An Almost-Linear Time Algorithm for Uniform Random Spanning Tree Generation

Aaron Schild
University of California, Berkeley, USA
aschild@berkeley.edu

ABSTRACT
We give an $m^{1+o(1)}\beta^{o(1)}$-time algorithm for generating uniformly random spanning trees in weighted graphs with max-to-min weight ratio $\beta$. In the process, we illustrate how fundamental tradeoffs in graph partitioning can be overcome by eliminating vertices from a graph using Schur complements of the associated Laplacian matrix.

Our starting point is the Aldous-Broder algorithm, which samples a random spanning tree using a random walk. As in prior work, we use fast Laplacian linear system solvers to shortcut the random walk from a vertex $v$ to the boundary of a set of vertices assigned to $v$ called a "shortcutter." We depart from prior work by introducing a new way of employing Laplacian solvers to shortcut the walk. To bound the amount of shortcutting work, we show that most random walk steps occur far away from an unvisited vertex. We apply this observation by charging uses of a shortcutter $S$ to random walk steps in the Schur complement obtained by eliminating all vertices in $S$ that are not assigned to it.

CCS CONCEPTS
• Theory of computation → Graph algorithms analysis; Generating random combinatorial structures; Sketching and sampling; Random projections and metric embeddings; Computational geometry;

KEYWORDS
random spanning trees, Schur complements, random walks

ACM Reference Format:
Aaron Schild. 2018. An Almost-Linear Time Algorithm for Uniform Random Spanning Tree Generation. In Proceedings of 50th Annual ACM SIGACT Symposium on the Theory of Computing (STOC’18). ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3188745.3188852

1 INTRODUCTION
In this paper, we give the first almost-linear time algorithm for the following problem:
Given an undirected graph $G$ with weights (conductances) $c_e \in E(G)$ on its edges, generate a spanning tree $T$ of $G$ with probability proportional to $\prod_{e \in E(T)} c_e$.

Random spanning tree generation has been studied for a long time [Kir77], has many connections to probability theory (for example [BS13]), and is a special case of determinantal point processes [AGR16]. They also have found applications in constructing cut sparsifiers [FH10, GRV09] and have played crucial roles in obtaining better approximation algorithms for both the symmetric [GSS11] and asymmetric [AGM+10] traveling salesman problem.

The uniform random spanning tree distribution is also one of the simplest examples of a negatively-correlated probability distribution that is not trivial to sample from. Much work has gone into efficiently sampling from the uniform spanning tree distribution in the past forty years [Gue83]. This work falls into three categories:

• Approaches centered around fast exact computation of effective resistances [Gue83, CDN89, Kul90, CMN96, HX16]. The fastest algorithm among these takes $O(n^ω)$ time for undirected, weighted graphs [CDN89].
• Approaches that approximate and sparsify the input graph using Schur complements [DKP*17, DPPR17]. [DKP*17] samples a truly uniform tree in $O(n^{13}m^{1/2} + n^2)$ time, while [DPPR17] samples a random tree in $O(n^2δ^2)$ time from a distribution with total variation distance $δ$ from the real uniform distribution for undirected, weighted graphs.
• Random-walk based approaches [Ald90, Bro89, Wil96, KM09, MST15]. [MST15] takes $O(m^{6/3})$ time for undirected, unweighted graphs.

Our main result is an algorithm for sampling a uniformly random spanning tree from a weighted graph with polynomial ratio of maximum to minimum weight in almost-linear time:

**Theorem 1.1.** Given a graph $G$ with edge weights $c_e$ and $β = (\max_{e \in E(G)} c_e) / (\min_{e \in E(G)} c_e)$, a uniformly random spanning tree of $G$ can be sampled in $m^{1+o(1)}β^{o(1)}$ time.

We also give a result whose runtime does not depend on the edge weights, but samples from a distribution that is approximately uniform rather than exactly uniform. However, the runtime dependence on the error is small enough to achieve $1/poly(n)$ error in almost-linear time, so it suffices for all known applications:

**Theorem 1.2.** Given a weighted graph $G$ and $ε \in (0, 1)$, a random spanning tree $T$ of $G$ can be sampled from a distribution with total variation distance at most $ε$ from the uniform distribution in time $m^{1+o(1)}ε^{−o(1)}$ time.

Our techniques are based on random walks and are inspired by [Ald90, Bro89, KM09, MST15]. Despite this, our runtime guarantees combine the best aspects of all of the former approaches. In particular, our $m^{1+o(1)}ε^{−o(1)}$ time algorithm has no dependence...
on the edge weights, like the algorithms from the first two categories, but has subquadratic running time on sparse graphs, like the algorithms in the third category.

1.1 Other Contributions
We use random walks to generate random spanning trees. The behavior of random walks can be understood through the lens of electrical networks. We prove several new results about electrical flows and find new uses for many prior results (for example [SRS17, Ind06, HHHN’08]). We highlight one of our new results here.

One particularly important quantity for understanding random walks is the effective resistance between two vertices:

**Definition 1.3 (Effective resistance).** The energy of a flow \( f \in \mathbb{R}^{E(G)} \) in an undirected graph \( G \) with weights \( \{c_e\}_{e \in E(G)} \) is

\[
\sum_{e \in E(G)} f_e^2 c_e
\]

For two vertices \( s, t \) in a graph \( G \), the \( G \)-effective resistance between \( s \) and \( t \), denoted \( \text{Reff}_{G}(s,t) \), is the minimum energy of any \( s-t \) flow that sends one unit of flow from \( s \) to \( t \).

We study the robustness of the \( s-t \) effective resistance to random changes in the graph \( G \). Specifically, we consider random graphs \( H \sim G[F] \) obtained by conditioning on the intersection of a random spanning tree in \( G \) with the set \( F \subseteq E(G) \), which amounts to sampling a tree \( T \) from \( G \), contracting all edges in \( E(T) \cap F \), and deleting all edges in \( F \setminus E(T) \).

Surprisingly, the \( s-t \) effective resistance in \( H \) is the same as the \( s-t \) effective resistance in \( G \) in expectation as long as \( G \setminus F \) is connected. However, the \( s-t \) effective resistance in \( H \) does not concentrate around the effective resistance in \( G \). In particular, \( s \) and \( t \) could be identified to one another in \( G \), in which case the \( s-t \) effective resistance is 0. We show that changing a small number of contractions to deletions makes the effective resistance not much smaller than its original value:

**Lemma 1.4.** Let \( G \) be a graph, \( F \subseteq E(G) \), \( e \in \{0,1\} \), and \( s,t \in V(G) \). Sample a uniformly random spanning tree \( T \) of \( G \). Then, with high probability, there is a set \( F' \subseteq E(T) \cap F \) that depends on \( T \) and satisfies both of the following guarantees:

- **(Effective resistance)** Let \( H' \) be the graph obtained from \( G \) by contracting all edges in \( E(T) \cap F \) \( F' \) and deleting all edges in \( F' \cup (F \setminus E(T)) \). Then \( \text{Reff}_{H'}(s,t) \geq (1-\epsilon)\text{Reff}_{G}(s,t) \)

- **(Size)** \( |F'| \leq O((\log n)/\epsilon^2) \)

Even better, we show that \( F' \) can be computed in almost-linear time. We discuss this in the full version [Sch17]. Our algorithm uses a combination of matrix sketching [AMS96, Ind06] and localization [SRS17] that may be of independent interest.

2 ALGORITHM OVERVIEW
Our algorithm, like those of [KM09] and [MST15], is based on the following beautiful result of Aldous [Ald90] and Broder [Bro89]:

**Theorem 2.1 (Aldous-Broder).** Pick an arbitrary vertex \( u_0 \) and run a random walk starting at \( u_0 \) in a weighted graph \( G \). Let \( T \) be the set of edges used to visit each vertex besides \( u_0 \) for the first time. Then \( T \) is a weighted uniformly random spanning tree of \( G \).

The runtime of Aldous-Broder is the amount of time it takes to visit each vertex for the first time, otherwise known as the cover time of \( G \). On one hand, the cover time can be as high as \( \Theta mn \). On the other hand, Aldous-Broder has the convenient property that only a small number of vertex visits need to be stored. In particular, only \( n-1 \) visits to vertices add an edge to the sampled tree (the first visits to each vertex besides the starting vertex). This observation motivates the idea of shortcutting the random walk.

2.1 The Shortcutting Meta-Algorithm
To motivate our algorithm, we classify all existing algorithms based on Aldous-Broder [Bro89, Ald90, KM09, MST15] at a very high level. We start by describing Aldous-Broder in a way that is more readily generalizable:

**Aldous-Broder**

- For each \( v \in V(G) \), let \( s_v^{(0)} = \{v\} \).
- Pick an arbitrary vertex \( u_0 \) and set \( u \leftarrow u_0 \).
- Until all vertices in \( G \) have been visited
  - Sample the first edge that the random walk starting at \( u \) uses to exit \( s_u^{(i)} \)
  - Replace \( u \) with the non-\( s_u^{(i)} \) endpoint of this edge.
- Return all edges used to visit each vertex besides \( u_0 \) for the first time.

We now generalize the above algorithm to allow it to use some number \( \sigma_0 \) of shortcutters \( \{v\} \subseteq s_v^{(i)} \subseteq V(G) \) for each vertex \( v \in V(G) \). If the \( \{s_v^{(i)}\}_{i=1}^{\sigma_0} \)’s are chosen carefully, then instead of running the random walk until it exits \( s_v^{(i)} \), one can sample the exiting edge much faster using Laplacian solvers.

Ideally, we could shortcut the random walk directly to the next unvisited vertex in order to minimize the number of wasted visits. Unfortunately, we do not know how to do such shortcutting efficiently. Instead, we use multiple shortcutters per vertex. More shortcutters per vertex means a better approximation to the set of previously visited vertices, which leads to fewer unnecessary random walk steps and a better runtime.

**Simple shortcutting meta-algorithm**

- For each \( v \in V(G) \), let \( s_v^{(0)} = \{v\} \) and pick shortcutters \( \{s_v^{(i)}\}_{i=1}^{\sigma_0} \).
- Pick an arbitrary vertex \( u_0 \) and set \( u \leftarrow u_0 \).
- Until all vertices in \( G \) have been visited
  - Let \( \tau^* \in \{0,1,\ldots,\sigma_0\} \) be the maximum value of \( i \) for which all vertices in \( s_v^{(i)} \) have been visited
  - Sample the first edge that the random walk starting at \( u \) uses to exit \( s_v^{(\tau^*)} \)
  - Replace \( u \) with the non-\( s_v^{(\tau^*)} \) endpoint of this edge.
- Return all edges used to visit each vertex besides \( u_0 \) for the first time.
To implement the above meta-algorithm, one must make two important choices, each of which is bolded above. Both of these choices only affect the runtime of the meta-algorithm; not its correctness:

- A set of shortcutters for each vertex \( v \in V(G) \)
- A method for sampling the first exit edge from \( S_u^{(2)} \), which we call a shortcutting method

The meta-algorithm could also choose an arbitrary starting location \( u_0 \), but this choice is not important to any shortcutting-based algorithm.

We now argue that the meta-algorithm correctly samples a uniformly random spanning tree, no matter the choice of the \( S_u^{(i)} \)’s or shortcutting method. First of all, \( i = 0 \) is always a valid choice for \( i^* \), so \( i^* \) exists and the algorithm is well-defined. Since all vertices in \( S_u^{(i^*)} \) have been previously visited, using the shortcutter \( S_u^{(i^*)} \) does not skip any first visits. Therefore, by Theorem 2.1, the edges returned form a uniformly random spanning tree.

Next, we summarize all algorithms based on this meta-algorithm [Ald90, Bro89, KM09, MST15] with \( \sigma_0 \) — the number of shortcutters per vertex — and the choice of shortcutting method. We also list bounds on the runtimes of these algorithms on unweighted graphs to offer context.

Table 1: Shortcutting methods, number of shortcutters, and runtimes for prior algorithms

| Algorithm   | Shortcutting method | \( \sigma_0 \) | Runtime                        |
|-------------|---------------------|---------------|--------------------------------|
| [Ald90, Bro89] | Offline            | 0             | \( O(mn) \)                    |
| [KM09]      | Offline             | 1             | \( O(m \sqrt{n}) \)            |
| [MST15]     | Offline             | 2             | \( O(m^{7/3}) \)               |
| This paper  | Online              | \( \Theta \left( \frac{\log \log n}{\log \log \log n} \right) \) | \( m^{1+O\left( \frac{\log \log \log n}{\log \log \log \log n} \right)} \) |

We now briefly describe how the algorithms of [KM09] and [MST15] fit into this framework. In [KM09], the authors partition the graph into clusters using ball-growing [LR99] and use these clusters to shortcut the random walk. Specifically, their algorithm is exactly the same as our meta-algorithm with \( S_u^{(1)} \) set to be the cluster in the partition containing the vertex \( v \). In [MST15], the authors form \( O(\log n) \) families of disjoint clusters such that every vertex in \( G \) is in at least one cluster in some family. For each vertex \( v \) in \( G \), \( S_u^{(1)} \) is the largest cluster in any of the families containing \( v \). [MST15] starts off by partitioning the graph into clusters with diameter \( O(m^{7/3}) \) using ball-growing [LR99]. For each vertex \( v \) in \( G \), \( S_u^{(2)} \) is the cluster in this partition containing \( v \). [MST15] uses a variant of our meta-algorithm that partially samples a random spanning tree in stages rather than fully sampling all at once. We describe this variant in Section 2.5.

While we have not yet discussed what “Offline” and “Online” shortcutting are, we highlight that our shortcutting method is different from that of [KM09] and [MST15]. This is one of the key reasons why we are able to obtain a faster algorithm and are able to effectively use more shortcutters per vertex.

2.2 Shortcutting Methods

The starting point for our improvement is an online shortcutting method. This method is based on the following observation:

**Key Idea 2.2 (Online shortcutting).** For a vertex \( u \) and a shortcutter \( S_u \) associated with \( u \), the probability that a random walk exits \( S_u \) through an edge \( e \) can be \( \epsilon \)-additively approximated for all \( e \in \partial S_u \) simultaneously using one \( \epsilon \)-approximate Laplacian system solve on a graph with \( |E(S_u) \cup \partial S_u| \) edges.

When combined with a trick due to Propp [Pro], one can exactly sample an escape edge in expected \( \tilde{O}(|E(S_u) \cup \partial S_u|) \) time with no preprocessing.

We call this method online due to its lack of preprocessing and the shortcutting technique used in [KM09] and [MST15] offline due to its fast query time with high preprocessing time. We summarize the runtime properties of these shortcutting methods for a shortcutter \( S_u \) here:

Table 2: Shortcutting methods and their runtimes

| Shortcutting method | Preprocessing | Query                  |
|---------------------|---------------|------------------------|
| Online              | None          | \( O(|E(S_u) \cup \partial S_u|) \) |
| Offline             | \( O(m^{7/3}) \) | \( O(1) \)              |

The upside to the online method is that its runtime does not depend quadratically on the size of \( S_u \)’s boundary. This is a critical barrier to improving the technique of [KM09] and [MST15] because it is impossible to obtain balanced cuts with arbitrarily small size that separate a graph into low-radius parts in most metrics. While the high query time for the online method may seem prohibitive initially, it is fine as long as online shortcutting does substantially less work than the random walk would have done to reach the boundary of \( S_u \). In the following path example, it takes the random walk \( \Theta(k^2) \) time for the random walk to escape \( S_u \) starting at \( u \), but online shortcutting only takes \( \tilde{O}(k) \) time to find the escape edge:

![Figure 1: How online shortcutting saves over the random walk.](image)

2.3 Properties of Shortcutters

Now, we describe the machinery that allows us to bound the total amount of work. We start with a bound that captures the idea that
most random walk steps happen far away from an unvisited vertex. This bound is a weighted generalization of Lemma A.4 given in [MST15]. We prove it in the full version:

**Lemma 2.3** (Key result for bounding the number of shortcutter uses). Consider an arbitrary vertex \( u_0 \) in a graph \( I \), an edge \( (u,v) = f \in E(I) \), and an \( R \geq 0 \). Let \( B(u,R) \subseteq V(I) \) denote the set of vertices in \( I \) with \( I \)-effective resistance distance at most \( R \) from \( u \). The expected number of times that the random walk starting at \( u_0 \) traverses \( f \) from \( u \to v \) before all vertices in \( B(u,R) \) have been visited is at most \( O(\epsilon f R) \), where \( \epsilon f \) is the conductance of the edge \( f \).

The effective resistance metric\(^1\) appears in the above lemma due to the relationship between random walks and electrical networks. Unlike recent work on sampling random spanning trees ([KM09, MST15]), we apply this lemma in the original graph and in other graphs obtained by “eliminating” vertices from the original graph. Specifically, we apply Lemma 2.3 to Schur complements of the input graph \( G \). We use the following combinatorial, folklore fact about Schur complements to employ Schur complements as an analysis tool:

**Theorem 2.4.** Consider a graph \( I \) and some set of vertices \( S \subseteq V(I) \). Let \( J \) be Schur\((I,S)\). Pick a vertex \( v \in S \) and generate two sequences of vertices as follows:

- Do a random walk in \( J \) starting at \( v \) and write down the sequence of visited vertices.
- Do a random walk in \( I \) starting at \( v \) and write down the sequence of visited vertices that are also in \( S \).

These two distributions over sequences are identical.

For a shortcutter \( S_u \), consider the graph \( H = \text{Schur}(G,(V(G) \setminus S_u) \cup \{u\}) \). Each use of the shortcutter \( S_u \) can be charged to crossing of at least one edge incident with \( u \) in the random walk on \( H \) by Theorem 2.4. Therefore, to bound the number of times a shortcutter \( S_u \) is used over the course of the algorithm, it suffices to bound the total conductance of edges between \( u \) and \( V(H) \setminus \{u\} \) in \( H \). This motivates one of the key properties of shortcutters, which we call conductivity in later sections:

**Key Idea 2.5** (Shortcutter Schur complement conductance bound). Let \( S_C \) denote a shortcutter for which \( S^{(i)}_u = S_C \) for all \( u \in C \). Then “on average,” the total conductance of the edges between \( C \) and \( V(G) \setminus S_C \), in the graph \( \text{Schur}(G,C \cup (V(G) \setminus S_C)) \), choose the Schur complement conductance of \( S_C \), is at most \( m^\sigma(1) a/(\alpha^{(i)}(m^\sigma(1)) r_{\text{min}}) \). \( \alpha \) is the ratio between the maximum and minimum effective resistance distance between any two vertices in \( G \) and \( r_{\text{min}} \) is the minimum effective resistance distance between any two vertices in \( G \).

By “on average,” we mean that the shortcutters \( S_C \) are organized into \( m^\sigma(1) \) sets and within each set, the total Schur complement conductance of the shortcutters \( S_C \subseteq C \) is at most \( m^\sigma(1) |C| a/(\alpha^{(i)}(m^\sigma(1)) r_{\text{min}}) \). For the rest of this section, we think of each shortcutter as having Schur complement conductance at most \( m^\sigma(1) a/(\alpha^{(i)}(m^\sigma(1)) r_{\text{min}}) \) in order to simplify the description. For a more formal description of the organization of our shortcutters, see Section 4.

\(^1\)The effective resistance \( \text{R}_{f}^R \) satisfies the triangle inequality and therefore forms a metric space on the vertices of \( G \)

To bound the number of times \( S_C \) is used, Lemma 2.3 requires two things:

**Sufficient properties for bounding shortcutter uses**

1. A bound on the Schur complement conductance of \( S_C \)
2. A bound on the effective resistance distance to the nearest unvisited vertex outside of \( S_C \)

The Schur complement conductance is at most \( m^\sigma(1) a/(\alpha^{(i)}(m^\sigma(1)) r_{\text{min}}) \) by conductivity. Therefore, we just need to bound the distance to the nearest unvisited vertex if there was an unvisited vertex within effective distance \( \alpha^{(i+1)}(m^\sigma(1)) r_{\text{min}} m^\sigma(1) \) of \( C \), Lemma 2.3 would imply that \( S_C \) is only used \( \left( \alpha^{(i+1)}(m^\sigma(1)) r_{\text{min}} m^\sigma(1) \right) \) times over the course of the shortcutter random walk. To bound the total work done, the following fact suffices:

**Key Idea 2.6** (Shortcutter overlap). Each vertex in \( G \) is in at most \( m^\sigma(1) \) shortcutters.

The above idea implies that the total size of all shortcutters is \( O(m^\sigma(1) \alpha^{(i)}(m^\sigma(1))) \). To use a shortcutter, we apply the online shortcutting method, which takes time proportional to the shortcutter’s size (see Table 2). If each shortcutter is used at most \( m^\sigma(1) a/(\alpha^{(i)}(m^\sigma(1)) \) times as described above, the total work due to all shortcutter uses is \( m^\sigma(1) \alpha^{(i)}(m^\sigma(1)) \leq m^\sigma(1) a/(\alpha^{(i)}(m^\sigma(1)) \) as desired.

Therefore, if we can obtain shortcutters with bounds on (1) and (2) that also respect Key Idea 2.6, we would have an almost-linear time algorithm for sampling random spanning trees on weighted graphs.

**2.4 Obtaining Shortcutters with Property (1) and Small Overlap**

We have not yet discussed how to actually obtain shortcutters with the desired conductance property. We discuss this in detail in the full version, but give a summary here for interested readers. We construct \( \{S^{(i)}_v\}_{v \in V(G)} \) for each \( i \) independently by

- constructing a small number of families of sets that are each well-separated in the effective resistance metric, have distance separation roughly \( \alpha^{(i)}(m^\sigma(1)) r_{\text{min}} \), and together cover the graph. These are the cores and the construction of these cores is similar to constructions of sparse covers of metric spaces (for example [AP90]).
- making the shortcutter around each core \( C \) be the set of vertices \( S_C \subseteq V(G) \) for which a random walk starting at \( x \in S_C \) is more likely to hit \( C \) before any other core in its family. The sparsity of the cover ensures that the shortcutters satisfy Key Idea 2.6, while their well-separatedness ensures Property (1).
2.5 Obtaining Property (2) using Partial Sampling and Carving

We now show that when $S_u^{(i)}$ is used, there is an unvisited vertex with effective resistance distance at most $m^{|E(G[S])|/r_{min}}$ from $u$. Review the shortcutting meta-algorithm. When $S_u^{(i)}$ is used, there is some vertex $v \in S_u^{(i+1)}$ that the random walk has not yet visited. $v$, however, may not be close to $u$. This motivates the following property of a shortcutter $S_u^{(i)}$, which we call being carved with respect to (a set of vertices) $S$:

**Key Idea 2.7 (Carving).** A shortcutter $S_u^{(i)}$ is carved with respect to $S \subseteq V(G)$ if $S_u^{(i)} \cap S$ only consists of vertices that are within effective resistance distance $m^{|E(G[S])|/r_{min}}$ of $u$.

If $S_u^{(i+1)}$ is carved with respect to $V(G)$, then the unvisited vertex $v$ is within distance $m^{|E(G[S])|/r_{min}}$ of $u$. As a result, there is an unvisited vertex within distance $m^{|E(G[S])|/r_{min}}$ of $u$, as desired.

It is difficult to directly build shortcutters to make them carved with respect to some set $S$. Instead, we remove vertices from shortcutters so that all shortcutters for all vertices are carved with respect to $V(G)$. To carve $V(G)$ out of all shortcutters $S_u^{(i)}$, one could just remove all vertices in $S_u^{(i)}$ that are farther than effective resistance distance $m^{|E(G[S])|/r_{min}}$ away from $u$. Unfortunately, this could remove almost all of the shortcutter in general.

Instead, we compute a partial sample of a random spanning tree in order to make it so that each shortcutter does not have to be carved with respect to as many vertices. Specifically, we modify the simple shortcutting meta-algorithm as follows:

**Full shortcutting meta-algorithm (one round of partial sampling)**

- **Choose** a set $S \subseteq V(G)$ for partial sampling.
- For each $v \in V(G)$, let $S_v^{(i)} = \{v\}$ and **pick** shortcutters $S_u^{(i)}$, $\sigma_0$.
- Pick an arbitrary vertex $u_0$ and set $u \leftarrow u_0$.
- Until all vertices in $S$ have been visited:
  - Let $i^* \in \{0, 1, \ldots, \sigma_0\}$ be the maximum value of $i$ for which all vertices in $S \cap S_u^{(i)}$ have been visited.
  - **Sample** the first edge that the random walk starting at $u$ uses to exit $S_u^{(i)}$.
  - Replace $u$ with the non-$S_u^{(i)}$ endpoint of this edge.
- Let $T'$ be all edges used to visit each vertex besides $u_0$ in $S$ for the first time that are in the induced subgraph $F := E(G[S])$.

**Condition** on the partial sample, which amounts to contracting all edges in $E(T') \cap F$ and deleting all edges of $F \setminus E(T')$ in $G$.

[MST15] also exploited partial sampling in this way. This algorithm correctly samples the intersection of a random spanning tree of $G$ with $E(G[S])$ because it does not skip any of the first visits to vertices in $S$ and only vertices in $S$ need to be visited in order to determine the edges in $G[S]$ that are in the sample. While this algorithm no longer samples the entirety of a tree, we only need shortcutters to be carved with respect to $S$ rather than all of $V(G)$ in order to show that the total work is $m^{1+o(1)}\sigma_0$.

Our algorithm and [MST15] exploit the full meta-algorithm in multiple rounds. During each round, we pick a set $S$ to condition on, run the meta-algorithm, and repeat until $G$ is a single vertex. At this point, we have sampled a complete spanning tree of $G$.

We want to choose $S$ to be small enough so that every shortcutter can be carved with respect to $S$ without increasing the Schur complement conductance of those shortcutters too much. As long as all shortcutters are carved with respect to $S$, the meta-algorithm takes $m^{1+o(1)}\sigma_0$ time. However, we also want $S$ to be large enough to make substantial progress.

When $\sigma_0 = 1$, let $S$ be the set of vertices $u$ assigned to the largest shortcutters. By Key Idea 2.6, there cannot be too many large shortcutters, which means that $S$ is the union of a small number of clusters with small effective resistance diameter ($\sqrt{r_{min}}$). Deleting each cluster from each shortcutter $S_C$ with a far-away core $C$ makes $S_C$ carved with respect to $S$. Furthermore, because the cluster was well-separated from $C$, its deletion did not increase the Schur complement conductance of $S_C$ much.

2.6 Bounding the Number of Rounds of Partial Sampling when $\sigma_0 = 1$

In the previous section for the $\sigma_0 = 1$ case, we saw that conditioning on the induced subgraph of the vertices assigned to the largest shortcutters was a good idea for carving. We now show that computing a partial sample for the induced subgraph of these vertices allows us to make substantial progress. Ideally, one could show that conditioning on the induced subgraph of vertices assigned to the largest shortcutters decreases the size of the graph by a constant fraction. Unfortunately, we do not know how to establish this in general.

To get around this, we do not rebuild shortcutters from scratch after each partial sampling round. Instead, we show that it is possible to make shortcutters $S_u^{(i)}$ that are contained within the shortcutter $S_u^{(i)}$ from the previous round. It is quite tricky to do this directly, as conditioning on a partial sample can change the metric structure of the graph $G$. In particular, the conductance of a shortcutter could dramatically increase after conditioning.

To cope with this, we show a concentration inequality that promises the existence of a small set of edges with high probability that, when deleted, restore the conductance of all shortcutters back to their value before conditioning. This result follows from a nontrivial generalization of Lemma 1.4 that we discuss in the full paper.

Given that $S_u^{(i)}$ is contained in $S_u^{(i)}$ for all $u \in V(G)$ and all $i \in \{\sigma_0\}$, conditioning on the vertices with the near-largest shortcutters decreases the maximum size of a remaining shortcutter by a large factor. Therefore, after $O(\log n)$ rounds of conditioning on the induced subgraph of the vertices assigned to the largest shortcutters, no shortcutters are left. At this point, the algorithm is done.
Each round takes \(m^{1+o(1)} \sqrt{d} \) time in the \( \sigma_0 = 1 \) case for a total of \( \widetilde{O}(m^{1+o(1)} \sqrt{d}) \) runtime.

### 2.7 Carving and Progress when \( \sigma_0 > 1 \)

The bottleneck in the algorithm for the \( \sigma_0 = 1 \) case is the sampling step. As discussed in Section 2.1, using more shortcutters allows us to approximate the set of previously visited vertices better, leading to a better runtime. In particular, the runtime-bounding argument presented earlier, given a carving and conductance bound, shows that using \( \sigma_0 \) shortcutters yields an \( m^{1+o(1)} \alpha^{1/(\sigma_0+1)} \)-runtime algorithm for sampling a spanning tree.

Unfortunately, carving shortcutters is more complicated when \( \sigma_0 > 1 \). We need to pick a relatively large set of vertices that can be carved out of all shortcutters for all vertices simultaneously. To do this, one could start by trying to generalize the strategy in the \( \sigma_0 = 1 \) case through repetition. Specifically, one could try the following strategy for picking a set \( S \) for use in one round of the meta-algorithm with partial sampling:

**First attempt at conditioning when \( \sigma_0 > 1 \)**

- \( S_{\sigma_0+1} \leftarrow V(G) \)
- For \( i = \sigma_0, \sigma_0 - 1, \ldots, 1 \):
  - \( S_i \leftarrow \) the vertices \( u \in S_{i+1} \) with near-maximum size \( u \) shortcutters; that is within a factor of \( m^{1/\sigma_i} \) of the maximum.
- Let \( S \leftarrow S_1 \)

This strategy has some benefits. If \( S_1 \) could be carved out of all shortcutters, the maximum size of \( S_u^{(1)} \) shortcutters for vertices \( v \in S_2 \) would decrease by a factor of \( m^{1/\sigma_1} \).

Before moving on, we elaborate on how conditioning on the induced subgraph of vertices assigned to a shortcutter renders it unnecessary in the future. Start by refining all cores of all \( S_u^{(i)} \) shortcutters to obtain \( \sigma_0 \) partitions \( \{\mathcal{P}_i\}_{i=0}^{\sigma_0} \) of \( V(G) \), with one for each \( i \in [\sigma_0] \). Standard ball-growing (for example [LR99]) ensures that the total conductance of all boundary edges of parts in \( \mathcal{P}_i \) is at most \( m^{1+o(1)}(\alpha^{1/(\sigma_0+1)} r_{\min}) \). Conditioning on the induced subgraph of a part \( P \) deletes or contracts all edges in the induced subgraph of \( P \), only leaving \( P \)'s boundary. Since \( P \)'s boundary is small, the random walk never needs to use \( P \)'s shortcutter again because the total number of steps across \( \mathcal{P}_i \) boundary edges is at most

\[
\frac{m^{1+o(1)} \alpha^{i+1)/(\sigma_0+1)} r_{\min}}{\alpha^{i+1}/(\sigma_0+1)} r_{\min} \leq m^{1+o(1)} \alpha^{o(1)}
\]

where the \( \alpha^{i+1}/(\sigma_0+1) r_{\min} \) bound follows from carving. Therefore, conditioning on a part replaces it with its boundary, thus rendering its shortcutter unnecessary.

Now, we go back to analyzing our first attempt at a \( \sigma_0 > 1 \) algorithm for selecting \( S \). If we could always carve all shortcutters with respect to \( S_1 \), conditioning \( \sigma_1 \) times on various \( S_i \)'s would make all shortcutters for \( \mathcal{P}_1 \) parts intersecting \( S_2 \) irrelevant, thus making it possible to condition on \( S_2 \) directly. More generally, if carving were not an issue, every \( \sigma_1 \) rounds of conditioning on \( S_1 \) would pave the way for one round of conditioning on \( S_{i+1} \). Combining this reasoning for all \( i \) implies that we have sampled the entire tree after \( \sigma_1 \) applications of the meta-algorithm.

Unfortunately, our first attempt does not produce carvable sets \( S \) in general because there could be a very large shortcutter with core just outside of some \( S_i \) that happens to intersect many of the vertices in \( S_j \). To cope with this, we incorporate a ball-growing type approach that switches to conditioning on this very large but nearby shortcutter if one exists. Once this procedure stops, one can carve the parts assigned to the selected shortcutters out of all other shortcutters because the selected shortcutters are larger than all other shortcutters that the selected parts intersect.

### 2.8 Coping with the Fixing Lemma in the Shortcutting Method

In Section 2.6, we established that we could obtain containment of shortcutters in past shortcutters if we deleted a small set of "fixing edges" from the graph. However, we cannot actually delete these edges from the graph, as we must do partial sampling in graphs resulting directly from conditioning in order to correctly sample a uniformly random spanning tree.

Instead of deleting these fixing edges from the graph, we just remove their endpoints from the shortcutters and use offline shortcutting [MST15, KM09] to make it so that shortcutting to the endpoints of these fixing edges only takes constant time rather than time proportional to the size of the shortcutter. Since there are a small number of fixing edges, the preprocessing time for offline shortcutting is small. Each shortcut to the endpoints of a removed edge takes \( \tilde{O}(1) \) time and can be charged to crossing a boundary edge of some core. Constructing the cores using standard ball-growing makes the total conductance of these boundary edges small, so Lemma 2.3 can be used to show that the number of such shortcutting steps is small.

While this completes the high-level description of our algorithm, it does not describe all of the contributions of this paper. Along the way, we prove many new results about effective resistance metrics that may be of independent interest.

### 3 Preliminaries

#### 3.1 Graphs

For a (directed or undirected) graph \( G \), let \( V(G) \) and \( E(G) \) denote its vertex and edge set respectively. \( n \) and \( m \) refer to the number of vertices and edges respectively of the input graph to our random spanning tree generation algorithm. For a set of edges \( F \subseteq E(G) \), \( G \setminus F \) denotes the graph obtained by deleting the edges in \( F \) from \( G \). For a set of vertices \( S \subseteq V(G) \), let \( G/S \) be the graph obtained by identifying all vertices in \( S \) to a single vertex; that is

\[
V(G/S) := \{V(G) \setminus S\} \cup \{s\}
\]

and each endpoint of an edge \( e \in E(G) \) that is also in \( S \) is replaced with \( s \). For a set of edges \( F \subseteq E(G) \), let \( V(F) \) denote the set of endpoints of edges in \( F \). For an edge \( f \in E(G) \), let \( G \setminus f \) and \( G/f \) denote the graph obtained by deleting and contracting \( f \) respectively.

For two sets of vertices \( S, S' \in V(G) \), let \( E_G(S, S') \) denote the set of edges with one endpoint in \( S \) and the other endpoint in \( S' \).
Let $G[S] := E_G(S) := E_G(S,S)$. When the graph $G$ is clear from context, we omit it from the subscript. For a graph $G$ with two sets $X, Y \subseteq V(G)$, let $G(X, Y)$ denote the graph obtained by identifying all vertices in $X$ to one vertex $x$ and all vertices in $Y$ to one vertex $y$.

For a set of vertices $S \subseteq V(G)$, let $\partial_G S := E_G(S, V(G) \setminus S)$ denote the boundary edges of $S$. For a singleton set $S = \{w\}$, $\partial_G S$ is abbreviated $\partial_G w$.

In this paper, graphs are sometimes weighted with conductances $(c_e)_{e \in E(G)}$. For a set $F \subseteq E(G)$, let $c_G^F := \sum_{e \in F} c_e^2$. Let $r^2_e = 1/c^2_e$. Let $\beta^G := (\max_{e \in E(G)} r^G_e) / (\min_{e \in E(G)} r^G_e)$. When the context is clear, the graph $G$ is omitted from all superscripts in the aforementioned definitions.

### 3.2 Laplacian Matrices, Electrical Flows, and Effective Resistances

For an undirected graph $G$ with two vertices $s, t \in V(G)$, let $b_{st} \in \mathbb{R}^{V(G)}$ denote the vector with $b_{st} = 1$ at $s$, $b_{st} = -1$ at $t$, and $b_{st} = 0$ for $v \not= s, t$. Direct all of the edges of $G$ arbitrarily. Suppose that $e = (a, b)$ and is directed from $a$ to $b$. Define $b_e := b_{ab}$. Define the Laplacian matrix of a weighted graph $G$ as

$$L_G := \sum_{e \in E(G)} c_e b_e b_e^T$$

This definition is invariant of the orientations of the edges. $L_G$ has nontrivial kernel, but still has a Moore-Penrose pseudoinverse $L^+_G$. The vector $L^+_G b_{st}$ is a vector of potentials for the electrical flow $C_G b_{st} L^+_G b_{st}$, where $C_G$ is the $|E(G)| \times |V(G)|$ matrix with rows equal to the vectors $b_e$ for $e \in E$ and $C_G$ is the $|E(G)| \times |E(G)|$ diagonal matrix of edge conductances.

The effective resistance between two vertices $s, t \in V(G)$ is the energy of the electrical flow from $s$ to $t$, which equivalently is

$$\text{Reff}_{G}(s, t) := b_{st}^T L^+_G b_{st}$$

For an edge $e = (a, b) \in E(G)$, let $\text{Reff}_{G}(e) := \text{Reff}_{G}(a, b)$. We use the following folklore fact about effective resistances extensively without reference:

**Remark 3.1.** The vertex set of $V(G)$ is a metric space with respect to the metric $\text{Reff}_{G}$. In particular, $\text{Reff}_{G}$ satisfies the triangle inequality, i.e.

$$\text{Reff}_{G}(s, t) \leq \text{Reff}_{G}(s, w) + \text{Reff}_{G}(w, t)$$

for any three vertices $s, t, w \in V(G)$.

For a set $S \subseteq V(G)$, define its effective resistance diameter to be

$$\max_{u, v \in S} \text{Reff}_{G}(u, v)$$

Often, for clarity, we call this the G-effective resistance diameter of $S$. Let $r_{\min} := \min_{u, v \in V(G)} \text{Reff}_{G}(u, v)$, $r_{\max} := \max_{u, v \in V(G)} \text{Reff}_{G}(u, v)$, and $\alpha = r_{\max} / r_{\min}$. Notice that $\beta \leq \alpha \leq m^2 \beta$. Therefore, to obtain an $m^2 \beta(1 + \log(1/\alpha))$-time algorithm, it suffices to obtain an $m^2(1 + \log(1/\alpha))$-time algorithm.

### 3.3 Laplacian Solvers

In this paper, we make extensive use of efficient approximate Laplacian solvers [ST14, KMP14, KOSZ13, CKM+14, PS14, LPS15, KS16]:

**Theorem 3.2 ([CKM+14]).** There is an $O(m \sqrt{\log n \log(np) / \varepsilon})$ time algorithm, that, given a demand vector $d \in \mathbb{R}^{V(G)}$ for some graph $G$ with edge weights between 1 and $\beta$, computes a vector $p \in \mathbb{R}^{V(G)}$ such that

$$||p - L^+_G d||_\infty \leq \varepsilon$$

with high probability.

### 3.4 Random Walks

For a weighted graph $G$ and some vertex $v \in V(G)$, let $\text{Pr}_{G}[v]$ denote the probability of an event $E$ over random walks starting at $v$ in the graph $G$. When the graph is clear from context, we denote this by $\text{Pr}[v]$. For a set of vertices $S \subseteq V(G)$, let $t_S$ be the random variable denoting the hitting time to the set $S$. When $S$ is a singleton $(b)$, we abbreviate $t_S$ as $t_b$.

We use the following fact about random walks extensively:

**Theorem 3.3 (Proposition 2.2 of [LP16]).** Let $G$ be a graph with conductances $(c_e)_e$. Consider two vertices $s, t \in V(G)$. For a vertex $u$, let $p_u = b_{st}^T L^+_G b_{st}$. Consider an edge $e = (u, v)$. Then

$$p_u c_e = E_s[\text{number of times } e \text{ is crossed from } u \rightarrow v \text{ before } t_t]$$

### 3.5 Basic Facts about Random Spanning Trees

Let $T \sim G$ denote the distribution over spanning trees of $G$ with each tree selected with probability proportional to $\prod_{e \in E(T)} c_e$. The following shows that conditioning on a partial sample is equivalent to modifying the input graph:

**Theorem 3.4 ([MST15]).** Consider a graph $G$ and a set of edges $F \subseteq E(G)$. Fix a spanning tree $T_0$ of $G$ and let $F_0 := E(T_0) \cap F$. Obtain a graph $H$ of $G$ by contracting all edges in $F_0$ and deleting all edges in $F \setminus F_0$. Then

$$\text{Pr}_{T \sim G}[T = T_0 | E(T) \cap F = F_0] = \frac{\text{Pr}_{T \sim H}[T' = T_0 \cap F_0]}{\text{Pr}_{T \sim H}[T' = T_0 \cap F_0]}$$

For any set $F \subseteq E(G)$, let $H \sim G[F]$ denote the distribution over minors $H$ of $G$ obtained by sampling a tree $T \sim G$, contracting all edges in $F \cap E(T)$, and deleting all edges in $F \setminus E(T)$. We also use the following folklore fact extensively:

**Theorem 3.5 ([Kir47]).** Consider a graph $G$ and an edge $e \in E(G)$. Then

$$\text{Pr}_{T \sim G}[e \in E(T)] = c_e \text{Reff}_{G}(e)$$

### 3.6 Schur Complements

**Definition 3.6 (Schur complements).** The Schur complement of a graph $I$ with respect to a subset of its vertices $S \subseteq V(I)$, denoted $\text{Schur}(I, S)$, is the weighted graph $I$ with $V(I) = S$ with Laplacian matrix

$$L_I = L_I[S, S] - L_I[S, S^c] L_I[S^c, S^c]^{-1} L_I[S^c, S]$$

220
where \(M[S_0,S_1]\) denotes the submatrix of a matrix \(M\) with rows and columns indexed by \(S_0\) and \(S_1\) respectively.

In the above definition, it is not immediately clear that \(L_J\) is the Laplacian matrix of a graph, but it turns out to be one. Furthermore, the following associativity property holds:

**Remark 3.7.** For any two disjoint sets of vertices \(S_0, S_1 \in V(I)\) for some graph \(I\),

\[
\text{Schur}(\text{Schur}(I, S_0 \cup S_1), S_0) = \text{Schur}(I, S_0)
\]

Also, Schur complements commute with edge deletions and contractions in the kept set \(S\):

**Remark 3.8 (Lemma 4.1 of [CDN89]).** Let \(S\) be a set of vertices in a graph \(G\) and \(f \in E_G(S)\). Then,

\[
\text{Schur}(G \setminus f, S) = \text{Schur}(G, S) \setminus f
\]

and

\[
\text{Schur}(G/f, S) = \text{Schur}(G, S)/f
\]

Schur complements also have the following combinatorial property, which is crucial to bounding the number of times shortcutters are used:

**Theorem 2.4.** Consider a graph \(I\) and some set of vertices \(S \subseteq V(I)\). Let \(J = \text{Schur}(I, S)\). Pick a vertex \(v \in S\) and generate two sequences of vertices as follows:

- Do a random walk in \(J\) starting at \(v\) and write down the sequence of visited vertices.
- Do a random walk in \(I\) starting at \(v\) and write down the sequence of visited vertices that are also in \(S\).

These two distributions over sequences are identical.

While this result is likely known, we include a proof for completeness. To prove this, we use the following folklore fact:

**Remark 3.9 (Chapter IX of [Bol13]).** Let \(S \subseteq V(G)\) be a set of vertices in a graph \(G\) and let \(v \in S, x \in V(G)\). Let \(H = G(S \setminus \{v\})\) and let \(s\) be the identification of the set \(S \setminus \{v\}\) in \(H\). Then

\[
\text{Pr}[t_x < t_S|v] = \frac{b_{c,s}^T L_{c,s} b_x}{b_{c,s}^T L_{c,s} b_v}
\]

**Proof of Theorem 2.4.** Let \(\{s_j\}_{j \geq 0}\) be the list of vertices visited by a random walk in \(J\) starting at \(v\). Let \(\{s'_j\}_{j \geq 0}\) be the list of vertices visited by a random walk in \(I\) starting at \(v\) outside of \(S\) omitted. To prove Theorem 2.4, it suffices to show that for any sequence \(\{t_j\}_{j \geq 0}\) of vertices in \(S\) and any \(j \geq 0\),

\[
\text{Pr}[s_j = t_j | (s_i)_{i < j} ] = \text{Pr}[s'_j = t_j | (s'_i)_{i < j} ]
\]

By the Markov property,

\[
\text{Pr}[s_j = t_j | s_i = t_i | (s_i)_{i < j} ] = \text{Pr}[s'_j = t_j | (s'_i)_{i < j} ]
\]

and

\[
\text{Pr}[s'_j = t_j | s_i = t_i | (s'_i)_{i < j} ] = \text{Pr}[s'_j = t_j | (s'_i)_{i < j} ]
\]

Let \(J' = J/(S \setminus \{t_j, t_{j+1}\})\) and \(J'' = I/(S \setminus \{t_j, t_{j+1}\})\). In both graphs, let \(s\) be the identification of \(S \setminus \{t_j, t_{j+1}\}\). By Remark 3.9 applied with \(S' \hookrightarrow S \setminus \{t_j\}\) and \(v \leftarrow t_{j+1}\),

\[
\text{Pr}[t_j < t_s | (t_i)_{i < j} ] = \frac{b_{i,j}^T L_{i,j} b_{t_{j+1}}}{b_{i,j}^T L_{i,j} b_{t_j}}
\]

and

\[
\text{Pr}[t_{j+1} < t_{s'} | (t'_{i})_{i < j+1} ] = \frac{b_{j+1,i}^T L_{j+1,i} b_{t_{j+2}}}{b_{j+1,i}^T L_{j+1,i} b_{t_{j+1}}}
\]

Since Schur complements preserve quadratic forms supported on the kept set,

\[
\frac{b_{i,j}^T L_{i,j} b_{t_{j+1}}}{b_{i,j}^T L_{i,j} b_{t_j}} = \frac{b_{j+1,i}^T L_{j+1,i} b_{t_{j+2}}}{b_{j+1,i}^T L_{j+1,i} b_{t_{j+1}}}
\]

By Remark 3.8,

\[
\text{Schur}(J', \{s, t_j, t_{j+1}\}) = \text{Schur}(J/(S \setminus \{t_j, t_{j+1}\}), \{s, t_j, t_{j+1}\})
\]

\[
= \text{Schur}(I/(S \setminus \{t_j, t_{j+1}\}))
\]

\[
= J'
\]

Therefore,

\[
\frac{b_{j+1,i}^T L_{j+1,i} b_{t_{j+2}}}{b_{j+1,i}^T L_{j+1,i} b_{t_{j+1}}} = \frac{b_{i,j}^T L_{i,j} b_{t_{j+1}}}{b_{i,j}^T L_{i,j} b_{t_j}}
\]

as desired. \(\square\)

4 KEY LEMMAS

Now, we formally introduce the concepts that were alluded to in Section 2. In the process, we outline the structure of the paper and prove the main result (Theorem 1.1) given the four main components of our algorithm: building shortcutters, selecting vertices to condition on, sampling, and computing a set of fixing edges.

Throughout this section, we use two key parameters: \(\sigma_0\) and \(\sigma_1\). These parameters should be thought of as distance and shortcutter-size-related parameters respectively. While there are other parameters (like the \(\mu\), which are all \(m^{o(1)}\)), these parameters are purely determined by proofs in the main sections. Only \(\sigma_0\) and \(\sigma_1\) are traded off in order to bound the runtime of the main algorithm ExactTree. For more details on parameter values, see the Appendix of the full version [Sch17].

4.1 Our Shortcutting Data Structure

Recall that in Section 2, we stated that no vertices were in more than \(m^{o(1)}\) different shortcutters. Here, we organize the shortcutters into a small number of families of disjoint shortcutters, which we call clans, in order to achieve this property.

221
An Almost-Linear Time Algorithm for Uniform Random Spanning Tree Generation

Definition 4.1 (Organization of shortcutters). Consider a graph $H$ obtained as a minor of $G$. A cluster is a set of vertices. In our algorithm, there are three kinds of clusters: parts, cores, and shortcutters. We define parts in Definition 4.3. A core is an arbitrary cluster. A shortcutter is a cluster $S_C$ that contains a core $C$ of vertices that are “assigned” to it. A clan is a set of (vertex-)disjoint shortcutters. A horde is a set of clans.

All hordes in our algorithm satisfy the following invariant:

Invariant 4.2. A horde $\mathcal{H}$ consists of at most $\ell_{\text{max}} \leq m^{o(1)}$ clans.

Definition 4.3 (Covering hordes and overlay partitions). A horde $\mathcal{H}$ is said to cover $H$ if every vertex in $H$ is in the core of some shortcutter in some clan of $\mathcal{H}$.

Given a collection of covering hordes $\{\mathcal{H}_i\}_{i=1}^{m}$, the overlays $\mathcal{P}_i(\{\mathcal{H}_i\}_{i=1}^{m})$ are formed by refining all shortcutters from all clans in $\cup_{j \geq i} \mathcal{H}_j$. More precisely, let $\chi_i$ denote the equivalence relation formed by letting $u \sim v$ if and only if for all clans $C \in \bigcup_{j \geq i} \mathcal{H}_j$, $u$ and $v$ are either (a) both in the same core of $C$ or (b) both not in any core of $C$. Let $\mathcal{P}_i(\{\mathcal{H}_i\}_{i=1}^{m})$ denote the equivalence classes of $\chi_i$.

Since all $\mathcal{H}_i$s are covering, each $\mathcal{P}_i(\{\mathcal{H}_i\}_{i=1}^{m})$ is a partition of $V(H)$. A part $P$ is some cluster in $\mathcal{P}_i(\{\mathcal{H}_i\}_{i=1}^{m})$ for some $i \in [m]$. Each part $P \in \mathcal{P}_i(\{\mathcal{H}_i\}_{i=1}^{m})$ is assigned to a single core $C_P$ of a shortcutter $S_P$ in some clan of $\mathcal{H}_i$.

Let $\partial \mathcal{P}_i(\{\mathcal{H}_i\}_{i=1}^{m})$ denote the set of boundary edges of parts in $\mathcal{P}_i(\{\mathcal{H}_i\}_{i=1}^{m})$.

Organizing shortcutters into clans allows us to define properties that hold for shortcutters in a clan “on average.” Now, we define various properties that cores, shortcutters, clans, and hordes should have. After defining these properties, we summarize their relevance to bounding the shortcutted random walk simulation time in Tables 3, 4, and 5.

For each of these definitions, fix a distance scale $R$. We start by insisting that each core consist of closely-by vertices.

Definition 4.4 (R-clans). Call a clan an $R$-clan if each shortcutter’s core has $H$-effective resistance diameter at most $R$.

$R$ may be referred to even if a clan is not an $R$-clan (i.e. the clan may not have bounded diameter cores).

Each clan contains shortcutters that are relatively similar to one another. This way, our analysis of the shortcutting scheme can focus on the clans within a horde independently. Specifically, a clan $C$ is said to be bucketed if the maximum size of a shortcutter in $C$ is at most $4m/|C|$.

Inverting this definition suggests a more convenient definition of the size of a clan.

Definition 4.5 (Effective size and bucketing). The effective size of a clan $C$, denoted $s_C$, is the following:

$$s_C := \max_{S \subseteq C} |E(S) \cup \partial S|$$

We say that a clan $C$ is bucketed if

$$|C| \leq 4s_C$$

Clans also contain shortcutters with the property that using a shortcutter bypasses many random walk steps. Specifically, the conductance of a shortcutter is relevant for assessing how many times it is used, as discussed in Section 2. For an arbitrary graph $H'$, let $\epsilon_{H'}(C)$, the conductance of $C$ with respect to $H'$, be

$$\epsilon_{H'}(C) := \sum_{e \in E(C,V(H')) \setminus \partial S_C} \chi_{e} \left( H', C \cup (V(H') \setminus S_C) \right)$$

We define the conductance with respect to $H'$, not $H$, because we need to delete edges from $H$ in order to maintain the condition that $\epsilon_{H'}(S_C)$ is low after conditioning. $H'$ will be a graph obtained by deleting some edges $\text{deleted}(C)$ from $H$.

Definition 4.6 (Deletion set and the deletion set condition). For a clan $C \in \mathcal{E}_i$, maintain a set $\text{deleted}(C)$ of edges. This set must satisfy the deletion set condition, which states that no deleted edge is incident with a nonempty part. Specifically, for any $P \in \mathcal{P}_i(\mathcal{E})$ for which $E(P) \neq \emptyset$,

$$\text{deleted}(C) \cap (\partial_P P) = \emptyset$$

The deletion set condition ensures that precomputed random walk steps to the endpoint of a deleted edge cross a boundary edge of some part in $\mathcal{P}_i(\mathcal{E})$. As a result, precomputed steps can be charged to boundary edges of parts and can be bounded using Lemma 2.3 as described in Table 5.

The following condition is used to bound the precomputation work during the shortcutting algorithm:

Definition 4.7 (Modifiedness). We say that a clan $C$ is $\tau$-modified if the number of deleted edges is not too high on average:

$$|\text{deleted}(C)| \leq \tau m^{1/m} s_C$$

For a clan $C$, let $H_C := H \setminus \text{deleted}(C)$. For a shortcutter $S_C \in C$, let $\epsilon_C(S_C) = \epsilon_{H_C}(S_C)$.
Lemma 2.3. 

Definition 4.9 (Well-spacedness). An R-clan C is well-spaced if no cluster C′ ⊆ V(H) is tied to more than one shortcutter S_C ∈ C.

The lack of ties for well-spaced clusters ensures that deleting C′ from all shortcutters in C does not increase the total conductance of shortcutters in C much.

All of the definitions that we have discussed leading up to this are used to show that conditioning once takes at most \(O(m^{1+\beta}r_{\min}^{(\beta+1)})\) time. Recall from Section 2 that sampling the intersection of a random tree with \(E(S)\) for some set of vertices \(S\) is supposed to allow us to get rid of some shortcutters because the boundary of \(S\) is small.

Definition 4.10 (Boundedness). Say that a clan associated with distance scale \(R\) is \(\kappa\)-bounded if

\[
\sum_{S_C \in C} c^H(\partial C) \leq \frac{\kappa m}{R}
\]

We now extend our definitions of clans to hordes. A horde \(H\) is an \(R\)-horde if each clan in \(H\) is an \(R\)-clan. A horde \(H\) is bucketed if each clan in it is bucketed. A horde is \(\tau\)-modified if each clan in it is \(\tau\)-modified. A horde \(H\) is \(\zeta\)-conductive if each clan in it is \(\zeta\)-conductive. A horde is well-spaced if each of its clans are well-spaced. A horde is \(\kappa\)-bounded if each of its clans is \(\kappa\)-bounded. A horde satisfies the deletion set condition if each of its clans satisfies it.

We now give definitions that are specific to hordes and to collections of hordes. Each vertex needs to have a shortcutter at every distance scale. Since a horde is associated with one distance scale \(R\), each vertex should have a shortcutter in each horde. Now, we define a special collection of hordes called an empire with which sampling can be performed:

Definition 4.11 (Empires). An empire \(E\) is a set of covering hordes \(\{E_i\}_{i=1}^{\infty}\), with \(E_i\) being an \(R\)\(_{\text{mod}}^{i/(\beta+1)}\)\(_{\min}\)-horde. Define bucketedness, \(\tau\)-modifiedness, \(\zeta\)-conductivity, well-spacing, \(\kappa\)-boundedness, and the deletion set condition for empires as well if these conditions hold for all constituent hordes.

Now, we show how these properties fit together to bound the runtime of our implementation of the full shortcutting meta-algorithm described in Section 2. When the random walk is at a vertex \(u\), the meta-algorithm first finds the maximum \(i \in [\sigma_0]\) for which the intersection of a shortcutter \(S_{P_i}\) with the set \(S\) is covered, where \(P_i \in P_i(E)\) is the unique part containing \(u\). If \(E(P_i) = 0\), it does a standard random walk step. Otherwise, it samples whether the random walk hits an endpoint of an edge in \(\text{deleted}(C)\) before exiting \(S_{P_i}\), where \(C\) is the clan containing \(S_{P_i}\). If so, it uses offline shortcutting to shortcut to \(\text{deleted}(C)\). Otherwise, it uses online shortcutting to shortcut to the boundary of \(S_{P_i}\).

The above discussion cites three kinds of random walk-related work and one kind of precomputation work. To bound the random walk-related work, we exploit Lemma 2.3. Lemma 2.3 requires bounds on two parameters: conductance of an edge and distance to an unvisited vertex. Work done using a part \(P_i\) is charged to the clan containing \(S_{P_i}\) as follows:
Table 3: Charging work during each sampling round to an edge

| Type of work     | Edge to charge to | Reason for charging |
|------------------|-------------------|---------------------|
| Walk across $\partial P_i$ | $P_i$ boundary edge | Normal random walk step |
| Shortcut to deleted(C) | $P_i$ boundary edge | Deletion set condition |
| Shortcut to $\delta S_{P_i}$ | An edge in Schur($H_i$, $C_{P_i} \cup V(H) \setminus S_{P_i}$) | Theorem 2.4 |

Table 4: Bounds on parameters required for Lemma 2.3

| Type of work     | Conductance bound for work charged to a clan $C$ | Distance to unvisited vertex |
|------------------|--------------------------------------------------|------------------------------|
| Walk across $\partial P_i$ | $\kappa m/(\sigma_1^{(n_0+1)} r_{\text{min}})$ (boundedness) | $\sigma_1^{(n_0+1)} r_{\text{min}}$ (carving, R-clan) |
| Shortcut to deleted(C) | $\kappa m/(\sigma_1^{(n_0+1)} r_{\text{min}})$ (boundedness) | $\sigma_1^{(n_0+1)} r_{\text{min}}$ (carving, R-clan) |
| Shortcut to $\delta S_{P_i}$ | $\xi m^{1/\sigma C_i}(1/\sigma_1^{(n_0+1)} r_{\text{min}})$ (conductivity) | $\sigma_1^{(n_0+1)} r_{\text{min}}$ (carving, R-clan) |

RebuildEmpire($\{H_i\}_{i=1}^{\alpha(P)}$) that, when given a set of covering hordes $\{H_i\}_{i=1}^{\alpha(P)}$ with $H_i$ associated with distance scale $\sigma_1^{(n_0+1)} r_{\text{min}}$ in a graph $H$, returns an empire $E = \{H_i\}_{i=1}^{\alpha(P)}$ with the following properties:

- (Bucketing) $E$ is bucketed.
- (Conductivity) If each horde $H_i$ is $\zeta$-conductive, then $E$ is $(8 \log n) \zeta + (16 \log n) \mu_{\text{app}}$-conductive.
- (Well-spacedness) $E$ is well-spaced.
- (Boundedness) If each horde $H_i$ is $\kappa$-bounded, then $E$ is $\kappa + \kappa_0$ bounded, for $\kappa_0 \leq \min(1)$.
- (Modifiedness and deletion set condition) If each horde $H_i$ is $\tau$-modified, then $E$ is $\tau$-modified as well. Furthermore, if the deletion set condition is satisfied in each clan of each $H_i$, it continues to be satisfied in $E$.
- (Clan growth) The number of clans in $E$ is at most $\mu_{\text{app}} \log n$ times as high as the number of clans in all of the $H_i$s.
- (Containment) For any $i \in [\sigma_0]$, consider any part $P \in P_i(E)$. There is a unique part $Q \in P_i((H_j)_i)$ for which $P \subseteq Q$. Furthermore, $C_P \subseteq C_Q$ and $S_P \subseteq S_Q$.

Our spanning tree generation algorithm starts by calling RebuildEmpire on the set of hordes consisting of one clan, each of which just contains the one shortcutter $V(G)$. These hordes are clearly covering and have $\zeta = 0$, $\kappa = 0$, and $\tau = 0$.

RebuildEmpire is useful to call on the remnants of empires after conditioning later on in order to achieve the containment property. Containment is essential to our notion of progress, as discussed in Section 2.

4.3 Selecting Parts to Condition On

Given an empire $E$ with respect to a graph $H$, we can choose a set of vertices $S$ to condition on. The set $S$ is small enough that, when carved out of shortcutters in $E$, does not increase their conductivity too much. The upside of carving is that each vertex in $S$ is close to the core of any shortcutter that it is in. We now define this precisely:

Definition 4.13 (Active parts and carving). A part $P$ is called active if it has nonempty interior, i.e. $E(P) \neq \emptyset$. A shortcutter is called active if any part assigned to it is active.

A shortcutter $S_C$ in an $R$-clan has been carved with respect to $S \subseteq V(G)$ if each vertex $v \in S \cap S_C$ is within $H$-effective resistance distance $\rho_{\text{curve}} R$ of all vertices in $C$. An $R$-clan $C$ in an empire $E$ has been carved with respect to $\zeta$ if all of its active shortcutters have been carved with respect to $S$. An $R$-horde $H$ in an empire $E$ has been carved with respect to $S$ if each clan in it has been carved with respect to $S$. An empire $E$ has been carved with respect to $S$ if each of its hordes has been carved with respect to $S$.

The routine ConditioningVerts both (a) selects parts $K$ for conditioning on and (b) removes vertices from the shortcutters of the input empire $E$ in order to ensure that it is carved with respect to $\cup_{P \in K} P$. The ConditioningVerts subroutine maintains internal state and is the only method that exploits the “Containment” guarantee of Lemma 4.12. The “Progress” input condition in the following definition captures the fact that partial sampling eliminates edges in the induced subgraph of the previously chosen parts:

Definition 4.14 (ConditioningVerts input conditions). Given an empire $E$ in a graph $H$, the algorithm ConditioningVerts($E$)
returns a set of parts \( \mathcal{K} \) to condition on and removes vertices from the shortcutters in the empire \( \mathcal{E} \) to obtain \( \mathcal{E}' \). Let \( \mathcal{E}_{\text{pre}} \) be the argument supplied to the previous call to \( \text{ConditioningVerts} \), let \( \mathcal{K}_{\text{pre}} := \text{ConditioningVerts}(\mathcal{E}_{\text{pre}}) \), and let \( \mathcal{E}_{\text{pre}}' \) be the empire \( \mathcal{E}_{\text{pre}}' \) after being modified by \( \text{ConditioningVerts} \). Let \( \mathcal{H}_{\text{pre}} \) be the graph in which \( \mathcal{E}_{\text{pre}}' \) lies. The following conditions are the input conditions for \( \text{ConditioningVerts} \):

- (Parameters) \( \mathcal{E} \) is a bucketed, \( \zeta \)-conductive, well-spaced, \( \tau \)-modified, and \( \kappa \)-bounded empire that satisfies the deletion set condition.
- (Containment) For any \( i \in [\sigma_0] \), consider any part \( P \in \mathcal{P}_i(\mathcal{E}) \). There is a unique part \( Q \in \mathcal{P}_i(\mathcal{E}_{\text{pre}}) \) for which \( P \subseteq Q \). Furthermore, \( C_P \subseteq C_Q \) and \( S_P \subseteq S_Q \).
- (Progress) For each \( P \in \mathcal{K}_{\text{pre}} \), \( E_{\text{H}}(P) = \emptyset \).

**Lemma 4.15.** Given an empire \( \mathcal{E} = \{E_i\}_{i=1}^{n_S} \) in a graph \( H \) that satisfies the input conditions given in Definition 4.14, \( \text{ConditioningVerts}(\mathcal{E}) \) returns a set of parts \( \mathcal{K} \) to condition on and removes vertices from the shortcutters in the empire \( \mathcal{E} \) to obtain \( \mathcal{E}' \). Let \( S = \cup_{P \in \mathcal{K}} P \subseteq V(H) \). Then the following guarantees are satisfied:

- (Conductivity) \( \mathcal{E}' \) is a bucketed, \( \tau \)-modified, \( \zeta + 10(\log m)\mu_{\text{app}}(\ell_{\max} + \tau) \)-conductive, well-spaced, \( \kappa \)-bounded empire that satisfies the deletion set condition.
- (Carving) \( \mathcal{E}' \) is carved with respect to \( S \).

### 4.4 Making Enough Progress during Each Round of Conditioning

In the previous section, we showed that \( S \) is small enough to ensure that carving \( S \) out of all shortcutters in \( \mathcal{E} \) does not increase the conductivity of \( \mathcal{E} \) too much. We now show that \( S \) is large enough to make a lot of progress. Specifically, we show the following:

**Lemma 4.16.** Consider a sequence of calls \( \mathcal{K}^j \leftarrow \text{ConditioningVerts}(\mathcal{E}^j) \) that modifies \( \mathcal{E}^j \) to obtain \( \mathcal{E}'^j \). Suppose that \( \mathcal{H}^j \) is the graph in which \( \mathcal{E}'^j \) is defined. Suppose that for each \( j > 0 \), \( E = \mathcal{E}^j, \mathcal{E}_{\text{pre}} = \mathcal{E}^{j-1}, \mathcal{K}_{\text{pre}} = \mathcal{K}^{j-1} \) satisfies the input conditions in Definition 4.14. Let

\[
\text{ifinal} = (2\sigma)^{2n_0}.
\]

Then \( E(H^{j=\text{ifinal}}) = \emptyset \).

This means that only \( (2\sigma)^{2n_0} \leq o(\log n) \) rounds of conditioning are necessary to sample a random spanning tree.

### 4.5 Conditioning on the Intersection of a Random Tree with the Selected Vertices

Now that each shortcutter \( S_C \) only intersects vertices to condition on that are close to \( C \), we can make the idea for using online shortcutting in Section 2 a reality:

**Lemma 4.17.** Let \( \mathcal{K} \subseteq \bigcup_{i=1}^{n_S} \mathcal{P}_i(\mathcal{E}) \) be a set of parts. Let \( F = \cup_{P \in \mathcal{K}} E(P) \) and \( S = \cup_{P \in \mathcal{K}} P \). Suppose that the empire \( \mathcal{E} \) is \( \zeta \)-conductive, \( \kappa \)-bounded, \( \tau \)-modified, satisfies the deletion set condition, and has been carved with respect to \( S \). Then, there is an algorithm \( \text{PartialSample}(\mathcal{E}, \mathcal{K}) \) that returns the intersection of a random spanning tree \( T \) in \( H \) with \( F \) in \( O((\zeta + \kappa)(\mu)\text{carve} + \tau)\ell_{\max}m^{1+1/\sigma_0} (\sigma_0 + 1) \) time.

### 4.6 Fixing Shortcutters

After computing \( T \cap F \leftarrow \text{PartialSample} \), contracting all edges in \( F \cap T \) in \( H \), and deleting all edges in \( F \setminus T \) from \( H \), \( \mathcal{E} \) is no longer an empire with respect to \( H \). In particular, the well-spacedness, \( \zeta \)-conductivity, and core diameter conditions break down. Well-spacedness and diameter can be fixed by applying \( \text{RebuildEmpire} \). However, the \( \zeta \)-conductivity constraint accumulates over an old value. We could recompute the empire from scratch, but that forgoes the containment property that is so important to establishing progress. We will deal with this by adding edges to deleted(\( C \)) for each clan \( C \) in \( \mathcal{E} \).

**Lemma 4.18.** Let \( H \) be a graph, \( \mathcal{E} \) be an empire in \( H \) and \( \mathcal{K} \) be a set of parts. Let \( S = \cup_{P \in \mathcal{K}} P \) and let \( \mathcal{F} = \cup_{P \in \mathcal{K}} E(P) \). Let \( H' = H[\mathcal{F}] \). Suppose that the following input conditions hold \( \mathcal{E} \):

- (Bucketing) The empire \( \mathcal{E} \) is bucketed.
- (Carving) \( \mathcal{E} \) is carved with respect to \( S \).

With high probability over \( H' \), \( \text{FixShortcutters}(\mathcal{E}, H', \mathcal{K}) \) adds edges to the deletion set of each clan of \( \mathcal{E} \) to obtain a set of covering hedges \( \{H'_i\} \) with the following properties:

- (Boundedness) For each \( i \), if \( E_i \) is \( \kappa \)-bounded, then \( H'_i \) is \( \ell \)-bounded, where \( \ell = \sum_{i=1}^{n_0} |E_{i}| \).
- (Modifiedness and deletion set condition) For each \( i \), if \( E_i \) is \( \tau \)-modified and satisfies the deletion set condition, then \( H'_i \) is \( \mu_{\text{mod}}(\tau + \zeta) \)-modified and also satisfies the deletion set condition.
- (Conductivity) For each \( i \), if \( E_i \) is \( \zeta \)-conductive with respect to \( H \), then \( H'_i \) is at most \( \zeta \)-conductive with respect to \( H' \).

Furthermore, it does so in \( m^{1+o(1)} \) time.

### 4.7 An \( m^{1+o(1)} \zeta^{o(1)} \)-Time Algorithm for Exact Random Spanning Tree Generation

We now tie the results from the previous sections together to prove Theorem 1.1. We prove this result using the algorithm \( \text{ExactTree} \), which simply chains the algorithms from the previous sections in order:

Most of the effort in proving the above result boils down to checking that all of the input conditions are satisfied for each of the subroutines that \( \text{ExactTree} \) calls.

**Proof of Theorem 1.1.** Invariant 4.2. Each of \( \text{RebuildEmpire}, \text{ConditioningVerts}, \text{PartialSample}, \) and \( \text{FixShortcutters} \) increases the number of clans by at most a factor of \( (\log m)\mu_{\text{app}} \). By Lemma 4.16, only \( (2\sigma)^{2n_0} \) iterations take place. Since there is only one clan initially, the number of clans at the end is at most

\[
((\log m)\mu_{\text{app}})^{(2\sigma)^{2n_0}} = \ell_{\max} \leq m^{o(1)}
\]

as desired.

\( \kappa \leq \kappa_{\text{max}} \). Each of the subroutines called in the while loop increases \( \kappa \) by at most a factor of \( \ell_{\max} \) and additively by at most \( k_0 \leq m^{o(1)} \). Therefore,
Algorithm 1: ExactTree(G)

Input: a weighted, undirected graph G
Output: a weighted uniformly random spanning tree of G
1 $H ← G$
   
// the set of hordes which contain one clan consisting of one shortcutter (the entire graph)
2 $E ← \{([V(G)])_{i=1}^{\sigma_0}\}$
3 $T ← \emptyset$
4 while $E(H) \neq \emptyset$
   5 \hspace{1em} $E ← \mathbf{RebuildEmpire}(E)$
   6 \hspace{1em} $K ← \mathbf{ConditioningVerts}(E)$
   7 \hspace{1em} $T ← T ∪ \mathbf{PartialSample}(E, K)$
   8 \hspace{1em} Contract all edges in $H$ added to $T$ and delete all other edges internal to parts of $K$
   9 \hspace{1em} $\mathbf{FixShortcutters}(E, H, K)$
10 \hspace{1em} return $T$

as desired.

$r ≤ r_{\text{max}}$ and $\zeta ≤ \zeta_{\text{max}}$. Each subroutine call increases $\max(r, \zeta)$ by a factor of at most $10(\log m)\mu_{\text{app}}(2\alpha_1)^{m_{\text{mod}}}$ and additively by at most $10(\log m)\mu_{\text{app}}\ell_{\text{max}}\mu_{\text{mod}}$. Therefore,

$$\begin{align*}
\kappa &\leq (\ell_{\text{max}})^{(2\alpha_1)^{m_{\text{mod}}}} \\
&\leq ((\log m)\mu_{\text{app}}(2\alpha_1)^{m_{\text{mod}}} \\
&= \kappa_{\text{max}} \\
&\leq m^{o(1)}
\end{align*}$$

as desired.

Well-definedness. Start with RebuildEmpire. At the beginning of the algorithm, $\zeta = 0$, $\kappa = 0$, and all of the deletion sets are empty, so the deletion set condition is satisfied. $E$ is not an empire when it is supplied to RebuildEmpire, but is a set of covering hordes because the $1$ in $m_{\text{mod}}$ is the first iteration and the cores are all $V(G)$ or (b) Lemma 4.18 states that the hordes $H_i$ are covering. Therefore, RebuildEmpire’s input conditions given in Lemma 4.12 are always respected.

Next, consider ConditioningVerts. The “Parameters” condition is the “Parameters” guarantee from Lemma 4.12. The “Containment” condition follows from the “Containment” guarantee of Lemma 4.12, with the fact that FixShortcutters only adds to the deletion sets of the clans and PartialSample does not change $E$. Line 8 of ExactTree contracts or deletes each edge internal to each part in $K$. Therefore, the “Progress” condition is satisfied afterwards.

The desired parameter bounds for PartialSample are given in the “Boundedness and covering” guarantee of Lemma 4.15. The carving condition of Lemma 4.17 is the “Carving” guarantee of Lemma 4.15.

Finally, deal with FixShortcutters. The input conditions for Lemma 4.18 are given directly as the “Carving” guarantee of Lemma 4.15, the “Bucketing” guarantee of Lemma 4.12, and the fact that removing vertices from shortcutters preserves the bucketing guarantee.

Correctness. By Theorem 3.4, sampling a random tree in some $H$ is equivalent to partial sampling with $F = \cup \mathcal{P} \in H E(P)$ and sampling a tree in the graph obtained by contracting the chosen edges in $F$ and deleting all others. By Lemma 4.17, PartialSample returns a valid sample from a uniformly random spanning tree of $H$ intersected with $F$. Therefore, once $E(H) = \emptyset$, $T$ has been completely sampled and is valid.

Runtime. By Lemma 4.16, the while loop runs at most $(2\alpha_1)^{2\sigma_1} \leq m^{o(1)}$ times. RebuildEmpire, ConditioningVerts, PartialSample, and FixShortcutters each take $m^{1+o(1)}(12^{-\alpha_1})^{o(1)}$ time by Lemmas 4.12, 4.15, 4.17, and 4.18 respectively and our bounds on $\ell_{\text{max}}$, $r_{\text{max}}$, $\kappa_{\text{max}}$, and $\zeta_{\text{max}}$. Contracting and deleting edges only takes $O(m)$ time. Therefore, the entire algorithm only takes $O(m^{1+o(1)}(12^{-\alpha_1}m^{o(1)}))$ time. Since $\sigma_0$ is superconstant, this runtime is $m^{1+o(1)}(12^{-\alpha_1}m^{o(1)})$, as desired.

4.8 An $m^{1+o(1)}e^{-\alpha_1}$. Time Algorithm for Generating a Random Spanning Tree from a Distribution with Total Variation Distance $\epsilon$ from Uniform

In the appendix of the full version, we give a simple reduction that proves Theorem 1.2 given just Theorem 1.1. The reduction samples the intersection of a random tree with a part of the graph with polynomial aspect ratio and smallest resistances. Conditioning on this part of the graph removes the edges with smallest resistance from the graph. A ball-growing-type technique and Theorem 1.1 ensures that each round of conditioning eliminates a number of edges from the graph proportional to the amount of work done.

ACKNOWLEDGMENTS

We would like to thank Nikhil Srivastava, Satish Rao, and Cheng Xiang for many helpful conversations. We would like to thank Alejandro Madry, Jim Pitman, Akshay Ramachandran, Alex Rusciano, Hong Zhou, and Qiuyi Zhang for helpful conversations. We thank Nima Anari, Antares Chen, Tselil Schramm, Di Wang, and Jason Wu for helpful edits.

REFERENCES

[AGM+10] Arash Asadpour, Michel X. 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 the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ‘10, pages 379–389, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[AGR16] Nima Anari, Shayan Oveis Gharan, and Alireza Rezaei. Monte carlo markov chain algorithms for sampling strongly rayleigh distributions and deterministic point processes. In Vitaly Feldman, Alexander Rakhlin,
and Ohad Shamir, editors, 29th Annual Conference on Learning Theory, volume 49 of Proceedings of Machine Learning Research, pages 103–115, Columbia University, New York, New York, USA, 23–26 Jun 2016. PMLR.

[Ald90] David J. Aldous. The random walk construction of uniform spanning trees and uniform labelled trees. SIAM J. Discret. Math., 3(4):450–465, November 1990.

[AMS96] Noga Alon, Yoshi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In Proceedings of the Twentieth-eight Annual ACM Symposium on Theory of Computing, STOC ’96, pages 20–29, New York, NY, USA, 1996. ACM.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions (extended abstract). In 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II, pages 593–593, 1990.

[Ald90] David J. Aldous. The random walk construction of uniform spanning trees and uniform labelled trees. SIAM J. Discret. Math., 3(4):450–465, November 1990.

[Bal13] Matthew Baker and Farbod Shokrieh. Chip-firing games, potential theory on graphs, and combinatorial theory. Journal of Combinatorial Theory, Series A, 120(1):164 – 182, 2013.

[CDN99] Charles J. Colbourn, Robert P. Day, and Louis D. Nel. Unranking and ranking spanning trees of a graph. Journal of Algorithms, 30(2):271 – 286, 1999.

[CKM*14] Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Sushant Sachdeva. Sampling spanning trees faster than matrix multiplication. In Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC ’17, pages 730–742, New York, NY, USA, 2017. ACM.

[DPR17] David Durfee, John Peebles, Richard Peng, and Anup B. Rao. Determinant-preserving sparsification of SDDM matrices with applications to counting and sampling spanning trees. CoRR, abs/1705.00962, 2017.

[FH10] Wai Shing Fung and Nicholas J. A. Harvey. Graph sparsification by edge-connectivity and random spanning trees. CoRR, abs/1005.0265, 2010.

[GRV09] Navin Goyal, Luis Rademacher, and Santosh Vempala. Expanders via random spanning trees. In Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’09, pages 576–585, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[GSS11] Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science, FOCS ’11, pages 550–559, Washington, DC, USA, 2011. IEEE Computer Society.

[Gue83] A. Guenoche. Random spanning tree. Journal of Algorithms, 4:214–220, 1983.

[HHN*08] Prabhakar Harsha, Thomas P. Hayes, Harisharay, Narayanan, Harald Räcke, and Jakumar Radhakrishnan. Minimum average latency in oblivious routing. In Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’08, pages 200–207, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[HX16] Nicholas J. A. Harvey and Keyulu Xu. Generating Random Spanning Trees via Fast Matrix Multiplication, pages 522–535. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[Ind06] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. J. ACM, 53(3):307–323, 2006.

[Kir04] G. Kirchhoff. Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird. Annalen der Physik, 148:497–508, 1847.

[KM09] Jonathan A. Kelner and Aleksander Madry. Faster generation of random spanning trees. In 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA, pages 13–21, 2009.

[KMP14] Joannis Konstantinidis, Gary L. Miller, and Richard Peng. Approaching optimality for solving sdd linear systems. SIAM Journal on Computing, 43(1):337–354, 2014.

[KOSZ13] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing, STOC ’13, pages 911–920, New York, NY, USA, 2013. ACM.

[KS16] R. Kyng and S. Sachdeva. Approximate gaussian elimination for laplacians - fast, sparse, and simple. In 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS), pages 573–582, Oct 2016.

[Kir47] G. Kirchhoff. Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird. Annalen der Physik, 148:497–508, 1847.

[LP16] Russell Lyons and Yuval Peres. Probability on Trees and Networks, volume 42 of Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, New York, 2016.

[LPS15] Yin Tat Lee, Richard Peng, and Daniel A. Spielman. Sparsified cholesky solvers for SDD linear systems. CoRR, abs/1506.08204, 2015.

[LR99] Frank Thomson Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. J. ACM, 46(5):787–823, 1999.

[MS15] Aleksander Madry, Damian Srasrak, and Jakub Tarnawski. Fast generation of random spanning trees and the effective resistance metric. In Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015, pages 2019–2036, 2015.

[MPS15] Jim Propp, Richard Peng, and Daniel A. Spielman. An efficient parallel solver for sdd linear systems. In Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing, STOC ’14, pages 333–342, New York, NY, USA, 2014. ACM.

[Sch17] Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. CoRR, abs/1711.06455, 2017.

[SR17] Aaron Schild, Satish Rao, and Nikhil Srivastava. Localization of electrical flows. CoRR, abs/1708.01632, 2017.

[ST14] Daniel A. Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. SIAM J. Matrix Analysis Applications, 35(3):835–885, 2014.

[Wul96] David Bruce Wilson. Generating random spanning trees more quickly than the cover time. In Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing, STOC ’96, pages 296–303, New York, NY, USA, 1996. ACM.