathcal{L}_t . We define $W_{i,0} = 1$.

Algorithm 3.2. (Updating approach of the lower chain.)

```

LowerCFTP(( $i, k$ ),  $U$ ,  $\mathcal{L}_t$ ,  $\mathcal{L}_{t+1}$ )
# Inputs: ( $i, k$ )  $\sim$  Unif $\{\mathcal{E}_d\}$ ,  $U \sim$  Unif $[0, 1]$  and  $\mathcal{L}_t$ ; Output:  $\mathcal{L}_{t+1}$ .
  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  $\mathcal{L}_t$ 
  If  $\{i, k\} \in \mathcal{L}_t$  02
    If  $U \leq W_{i,k}/(1 + W_{i,k})$  then  $\mathcal{L}_{t+1} = \mathcal{L}_t \cup e$  03
    Else then  $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus e$  04
  Else if  $i \leftrightarrow k | \mathcal{L}_t$  05
    Find the path  $i \rightarrow j (\neq k) \rightarrow \dots \rightarrow k \in \mathcal{L}_t$  06
    If  $U \leq W_{i,k}/(W_{i,j} + W_{i,k})$  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  $U \leq \min_{j \in A \cup \{0\}} W_{i,k}/(W_{i,j} + W_{i,k})$  10
       $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i, C\} \cup \{\{i, k\}\}$  11
    Else 12
      Find  $D = \{j | j \in C, j \neq k, U \leq W_{i,k}/(W_{i,k} + W_{i,j})\}$  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 Algorithms 3.2, 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 3.2, lines 02 to 08 update the lower chain according to updating rules of the MCMC algorithm (Algorithm 3.1). 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 $i \leftrightarrow k | \mathcal{L}_t$, lines 10 to 14 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 have Algorithm 3.3 to sample a forest from the target distribution.

Algorithm 3.3. (CFTP with the lower chain.)

FLAG = 0, $t = -M$ and $\mathcal{L}_t = \emptyset$	01
Generate $U_t \sim \text{Unif}[0, 1]$ and $(i, k)_t \sim \text{Unif}\{\mathcal{E}_d\} : t = -\infty, \dots, -1$	02
Repeat until 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 $\mathcal{L}_t = \emptyset$	12

Proposition 3. *Algorithm 3.3 returns a forest with probability distribution (1).*

Proof. These follows since all the Markov chains coalesce.

The running time of this CFTP algorithm is at least the waiting time that the lower chain becomes a tree (see 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 section.

3.4. Coupling from the past with bounding chains

We follow the notations in 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 $\mathcal{U}_t \supset \cup \mathcal{E}(F_t) = \cup_{F \in \mathcal{F}_t} \mathcal{E}(F)$. 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 3.4 and Algorithm 3.5 together provide us a way of doing this.

In Algorithm 3.5, $B = \phi$ or $B = \{k\}$ means that $i \leftrightarrow k | \mathcal{U}_t$ or i and k are connected

Algorithm 3.4. (Updating approach of 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 $U \leq W_{i,k}/(1 + W_{i,k})$	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 $i \rightarrow j (\neq k) \rightarrow \dots \rightarrow k \in \mathcal{L}_t$	06
If $U \leq W_{i,k}/(W_{i,j} + W_{i,k})$	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

only via edge $\{i, k\}$. Vertex set C and D are defined similarly as that in Algorithm 3.2.

The following proposition proves that \mathcal{L}_t and \mathcal{U}_t governed by Algorithm 3.4 and Algorithm 3.5 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 3.4 and Algorithm 3.5 guarantees $\mathcal{U}_{t+1} \supset \cup \mathcal{E}(F_{t+1})$ 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 3.4, lines 01 to 09 update \mathcal{L}_t and \mathcal{U}_t according to the MCMC algorithm (Algorithm 3.1). When $i \leftrightarrow k | \mathcal{L}_t$ (line 10), it will call the subroutine **UPDATE**, given by Algorithm 3.5. In Algorithm 3.5, 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 3.5 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 $i \leftrightarrow k \leftrightarrow l | \mathcal{U}_t$,

Algorithm 3.5. (The subroutine of Algorithm 3.4.)

UPDATE ((i, k), U , $\mathcal{L}_t, \mathcal{U}_t, \mathcal{L}_{t+1}, \mathcal{U}_{t+1}$)	
# Inputs: (i, k) \sim Unif $\{\mathcal{E}_d\}$, $U \sim$ Unif $[0, 1]$, $\mathcal{L}_t, \mathcal{U}_t$; Outputs: $\mathcal{L}_{t+1}, \mathcal{U}_{t+1}$.	
Find the set $B = \{j j \neq i, i \rightarrow j \rightarrow \dots \rightarrow k \in \mathcal{U}_t\}$	01
If $B = \emptyset$ or $B = \{k\}$	02
If $U \leq W_{ik}/(W_{ik} + 1)$	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
$C = \{j j \neq k, j \neq i, \{i, j\} \in \mathcal{L}_t\}$	
$D = \{j j \in C, j \neq k, U \leq W_{ik}/(W_{ik} + W_{ij})\}$	
$A = B \cup \{0\} \setminus \{k\}$	
If $U \leq \min_{j \in A} \{W_{i,k}/(W_{i,j} + W_{i,k})\}$	08
Find $E = \{l l \neq k \text{ and } k \leftrightarrow l \mathcal{L}_t\}$	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 $U \geq \max_{j \in A} \{W_{i,k}/(W_{i,j} + W_{i,k})\}$	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

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 3.6, the CFTP algorithm with two bounding chains, which returns a forest from the target distribution.

Proposition 5. *Algorithm 3.6 returns a forest with the probability distribution given by (1).*

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

Algorithm 3.6 is more efficient than Algorithm 3.3. This is because we have two bounding chains, the upper chain and the lower chain. The upper chain will remove

Algorithm 3.6. (CFTP with bounding chains.)	
FLAG = 0, $t = -M$, $\mathcal{U}_t = \mathcal{E}(G)$ and $\mathcal{L}_t = \emptyset$	01
Generate $U_t \sim \text{Unif}[0, 1]$ and $(i, k)_t \sim \text{Unif}\{\mathcal{E}_d\}$: $t = -\infty, \dots, 0$	02
Repeat until FLAG= 1 and $t = 0$	03
If FLAG= 0	04
LowerUpperCFTP $((i, k)_t, U_t, \mathcal{L}_t, \mathcal{U}_t, \mathcal{L}_{t+1}, \mathcal{U}_{t+1})$	05
Else	06
UpdateForest $((i, k)_t, U_t, \mathcal{L}_t, \mathcal{L}_{t+1})$	07
If FLAG= 0, and $\mathcal{L}_{t+1} = \mathcal{U}_{t+1}$ or \mathcal{L}_{t+1} is a spanning tree	08
FLAG= 1	09
$t = t + 1$	10
If FLAG= 0 and $t = 0$	11
$M = 2M$, $t = -M$, $\mathcal{U}_t = \mathcal{E}(G)$ and $\mathcal{L}_t = \emptyset$	12

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 3.4 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 running time at most $pO(\omega)$, where ω is the complexity of find a path from i to k in \mathcal{U}_t . So B can be found 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 4.1. (Bernoulli rejection sampler.)

# Inputs: $G, W_e, e \in \mathcal{E} = \{e_1, \dots, e_m\}$; output: F .	
$g = \emptyset$	01
Repeat $i = 1, \dots, m$	02
$U \sim \text{Unif}[0, 1]$	03
If $U \leq W_{e_i} / (1 + W_{e_i})$	04
$g = g \cup e_i$	05
If g is not a forest	06
Return to line 02 and $g = \emptyset$	07
$F = g$	08

Algorithm 4.1 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 is rejected. It can easily be proved that Algorithm 4.1 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 $\{0, i\}, i = 1, \dots, p$ 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 in [1, 5, 10, 17]. 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 $\hat{P}(F^*)$ given by*

$$\hat{P}(F^*) \propto \hat{h}(F^*) = c(F^*) \prod_{e \in \mathcal{E}(F^*)} W_e, \quad (4)$$

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 i th component.

Proof. If we delete vertex 0 and its edges in \tilde{T} , sampled from \tilde{G} , then we get a

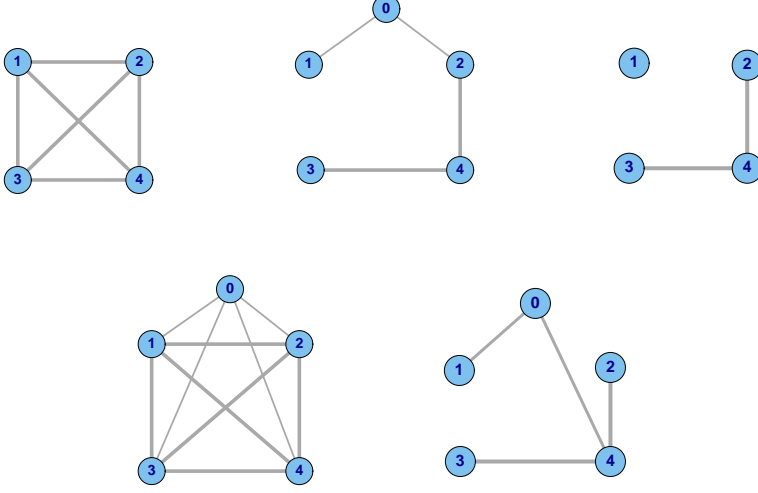


FIGURE 2: Rejection sampler. Top-left: original graph G ; Bottom-left: graph \tilde{G} ; Top-middle and bottom-right: two trees sampled from \tilde{G} ; Top-right: the sampled forest.

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

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

Obviously all \tilde{T} s in \mathcal{A} have the same probability. Then it follows that $\hat{P}(F^*) = c(F^*) \prod_{e \in \mathcal{E}(F^*)} W_e / \left[\sum_{F^* \in \mathcal{F}(G)} c(F^*) \prod_{e \in \mathcal{E}(F^*)} W_e \right]$, where $c(F^*)$ denotes the number of trees that correspond to the same forest F^* . We may also write $\hat{P}(F^*) \propto \hat{h}(F^*) = c(F^*) \prod_{e \in \mathcal{E}(F^*)} W_e$.

Suppose that F^* has k components and the i th component has size s_i . Then the i th 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 $h(F) \leq \hat{h}(F)$, according to $c(F) \geq 1$. We can then use rejection sampling to sample F from $P(F)$, since we can sample from $\hat{h}(F)$, by using Proposition 6. Therefore, summarizing the above, we have the following algorithm which returns a forest with probability distribution given by (1).

Algorithm 4.2. (Rejection sampler based on tree sampling.)

# Inputs: $G, W_e, e \in \mathcal{E}$; output: F .	
Sample a tree T from \tilde{G} and U from $\text{Unif}[0, 1]$	01
Delete vertex 0 and its edges in \tilde{T} and achieve an undirected forest F^*	02
# F^* has the probability $\hat{P}(F^*)$.	
If $U \leq P(F^*)/\hat{P}(F^*)$ 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.

There are various tree sampling algorithms. The methods in [5, 10] 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, 17] 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 4.2) as determinant rejection sampling and random-walk rejection sampling.

5. Complexity analysis for special graphs

All existing tree sampling algorithms [1, 5, 10, 17] 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, analyzing the complexity of algorithms in sections 3 and 4 is non-trivial. 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 $V = \{1, \dots, n\} \times \{1, \dots, n\}$ and two vertices (i, j) and (i', j') are 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 3.3, we can show that the running time of the algorithm grows exponentially, as the number of vertices increases. Algorithm 3.2 can be simplified to Algorithm 5.1 for a lattice graph where each edge has weight 1.

Algorithm 5.1. (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 $\{i, k\} \in \mathcal{L}_t$	01
If $U \leq 0.5$ then $\mathcal{L}_{t+1} = \mathcal{L}_t$	02
Else then $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i, k\}$	03
Else if $i \leftrightarrow k \mathcal{L}_t$	04
Find the path $i \rightarrow j \neq k \rightarrow \dots \rightarrow k \in \mathcal{L}_t$	05
If $U \leq 0.5$ then $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i, j\} \cup \{i, k\}$	06
Else $\mathcal{L}_{t+1} = \mathcal{L}_t$	07
Else	08
If $U \leq 0.5$ then $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i, C\} \cup \{i, k\}$	09
Else $\mathcal{L}_{t+1} = \mathcal{L}_t$	10

Proposition 7. *Given G is a uniformly weighted square lattice, if we start \mathcal{L}_t from the empty forest and update it with Algorithm 5.1, then it takes exponential time to become a spanning tree.*

Proof. See Appendix C.

When we sample from a square lattice, Algorithm 3.6 is more efficient than Algorithm 3.3, since Algorithm 3.6 uses two bounding chains and the amount of coalescent time is always less than the amount of time required by CFTP with the lower chain. But it is non-trivial to analyse theoretically the complexity of Algorithm 3.6 for sampling from a square lattice. We therefore leave this as future research work.

5.2. Complexity analysis for Algorithm 4.2

Algorithm 4.2 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 4.2 for the complete graph with weight 1 on each edge, is $O((p+1)\gamma)$, where γ is 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 4.2) 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 [18], we know that in a complete graph $\frac{\mathcal{W}_{\mathcal{F}(G)}}{\mathcal{W}_{\mathcal{T}(G)}}$, the ratio of the number of forests and the number of spanning trees, is approximately \sqrt{e} when p is large. We then have $\frac{\mathcal{W}_{\mathcal{F}(G)}}{\mathcal{W}_{\mathcal{T}(\tilde{G})}} \approx \frac{\sqrt{e}p^{p-2}}{(p+1)^{p-1}} = \frac{1/(p+1)}{\sqrt{e}}$, for large p .

Therefore Algorithm 4.2 is a polynomial-time algorithm when sampling from complete graphs.

Proposition 9. *Algorithm 4.2 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 that

$$\mathcal{W}_{\mathcal{T}(\tilde{G}_n)}/\mathcal{W}_{\mathcal{F}(G_n)} \geq 4.2655^{(n-1)(n-2)}/3.74101n^2$$

which increases exponentially with n^2 . This means that the acceptance probability decreases to zero exponentially with n^2 . Therefore Algorithm 4.2 needs 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

$$3.64497 \leq \lim_{n \rightarrow \infty} [\mathcal{W}_{\mathcal{F}(G_n)}]^{1/n^2} \leq 3.74101. \quad (5)$$

Lemma 3. Let \tilde{G}_n be the graph derived from the $n \times n$ square lattice by Algorithm 4.2. 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 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.

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.

The number of nodes	16	25	36
Bernoulli RS	< 1s	1s	3s
CFTP	14s	66s	234s
Random walk RS	8s	149s	4120s
Mean number of trees	55	443	4590

TABLE 1: Running time 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.

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 36s to obtain 1000 forests. Bernoulli rejection sampling continues indefinitely and fails to get an output in realistic time, and CFTP

The number of nodes	15	25	35
Bernoulli RS	--	--	--
CFTP	1.2×10^3 s	2.8×10^4 s	7.9×10^5 s
Random walk RS	2.5s	10s	36s
Mean number of trees	19	32	50

TABLE 2: Running time 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.

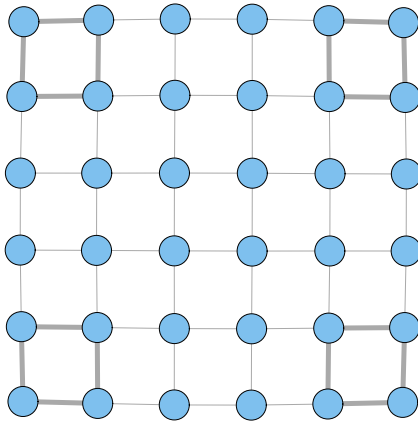


FIGURE 3: Lattice with unequal weights. The thin lines have weight 1 and the thick lines have weight 20.

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

4.2 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 CFTP with bounding chains algorithm. For sparse graphs without heavily weighted loops, we recommend using Bernoulli rejection sampling.

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 3.2). But for the same problem, the complexity of CFTP with bounding chains (Algorithm 3.6) 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 selection of 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 [8, 9, 12], the posterior distribution of the forests has a form as (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 [7].

Appendix A. Proof of Proposition 2

Proof. Algorithm 3.2 updates the lower chain \mathcal{L}_t with 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 $\mathcal{L}_{t+1} \subset \cap \mathcal{E}(F_{t+1})$.

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 3.1 and 3.2, if $U \leq W_{i,k}/(1 + W_{i,k})$, 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 have that if $U > W_{i,k}/(1 + W_{i,k})$, then $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \setminus \{i, k\}$, $s = 1, \dots, m$ 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. If $\{i, k\} \notin \mathcal{L}_t$ and $i \rightarrow j \rightarrow \dots \rightarrow k \in \mathcal{E}(F_t)$, $j \neq k$, $F_t \in \mathcal{F}_t$. According to Algorithm 3.1, if $U \leq W_{i,k}/(W_{i,j} + W_{i,k})$, then $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \setminus \{i, j\} \cup \{i, k\}$; otherwise $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t)$. And according to lines 05 to 08 of Algorithm 3.2, if $U \leq W_{i,k}/(W_{i,j} + W_{i,k})$, then $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i, j\} \cup \{i, k\}$; otherwise $\mathcal{L}_{t+1} = \mathcal{L}_t$. Obviously if $U \leq W_{i,k}/(W_{i,j} + W_{i,k})$ 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 $i \not\leftrightarrow k | \mathcal{L}_t$, 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 $i \rightarrow j_{F_t} \rightarrow \dots \rightarrow k \in \mathcal{E}(F_t)$, $F_t \in \mathcal{F}_t$. Note, if i and k are not connected in F_t or they are neighbors, we define $j_{F_t} = 0$.

Remember the definition of A in Algorithm 3.2. If $U \leq \min_{j \in A \cup \{0\}} \frac{W_{i,k}}{W_{i,j} + W_{i,k}}$, 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 3.2 also tells us $\mathcal{L}_{t+1} = \mathcal{L}_t \setminus \{i, C\} \cup \{i, k\} = \mathcal{L}_t \setminus \{i, A\} \cup \{i, k\}$. 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 $U > \min_{j \in A \cup \{0\}} \frac{W_{i,k}}{W_{i,j} + W_{i,k}}$, we have to introduce extra notations to simplify the proof. For simplicity, We use F_t^* to denote F_t , if $j_{F_t} = 0$. When $j_{F_t} \neq 0$, 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 these notations, we divide all the forests F_t into three groups: F_t^* , \tilde{F}_t and \bar{F}_t . According to $D = \{j | j \in C, j \neq k, U \leq W_{i,k}/(W_{i,k} + W_{i,j})\}$ in line 13 of Algorithm 3.2, we have

$\cup_{\bar{F}_t} \{i, j_{\bar{F}_t}\} \cup \{i, k\} \supset \{i, D\}$. Therefore

$$\begin{aligned}
\mathcal{L}_{t+1} &= \mathcal{L}_t \setminus \{i, D\} = \mathcal{L}_t \setminus [\cup_{\bar{F}_t} \{i, j_{\bar{F}_t}\} \cup \{i, k\}] \\
&\subset [\cap_{\bar{F}_t} \mathcal{E}(\bar{F}_t)] \cap [\cap_{F_t^*} \mathcal{E}(F_t^*)] \cap [\cap_{\bar{F}_t} \mathcal{E}(\bar{F}_t)] \setminus [\cup_{\bar{F}_t} \{i, j_{\bar{F}_t}\} \cup \{i, k\}] \\
&\subset [\cap_{\bar{F}_t} \mathcal{E}(\bar{F}_t)] \cap [\cap_{F_t^*} \mathcal{E}(F_t^*) \setminus \{i, k\}] \cap [\cap_{\bar{F}_t} \mathcal{E}(\bar{F}_t) \setminus \cup_{\bar{F}_t} \{i, j_{\bar{F}_t}\}] \\
&= [\cap_{\bar{F}_t} \mathcal{E}(\bar{F}_t)] \cap [\cap_{F_t^*} \mathcal{E}(F_t^*) \setminus \{i, k\}] \cap [\cap_{\bar{F}_t} (\mathcal{E}(\bar{F}_t) \setminus \{i, j_{\bar{F}_t}\})] \\
&\subset [\cap_{\bar{F}_t} \mathcal{E}(\bar{F}_{t+1})] \cap [\cap_{F_t^*} \mathcal{E}(F_{t+1}^*)] \cap [\cap_{\bar{F}_t} \mathcal{E}(\bar{F}_{t+1})] \\
&= \cap \mathcal{E}(F_{t+1}).
\end{aligned} \tag{6}$$

Note that, we should prove the relation \subset in (6) is correct. According to Algorithm 3.1, if (a): $j_{F_t} \neq 0$, then $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \setminus \{i, j_{F_t}\} \cup \{i, k\}$ given $U \leq W_{i,k}/(W_{i,k} + W_{i,j^s})$ which implies $\mathcal{E}(\bar{F}_t) \setminus \{i, j_{\bar{F}_t}\} \subset \mathcal{E}(\bar{F}_{t+1})$; if (b): $j_{F_t} \neq 0$, then $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t)$ given $U > W_{i,k}/(W_{i,k} + W_{i,j^s})$ which implies $\mathcal{E}(\bar{F}_{t+1}) = \mathcal{E}(\bar{F}_t)$. 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 (6) is correct.

Appendix B. Proof of Proposition 4

Proof. Algorithm 3.4 updates the upper chain and lower chain with 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. $i \leftrightarrow k | \mathcal{L}_t$.

Under cases 1 and 2, the proposition follows from the proof of Proposition 2. In case 3, Algorithm 3.4 calls Algorithm 3.5. So we only need to prove that Algorithm 3.5 guarantees the proposition under $i \leftrightarrow k | \mathcal{L}_t$.

Algorithm 3.5 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, k are either neighbors 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 $\mathcal{L}_{t+1} \subset \cap_s \mathcal{E}(F_{t+1}^s)$, as in the proof of Proposition 2. Remember the definition of vertex set E in Algorithm 3.5, 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, k are neighbors. 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 $\{i, j\} \notin \mathcal{E}(F_{t+1}^s), s = 1, \dots, m$ and further $\{i, E\} \notin \cup \mathcal{E}(F_{t+1})$. 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

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

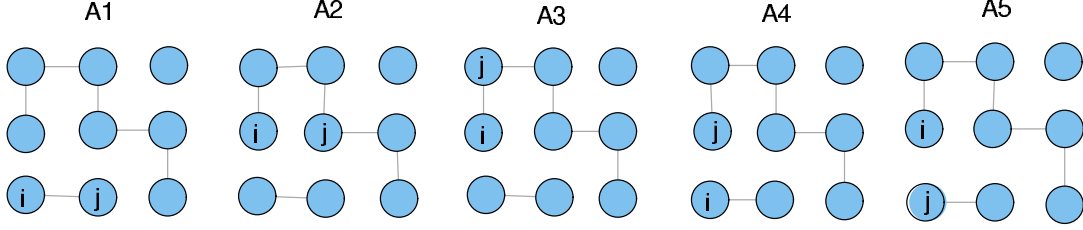
If $\min_{j \in A} \{W_{i,k}/(W_{i,j} + W_{i,k})\} \leq U \leq \max_{j \in A} \{W_{i,k}/(W_{i,j} + W_{i,k})\}$, then for some $F_t \in \mathcal{F}_t$, $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t)$, for some F_t , $\mathcal{E}(F_{t+1}) = \mathcal{E}(F_t) \setminus \{i, k\}$ and for the other F_t , $\mathcal{E}(F_{t+1})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

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

And we can prove $\cap \mathcal{E}(F_{t+1}) \supset \mathcal{L}_{t+1}$ 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 notations. Let $G_n = (V, \mathcal{E})$ be the $n \times n$ square lattice and S_t be the largest subtree in \mathcal{L}_t and $|S_t|$ 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 $\xi_{t+1} = 0$. Assume $1 > \alpha \geq 1/2$. Let $\sigma_t = \{\mathcal{L}_t : \alpha(n^2 + 1) < |S_t| < n^2 - 1\}$. We define the edge

FIGURE 4: The events $\{e \in A_i\}, i = 1, \dots, 5$.

sets

$$\begin{aligned}
A_1 &= \{(i, j) \mid i, j \in G_n \setminus S_t\}, \\
A_2 &= \{(i, j) \mid i, j \in S_t\} \cap \{(i, j) \mid (i, j) \notin S_t\}, \\
A_3 &= \{(i, j) \mid (i, j) \in S_t\}, \\
A_4 &= \{(i, j) \mid i \in G_n \setminus S_t\} \cap \{(i, j) \mid j \in S_t\}, \\
A_5 &= \{(i, j) \mid i \in S_t\} \cap \{(i, j) \mid j \in G_n \setminus S_t\}.
\end{aligned} \tag{7}$$

As when running the lower chain in Algorithm 5.1, 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 $P(A_i \mid \sigma_t) = \alpha_i, i = 1, \dots, 5$. 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, where the first graph is an example of event A_1 where vertices i and j do not belong to the largest subtree S_t . The second graph 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 the third graph $\{i, j\}$ belongs to S_t , therefore it is event A_3 . The fourth graph is event A_4 where j belongs to S_t but i does not. And the fifth graph is event A_5 .

The following lemma is also necessary for proving Proposition 7.

Lemma 4. *With the previous definitions and notations, $\alpha_4 = \alpha_5$ and $\alpha_3 \geq (1/3)(\alpha_4 + \alpha_5)$.*

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 3 vertices which are not leaves (vertices with degree

1), since a lattice has maximum vertex degree 4. Then we can find 3 vertices which have at most two neighbors not in S_t since it has at most 4 neighbors and at least two in S_t . And similarly, each of the other vertices has at most three neighbors not in S_t .

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

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

Now we prove Proposition 7.

Proof. 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 $\xi_0 = n^2 - 1$. For some $\delta > 1$, we immediately attain the following inequality

$$\begin{aligned} P(T \leq t | \xi_0) &= P(\xi_t = 0 | \xi_0) \\ &= P(\delta^{-\xi_t} = 1 | \xi_0) \\ &\leq E(\delta^{-\xi_t} | \xi_0). \end{aligned} \tag{8}$$

Given $A_1 = \{(i, j) | i, j \in G_n \setminus S_t\}$, we have $P(|S_{t+1}| = |S_t| | A_1, \sigma_t) = 1$, since according to Algorithm 5.1, given $e \in A_1$ the choice of e has no effect on $|S_{t+1}|$. So $E_{A_1}(\delta^{|S_{t+1}| - |S_t|} | \sigma_t) = \alpha_1$.

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

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

$$P(|S_{t+1}| = |S_t| | A_3, \sigma_t) = P(|S_{t+1}| - |S_t| \leq -1 | A_3, \sigma_t) = 1/2.$$

This is because when $(i, j) \in S_t$, the edge (i, j) has probability 0.5 to be removed from S_t and probability 0.5 not to be removed. If (i, j) is removed, then the number of edges of the largest subtree in \mathcal{L}_t will decrease at least by 1. Thus $E_{A_3}(\delta^{|S_{t+1}| - |S_t|} | \sigma_t) \leq \alpha_3(0.5 + 0.5\delta^{-1})$.

Recall $A_4 = \{(i, j) | i \in G_n \setminus S_t\} \cap \{(i, j) | j \in S_t\}$. In this case, edge (i, j) has probability 0.5 to be added to S_t and probability 0.5 not to be added. Then

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

So $E_{A_4}(\delta^{|S_{t+1}|-|S_t|}|\sigma_t) = \alpha_4(0.5 + 0.5\delta)$.

Remember that $A_5 = \{(i, j)|i \in S_t\} \cap \{(i, j)|j \in \mathcal{G} \setminus S_t\}$. Similar to the case of A_3 , we have $P(|S_{t+1}| = |S_t| | A_5, \sigma_t) = 1/2$ and $P(|S_{t+1}| - |S_t| \leq -1 | A_5, \sigma_t) = 1/2$. Therefore, $E_{A_5}(\delta^{|S_{t+1}|-|S_t|}|\sigma_t) \leq \alpha_5(0.5 + 0.5\delta^{-1})$.

With all the previous illustration and Lemma B.1, given $\delta \leq 5/3$, we have

$$\begin{aligned}
& E\left(\delta^{|S_{t+1}|-|S_t|}|\alpha(n^2 + 1) < |S_t| < n^2 - 1\right) \\
&= \sum_{i=1}^5 E_{A_i}\left(\delta^{|S_{t+1}|-|S_t|}|\sigma_t\right) \\
&\leq \alpha_1 + \alpha_2 + \alpha_3\left(\frac{1}{2} + \frac{1}{2}\delta^{-1}\right) + \alpha_4\left(\frac{1}{2} + \frac{1}{2}\delta\right) + \alpha_5\left(\frac{1}{2} + \frac{1}{2}\delta^{-1}\right) \\
&= 1 + \frac{1}{2}[(\alpha_3 + \alpha_5)(\delta^{-1} - 1) + \alpha_4(\delta - 1)] \\
&\leq 1 + \frac{1}{2}\left[\left(\frac{5}{3}\alpha_5\right)(\delta^{-1} - 1) + \alpha_5(\delta - 1)\right] \\
&\leq 1.
\end{aligned} \tag{9}$$

From (9), we have

$$\begin{aligned}
& E(\delta^{-\xi_{t+1}}|\xi_t) \\
&\leq E(\delta^{-\xi_{t+1}}|0 < \xi_t \leq (1 - \alpha)(n^2 - 1) - 2\alpha) \\
&\quad + E(\delta^{-\xi_{t+1}}|(1 - \alpha)(n^2 - 1) - 2\alpha < \xi_t < n^2 - 1) \\
&\leq \delta^{-\xi_t} E(\delta^{-(\xi_{t+1} - \xi_t)}|0 < \xi_t \leq (1 - \alpha)(n^2 - 1) - 2\alpha) + \delta^{-[(1 - \alpha)(n^2 - 1) - 2\alpha]} \\
&= \delta^{-\xi_t} E(\delta^{|S_{t+1}|-|S_t|}|\alpha(n^2 + 1) < |F_t| < (n^2 - 1)) + \delta^{-[(1 - \alpha)(n - 1) - 2\alpha]} \\
&\leq \delta^{-\xi_t} + \delta^{-[(1 - \alpha)(n - 1) - 2\alpha]}.
\end{aligned} \tag{10}$$

Therefore,

$$\begin{aligned}
P(T \leq t | \xi_0) &\leq E(\delta^{-\xi_t} | \xi_0) \\
&= E(E(\delta^{-\xi_t} | \xi_{t-1}) | \xi_0) \\
&\leq E(\delta^{-\xi_{t-1}} | \xi_0) + \delta^{-[(1 - \alpha)(n^2 - 1) - 2\alpha]} \\
&\leq \delta^{-\xi_0} + t\delta^{-[(1 - \alpha)(n^2 - 1) - 2\alpha]}.
\end{aligned} \tag{11}$$

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

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

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 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 .

The weighted degree matrix of G is a diagonal matrix with the i th 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 $A = (a_{ij})$, where $a_{ii} = 0$ and $a_{ij} = W_{ij}, i \neq j, \forall i, j$. See [13], for example, for a proof of the theorem.

Now we can prove Lemma 3.

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

According to the Matrix Tree Theorem, we have

$$\mathcal{W}_{\mathcal{T}(\tilde{G}_n)} = \det \begin{pmatrix} A - I & -I & 0 & \cdots & 0 & 0 \\ -I & A & -I & \cdots & 0 & 0 \\ 0 & -I & A & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & & \\ 0 & 0 & 0 & \cdots & A & -I \\ 0 & 0 & 0 & \cdots & -I & A - I \end{pmatrix}, \quad (12)$$

which is a determinant of an $n^2 \times n^2$ partitioned tri-diagonal matrix. In (12) I is the $n \times n$ identity matrix and A is an $n \times n$ tri-diagonal matrix, given by

$$A = \begin{pmatrix} 4 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 5 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 5 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & & \\ 0 & 0 & 0 & \cdots & 5 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 4 \end{pmatrix}. \quad (13)$$

Let J_n be the determinant of an $n \times n$ tri-diagonal matrix with 4.5 in diagonal entries and -1 in sub-diagonal and super-diagonal entries. We have the recursive formula $J_n = 4.5J_{n-1} - J_{n-2}$. It is easy to have $J_n = \tilde{\alpha}\tilde{\alpha}_1^n + \tilde{\beta}\tilde{\alpha}_2^n$, where $\tilde{\alpha} = \frac{36.5 + 9\sqrt{16.25}}{32.5 + 9\sqrt{16.25}} > 0$ and $\tilde{\beta} = \frac{36.5 - 9\sqrt{16.25}}{32.5 - 9\sqrt{16.25}} < 0$. Therefore

$$|A - I/2| \geq 3.5^2 J_{n-2} - 7J_{n-3} + J_{n-4} \geq 10.5J_{n-2} \geq \tilde{\alpha}_1^{n-1}, \quad (14)$$

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

From

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

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

Denote the eigenvalues of A by λ_l , for $l = 1, \dots, n$. Obviously for all l , $\lambda_l > 3$. Thus we have $\det(\tilde{A}_2) = \det(A - (A - I)^{-1}) = \prod_{l=1}^n (\lambda_l - (\lambda_l - 1)^{-1})$, which is larger than $\prod_{l=1}^n (\lambda_l - 0.5) = \det(A - I/2)$. Similarly, for all $i = 1, \dots, n-1$, $\det(\tilde{A}_i) > \det(A - I/2)$. With the results $\det(\tilde{A}_1) > 1$, $\det(\tilde{A}_n) > 1$ and (14), we have

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

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. on Foundations of Computer Science*, 442-447.
- [3] CALKIN, N. AND MERINO, C. AND NOBLE, S. AND NOY, M. (2003). Improved bounds for the number of forests and acyclic orientations in the square lattice. *The Electronic Journal of Combinatorics*, **10**:R4.
- [4] CALLAN, D. (2003). A combinatorial derivation of the number of labelled forests. *Journal of Integer Sequences*, **6**:Article 03.4.7.
- [5] COLBOURN, C. J. AND DAY, R. R. J. AND NEL, L. D. (1989). Unranking and ranking spanning trees of a graph. *Journal of Algorithms*, **10**:271-286.
- [6] COLBOURN, C. J. AND MYRVOLD, W. J. AND NEUFELD, E. (1996). Two algorithms for unranking arborescences. *Journal of Algorithms*, **20**:268-281.
- [7] DAI, H. (2007). Perfect simulation methods for Bayesian applications. *PhD thesis, University of Oxford. Advisor: Dr. Peter Clifford.*
- [8] DAWID, A. P. AND LAURITZEN, S. L. (1993). Hyper Markov laws in the statistical analysis of decomposable graphical models. *Annals of Statistics*, **21**:1272-1317.
- [9] GIUDICI, P. (1996). Learning in graphical Gaussian models. *Bayesian Statistics*, **5**:621-628, Oxford University Press.
- [10] GUENOCHÉ, A. (1983). Random spanning tree. *Journal of Algorithms*, **4**:214-220.
- [11] HUBER, M. (2004). Perfect sampling using bounding chains. *The Annals of Applied Probability*, **14**:734-753.
- [12] JONES, BEATRIX AND CARVALHO, C. AND DOBRA, A. AND HANS, C. AND CARTER, C. AND WEST, M. (2005). Experiments in stochastic computation for high-dimensional graphical models. *Statistical Science*, **4**:388-400.
- [13] JONES, BRIAN D. AND PITTEL, B. G. AND VERDUCCI, J. S. (1999). Tree and forest weights and their application to nonuniform random graphs. *The Annals of Applied Probability*, **9**:197-215.
- [14] LAURITZEN, S. L. (1996). *Graphical models*. Clarendon Press, Oxford.
- [15] MØLLER, J. (1999). Perfect simulation of conditionally specified models. *Journal of the Royal Statistical Society, B*, **61**:251-264.
- [16] PROPP, J. G. AND WILSON, D. B. (1996). Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, **9**:223-252.
- [17] 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. *Journal of Algorithms*, **27**:170-217.

- [18] TAKÁCS, L. (1990). On the number of distinct forests. *SIAM Journal on Discrete Mathematics*, **3**(4): 574-581.