Generating Random Spanning Trees via Fast Matrix Multiplication

Nicholas J. A. Harvey and Keyulu Xu

University of British Columbia,
Vancouver, BC, Canada
nickhar@cs.ubc.ca,keyulu.x@gmail.com

Abstract. We consider the problem of sampling a uniformly random spanning tree of a graph. This is a classic algorithmic problem for which several exact and approximate algorithms are known. Random spanning trees have several connections to Laplacian matrices; this leads to algorithms based on fast matrix multiplication. The best algorithm for dense graphs can produce a uniformly random spanning tree of an n-vertex graph in time $O(n^{2.38})$. This algorithm is intricate and requires explicitly computing the LU-decomposition of the Laplacian.
We present a new algorithm that also runs in time $O(n^{2.38})$ but has several conceptual advantages. First, whereas previous algorithms need to introduce *directed* graphs, our algorithm works only with undirected graphs. Second, our algorithm uses fast matrix inversion as a black-box, thereby avoiding the intricate details of the LU-decomposition.

Keywords: Uniform Spanning Trees · Spectral Graph Theory · Fast Matrix Multiplication · Laplacian Matrices

1 Introduction

Enumerating and sampling spanning trees of a graph is a classic problem in combinatorics dating back to Kirchhoff's celebrated matrix-tree theorem [16] from 1847. From this result, one can fairly easily derive a polynomial-time algorithm to generate a uniformly random spanning tree. Over the past few decades, researchers have developed several startling algorithms for this problem with improved running times.

The existing algorithms fall into three broad classes.

Laplacian-based algorithms Properties of the graph's Laplacian matrix allow one to compute the number of spanning trees in the graph. Similarly, one can compute the probability that a given edge is in a uniformly random spanning tree. A sequence of papers [12, 18, 8, 9] developed improved algorithms following this approach. This culminated in the algorithm of Colbourn, Myrvold and Neufeld which has running time $O(n^\omega)$, where $\omega < 2.373$ is the best-known exponent for matrix multiplication. These algorithms are most efficient on dense graphs.

Random walks Aldous [1], Broder [3] and Wilson [21] showed that remarkably simple algorithms using random walks can be used to generate a uniformly random spanning tree. These algorithms are particularly efficient on graphs whose cover time or mean hitting time is small.

Approximate algorithms Recent advances in algorithmic spectral graph theory have led to nearly-linear time algorithms for approximately solving linear systems involving Laplacian matrices [17]. These methods can be used to accelerate the random walk algorithms by identifying regions of the graph where the random walk will be slow [15, 20]. These algorithms are most efficient on sparse graphs.

1.0.0.1 Applications. The interest in enumerating and sampling spanning trees is not only due to its origins as a foundational problem in combinatorics. Random spanning trees have also turned out to be useful in many other contexts in combinatorics and computer science. For example, Colbourn et al. [7] showed how the coefficients of the reliability polynomial can be estimated using random spanning trees. Goyal, Rademacher and Vempala [11] have used random spanning trees to generate expander graphs. Recent breakthroughs on the traveling salesman problem [2, 10] involve so-called "λ-random spanning trees", which are essentially uniformly random spanning trees in multigraphs. Other distributions on spanning trees have been used to show results in spectral graph theory [14]. More generally, random distributions on matroid bases have had interesting applications in submodular optimization [6].

1.1 Related Work

Consider the following algorithm for sampling any subgraph [18, Algorithm A]. Consider the edges in order; for each edge, decide if it is in the subgraph or not with probability conditioned on the previous decisions. It is a trivial consequence of the chain rule for conditional probabilities that this generates a random subgraph according to the desired distribution.

This algorithm can be used to generate uniformly random spanning trees if one can determine the probability of an edge being in the tree, conditioned on all previous decisions. It turns out that conditioning on an edge *not* being in the tree is the same as deleting the edge, whereas conditioning on an edge being in the tree is the same as *contracting* the edge. Thus, we may use the matrix-tree theorem to determine the sampling probability for each edge, by considering the graph with all the necessary deletions and contractions. Guenoche [12] and Kulkarni [18] discussed this method and showed that it can be implemented in time $O(n^3 m)$. A more detailed discussion of this method is given in Section 3.

Colbourn, Day and Nel [8] showed that the runtime of this method can be improved to $O(n^3)$. Their algorithm is recursive and applies partial Gaussian elimination. Colbourn, Myrvold and Neufeld [9] presented a different algorithm that also has runtime $O(n^3)$. Their first observation is that the desired sampling probabilities can be determined in constant time from the inverse of the (modified) Laplacian matrix (which they call the Kirchhoff matrix). Then, they observe that, after contracting an edge, the new inverse of the Laplacian matrix

can be computed in $O(n^2)$ time by the Sherman-Morrison formula. Since the algorithm performs $n-1$ contractions, the total runtime is $O(n^3)$.

The best running time for dense graphs is obtained by another algorithm of Colbourn, Myrvold and Neufeld (CMN) [9]. They show that fast matrix multiplication can be used to give an algorithm with runtime $O(n^\omega)$. This algorithm abandons the Sherman-Morrison formula and instead computes the LU-decomposition of the Laplacian matrix via a "six-way divide-and-conquer algorithm". The rather intricate details of this approach are strongly reminiscent of the Bunch-Hopcroft algorithm [4] for fast matrix inversion.

1.2 Our Techniques

In this paper, we present a new algorithm for sampling a uniformly random spanning tree in $O(n^\omega)$ time. Our approach is different from, and arguably simpler than, the CMN algorithm. We recursively enumerate all edges in the graph, and lazily update the inverse of the Laplacian matrix as edges are chosen to be added to the tree or not. The updates are determined by an extension of the Sherman-Morrison formula and can be performed using fast matrix inversion as a black box. This avoids many of the intricacies of the approach based on LU-decomposition. Our idea for this approach originates from a similar algorithm for non-bipartite matching that also uses fast matrix inversion [13].

Nevertheless, there are numerous challenges that must be addressed in the present work. One challenge is that the Laplacian matrix is not invertible. Previous algorithms dealt with that by deleting the row and column associated with an arbitrary vertex and inverting the resulting matrix instead. We avoid this issue by working with the Moore-Penrose *pseudoinverse* of the Laplacian, which always exists. We must then derive a new extension of the Sherman-Morrison formula for updating the pseudoinverse. Such formulas are known, but quite complicated in general — a standard reference [5, §3.1] describes an algorithm that involves six different cases! Our formulas are much simpler.

Another challenge relates to the contraction of edges. Normally contracting an edge involves decreasing the number of vertices by one. Performing the corresponding operation to the Laplacian and its pseudoinverse is quite cumbersome. The CMN algorithm avoids this issue by working with directed graphs and sampling arboresences. In a directed graph, the analog of this contraction operation is to delete all-but-one incoming arc to a vertex; this does not affect the number of vertices. We adopt a different approach that avoids unnecessarily resorting to directed graphs. We effectively contract an edge by increasing its weight to be a large value k. In the limit $k \to \infty$, this is equivalent to contracting the edge, from the point of view of electrical networks and spanning trees.

2 Preliminaries

The graph G is assumed to be undirected, simple, connected and unweighted.

2.1 Notations

In this section, we explain the notations that we use in the algorithms and theorems.

Definition 1. *Given an unweighted graph $G = (V_G, E_G)$ with $|V_G| = n$, its Laplacian matrix $L_G = (l_{i,j})_{n\times n}$ is defined as $L_G = D - A$, where D is the degree matrix and A is the adjacency matrix, i.e.*

$$l_{i,j} = \begin{cases} deg(v_i) & (\text{if } i = j) \\ -1 & (\text{if } i \neq j \text{ and } v_iv_j \in E_G) \\ 0 & \text{otherwise} \end{cases}.$$

Given any set $E \subseteq E_G$, we may define its Laplacian L_E to be the Laplacian of the subgraph (V_G, E).

We also define the Laplacian of a graph with finite weights. Suppose that $w : E \to \mathbb{R}_{\geq 0}$ assigns weights to the edges of G. Then the weighted Laplacian is $L_w = (l_{i,j})_{n\times n}$ where

$$l_{i,j} = \begin{cases} \sum_{e \text{ incident on } i} w_e & (\text{if } i = j) \\ -w_e & (\text{if } e = \{i, j\} \in E) \\ 0 & (\text{otherwise}) \end{cases}$$

Definition 2. *Let A be a matrix. A submatrix containing rows S and columns T is denoted $A_{S,T}$. A submatrix containing all rows (resp., columns) is denoted $A_{*,T}$ (resp., $A_{S,*}$).*

Remark 1. Throughout this paper we will use the notation of Definition 2 for matrices such as L_G whose notation already involves a subscript. Mathematical correctness would suggest using the notation $(L_G)_{S,T}$ but for typographical clarity we will instead use the notation $L_{G_{S,T}}$.

Definition 3. *Let $A \in \mathbb{M}_{m\times n}$, a pseudoinverse of A is defined as $A^+ \in \mathbb{M}_{n\times m}$ satisfying all of the following criteria: $AA^+A = A$, $A^+AA^+ = A^+$, $(AA^+)^T = AA^+$, $(A^+A)^T = A^+A$.*

Definition 4. *Define $\omega \in \mathbb{R}$ as the infimum over all $c \in \mathbb{R}$ such that multiplying two $n \times n$ matrices takes $O(n^c)$ time. Matrix inverse of an $n \times n$ matrix can also be computed in $O(n^\omega)$ time.*

2.2 Facts

We will use the following basic facts. Proofs of these facts can be found in books on linear algebra and spectral graph theory.

Fact 1 (Sherman-Morrison-Woodbury formula). *Let $M \in \mathbb{M}_{n\times n}, U \in \mathbb{M}_{n\times k}, V \in \mathbb{M}_{n\times k}$. Suppose M is non-singular. Then $M + UV^T$ is non-singular if and only if $I + V^TM^{-1}U$ is non-singular. If $M + UV^T$ is non-singular, then*

$$(M + UV^T)^{-1} = M^{-1} - M^{-1}U(I + V^TM^{-1}U)^{-1}V^TM^{-1}$$

$P := I - \mathbf{1}\mathbf{1}^T/n$

Fact 2. *For any $L \in \mathbb{M}_{n\times n}$ with kernel* span$(\mathbf{1})$*, we have $LL^+ = I - \frac{\mathbf{1}\mathbf{1}^T}{n}$. We call $I - \frac{\mathbf{1}\mathbf{1}^T}{n}$ the projection matrix P.*

Fact 3 (Facts about Submatrices).

1. *For any $A, B \in \mathbb{M}_{m\times n}$ and index set S, $(A+B)_{S,S} = A_{S,S} + B_{S,S}$.*
2. *For any matrices C, D, E, F and index set S, if $C = DEF$, then $C_{S,S} = D_{S,*}EF_{*,S}$.*
3. *For any $A \in \mathbb{M}_{m\times n}, B \in \mathbb{M}_{n\times l}$ and index set S, if A or B is only non-zero in S, S, then $(AB)_{S,S} = A_{S,S} \times B_{S,S}$.*
4. *For any matrices $C = DEF$ and index set S. If $D_{*,S^c} = 0$ and $F_{S^c,*} = 0$, then $C = D_{*,S}E_{S,S}F_{S,*}$.*
5. *Suppose $D = \left[\begin{smallmatrix} M & 0 \\ 0 & 0 \end{smallmatrix}\right]$ and $E = \left[\begin{smallmatrix} A & B \\ X & Y \end{smallmatrix}\right]$ where M, A are n-by-n and $MA - I$ is non-singular. Then we have*

$$(DE - I)^{-1} = \begin{bmatrix} (MA-I)^{-1} & (MA-I)^{-1}MB \\ 0 & -I \end{bmatrix}$$

Fact 4. *Let $A, B \in \mathbb{M}_{n\times n}$ with B symmetric positive semi-definite. Suppose x is an eigenvector of AB corresponding to eigenvalue λ. Then $B^{1/2}x$ is an eigenvector of $B^{1/2}AB^{1/2}$ corresponding to eigenvalue λ.*

Fact 5. *Let G be a graph with n vertices. Let $\lambda_1 \le \cdots \le \lambda_n$ be the eigenvalues of L_G with the corresponding eigenvectors $v_1, \cdots, v_n$. Then L_G is symmetric positive semi-definite. $\lambda_1 = 0$ and $v_1 = \mathbf{1}$. Moreover, $\lambda_2 > 0$ if and only if G is connected, i.e. G is disconnected if and only if $\exists z$ with $z^T\mathbf{1} = 0$ and $z^T L_G z = 0$. Everything above holds for L_G^+ as well.*

3 The Chain-Rule Algorithm

Given a simple undirected connected graph $G = (V_G, E_G)$, let $\mathcal{T}$ be the set of all spanning trees of G. We want to sample a uniformly random spanning tree $\hat{T} \subseteq E_G$ such that for any $T \in \mathcal{T}$, $\mathbb{P}(\hat{T} = T) = 1/|\mathcal{T}|$.

As described in Section 1.1, there is a simple algorithm for generating uniformly random spanning trees based on the chain-rule for conditional probabilities [12] [18, Algorithm A8] [19, §4.2]. The algorithm traverses the graph and samples an edge with the conditional probability of it belonging to the tree. Fact 6 below shows that this conditional probability is determined by effective resistances in the graph where edges are contracted or deleted in accordance with the algorithm's previous decisions. This algorithm is shown in Algorithm 1.

Fact 6. *Given an graph $G = (V_G, E_G)$ with Laplacian L_G, the effective resistance of an edge $e = \{u, v\} \in E_G$ is defined as*

$$R_e^{\mathit{eff}} \;=\; (\mathcal{X}_u - \mathcal{X}_v)^T L_G^+ (\mathcal{X}_u - \mathcal{X}_v).$$

where $\mathcal{X}_u$ is a unit vector of size $|V_G|$ with $\mathcal{X}_u(u) = 1$ and 0 otherwise. Let $\hat{T}$ be a random variable denoting a uniformly random spanning tree, i.e. $\mathbb{P}(\hat{T} = T) = 1/|\mathcal{T}|$ for any $T \in \mathcal{T}$, where $\mathcal{T}$ is the set of all spanning trees of G. Then for any $e \in E_G$, we have $\mathbb{P}(e \in \hat{T}) = R_e^{\mathit{eff}}$.

Algorithm 1 Sampling a uniformly random spanning tree using the chain-rule.

```
function SampleSpanningTree(G = (V, E))
    for e = {u, v} ∈ E do
        R_e^eff ← (χ_u − χ_v)^T L_G^+ (χ_u − χ_v)
        Flip a biased coin that turns head with probability R_e^eff
        if head then
            Add e to the spanning tree
            Contract e from G and update L_G^+
        else
            Delete e from G and update L_G^+
```

The algorithm involves three key properties that guarantee correctness.

- **P1:** It visits every edge of E_G exactly once.
- **P2:** It examines L_G^+ to compute the correct conditional probability of sampling an edge.
- **P3:** It updates L_G^+ to incorporate the contraction or deletion of that edge.

The naive method to update L_G^+ is to recompute it from scratch, which would require $O(n^3)$ time. There are at most n^2 edges, so overall the algorithm runs in $O(n^5)$ time.

4 A Recursive Algorithm with Lazy Updates

In this section, we present Algorithm 2, which, based on Algorithm 1, provides a faster way to update the Laplacian pseudoinverse and reduces the runtime to $O(n^\omega)$. The only difference between Algorithm 2 and Algorithm 1 is that Algorithm 2 visits the edges in a specific order to exploit lazy updates to L_G^+.

4.1 Update Formulas

In this subsection, we present our update formulas for L_G^+. We first observe that the effective resistance of any edge only depends on one entry of L_G^+. To see that, for any edge $\{u, v\}$, it follows from Fact 3.4 that

$$R_e^{\text{eff}} = (\chi_u - \chi_v)^T L_G^+ (\chi_u - \chi_v) = [1, -1] L^+_{G_{\{u,v\},\{u,v\}}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Therefore, when we are deciding whether to sample an edge, all we need to ensure is that the value of the corresponding entry in the Laplacian pseudoinverse is correct, which makes lazy updates desirable. Suppose we have made sampling decisions for some edges of a graph G but have not changed L_G^+ to reflect these decisions. Let F be the set of edges sampled and D be the set of edges discarded. We want to (partially) update L_G^+ to the Laplacian pseudoinverse of the graph obtained by contracting edges in F and deleting edges in D from G.

Because the order of updates does not matter, we make the deletion updates altogether before making the contraction updates. Theorem 1 and Corollary 1 give update formulas for deletion. Lemma 1 states that these formulas are well-defined.

Lemma 1. *Let $G = (V_G, E_G)$ be a connected graph and $D \subseteq E_G$. $I - L_D L_G^+$ is non-singular iff $G \setminus D$ contains at least one spanning tree.*

Proof. $I - L_D L_G^+$ is singular iff $1 \in eig(L_D L_G^+)$ because I only has eigenvalue 1. $eig(L_D L_G^+) = eig((L_G - L_{G\setminus D})L_G^+)$. By Fact 5, $\mathbf{1}$ lies in the kernel of L_G^+. Suppose $1 \in eig(L_D L_G^+)$. Let $x \perp \mathbf{1}$ be an eigenvector of $(L_G - L_{G\setminus D})L_G^+$ corresponding to eigenvalue 1. Let $y = (L_G^+)^{1/2}x / \|(L_G^+)^{1/2}x\|$. By Fact 4, y is an eigenvector of $(L_G^+)^{1/2}(L_G - L_{G\setminus D})(L_G^+)^{1/2}$ corresponding to eigenvalue 1. We have

$$y^T (L_G^+)^{1/2}(L_G - L_{G\setminus D})(L_G^+)^{1/2} y = 1$$

Also, it is clear that

$$y^T (L_G^+)^{1/2} L_G (L_G^+)^{1/2} y = y^T L_G^+ L_G y = y^T P y = y^T (I - \mathbf{1}^T\mathbf{1}/n) y = y^T y = 1$$

It follows that $y^T (L_G^+)^{1/2} L_{G\setminus D} (L_G^+)^{1/2} y = 0$. Also, $y^T (L_G^+)^{1/2}\mathbf{1} = x^T L_G^+ \mathbf{1} = 0$. By Fact 5, $G \setminus D$ is disconnected. Hence $L_D L_G^+$ is non-singular if $G \setminus D$ contains at least one spanning tree.

Conversely, suppose $G \setminus D$ is disconnected. Then by Fact 5 and Fact 4, there exists $y \perp \mathbf{1}$ of length 1 such that $y^T (L_G^+)^{1/2} L_{G\setminus D} (L_G^+)^{1/2} y = 0$. Also, $y^T (L_G^+)^{1/2} L_G (L_G^+)^{1/2} y = y^T y = 1$. Hence $y^T (L_G^+)^{1/2}(L_G - L_{G\setminus D})(L_G^+)^{1/2} y = 1$. It follows that $1 \in eig(L_D L_G^+)$ and $I - L_D L_G^+$ is singular.

$(L_G - L_D)^+$ is the Laplacian pseudoinverse of the graph obtained by deleting edges in D from G. The runtime of each update in Theorem 1 is $O(|V_G|^\omega)$.

Theorem 1. *Let $G = (V_G, E_G)$ be a connected graph and $D \subseteq E_G$. If $G \setminus D$ contains at least one spanning tree, then*

$$(L_G - L_D)^+ = L_G^+ - L_G^+ \left(L_D L_G^+ - I\right)^{-1} L_D L_G^+$$

Proof. By Lemma 1, $(L_D L_G^+ - I)^{-1}$ is well-defined. Since G and $G \setminus D$ are connected, by Fact 5 and Fact 2, $(L_G - L_D)(L_G - L_D)^+ = P$. We have

$$\begin{aligned}
&(L_G - L_D)(L_G^+ - L_G^+(L_D L_G^+ - I)^{-1} L_D L_G^+) \\
=&L_G L_G^+ - L_D L_G^+ - ((L_G L_G^+ - L_D L_G^+)(L_D L_G^+ - I)^{-1} L_D L_G^+) \\
=&P - L_D L_G^+ + ((L_D L_G^+ - I + \mathbf{1}\cdot\mathbf{1}^T/n)(L_D L_G^+ - I)^{-1} L_D L_G^+) \\
=&P - L_D L_G^+ + L_D L_G^+ + \mathbf{1}\cdot\mathbf{1}^T/n (L_D L_G^+ - I)^{-1} L_D L_G^+
\end{aligned}$$

We claim $\mathbf{1}^T (L_D L_G^+ - I)^{-1} = -\mathbf{1}^T$. To see that,

$$\begin{aligned}
-\mathbf{1}^T (L_D L_G^+ - I) &= \mathbf{1}^T (I - L_D L_G^+) \\
&= \mathbf{1}^T (I - L_G L_G^+ + L_{G\setminus D} L_G^+) \\
&= \mathbf{1}^T (\mathbf{1}\cdot\mathbf{1}^T/n + L_{G\setminus D} L_G^+) \\
&= \mathbf{1}^T + \mathbf{1}^T (L_{G\setminus D} L_G^+) = \mathbf{1}^T
\end{aligned}$$

It follows from the claim that $\mathbf{1}\cdot\mathbf{1}^T/n (L_D L_G^+ - I)^{-1} L_D L_G^+ = 0$ because $\mathbf{1}^T L_D = 0$. Hence $(L_G - L_D)(L_G^+ - L_G^+(L_D L_G^+ - I)^{-1} L_D L_G^+) = P$.

The formula in Theorem 1 updates the entire L_G^+, which is unnecessary because we will not be using most entries of L_G^+ immediately. Corollary 1 gives a formula that updates a submatrix of L_G^+, using only the values of that submatrix. The updated submatrix has the same value as the submatrix of the Laplacian pseudoinverse of the graph obtained by deleting edges in D from G. The runtime of each update is improved to $O(|S|^\omega)$.

Corollary 1. *Let $G = (V_G, E_G)$ be a connected graph and $D \subseteq G$. Let $S \subseteq V_G$. Define $E[S]$ as the set of edges whose vertices are in S. Suppose $D \subseteq E[S]$ and $G \setminus D$ contains at least one spanning tree, then*

$$(L_G - L_D)^+_{S,S} = L^+_{G_{S,S}} - L^+_{G_{S,S}}(L_{D_{S,S}} L^+_{G_{S,S}} - I)^{-1} L_{D_{S,S}} L^+_{G_{S,S}}.$$

Proof. L_D is only non-zero on the rows and columns indexed by S, since $D \subseteq E[S]$. Fact 3.5 implies that

$$(L_D L_G^+ - I)^{-1} \;=\; \begin{bmatrix} (L_{D_{S,S}} L^+_{G_{S,S}} - I)^{-1} & (L_{D_{S,S}} L^+_{G_{S,S}} - I)^{-1} L_{D_{S,S}} L_{G_{S,S^c}} \\ 0 & -I \end{bmatrix} \quad (1)$$

and in particular that

$$(L_D L_G^+ - I)^{-1}_{S,S} \;=\; (L_{D_{S,S}} L^+_{G_{S,S}} - I)^{-1}. \quad (2)$$

Combining Theorem 1, Fact 3.1 and 3.3 gives

$$(L_G - L_D)^+_{S,S} = L^+_{G_{S,S}} - L^+_{G_{S,S}}(L_D L_G^+ - I)^{-1}_{S,S} L_{D_{S,S}} L^+_{G_{S,S}}.$$

The result now follows from (2).

We present similar update formulas for contraction. As mentioned in Section 1.2, algorithms for generating random spanning trees must contract edges but somehow avoid the cumbersome updates to the Laplacian that result from decreasing the number of vertices. Our approach is to increase the edge's weight to a large value k. By Fact 7 below, this is equivalent to contracting the edge in the limit as $k \to \infty$. One must be careful to specify formally what this means, because we have only defined the Laplacian of a weighted graph when the weights are finite. However, this does not matter. The main object of interest to us is L_G^+, and this *does* have a finite limit as $k \to \infty$.

To emphasize the graph under consideration, we use the following notation: $R_e^{\text{eff}}[H]$ denotes the effective resistance of edge e in the graph H.

Fact 7. *Let G be a weighted graph. Let e, f be distinct edges in G. Let G/e be the graph obtained by contracting edge e. Let $G + ke$ be the weighted graph obtained by increasing e's weight by k. Then*

$$R_f^{\mathit{eff}}[G/e] \;=\; \lim_{k\to\infty} R_f^{\mathit{eff}}[G + ke].$$

Let us make explicit the dependence on k in the graphs and matrices used by the algorithm. For any finite k, define $G(k) := G \setminus D + kF$, the graph obtained

by deleting the edges D then increasing the weight of edges in F by k. For any edge $e = \{u, v\}$, we have

$$\begin{aligned} R_e^{\text{eff}}[G \setminus D/F] &= \lim_{k\to\infty} R_e^{\text{eff}}[G(k)] \qquad \text{(by Fact 7)} \\ &= \lim_{k\to\infty} (\mathcal{X}_u - \mathcal{X}_v)^T L_{G(k)}^+ (\mathcal{X}_u - \mathcal{X}_v) \qquad \text{(by Fact 6)} \\ &= (\mathcal{X}_u - \mathcal{X}_v)^T \lim_{k\to\infty} L_{G(k)}^+ (\mathcal{X}_u - \mathcal{X}_v) \end{aligned}$$

Thus, if the Laplacian pseudoinverse is updated to $\lim_{k\to\infty} L_{G(k)}^+$, then the algorithm will sample edges with the correct probability. The next few theorems give the update formulas. Let us first give a definition of incidence matrices.

Definition 5. *Let $G = (V_G, E_G)$ be a graph with n vertices. Given an edge $e = \{u, v\} \in E_G$, we define the incidence vector of e as $v_e = (\mathcal{X}_u - \mathcal{X}_v)$. Given a set of edges $E = \{e_1, e_2, \cdots, e_m\} \subseteq E_G$, we define the incidence matrix of E as $V_E = [v_{e_1} | v_{e_2} | \cdots | v_{e_m}]$.*

By the definition of the weighted Laplacian, $L_{G+kF} = L_G + kV_F V_F^T$. The next two lemmas state that our contraction update formulas are well-defined.

Lemma 2. *Let $G = (V_G, E_G)$ be a connected graph. Given $F \subseteq E_G$ with $|F| = r$, let V be the incidence matrix of F. $V^T L_G^+ V$ is non-singular iff F is a forest.*

Proof. Suppose F is a forest. For any $x \in \mathbb{R}^r$, $x \neq 0$, let $y = Vx$. Since F is a forest, V has full column rank. Therefore $y \neq 0$. Clearly $y^T \mathbf{1} = x^T (V^T \mathbf{1}) = 0$. By Fact 5, L_G^+ is PSD and $\ker(L_G^+) = \mathbf{1}$. Thus $y \perp \ker(L_G^+)$. We have

$$x^T V^T L_G^+ V x = y^T L_G^+ y > 0$$

Hence $V^T L_G^+ V$ is positive definite and thus non-singular. The converse is trivial.

Lemma 3. *Let G be a connected graph. Given $F \subseteq E_G$, let V be the incidence matrix of F. If F is a forest, then $I/k + V^T L_G^+ V$ is non-singular for any $k > 0$.*

Proof. By Lemma 2, $V^T L_G^+ V$ is positive definite. Since $k > 0$, I/k is also positive definite. The lemma follows from the sum of two positive definite matrices is positive definite.

Theorem 2 and Corollary 2 give contraction update formulas for a finite k. Corollary 2 improves on Theorem 2 by only updating a submatrix. The runtime of each update in Corollary 2 is $O(|S|^\omega)$.

Theorem 2. *Let $G = (V_G, E_G)$ be a connected graph. Given a forest $F \subseteq E_G$, let V be the incidence matrix of F. For any $k > 0$,*

$$(L_G + k \cdot L_F)^+ = L_G^+ - L_G^+ V (I/k + V^T L_G^+ V)^{-1} V^T L_G^+$$

Proof. Let $M_k = L_G + k \cdot L_F = L_G + k \cdot VV^T$ and $N_k = L_G^+ - L_G^+V(I/k + V^TL_G^+V)^{-1}V^TL_G^+$. By Lemma 3, N_k is well-defined. By Fact 5, $\ker(L_G^+) = span(\mathbf{1})$. By Fact 2, $L_GL_G^+ = P = I - \mathbf{1}\cdot\mathbf{1}^T/|V_E|$. We have

$$\begin{aligned} M_kN_k &= (L_G + kVV^T)(L_G^+ - L_G^+V(I/k + V^TL_G^+V)^{-1}V^TL_G^+) \\ &= P + kVV^TL_G^+ - (L_GL_G^+V + kVV^TL_G^+V)(I/k + V^TL_G^+V)^{-1}V^TL_G^+ \\ &= P + kVV^TL_G^+ - kV(I/k + V^TL_G^+V)(I/k + V^TL_G^+V)^{-1}V^TL_G^+ \qquad (3) \\ &= P + kVV^TL_G^+ - kVV^TL_G^+ = P \end{aligned}$$

where (3) follows from the sum of any column of an incidence matrix is 0. Since $G + kF$ is connected, we have $M_k^+ = N_k$.

Corollary 2. *Let $G = (V_G, E_G)$ be a connected graph. Given a forest $F \subseteq E_G$, let V be the incidence matrix of F. Suppose $F \subseteq E[S]$, where $S \subseteq V_G$. Then for any $k > 0$,*

$$(L_G + k \cdot L_F)^+_{S,S} = L^+_{G_{S,S}} - L^+_{G_{S,S}}V_{S,*}(I/k + V^T_{S,*}L^+_{G_{S,S}}V_{S,*})^{-1}V^T_{S,*}L^+_{G_{S,S}}$$

Proof. V is only non-zero in rows in S. By Fact 3.4 $V^T_{S,*}L^+_{G_{S,S}}V_{S,*} = V^TL_G^+V$. The corollary then follows from Fact 3.1, 3.2 and 3.3.

Remark 2. Because the set of sampled edges, i.e. contracted edges F is a forest, V has at most $|S|$ columns.

The following theorem extends the result in Theorem 2 to $k = \infty$ and gives a contraction update formula that we use in Algorithm 2.

Theorem 3. *Let G be a graph with finite weights. Let $G(k) = G + kF_1$ for a forest $F_1 \subseteq E_G$. Let $F_2 \subseteq E_G$ be disjoint from F_1 such that $F_1 \cup F_2$ is a forest. Let V be the incidence matrix of F_2. For $k > 0$, define $N = \lim_{k\to\infty} L^+_{G(k)}$. Then*

$$\lim_{k\to\infty} L^+_{G(k)+kF_2} \;=\; N - NV(V^TNV)^{-1}V^TN.$$

Furthermore $\ker(\lim_{k\to\infty} L^+_{G(k)+kF_2}) = \mathrm{span}(V_{F_1\cup F_2} \cup \mathbf{1})$.

Proof. We first show that $\lim_{k\to\infty} L^+_{G+kF} \;=\; L_G^+ - L_G^+V(V^TL_G^+V)^{-1}V^TL_G^+$, where V is the incidence matrix of F. By Lemma 2, $V^TL_G^+V$ is invertible so the RHS of the formula above is well-defined. Let $N_k = (L_G + k \cdot L_F)^+ = L_G^+ - L_G^+V(I/k + V^TL_G^+V)^{-1}V^TL_G^+$ and $N = L_G^+ - L_G^+V(V^TL_G^+V)^{-1}V^TL_G^+$. We show as $k \to \infty$, N_k converges to N with respect to any matrix norm. Let $A = V^TL_G^+V$. We have

$$\begin{aligned} \|N_k - N\| &= \|L_G^+V((I/k + A)^{-1} - A^{-1})V^TL_G^+\| \\ &\le \|L_G^+\|^2 \cdot \|V\| \cdot \|V^T\| \cdot \|(I/k + A)^{-1} - A^{-1}\| \qquad (4) \end{aligned}$$

By the Sherman-Morrison-Woodbury formula (Fact 1),

$$\begin{aligned}\|(I/k + A)^{-1} - A^{-1}\| &= \|A^{-1} - A^{-1}(I + A^{-1}/k)^{-1}A^{-1}/k - A^{-1}\| \\ &= \|A^{-1}(I + A^{-1}/k)^{-1}A^{-1}/k\| \\ &\leq \|A^{-1}\|^2 \cdot \|(I + A^{-1}/k)^{-1}\|/k \\ &\to \|A^{-1}\|^2\|I\|/k \qquad (5) \\ &\to 0 \qquad (6)\end{aligned}$$

where (5) follows from the fact that $I + A^{-1}/k \to I$ uniformly as $k \to \infty$, and the facts that matrix norm and matrix inverse are continuous functions for non-singular matrices. Hence, combining (4) and (6), $\|N_k - N\| \to 0$ as $k \to \infty$. The theorem then follows from the fact that the order of applying the update formulas does not matter and that applying the formula for F_1 and F_2 is the same as for $F_1 \cup F_2$.

A similar argument as Corollary 1 can show that the submatrix version of Theorem 3 holds as well. The only remaining detail is to establish that $V^T NV$ is non-singular. This follows by the same argument as Lemma 2 because the columns of V_{F_2} are not spanned by the columns of V_{F_1}, since $F_1 \cup F_2$ is a forest.

4.2 The Recursive Algorithm

We say an edge $\{u, v\}$ is in a submatrix if entries (u, v) and (v, u) are inside the submatrix. Corollary 1 and Corollary 2 say that if we have only made sampling decisions for edges in a submatrix, then we can update the submatrix of the Laplacian pseudoinverse with a small cost, using only the values of that submatrix. Algorithm 2 samples the edges in a matrix by diving the matrix into submatrices and recursively samples the edges in each submatrix. Whenever the algorithm returns from a recursive call to a submatrix, it updates the current matrix with the formulas given by Corollary 1 and Theorem 3 to ensure that the next submatrix it enters has been updated, which is enough for the algorithm to correctly sample the edges in that submatrix. Let us formally define the way we recurse on the edges.

Definition 6. *Let $G = (V_G, E_G)$ be an graph and S, R be disjoint sets of V_G. We define the following subsets of edges.*

$$\begin{aligned}E[S] &= \{\{u, v\} \in E_G : u, v \in S\} \\ E[R, S] &= \{\{u, v\} \in E_G : u \in R, v \in S\}\end{aligned}$$

Remark 3. Suppose that $R = R_1 \cup R_2$ and $S = S_1 \cup S_2$. Then

$$\begin{aligned}E[S] &= E[S_1] \cup E[S_2] \cup E[S_1, S_2] \\ E[R, S] &= E[R_1, S_1] \cup E[R_1, S_2] \cup E[R_2, S_1] \cup E[R_2, S_2]\end{aligned}$$

The formulas in Remark 3 give a recursive way to traverse the graph, visiting each edge exactly once. This is the approach adopted by Algorithm 2.

The algorithm samples the edges in $E[S]$ with SampleEdgesWithin(S), where we partition the current vertex set S into $S = S_1 \cup S_2$ and then recurse to visit edges in $E[S_1]$, $E[S_2]$ and $E[S_1, S_2]$, calling SampleEdgesWithin(S_1) and SampleEdgesWithin(S_2) respectively on $E[S_1]$, $E[S_2]$ and calling SampleEdgesCrossing(S_1, S_2) on $E[S_1, S_2]$. In SampleEdgesCrossing(S_1, S_2) We do a similar splitting and recursion. So, Algorithm 2 satisfies the property **P1** mentioned in Section 3.

Because Algorithm 2 does lazy updates, in order not to confuse with the true L_G^+, we denote the matrix that Algorithm 2 maintains by N. The way N is updated ensures that the following invariants are satisfied.

Invariant 1: SampleEdgesWithin(S) initially has $N_{S,S} = L^+_{G_{S,S}}$. The algorithm restores this property after each recursive call to the functions SampleEdgesWithin(S_i) and SampleEdgesCrossing(S_i, S_j).

Invariant 2: SampleEdgesCrossing(R, S) initially has $N_{R\cup S, R\cup S} = L^+_{G_{R\cup S, R\cup S}}$. The algorithm restores this property after each recursive call to the function SampleEdgesCrossing(R_i, S_j).

Since the two invariants guarantee that for any edge $\{r, s\}$, $N_{\{r,s\},\{r,s\}}$ is equal to $L^+_{G_{\{r,s\},\{r,s\}}}$ when we are deciding whether to keep the edge, the values of the effective resistances are correct for all edges. So, Algorithm 2 satisfies the properties **P2** and **P3**.

4.3 Analysis of Runtime

Let $f(n)$ and $g(n)$ respectively denote the runtime of SampleEdgesWithin(S) and SampleEdgesCrossing(R, S), where $n = |R| = |S|$. Updating N requires $O(|S|^\omega)$ time. Therefore, we have

$$f(n) = 2f(n/2) + g(n) + O(n^\omega)$$
$$g(n) = 4g(n/2) + O(n^\omega)$$

By standard theorems on recurrence relations, the solutions of these recurrences are $g(n) = O(n^\omega)$ and $f(n) = O(n^\omega)$. Thus, the runtime of Algorithm 2 is $O(n^\omega)$.

5 Conclusions

In this paper, we have shown a new algorithm for sampling random spanning trees, which is arguably simpler and cleaner than the algorithm of Colbourn, Myrvold and Neufeld (CMN)[9]. Our algorithm uses a similar framework as the algorithm for non-bipartite matching of Harvey [13]. Some open questions are whether the same type of framework can be applied to other graph-theoretic problems, and whether it is possible to bring this line of work and the recent results on the sparse graph case of random spanning trees generation closer together.

Algorithm 2 A Recursive Algorithm

function SampleSpanningTree($G = (V_G, E_G)$)
  $N \leftarrow L_G^+$
  SampleEdgesWithin(V_G)
  return the uniform spanning tree T
function SampleEdgesWithin(S)
  if $|S| = 1$ **then return**
  Divide S in half: $S = S_1 \cup S_2$
  for $i \in \{1, 2\}$ **do**
    SampleEdgesWithin(S_i)
    Restore N_{S_i,S_i} to its value before entering the recursion
    $F \leftarrow$ the set of edges contracted in SampleEdgesWithin(S_i)
    $D \leftarrow$ the set of edges deleted in SampleEdgesWithin(S_i)
    Update(S, F, D)
  SampleEdgesCrossing(S_1, S_2)
function SampleEdgesCrossing(R, S)
  if $|R| = 1$ **then**
    Let $r \in R$ and $s \in S$, $R^{\text{eff}} \leftarrow (\mathcal{X}_r - \mathcal{X}_s)^T N (\mathcal{X}_r - \mathcal{X}_s)$
    Flip a biased coin that turns head with probability R^{eff}
    if head **then**
      Add $e_{r,s}$ to the uniform spanning tree T and the set of contracted edges
    else
      Add $e_{r,s}$ to the set of deleted edges
  else
    Divide R and S each in half: $R = R_1 \cup R_2$ and $S = S_1 \cup S_2$
    for $i \in \{1, 2\}$ and $j \in \{1, 2\}$ **do**
      SampleEdgesCrossing(R_i, S_j)
      Restore $N_{R_i \cup S_j, R_i \cup S_j}$ to its value before entering the recursion
      $F \leftarrow$ the set of edges contracted in SampleEdgesCrossing(R_i, S_j)
      $D \leftarrow$ the set of edges deleted in SampleEdgesCrossing(R_i, S_j)
      Update($R \cup S, F, D$)
procedure Update(S, F, D)
  Let V be the incidence matrix for F
  Let L_D be the Laplacian matrix for D
  $N_{S,S} \leftarrow N_{S,S} - N_{S,S} V_{S,*} (V_{S,*}^T N_{S,S} V_{S,*})^{-1} V_{S,*}^T N_{S,S}$
  $N_{S,S} \leftarrow N_{S,S} - N_{S,S} (L_{D_{S,S}} N_{S,S} - I)^{-1} L_{D_{S,S}} N_{S,S}$

References

1. David Aldous. The random walk construction of uniform spanning trees and uniform labelled trees. *SIAM J. Discrete Math.*, 3:450–465, 1990.
2. Arash Asadpour, Michel Goemans, Aleksander Madry, Shayan Oveis Gharan, and Amin Saberi. An $O(\log n/\log\log n)$-approximation algorithm for the asymmetric traveling salesman problem. In *Proceedings of SODA*, 2010.
3. Andrei Broder. Generating random spanning trees. In *Proceedings of FOCS*, pages 442–447, 1989.
4. James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 1974.
5. Stephen L. Campbell and Carl D. Meyer. *Generalized Inverses of Linear Transformations*. SIAM, 1973.
6. Chandra Chekuri, Jan Vondrak, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *Proceedings of FOCS*, 2010.
7. C. J. Colbourn, B. M. Debroni, and W. J. Myrvold. Estimating the coefficients of the reliability polynomial. *Congr. Numer.*, 62:217–223, 1988.
8. Charles J. Colbourn, Robert P. J. Day, and Louis D. Nel. Unranking and ranking spanning trees of a graph. *Journal of Algorithms*, 10:271–286, 1989.
9. Charles J. Colbourn, Wendy J. Myrvold, and Eugene Neufeld. Two algorithms for unranking arborescences. *Journal of Algorithms*, 20:268–281, 1996.
10. Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *Proceedings of FOCS*, 2011.
11. Vempala Goyal, Rademacher. Expanders via random spanning trees. In *Proceedings of SODA*, 2009.
12. A. Guénoche. Random spanning tree. *Journal of Algorithms*, 4:214–220, 1983.
13. Nicholas J. A. Harvey. Algebraic algorithms for matching and matroid problems. *SIAM Journal on Computing*, 2009.
14. Nicholas J. A. Harvey and Neil Olver. Pipage rounding, pessimistic estimators and matrix concentration. In *Proceedings of SODA*, 2014.
15. Jonathan A. Kelner and Aleksander Madry. Faster generation of random spanning trees. In *Proceedings of FOCS*, 2009.
16. G. Kirchhoff. Über die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird. *Ann. Phys. und Chem.*, 72:497–508, 1847.
17. Ioannis Koutis, Gary L. Miller, and Richard Peng. A fast solver for a class of linear systems. *Communications of the ACM*, 55(10), 2012.
18. V. G. Kulkarni. Generating random combinatorial objects. *Journal of Algorithms*, 11(2):185–207, 1990.
19. Russell Lyons and Yuval Peres. *Probability on Trees and Networks*. Cambridge University Press. In preparation. Current version available at `http://pages.iu.edu/~rdlyons/`.
20. Aleksander Madry, Damian Straszak, and Jakub Tarnawski. Fast generation of random spanning trees and the effective resistance metric. In *Proceedings of SODA*, 2015.
21. David B. Wilson. Generating random spanning trees more quickly than the cover time. In *Proceedings of STOC*, 1996.