58th Annual IEEE Symposium on Foundations of Computer Science

Determinant-Preserving Sparsification of SDDM Matrices with Applications to Counting and Sampling Spanning Trees

David Durfee*, John Peebles†, Richard Peng*, and Anup B. Rao‡
* School of Computer Science, Georgia Institute of Technology, Atlanta, USA
ddurfee@gatech.edu, rpeng@cc.gatech.edu
† EECS Department, Massachusetts Institute of Technology, Cambridge, USA
jpeebles@mit.edu
‡ Adobe Research, San José, USA
anuprao@adobe.com

***Abstract*—We show variants of spectral sparsification routines can preserve the total spanning tree counts of graphs, which by Kirchhoff's matrix-tree theorem, is equivalent to determinant of a graph Laplacian minor, or equivalently, of any SDDM matrix. Our analyses utilizes this combinatorial connection to bridge between statistical leverage scores / effective resistances and the analysis of random graphs by [Janson, Combinatorics, Probability and Computing '94]. This leads to a routine that in quadratic time, sparsifies a graph down to about $n^{1.5}$ edges in ways that preserve both the determinant and the distribution of spanning trees (provided the sparsified graph is viewed as a random object). Extending this algorithm to work with Schur complements and approximate Cholesky factorizations leads to algorithms for counting and sampling spanning trees which are nearly optimal for dense graphs.**

We give an algorithm that computes a $(1\pm\delta)$ approximation to the determinant of any SDDM matrix with constant probability in about $n^2\delta^{-2}$ time. This is the first routine for graphs that outperforms general-purpose routines for computing determinants of arbitrary matrices. We also give an algorithm that generates in about $n^2\delta^{-2}$ time a spanning tree of a weighted undirected graph from a distribution with total variation distance of δ from the w-uniform distribution .

I. Introduction

This document is an extended abstract which has neither full proofs nor our full set of results. Readers are strongly encouraged to instead read the full version of the paper, which can be found at https://arxiv.org/abs/1705.00985

The determinant of a matrix is a fundamental quantity in numerical algorithms due to its connection to the rank of the matrix and its interpretation as the volume of the ellipsoid corresponding of the matrix. For graph Laplacians, which are at the core of spectral graph theory and spectral algorithms, the matrix-tree theorem gives that the determinant of the minor obtained by removing one row and the corresponding column equals to the total weight of all the spanning trees in the graph [1] . Formally on a weighted graph G with n vertices we have:

$$\det\left(\boldsymbol{L}^G_{1:n-1,1:n-1}\right) = \mathcal{T}_G,$$

where $\boldsymbol{L}^G$ is the graph Laplacian of G and and $\mathcal{T}_G$ is the total weight of all the spanning trees of G. As the all-ones vector is in the null space of $\boldsymbol{L}^G$, we need to drop its last row and column and work with $\boldsymbol{L}^G_{1:n-1,1:n-1}$, which is precisely the definition of SDDM matrices in numerical analysis [2]. The study of random spanning trees builds directly upon this connection between tree counts and determinants, and also plays an important role in graph theory [3], [4], [5].

While there has been much progress in the development of faster spectral algorithms, the estimation of determinants encapsulates many shortcomings of existing techniques. Many of the nearly linear time algorithms rely on sparsification procedures that remove edges from a graph while provably preserving the Laplacian matrix as an operator, and in turn, crucial algorithmic quantities such as cut sizes, Rayleigh quotients, and eigenvalues. The determinant of a matrix on the other hand is the product of all of its eigenvalues. As a result, a worst case guarantee of $1 \pm (\epsilon/n)$ per eigenvalue is needed to obtain a good overall approximation, and this in turn leads to additional factors of n in the number of edges needed in the sparse approximate.

Due to this amplification of error by a factor of n, previous works on numerically approximating determinants without dense-matrix multiplications [6], [7], [8] usually focus on the log-determinant, and (under a nearly-linear running time) give errors of additive ϵn in the log determinant estimate, or a multiplicative error of $\exp(\epsilon n)$ for the determinant. The lack of a sparsification procedure also led to the running time of random spanning tree sampling algorithms to be limited by the sizes of the dense graphs generated in intermediate steps [9], [10], [11].

In this paper, we show that a slight variant of spectral sparsification preserves determinant approximations to a much higher accuracy than applying the guarantees to individual edges. Specifically, we show that sampling $\omega(n^{1.5})$ edges from a distribution given by leverage scores, or weight times effective resistances, produces a sparser graph whose determinant approximates that of the original graph. Furthermore,

0272-5428/17 $31.00 © 2017 IEEE
DOI 10.1109/FOCS.2017.90

by treating the sparsifier itself as a random object, we can show that the spanning tree distribution produced by sampling a random tree from a random sparsifier is close to the spanning tree distribution in the original graph in total variation distance. Combining extensions of these algorithms with sparsification based algorithms for graph Laplacians then leads to quadratic time algorithms for counting and sampling random spanning trees, which are nearly optimal for dense graphs with $m = \Theta(n^2)$.

This determinant-preserving sparsification phenomenon is surprising in several aspects: because we can also show—both experimentally and mathematically—that on the complete graph, about $n^{1.5}$ edges are necessary to preserve the determinant, this is one of the first graph sparsification phenomenons that requires the number of edges to be between $>> n$. The proof of correctness of this procedure also hinges upon combinatorial arguments based on the matrix-tree theorem in ways motivated by a result for Janson for complete graphs [12], instead of the more common matrix-concentration bound based proofs [13], [14], [15], [16]. Furthermore, this algorithm appears far more delicate than spectral sparsification: it requires global control on the number of samples, high quality estimates of resistances (which is the running time bottleneck in Theorem 18 below), and only holds with constant probability. Nonetheless, the use of this procedure into our determinant estimation and spanning tree generation algorithms still demonstrates that it can serve as a useful algorithmic tool.

A. Our Results

We will use $G = (V, E, \boldsymbol{w})$ to denote weighted multigraphs, and $\boldsymbol{d}_u \stackrel{\text{def}}{=} \sum_{e:e\ni u} \boldsymbol{w}_e$ to denote the weighted degree of vertex u. The weight of a spanning tree in a weighed undirected multigraph is:

$$\boldsymbol{w}(T) \stackrel{\text{def}}{=} \prod_{e\in T} \boldsymbol{w}_e.$$

We will use $\mathcal{T}_G$ to denote the total weight of trees, $\mathcal{T}_G \stackrel{\text{def}}{=} \sum_{T\in\mathcal{T}} \boldsymbol{w}(T)$. Our key sparsification result can be described by the following theorem:

Theorem 1. *Given any graph G and any parameter δ, we can compute in $O(n^2\delta^{-2})$ time a graph H with $O(n^{1.5}\delta^{-2})$ edges such that with constant probability we have*

$$(1-\delta)\,\mathcal{T}_G \le \mathcal{T}_H \le (1+\delta)\,\mathcal{T}_G.$$

This implies that graphs can be sparsified in a manner that preserves the determinant, albeit to a density that is not nearly-linear in n.

We show how to make our sparsification routine to errors in estimating leverage scores, and how our scheme can be adapted to implicitly sparsify dense objects that we do not have explicit access to. In particular, we utilize tools such as rejection sampling and high quality effective resistance estimation via projections to extend this routine to give determinant-preserving sparsification algorithms for Schur complements, which are intermediate states of Gaussian elimination on graphs, using ideas from the sparsification of random walk polynomials.

We use these extensions of our routine to obtain a variety of algorithms built around our graph sparsifiers. Our two main algorithmic applications are as follows. We achieve the first algorithm for estimating the determinant of an SDDM matrix that is faster than general purpose algorithms for the matrix determinant problem. Since the determinant of an SDDM m corresponds to the determinant of a graph Laplacian with one row/column removed.

Theorem 2. *Given an SDDM matrix $\boldsymbol{M}$, there is a routine* DETAPPROX *which in $\widetilde{O}\left(n^2\delta^{-2}\right)$ time outputs D such that $D = (1\pm\delta)\det(\boldsymbol{M})$ with high probability*

A crucial thing to note which distinguishes the above guarantee from most other similar results is that we give a multiplicative approximation of the $\det(M)$. This is much stronger than giving a multiplicative approximation of $\log\det(M)$, which is what other work typically tries to achieve.

The sparsifiers we construct will also approximately preserve the spanning tree distribution, which we leverage to yield a faster algorithm for sampling random spanning trees. Our new algorithm improves upon the current fastest algorithm for general weighted graphs when one wishes to achieve constant—or slightly sub-constant—total variation distance.

Theorem 3. *Given an undirected, weighted graph $G = (V, E, \boldsymbol{w})$, there is a routine* APPROXTREE *which in expected time $\widetilde{O}\left(n^2\delta^{-2}\right)$ outputs a random spanning tree from a distribution that has total variation distance $\le \delta$ from the $\boldsymbol{w}$-uniform distribution on G.*

We prove this theorem in the full version of the paper.

B. Prior Work

1) Graph Sparsification: In the most general sense, a graph sparsification procedure is a method for taking a potentially dense graph and returning a sparse graph called a *sparsifier* that approximately still has many of the same properties of the original graph. It was introduced in [17] for preserving properties related to minimum spanning trees, edge connectivity, and related problems. [18] defined the notion of *cut sparsification* in which one produces a graph whose cut sizes approximate those in the original graph. [19] defined the more general notion of *spectral sparsification* which requires that the two graphs' Laplacian matrices approximate each other as quadratic forms.[1] In particular, this spectral sparsification samples $\widetilde{O}(n/\epsilon^2)$ edges from the original graph, yielding a graph with $\widetilde{O}(n/\epsilon^2)$ whose quadratic forms—and hence, eigenvalues—approximate each other within a factor of $(1\pm\epsilon)$. This implies that their determinants approximate each other within $(1\pm\epsilon)^n$. This is not useful from the perspective of preserving the determinant: since one would need to samples

[1] If two graphs Laplacian matrices approximate each other as quadratic forms then their cut sizes also approximate each other.

$\Omega(n^3)$ edges to get a constant factor approximation, one could instead exactly compute the determinant or sample spanning trees using exact algorithms with this runtime.

All of the above results on sparsification are for undirected graphs. Recently, [20] has defined a useful notion of sparsification for directed graphs along with a nearly linear time algorithm for constructing sparsifiers under this notion of sparsification.

2) Determinant Estimation: Exactly calculating the the determinant of an arbitrary matrix is known to be equivalent to matrix multiplication [21]. For approximately computing the log of the determinant, [22] uses the identity $\log(\det(A)) = \text{tr}(\log(B)) + \text{tr}(\log(B^{-1}A))$ to do this whenever one can find a matrix B such that the $\text{tr}(\log(B)) = \log(\det(B))$ and $\text{tr}(\log(B^{-1}A)) = \log(\det(B^{-1}A)$ can both be quickly approximated.[2]

For the special case of approximating the log determinant of an SDD matrix, [23] applies this same identity recursively where the B matrices are a sequence of ultrasparsifiers that are inspired by the recursive preconditioning framework of [2]. They obtain a running time of $O(m(n^{-1}\epsilon^{-2} + \epsilon^{-1})\text{polylog}(n\kappa/\epsilon))$ for estimating the log determinant to additive error ϵ.

[24] estimates the log determinant of arbitrary positive definite matrices, but has runtime that depends linearly on the condition number of the matrix.

In contrast, our work is the first we know of that gives a multiplicative approximation of the determinant itself, rather than its log. Despite achieving a much stronger approximation guarantee, our algorithm has essentially the same runtime as that of [23] when the graph is dense. Note also that if one wishes to conduct an "apples to apples" comparison by setting their value of ϵ small enough in order to match our approximation guarantee, their algorithm would only achieve a runtime bound of $O(mn\delta^{-2}\text{polylog}(n\kappa/\epsilon))$, which is never better than our runtime and can be as bad as a factor of n worse.[3]

3) Sampling Spanning Trees: Previous works on sampling random spanning trees are a combination of two ideas: that they could be generated using random walks, and that they could be mapped from a random integer via Kirchoff's matrix tree theorem. The former leads to running times of the form $O(nm)$ [25], [26], while the latter approach[27], [28], [29], [30] led to routines that run in $O(n^\omega)$ time, where $\omega \approx 2.373$ is the matrix multiplication exponent [31].

These approaches have been combined in algorithms by Kelner and Madry [9] and Madry, Straszak and Tarnawski [10]. These algorithms are based on simulating the walk more efficiently on parts of the graphs, and combining this with graph decompositions to handle the more expensive portions of the walks globally. Due to the connection with random-walk based spanning tree sampling algorithms, these routines often have inherent dependencies on the edge weights. Furthermore, on dense graphs their running times are still worse than the matrix-multiplication time routines.

The previous best running time for generating a random spanning tree from a weighted graph was $\widetilde{O}\left((n^{4/3}m^{1/2} + n^2)\log^2(1/\delta)\right)$ achieved by [11]. It works by combining a recursive procedure similar to those used in the more recent $O(n^\omega)$ time algorithms [30] with spectral sparsification ideas. When $m = \Theta\left(n^2\right)$, the algorithm in [11] takes $\widetilde{O}\left(n^{7/3}\right)$ time to produce a tree from a distribution that is $o(1)$ away from the $\boldsymbol{w}$-uniform distribution, which is slower by nearly a $n^{1/3}$ factor than the algorithm given in this paper.

Our algorithm can be viewed as a natural extension of the sparsification0-based approach from [11]: instead of preserving the probability of a single edge being chosen in a random spanning tree, we instead aim to preserve the entire distribution over spanning trees, with the sparsifier itself also considered as a random variable. This allow us to significantly reduce the sizes of intermediate graphs, but at the cost of a higher total variation distance in the spanning tree distributions. This characterization of a random spanning tree is not present in any of the previous works, and we believe it is an interesting direction to combine our sparsification procedure with the other algorithms.

II. Background

A. Graphs, Matrices, and Random Spanning Trees

The goal of generating a random spanning tree is to pick tree T with probability proportional to its weight, which we formalize in the following definition.

Definition 4 ($\boldsymbol{w}$-uniform distribution on trees)**.** Let $\boldsymbol{Pr}_T^G(\cdot)$ be a probability distribution on $\mathcal{T}_G$ such that

$$\boldsymbol{Pr}_T^G(T = T_0) = \frac{\Pi_{e\in T_0}\boldsymbol{w}_e}{\mathcal{T}_G}.$$

We refer to $\boldsymbol{Pr}_T^G(\cdot)$ as the $\boldsymbol{w}$-uniform distribution on the trees of G.

When the graph G is unweighted, this corresponds to the uniform distribution on $\mathcal{T}_G$.

We refer to $\boldsymbol{Pr}_T^G(\cdot)$ as the $\boldsymbol{w}$-uniform distribution on $\mathcal{T}_G$. When the graph G is unweighted, this corresponds to the uniform distribution on $\mathcal{T}_G$. Furthermore, as we will manipulate the probability of a particular tree being chosen extensively, we will denote such probabilities with $\boldsymbol{Pr}^G(\widehat{T})$, aka:

$$\boldsymbol{Pr}^G\left(\widehat{T}\right) \stackrel{\text{def}}{=} \boldsymbol{Pr}_T^G\left(T = \widehat{T}\right).$$

The Laplacian of a graph $G = (V, E, \boldsymbol{w})$ is an $n \times n$ matrix specified by:

$$\boldsymbol{L}_{uv} \stackrel{\text{def}}{=} \begin{cases} \boldsymbol{d}_u & \text{if } u = v \\ -\boldsymbol{w}_{uv} & \text{if } u \neq v \end{cases}$$

[2]Specifically, they take B as the diagonal of A and prove sufficient conditions for when the log determinant of $B^{-1}A$ can be quickly approximated with this choice of B.

[3]This simplification of their runtime is using the substitution $\epsilon = \delta/n$ which gives roughly $(1 \pm \delta)$ multiplicative error in estimating the determinant for their algorithm. This simplification is also assuming $\delta \leq 1$, which is the only regime we analyze our algorithm in and thus the only regime in which we can compare the two.

We will write $\boldsymbol{L}^G$ when we wish to indicate which graph G that the Laplacian corresponds to and $\boldsymbol{L}$ when the context is clear. When the graph has multi-edges, we define $\boldsymbol{w}_{uv}$ as the sum of weights of all the edges e that go between vertices u, v. Laplacians are natural objects to consider when dealing with random spanning trees due to the matrix tree theorem, which states that the determinant of $\boldsymbol{L}$ with any row/column corresponding to some vertex removed is the total weight of spanning trees. We denote this removal of a vertex u as $\boldsymbol{L}_{-u}$. As the index of vertex removed does not affect the result, we will usually work with $\boldsymbol{L}_{-n}$. Furthermore, we will use $\det(\boldsymbol{M})$ to denote the determinant of a matrix. As we will work mostly with graph Laplacians, it is also useful for us to define the 'positive determinant' $\det_+$, where we remove the last row and column. Using this notation, the matrix tree theorem can be stated as:

$$\mathcal{T}_G = \det(\boldsymbol{L}^G_{-n}) = \det_+\left(\boldsymbol{L}^G\right).$$

We measure the distance between two probability distributions by total variation distance.

Definition 5. Given two probability distributions p and q on the same index set Ω, the *total variation distance* between p and q is given by

$$d_{TV}(p,q) \stackrel{\text{def}}{=} \frac{1}{2}\sum_{x\in\Omega}|p(x)-q(x)|.$$

Let $G=(V,E,\boldsymbol{w})$ be a graph and $e\in E$ an edge. We write G/e to denote the graph obtained by contracting the edge e, i.e., identifying the two endpoints of e and deleting any self loops formed in the resulting graph. We write $G\backslash e$ to denote the graph obtained by deleting the edge e from G. We extend these definitions to G/F and $G\backslash F$ for $F\subseteq E$ to refer to the graph obtained by contracting all the edges in F and deleting all the edges in F, respectively.

Also, for a subset of vertices V_1, we use $G[V_1]$ to denote the graph induced on the vertex of V_1. letting $G(V_1)$ be the edges associated with $\boldsymbol{L}_{[V_1,V_1]}$ in the Schur complement.

B. Effective Resistances and Leverage Scores

The matrix tree theorem also gives connections to another important algebraic quantity: the *effective resistance* between two vertices. This quantity is formally given as $\mathcal{R}_{eff}(u,v) \stackrel{\text{def}}{=} \boldsymbol{\chi}_{uv}^\intercal \boldsymbol{L}^{-1}\boldsymbol{\chi}_{uv}$ where $\boldsymbol{\chi}_{uv}$ is the indicator vector with 1 at u, -1 at v, and 0 everywhere else. Via the adjugate matrix, it can be shown that the effective resistance of an edge is precisely the ratio of the number of spanning trees in G/e over the number in G:

$$\mathcal{R}_{eff}(u,v) = \frac{\mathcal{T}_{G/e}}{\mathcal{T}_G}.$$

As $\boldsymbol{w}_e\cdot\mathcal{T}_{G/e}$ is the total weight of all trees in G that contain edge e, the fraction[4] of spanning trees that contain $e=uv$ is given by $\boldsymbol{w}_e\mathcal{R}_{eff}(u,v)$. This quantity is called the *statistical leverage score* of an edge, and we denote it by $\overline{\boldsymbol{\tau}}_e$. It is fundamental component of many randomized algorithms for sampling / sparsifying graphs and matrices [13], [32], [14].

The fact that $\overline{\boldsymbol{\tau}}_e$ is the fraction of trees containing e also gives one way of deriving the sum of these quantities:

Fact 6. *(Foster's Theorem) On any graph G we have*

$$\sum_e \overline{\boldsymbol{\tau}}_e = n-1.$$

The resistance $\mathcal{R}_{eff}(u,v)$, and in turn the statistical leverage scores $\overline{\boldsymbol{\tau}}_e$ can be estimated using linear system solves and random projections [13]. For simplicity, we follow the abstraction utilized by Madry, Straszak, and Tarnawski [10], except we also allow the intermediate linear system solves to utilize a sparsifier instead of the original graph.

Lemma 7. *(Theorem 2.1. of [10])*
Let $G=(V,E)$ be a graph with m edges. For every $\epsilon>0$ we can find in $\tilde{O}(\min\{m\epsilon^{-2}, m+n\epsilon^{-4}\})$ time an embedding of the effective resistance metric into $\Re^{O(\epsilon^{-2}\log m)}$ such that with high probability allows one to compute an estimate $\widetilde{\mathcal{R}}_{eff}(u,v)$ of any effective resistance satisfying

$$(1-\epsilon)\widetilde{\mathcal{R}}_{eff}(u,v) \le \mathcal{R}_{eff}(u,v) \le (1+\epsilon)\widetilde{\mathcal{R}}_{eff}(u,v)$$

$\forall u,v\in V$. Specifically, each vertex u in this embedding is associated with an (explicitly stored) $\boldsymbol{z}_u\in\Re^{O(\epsilon^{-2}\log m)}$, and for any pair of vertices, the estimate $\widetilde{\mathcal{R}}_{eff}(u,v)$ is given by:

$$\widetilde{\mathcal{R}}_{eff}(u,v) = \|\boldsymbol{z}_u-\boldsymbol{z}_v\|_2^2,$$

which takes $O(\epsilon^{-2}\log m)$ time to compute once we have the embedding.

C. Schur Complements

For our applications, we will utilize our determinant-preserving sparsification algorithms in recursions based on Schur complements. A partition of the vertices, which we will denote using

$$V = V_1 \sqcup V_2,$$

partitions the corresponding graph Laplacian into blocks which we will denote using indices in the subscripts:

$$\boldsymbol{L} = \begin{bmatrix} \boldsymbol{L}_{[V_1,V_1]} & \boldsymbol{L}_{[V_1,V_2]} \\ \boldsymbol{L}_{[V_2,V_1]} & \boldsymbol{L}_{[V_2,V_2]} \end{bmatrix}.$$

The Schur complement of G, or $\boldsymbol{L}$, onto V_1 is then:

$$\begin{aligned}\textsc{Sc}(G,V_1) &= \textsc{Sc}\left(\boldsymbol{L}^G,V_1\right)\\ &\stackrel{\text{def}}{=} \boldsymbol{L}^G_{[V_1,V_1]} - \boldsymbol{L}^G_{[V_1,V_2]}\left(\boldsymbol{L}^G_{[V_2,V_2]}\right)^{-1}\boldsymbol{L}^G_{[V_2,V_1]},\end{aligned}$$

and we will use $\textsc{Sc}(G,V_1)$ and $\textsc{Sc}\left(\boldsymbol{L}^G,V_1\right)$ interchangeably. We further note that we will always consider V_1 to be the vertex set we Schur complement onto, and V_2 to be the vertex set we eliminate, except for instances in which we need to consider both $\textsc{Sc}(G,V_1)$ and $\textsc{Sc}(G,V_2)$.

[4] provided one thinks of an edge with weight w as representing w parallel edges, or equivalently, counts spanning trees with multiplicity according to their weight

Schur complements behave nicely with respect to determinants determinants, which suggests the general structure of the recursion we will use for estimating the determinant.

Fact 8. *For any matrix $\boldsymbol{M}$ where $\boldsymbol{M}_{[V_2,V_2]}$ is invertible,*

$$\det\left(\boldsymbol{M}_{-n}\right) = \det\left(\boldsymbol{M}_{[V_2,V_2]}\right) \cdot \det_{+}(\textsc{Sc}\left(\boldsymbol{M}, V_1\right)).$$

This relationship also suggests that there should exist a bijection between spanning tree distribution in G and the product distribution given by sampling spanning trees independently from $\textsc{Sc}\left(\boldsymbol{L}, V_1\right)$ and the graph Laplacian formed by adding one row/column to $\boldsymbol{L}_{[V_2,V_2]}$. We examine this further in the full version.

III. Sketch of the Results

The starting point for us is the paper by Janson [12] which gives (among other things) the limiting distribution of the number of spanning trees in the $\mathcal{G}_{n,m}$ model of random graphs. Our concentration result for the number of spanning trees in the sparsified graph is inspired by this paper, and our algorithmic use of this sparsification routine is motivated by sparsification based algorithms for matrices related to graphs [33], [34], [35]. The key result we will prove is a concentration bound on the number of spanning trees when the graph is sparsified by sampling edges with probability approximately proportional to effective resistance.

A. Concentration Bound

Let G be a weighted graph with n vertices and m edges, and H be a random subgraph obtained by choosing a subset of edges of size s uniformly randomly. The probability of a subset of edges, which could either be a single tree, or the union of several trees, being kept in H can be bounded precisely. Since we will eventually choose $s > n^{1.5}$, we will treat the quantity n^3/s^2 as negligible. The probability of H containing a fixed tree was shown by Janson to be:

Lemma 9. *If $m \geq \frac{s^2}{n}$, then for any tree T, the probability of it being included in H is*

$$Pr_H\left[T \in H\right] = \frac{(s)_{n-1}}{(m)_{n-1}} = p^{n-1} \cdot \exp\left(-\frac{n^2}{2s} - O\left(\frac{n^3}{s^2}\right)\right).$$

where $(a)_b$ denotes the product $a \cdot (a-1) \cdots (a-(b-1))$.

By linearity of expectation, the expected total weight of spanning trees in H is:

$$\mathbb{E}_H\left[\mathcal{T}_H\right] = \mathcal{T}_G \cdot p^{n-1} \cdot \exp\left(-\frac{n^2}{2s} - O\left(\frac{n^3}{s^2}\right)\right). \quad (1)$$

As in [12], the second moment, $\mathbb{E}_H\left[\mathcal{T}_H^2\right] = \mathbb{E}_H\left[\sum_{(T_1,T_2)} \boldsymbol{w}(T_1)\boldsymbol{w}(T_2)\boldsymbol{Pr}\left(T_1, T_2 \in H\right)\right]$, can be written as a sum over all pairs of trees (T_1, T_2). Due to symmetry, the probability of a particular pair of trees T_1, T_2 both being subgraphs of H depends only on the size of their intersection. The following bound is shown in the appendix of the full version.

Lemma 10. *Let G be a graph with n vertices and m edges, and H be a uniformly random subset of $s > 10n$ edges chosen from G, where $m \geq \frac{s^2}{n}$. Then for any two spanning trees T_1 and T_2 of G with $|T_1 \cap T_2| = k$, we have:*

$$Pr_H\left[T_1, T_2 \in H\right] \leq p^{2n-2} \exp\left(-\frac{2n^2}{s}\right)\left(\frac{1}{p}\left(1 + \frac{2n}{s}\right)\right)^k,$$

where $p = s/m$.

The crux of the bound on the second moment in Janson's proof is getting a handle on the number of tree pairs (T_1, T_2) with $|T_1 \cap T_2| = k$ in the complete graph where all edges are symmetric. An alternate way to obtain a bound on the number of spanning trees can also be obtained using leverage scores, which describe the fraction of spanning trees that utilize a single edge. A well known fact about random spanning tree distributions [36] is that the edges are negatively correlated:

Fact 11 (Negative Correlation)**.** *Suppose F is subset of edges in a graph G, then*

$$\boldsymbol{Pr}_T^G\left(F \subseteq T\right) \leq \Pi_{e \in F} \boldsymbol{Pr}_T^G\left(e \in T\right).$$

An easy consequence of Fact 11 is

Lemma 12. *For any subset of edges F we have that the total weight of all spanning trees containing F is given by*

$$\sum_{\substack{T \text{ is a spanning tree of } G \\ F \subseteq T}} \boldsymbol{w}\left(T\right) \leq \mathcal{T}_G \prod_{e \in F} \overline{\boldsymbol{\tau}}_e.$$

The combinatorial view of all edges being interchangable in the complete graph can therefore be replaced with an algebraic view in terms of the leverage scores. Specifically, invoking Lemma 12 in the case where all edges have leverage score at most $\frac{n}{m}$ gives the following lemma which is proven in appendix of the full version.

Lemma 13. *In a graph G where all edges have leverage scores at most $\frac{n}{m}$, we have*

$$\sum_{\substack{T_1,T_2 \\ |T_1 \cap T_2| = k}} \boldsymbol{w}\left(T_1\right) \cdot \boldsymbol{w}\left(T_2\right) \leq \mathcal{T}_G^2 \cdot \frac{1}{k!}\left(\frac{n^2}{m}\right)^k$$

With Lemma 13, we can finally prove the following bound on the second moment which gives our concentration result.

Lemma 14. *Let G be a graph on n vertices and m edges such that all edges have statistical leverage scores $\leq \frac{n}{m}$. For a random subset of $s > 10n$ edges, H, where $m \geq \frac{s^2}{n}$ we have:*

$$\begin{aligned}\mathbb{E}_H\left[\mathcal{T}_H^2\right] &\leq \mathcal{T}_G^2 p^{2n-2} \exp\left(-\frac{n^2}{s} + O\left(\frac{n^3}{s^2}\right)\right) \\ &= \mathbb{E}_H\left[\mathcal{T}_H\right]^2 \exp\left(O\left(\frac{n^3}{s^2}\right)\right).\end{aligned}$$

Proof. By definition of the second moment, we have:

$$\mathbb{E}_H\left[\mathcal{T}_H^2\right] = \sum_{T_1,T_2} \boldsymbol{w}\left(T_1\right) \cdot \boldsymbol{w}\left(T_2\right) \cdot \Pr_H\left[T_1 \cup T_2 \subseteq H\right].$$

Re-writing the above sum in terms of the size of the intersection k, and invoking Lemma 10 gives:

$$\mathbb{E}_H\left[\mathcal{T}_H^2\right] \leq \sum_{k=0}^{n-1} \sum_{\substack{T_1,T_2 \\ |T_1\cap T_2|=k}} \boldsymbol{w}\left(T_1\right)\cdot \boldsymbol{w}\left(T_2\right)\cdot p^{2n-2}\exp\left(-\frac{2n^2}{s}\right)\left(\frac{1}{p}\left(1+\frac{2n}{s}\right)\right)^k.$$

Note that the trailing term only depends on k and can be pulled outside the summation of T_1, T_2, so we then use Lemma 13 to bound this by:

$$\mathbb{E}_H\left[\mathcal{T}_H^2\right] \leq \sum_{k=0}^{n-1} \mathcal{T}_G^2\cdot\frac{1}{k!}\left(\frac{n^2}{m}\right)^k\cdot p^{2n-2} \cdot\exp\left(-\frac{2n^2}{s}\right)\left(\frac{1}{p}\left(1+\frac{2n}{s}\right)\right)^k.$$

Pulling out the terms that are independent of k, and substituting in $p = s/m$ gives:

$$\mathbb{E}_H\left[\mathcal{T}_H^2\right] \leq \mathcal{T}_G^2\cdot p^{2n-2}\cdot\exp\left(-\frac{2n^2}{s}\right)\cdot\sum_{k=0}^{n-1}\frac{\cdot\left(\frac{n^2}{s}\left(1+\frac{2n}{s}\right)\right)^k}{k!}.$$

From the Taylor expansion of $\exp(\cdot)$, we have:

$$\begin{aligned}\mathbb{E}_H\left[\mathcal{T}_H^2\right] &\leq \mathcal{T}_G^2\cdot p^{2n-2}\cdot\exp\left(-\frac{2n^2}{s}+\frac{n^2}{s}\left(1+\frac{2n}{s}\right)\right)\\ &= \mathcal{T}_G^2\cdot p^{2n-2}\cdot\exp\left(-\frac{n^2}{s}\right)\cdot\exp\left(O\left(\frac{n^3}{s^2}\right)\right).\end{aligned}$$

□

This bound implies that once we set $s^2 > n^3$, the variance becomes less than the square of the expectation. It forms the basis of our key concentration results, which we show in Section IV, and also leads to Theorem 1. In particular, we demonstrate that this sampling scheme extends to importance sampling, where edges are picked with probabilities proportional to (approximations of) of their leverage scores.

A somewhat surprising aspect of this concentration result is that there is a difference between models $\mathcal{G}_{n,m}$ and the Erdos-Renyi model $\mathcal{G}_{n,p}$ when the quantity of interest is the number of spanning trees. In particular, the number of spanning trees of a graph $G \sim \mathcal{G}_{n,m}$ is approximately normally distributed when $m = \omega\left(n^{1.5}\right)$, whereas it has approximate log-normal distribution when $G \sim \mathcal{G}_{n,p}$ and $p < 1$.

An immediate consequence of this is that we can now approximate $\det_+(\boldsymbol{L}^G)$ by computing $\det_+(\boldsymbol{L}^H)$. It also becomes natural to consider speedups of random spanning tree sampling algorithms that generate a spanning tree from a sparsifier. Note however that we cannot hope to preserve the distribution over all spanning trees via a single sparsifier, as some of the edges are no longer present.

To account for this change in support, we instead consider the randomness used in generating the sparsifier as also part of the randomness needed to produce spanning trees. In the full version, we show that just bounds on the variance of $\mathcal{T}_H$ suffices for a bound on the TV distances of the trees.

Lemma 15. *Suppose $\mathcal{H}$ is a distribution over rescaled subgraphs of G such that for some parameter some $0 < \delta < 1$ we have*

$$\frac{\mathbb{E}_{H\sim\mathcal{H}}\left[\mathcal{T}_H^2\right]}{\mathbb{E}_{H\sim\mathcal{H}}\left[\mathcal{T}_H\right]^2} \leq 1+\delta,$$

and for any tree $\hat{T}$ and any graph from the distribution that contain it, H we have:

$$\boldsymbol{w}^H\left(\widehat{T}\right) = \boldsymbol{w}^G\left(\widehat{T}\right)\cdot Pr_{H'\sim\mathcal{H}}\left[\widehat{T}\subseteq H'\right]^{-1}\cdot\frac{\mathbb{E}_{H'\sim\mathcal{H}}\left[\mathcal{T}_{H'}\right]}{\mathcal{T}_G},$$

then the distribution given by $\boldsymbol{Pr}^G(T)$, p, and the distribution induced by $\mathbb{E}_{H\sim\mathcal{H}}\left[\boldsymbol{Pr}^H(T)\right]$, $\tilde{p}$ satisfies

$$d_{TV}\left(p,\tilde{p}\right) \leq \sqrt{\delta}.$$

Note that uniform sampling meets the property about $\boldsymbol{w}^H(T)$ because of linearity of expectation. One can also check that the importance sampling based routine that we discuss in the full version of this paper also meets this criteria. Combining this with the running time bounds from Theorem 1, as well as the $\tilde{O}(m^{1/2}n^{4/3})$ time random spanning tree sampling algorithm from [11] then leads to a faster algorithm.

Corollary 16. *For any graph G on n vertices and any $\delta > 0$, there is an algorithm that generates a tree from a distribution whose total variation is at most δ from the random tree distribution of G in time $\tilde{O}(n^{\frac{25}{12}=2.0833\ldots}\delta^{-2/3} + n^2\delta^{-2})$.*

B. Integration Into Recursive Algorithms

As a one-step invocation of our concentration bound leads to speedups over previous routines, we investigate tighter integrations of the sparsification routine into algorithms. In particular, the sparsified Schur complement algorithms [35] provide a natural place to substitute spectral sparsifiers with determinant-preserving ones. In particular, the identity of

$$\det_+(\boldsymbol{L}) = \det\left(\boldsymbol{L}_{[V_2,V_2]}\right)\cdot\det_+(\textsc{Sc}\left(\boldsymbol{L},V_1\right)).$$

where $\det_+$ is the determinant of the matrix minor, suggests that we can approximate $\det\left(\boldsymbol{L}_{-n}\right)$ by approximating $\det\left(\boldsymbol{L}_{[V_2,V_2]}\right)$ and $\det_+(\textsc{Sc}\left(\boldsymbol{L},V_1\right))$ instead. Both of these subproblems are smaller by a constant factor, and we also have $|V_1| + |V_2| = n$. So this leads to a recursive scheme where the total number of vertices involved at all layers is $O(n\log n)$. This type of recursion underlies both our determinant estimation and spanning tree sampling algorithms.

The main difficulty remaining for the determinant estimation algorithm is then sparsifying $\textsc{Sc}\left(G,V_1\right)$ while preserving its determinant. For this, we note that some V_1 are significantly easier than others: in particular, when $V_2 = V\setminus V_1$ is an independent set, the Schur complement of each of the vertices in V_2 can be computed independently. Furthermore, it is well understood how to sample these complements, which

are weighted cliques, by a distribution that exceeds their true leverage scores.

Lemma 17. *There is a procedure that takes a graph G with n vertices, a parameter δ, and produces in $\tilde{O}(n^2\delta^{-1})$ time a subset of vertices V_1 with $|V_1| = \Theta(n)$, along with a graph H^{V_1} such that*

$$\mathcal{T}_{\mathrm{Sc}(G,V_1)} \exp(-\delta) \leq \mathbb{E}_{H^{V_1}}\left[\mathcal{T}_{H^{V_1}}\right] \leq \mathcal{T}_{\mathrm{Sc}(G,V_1)} \exp(\delta),$$

and

$$\frac{\mathbb{E}_{H^{V_1}}\left[\mathcal{T}^2_{H^{V_1}}\right]}{\mathbb{E}_{H^{V_1}}\left[\mathcal{T}_{H^{V_1}}\right]^2} \leq \exp(\delta).$$

Lemma 7 holds w.h.p., and we condition on this event. In our algorithmic applications we will be able to add the polynomially small failure probability of Lemma 7 to the error bounds.

The bound on variance implies that the number of spanning trees is concentrated close to its expectation, $\mathcal{T}_{\mathrm{Sc}(G,V_1)}$, and that a random spanning tree drawn from the generated graph H^{V_1} is —over the randomness of the sparsification procedure—close in total variation distance to a random spanning tree of the true Schur complement.

As a result, we can design schemes that:

1) Finds an $O(1)$-DD subset V_2, and set $V_1 \leftarrow V \setminus V_2$.
2) Produce a determinant-preserving sparsifier H^{V_1} for $\mathrm{Sc}(G, V_1)$.
3) Recurse on both $\boldsymbol{L}_{[V_2,V_2]}$ and H^{V_1}.

However, in this case, the accumulation of error is too rapid for yielding a good approximation of determinants. Instead, it becomes necesary to track the accumulation of variance during all recursive calls. Formally, the cost of sparsifying so that the variance is at most δ is about $n^2\delta^{-1}$, where δ is the size of the problem. This means that for a problem on G_i of size $\beta_i n$ for $0 \leq \beta_i \leq 1$, we can afford an error of $\beta_i\delta$ when working with it, since:

1) The sum of β_i on any layer is at most 2, [5] so the sum of variance per layer is $O(\delta)$.
2) The cost of each sparsification step is now $\beta_i n^2\delta^{-1}$, which sums to about $n^2\delta^{-1}$ per layer.

Our random spanning tree sampling algorithm in the full version is similarly based on this careful accounting of variance.

IV. Determinant Preserving Sparsification

In this section we give some of the key steps in proving Theorem 1 (proven in the full version), our primary result regarding determinant-preserving sparsification. The main step is proving the following general determinant-preserving sparsification routine that also forms the core of subsequent algorithms:

Theorem 18. *Given an undirected, weighted graph $G = (V, E, \boldsymbol{w})$, an error threshold $\epsilon > 0$, parameter ρ along with routines:*

1) $\textsc{SampleEdge}_G()$ *that samples an edge e from a probability distribution $\boldsymbol{p}$ ($\sum_e \boldsymbol{p}_e = 1$), as well as returning the corresponding value of $\boldsymbol{p}_e$. Here $\boldsymbol{p}_e$ must satisfy:*

$$\frac{\overline{\boldsymbol{\tau}}_e}{n-1} \leq \rho \cdot \boldsymbol{p}_e$$

where $\overline{\boldsymbol{\tau}}_e$ is the true leverage score of e in G.
2) $\textsc{ApproxLeverage}_G(u, v, \epsilon)$ *that returns the leverage score of an edge u, v in G to an error of ϵ. Specifically, given an edge e, it returns a value $\widetilde{\boldsymbol{\tau}}_e$ such that:*

$$(1-\epsilon)\,\overline{\boldsymbol{\tau}}_e \leq \widetilde{\boldsymbol{\tau}}_e \leq (1+\epsilon)\,\overline{\boldsymbol{\tau}}_e.$$

There is a routine $\textsc{DetSparsify}(G, s, \epsilon)$ that computes a graph H with s edges such that its tree count, $\mathcal{T}_H$, satisfies:

$$\mathbb{E}_H\left[\mathcal{T}_H\right] = \mathcal{T}_G\left(1 \pm O\left(\frac{n^3}{s^2}\right)\right),$$

and:

$$\frac{\mathbb{E}_H\left[\mathcal{T}^2_H\right]}{\mathbb{E}_H\left[\mathcal{T}_H\right]^2} \leq \exp\left(\frac{\epsilon^2 n^2}{s} + O\left(\frac{n^3}{s^2}\right)\right)$$

Furthermore, the expected running time is bounded by:

1) $O(s \cdot \rho)$ *calls to* $\textsc{SampleEdge}_G(e)$ *and* $\textsc{ApproxLeverage}(e)$ *with constant error,*
2) $O(s)$ *calls to* $\textsc{ApproxLeverage}(e)$ *with ϵ error.*

We establish all guarantees for this algorithm in the full version, and only show here that the concentration bounds as sketched in Section III holds for approximate leverage scores in Section IV-A.

A. Concentration Bound with Approximately Uniform Leverage Scores

Similar to the simplified proof as outlined in Section III, our proofs relied on uniformly sampling s edges from a multigraph with $m \geq \frac{s^2}{n}$ edges, such that all edges have leverage score within multiplicative $1 \pm \epsilon$ of $\frac{n-1}{s}$, aka. approximately uniform. The bound that we prove is an analog of Lemma 14

Lemma 19. *Given a weighted multi-graph G such that $m \geq \frac{s^2}{n}$, $s \geq n$, and all edges $e \in E$ have $\frac{(1-\epsilon)(n-1)}{m} \leq \overline{\boldsymbol{\tau}}_e \leq \frac{(1+\epsilon)(n-1)}{m}$, with $0 \leq \epsilon < 1$, then*

$$\frac{\mathbb{E}_H\left[\mathcal{T}^2_H\right]}{\mathbb{E}_H\left[\mathcal{T}_H\right]^2} \leq \exp\left(\frac{n^2\epsilon^2}{s} + O\left(\frac{n^3}{s^2}\right)\right)$$

Similar to the proof of Lemma 14 in Section III, we can utilize the bounds on the probability of k edges being chosen using Lemma 10. The only assumption that changed was the bounds on $\overline{\boldsymbol{\tau}}_e$, which does not affect $\mathbb{E}_H\left[\mathcal{T}_H\right]^2$. The only term that changes is our upper bound the total weight of trees that contain some subset of k edges that was the produce of k leverage scores. At a glance, this product can change by a factor of up to $(1+\epsilon)^k$, which when substituted naively into the proof of Lemma 10 directly would yield an additional term of

$$\exp\left(\frac{n^2\epsilon}{s}\right),$$

[5] each recursive call may introduce one new vertex

and in turn necessitating $\epsilon < n^{-1/2}$ for a sample count of $s \approx n^{1.5}$.

However, note that this is the worst case distortion over a subset F. The upper bound that we use, Lemma 13 sums over these bounds over all subsets, and over all edges we still have

$$\sum_{e \in G} \overline{\boldsymbol{\tau}}_e = n - 1.$$

Incorporating this allows us to show a tighter bound that depends on ϵ^2.

Similar to the proof of Lemma 13, we can regroup the summation over all $\binom{m}{k}$ subsets of $E(G)$, and bound the fraction of trees containing each subset F via $\sum_{T:F \subseteq T} \boldsymbol{w}(T) \leq \mathcal{T}_G \prod_{e \in F} \boldsymbol{\tau}_e$ via Lemma 12.

$$\sum_{\substack{T_1, T_2 \\ |T_1 \cap T_2| = k}} \boldsymbol{w}(T_1) \cdot \boldsymbol{w}(T_2) \leq \sum_{\substack{F \subseteq E \\ |F| = k}} \mathcal{T}_G^2 \prod_{e \in F} \overline{\boldsymbol{\tau}}_e^2$$

The proof heavily uses the identity $\sum_{e \in E} \overline{\boldsymbol{\tau}}_e = n - 1$. We bound this in first two steps: first treat it as a symmetric product over $\overline{\boldsymbol{\tau}}_e^2$, and bound the total as a function of

$$\sum_e \overline{\boldsymbol{\tau}}_e^2,$$

then we bound this sum using the fact that $\sum_e \overline{\boldsymbol{\tau}}_e = n - 1$.

The first step utilizes the concavity of the product function, and bound the total by the sum:

Lemma 20. *For any set of non-negative values $\boldsymbol{x}_1 \dots \boldsymbol{x}_m$ with $\sum_i \boldsymbol{x}_i \leq z$, we have*

$$\sum_{\substack{F \subseteq [1 \dots m] \\ |F| = k}} \prod_{i \in F} \boldsymbol{x}_i \leq \binom{m}{k} \left(\frac{z}{m}\right)^k.$$

Proof. We claim that this sum is maximized when $\boldsymbol{x}_i = \left(\frac{z}{m}\right)$ for all e.

Consider fixing all variables other than some $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, which we assume to be $\boldsymbol{x}_1 \leq \boldsymbol{x}_2$ without loss of generality as the function is symmetric on all variables:

$$\sum_{\substack{F \subseteq [1 \dots m] \\ |F| = k}} \prod_{i \in F} \boldsymbol{x}_i = \boldsymbol{x}_1 \boldsymbol{x}_2 \left(\sum_{\substack{F \subseteq [3 \dots m] \\ |F| = k-2}} \prod_{i \in F} \boldsymbol{x}_i \right) + (\boldsymbol{x}_1 + \boldsymbol{x}_2) \cdot \left(\sum_{\substack{F \subseteq [3 \dots m] \\ |F| = k-1}} \prod_{i \in F} \boldsymbol{x}_i \right) + \sum_{\substack{F \subseteq [3 \dots m] \\ |F| = k}} \prod_{i \in F} \boldsymbol{x}_i.$$

Then if $\boldsymbol{x}_1 < \boldsymbol{x}_2$, locally changing their values to $\boldsymbol{x}_1 + \epsilon$ and $\boldsymbol{x}_2 - \epsilon$ keeps the second term the same. While the first term becomes

$$(\boldsymbol{x}_1 + \epsilon)(\boldsymbol{x}_2 - \epsilon) = \boldsymbol{x}_1 \boldsymbol{x}_2 + \epsilon(\boldsymbol{x}_2 - \boldsymbol{x}_1) - \epsilon^2,$$

which is greater than $\boldsymbol{x}_1 \boldsymbol{x}_2$ when $0 < \epsilon < (\boldsymbol{x}_2 - \boldsymbol{x}_1)$.

This shows that the overall summation is maximized when all $\boldsymbol{x}_i$ are equal, aka

$$\boldsymbol{x}_i = \frac{z}{m},$$

which upon substitution gives the result. □

The second step is in fact the $k = 1$ case of Lemma 13.

Lemma 21. *For any set of values $\boldsymbol{y}_e$ such that*

$$\sum_e \boldsymbol{y} = n - 1,$$

and

$$\frac{(1-\epsilon)\, n}{m} \leq \boldsymbol{y}_e \leq \frac{(1+\epsilon)\, n}{m},$$

we have

$$\sum_e \boldsymbol{y}_e^2 \leq \frac{(1+\epsilon^2)(n-1)^2}{m}.$$

Proof. Note that for any $a \leq b$, and any ϵ, we have

$$(a - \epsilon)^2 + (b + \epsilon)^2 = a^2 + b^2 + 2\epsilon^2 + 2\epsilon(b - a),$$

and this transformation must increase the sum for $\epsilon > 0$. This means the sum is maximized when half of the leverage scores are $\frac{(1-\epsilon)(n-1)}{m}$ and the other half are $\frac{(1+\epsilon)(n-1)}{m}$. This then gives

$$\sum_{e \in E} \boldsymbol{y}_e^2 \leq \frac{m}{2} \left(\frac{(1+\epsilon)(n-1)}{m} \right)^2 + \frac{m}{2} \left(\frac{(1-\epsilon)(n-1)}{m} \right)^2 = \frac{(1+\epsilon^2)(n-1)^2}{m}.$$

□

Proof. (of Lemma 19)

We first derive an analog of Lemma 13 for bounding the total weights of pairs of trees containing subsets of size k, where we again start with the bounds

$$\sum_{\substack{T_1, T_2 \\ |T_1 \cap T_2| = k}} \boldsymbol{w}(T_1) \cdot \boldsymbol{w}(T_2) \leq \sum_{\substack{F \subseteq E \\ |F| = k}} \sum_{\substack{T_1, T_2 \\ F \subseteq T_1 \cap T_2}} \boldsymbol{w}(T_1) \cdot \boldsymbol{w}(T_2) = \sum_{\substack{F \subseteq E \\ |F| = k}} \left(\sum_{T : F \subseteq T} \boldsymbol{w}(T) \right)^2$$

Applying Lemma 12 to the inner term of the summation then gives

$$\sum_{\substack{T_1, T_2 \\ |T_1 \cap T_2| = k}} \boldsymbol{w}(T_1) \cdot \boldsymbol{w}(T_2) \leq \sum_{\substack{F \subseteq E \\ |F| = k}} \mathcal{T}_G^2 \cdot \prod_{e \in F} \overline{\boldsymbol{\tau}}_e^2$$

The bounds on $\overline{\boldsymbol{\tau}}_e$ and $\sum_e \overline{\boldsymbol{\tau}}_e = n - 1$ gives, via Lemma 21

$$\sum_e \overline{\boldsymbol{\tau}}_e^2 \leq \frac{(1+\epsilon^2)(n-1)^2}{m}.$$

Substituting this into Lemma 20 with $\boldsymbol{x}_i = \overline{\boldsymbol{\tau}}_e^2$ then gives

$$\sum_{\substack{F\subseteq E\\|F|=k}}\prod_{e\in F}\overline{\boldsymbol{\tau}}_e^2 \le \binom{m}{k}\left(\frac{(1+\epsilon^2)n^2}{m^2}\right)^k \le \frac{m^k}{k!}\left(\frac{(1+\epsilon^2)n^2}{m^2}\right)^k = \frac{1}{k!}\left(\frac{(1+\epsilon^2)n^2}{m}\right)^k.$$

which implies our analog of Lemma 13

$$\sum_{\substack{T_1,T_2\\|T_1\cap T_2|=k}}\boldsymbol{w}(T_1)\cdot\boldsymbol{w}(T_2) \le \mathcal{T}_G^2\cdot\frac{1}{k!}\left(\frac{(1+\epsilon^2)n^2}{m}\right)^k.$$

We can then duplicate the proof of Lemma 14. Similar to that proof, we can regroup the summation by $k = |T_1 \cap T_2|$ and invoking Lemma 10 to get:

$$\mathbb{E}_H\left[\mathcal{T}_H^2\right] \le \sum_{k=0}^{n-1}\sum_{\substack{T_1,T_2\\|T_1\cap T_2|=k}}\boldsymbol{w}(T_1)\cdot\boldsymbol{w}(T_2)\cdot p^{2n-2} \cdot\exp\left(-\frac{2n^2}{s}\right)\left(\frac{1}{p}\left(1+\frac{2n}{s}\right)\right)^k.$$

where $p = s/m$. When incorporated with our analog of Lemma 13 gives:

$$\mathbb{E}_H\left[\mathcal{T}_H^2\right] \le \sum_{k=0}^{n-1} p^{2n-2}\exp\left(-\frac{2n^2}{s}\right)\left(\frac{1}{p}\left(1+\frac{2n}{s}\right)\right)^k \cdot\mathcal{T}_G^2\frac{1}{k!}\left(\frac{(1+\epsilon^2)n^2}{m}\right)^k$$
$$= \mathcal{T}_G^2 p^{2n-2}\cdot\exp\left(-\frac{2n^2}{s}\right)\cdot\sum_{k=0}^{n-1}\frac{\left(\frac{(1+\epsilon^2)n^2}{s}\left(1+\frac{2n}{s}\right)\right)^k}{k!}.$$

Substituting in the Taylor expansion of $\sum_k \frac{z^k}{k!} \le \exp(z)$ then leaves us with:

$$\mathbb{E}_H\left[\mathcal{T}_H^2\right] \le \mathcal{T}_G^2\cdot p^{2n-2}\cdot\exp\left(-\frac{n^2}{s}+\frac{n^2\epsilon^2}{s}+O\left(\frac{n^3}{s^2}\right)\right)$$

and finishes the proof. □

B. Implicit Sparsification of the Schur Complement

To obtain speedups in our applications, we also need access to a fast determinant-preserving sparsification routine for Schur complement. This will first require finding $(1+\alpha)$-diagonally-dominant subsets.

Definition 22. In a weighted graph $G = (V, E, \boldsymbol{w})$, a subset of vertices $V_2 \subseteq V$ is $(1+\alpha)$-diagonally-dominant, or $(1+\alpha)$-DD if for every $u \in V_2$ with weighted degree $\boldsymbol{d}_u$ we have:

$$\sum_{v\sim u, v\notin V_2}\boldsymbol{w}_{uv} \ge \frac{1}{1+\alpha}\boldsymbol{d}_u = \frac{1}{1+\alpha}\sum_{v\sim u}\boldsymbol{w}_{uv}.$$

It was shown in [35] that large sets of such vertices can be found by trimming a uniformly random sample.

Lemma 23. *(Lemma 3.5. of [35] instantiated on graphs) There is a routine* ALMOSTINDEPENDENT(G,α) *that for a graph G with n vertices, and a parameter $\alpha \ge 0$, returns in $O(m)$ expected time a subset V_2 with $|V_2| \ge n/(8(1+\alpha))$ such that $\boldsymbol{L}_{G,[V_2,V_2]}$ is $(1+\alpha)$-DD.*

Given such a subset V_2, we then proceed to sample edges in $\text{SC}(G, V_1)$ via the following simple random walk sampling algorithm:

1) Pick a random edge in G.
2) Extend both of its endpoints in random walks until they first reach somewhere in V_1.

Incorporating this scheme into the determinant preserving sparsification schemes then leads these guarantees (proven in the full version):

Theorem 24. *Conditioned on Lemma 7 holding, there is a procedure* SCHURSPARSE *that takes a graph G, and an 1.1-DD subset of vertices V_2, returns a graph H^{V_1} in $\tilde{O}(n^2\delta^{-1})$ expected time such that the distribution over H^{V_1} satisfies:*

$$\mathcal{T}_{\text{SC}(G,V_1)}\exp(-\delta) \le \mathbb{E}_{H^{V_1}}\left[\mathcal{T}_{H^{V_1}}\right] \le \mathcal{T}_{\text{SC}(G,V_1)}\exp(\delta),$$

and

$$\frac{\mathbb{E}_{H^{V_1}}\left[\mathcal{T}_{H^{V_1}}^2\right]}{\mathbb{E}_{H^{V_1}}\left[\mathcal{T}_{H^{V_1}}\right]^2} \le \exp(\delta).$$

Furthermore, the number of edges of H^{V_1} can be set to anywhere between $O(n^{1.5}\delta^{-1})$ and $O(n^2\delta^{-1})$ without affecting the final bound.

V. APPROXIMATE DETERMINANT OF SDDM MATRICES

In this section, we provide an algorithm for computing an approximate determinant of SDDM matrices, which are minors of graph Laplacians formed by removing one row/column.

Theorem 1 allows us to sparsify a dense graph while still approximately preserving the determinant of the graph minor. If there were some existing algorithm for computing the determinant that had good dependence on sparsity, we could achieve an improved runtime for determinant computation by simply invoking such an algorithm on a minor of the sparsified graph.[6] Unfortunately, current determinant computation algorithms (that achieve high-accuracy) are only dependent on n, so simply reducing the edge count does not directly improve the runtime for determinant computation. Instead the algorithm we give will utilize Fact 8

$$\det_+(\boldsymbol{L}) = \det\left(\boldsymbol{L}_{[V_2,V_2]}\right)\cdot\det_+(\text{SC}(\boldsymbol{L}, V_1)).$$

(where we recall that $\det_+$ is the determinant of the matrix minor) to recursively split the matrix. Specifically, we partition the vertex set based upon the routine ALMOSTINDEPENDENT from Lemma 23, then compute Schur complements according to SCHURSPARSE in Theorem 24. Our algorithm will take as input a Laplacian matrix. However, this recursion naturally produces two matrices, the second of which is a Laplacian and

[6]To get with high probability one could use standard boosting tricks involving taking the median of several estimates of the determinant obtained in this fashion.

the first of which is a submatrix of a Laplacian. Therefore, we need to convert $\boldsymbol{L}_{[V_2,V_2]}$ into a Laplacian. We do this by adding one vertex with appropriate edge weights such that each row and column sums to 0. Pseudocode of this routine is in Algorithm 1, and we call it with the parameters $\boldsymbol{L}^{V_2} \leftarrow \text{ADDROWCOLUMN}(\boldsymbol{L}_{[V_2,V_2]})$.

Algorithm 1: ADDROWCOLUMN($\boldsymbol{M}$) : complete $\boldsymbol{M}$ into a graph Laplacian by adding one more row/column

Input: SDDM Matrix $\boldsymbol{M}$
Output: Laplacian matrix $\boldsymbol{L}$ with one extra row / column than $\boldsymbol{M}$

Let n be the dimension of $\boldsymbol{M}$;
for $i = 1$ *to* n **do**
  Sum non-zero entries of row i, call $\boldsymbol{s}_i$;
  Set $\boldsymbol{L}(n+1,i), \boldsymbol{L}(i,n+1) \leftarrow -\boldsymbol{s}_i$;
Let $\boldsymbol{L}(n+1,n+1) \leftarrow \sum_{i=1}^{n} \boldsymbol{s}_i$;
Output $\boldsymbol{L}$;

The procedure ADDROWCOLUMN outputs a Laplacian $\boldsymbol{L}^{V_2}$ such that $\boldsymbol{L}_{[V_2,V_2]}$ can be obtained if one removes this added row/column. This immediately gives $\det_+(\boldsymbol{L}^{V_2}) = \det(\boldsymbol{L}_{[V_2,V_2]})$ by definition, and we can now give our determinant computation algorithm of the minor of a graph Laplacian.

Algorithm 2: DETAPPROX($\boldsymbol{L}, \delta, \overline{n}$) : Compute $\det_+(\boldsymbol{L})$ with error parameter δ

Input: Laplacian matrix $\boldsymbol{L}$, top level error threshold δ, and top level graph size $\overline{n}$
Output: Approximate $\det_+(\boldsymbol{L})$

if *this is the top-level invocation of this function in the recursion tree* **then**
  $\delta' \leftarrow \Theta(\delta^2 / \log^3 n)$
else
  $\delta' \leftarrow \delta$
if $\boldsymbol{L}$ *is* 2×2 **then**
  return *the weight on the (unique) edge in the graph*
$V_2 \leftarrow \text{ALMOSTINDEPENDENT}(\boldsymbol{L}, \frac{1}{10})$ {Via Lemma 23}
$V_1 \leftarrow V \setminus V_2$;
$\boldsymbol{L}^{V_1} \leftarrow \text{SCHURSPARSE}(\boldsymbol{L}, V_1, \delta')$; {$|V_1|/\overline{n}$ is the value of β in Lemma 25.}
$\boldsymbol{L}^{V_2} \leftarrow \text{ADDROWCOLUMN}(\boldsymbol{L}_{[V_2,V_2]})$;
Output $\text{DETAPPROX}(\boldsymbol{L}^{V_1}, \delta' |V_1| / \overline{n}, \overline{n}) \cdot \text{DETAPPROX}(\boldsymbol{L}^{V_2}, \delta' |V_2| / \overline{n}, \overline{n})$;

Our analysis of this recursive routine consists of bounding the distortions incurred at each level of the recursion tree. This in turn uses the fact that the number of vertices across all calls within a level and the total "amount" of δ across all calls within a level both remain unchanged from one level to the next. This can be summarized by the following Lemma which bounds the error accumulated within one level of recursion in our algorithm.

Lemma 25. *Suppose we are given some small $\delta \geq 0$ and non-negative $\beta_1, ..., \beta_k$ such that $\sum_{i=1}^{k} \beta_i = O(1)$, along with Laplacian matrices $\boldsymbol{L}(1), \ldots, \boldsymbol{L}(k)$ and each having a corresponding vertex partition $V_1(i), V_2(i)$, where*

$$\boldsymbol{L}(i) = \begin{bmatrix} \boldsymbol{L}(i)_{[V_1(i),V_1(i)]} & \boldsymbol{L}(i)_{[V_1(i),V_2(i)]} \\ \boldsymbol{L}(i)_{[V_2(i),V_1(i)]} & \boldsymbol{L}(i)_{[V_2(i),V_2(i)]} \end{bmatrix}.$$

Let $\boldsymbol{L}^{V_1(i)}$ denote the result of running SCHURSPARSE *to remove the $V_2(i)$ block in each of these matrices:*[7]

$$\boldsymbol{L}^{V_1(i)} \stackrel{\text{def}}{=} \text{SCHURSPARSE}\left(\boldsymbol{L}(i), V_1(i), \beta_i \delta\right).$$

Then conditioning upon a with high probability event[8] *in each of these calls to* SCHURSPARSE*, for any p we have with probability at least $1 - p$:*

$$\prod_{i=1}^{k} \det_+\left(\boldsymbol{L}(i)\right) =$$
$$\left(1 \pm O\left(\sqrt{\delta/p}\right)\right) \prod_{i=1}^{k} \det\left(\boldsymbol{L}_{[V_2(i),V_2(i)]}(i)\right) \det_+(\boldsymbol{L}^{V_1(i)}).$$

Here the β_i corresponds to the $|V_1|/n$ and $|V_2|/n$ values that δ is multiplied against in each call parameter to SCHURSPARSE.

Applying Lemma 25 to all the layers of the recursion tree gives the overall guarantees.

Proof of Theorem 2.

Running Time: Let the number of vertices and edges in the current graph corresponding to $\boldsymbol{L}$ be n and m respectively. Calling ALMOSTINDEPENDENT takes expected time $O(m)$ and guarantees

$$\frac{n}{16} \leq |V_2| \leq \frac{n}{8},$$

which means the total recursion terminates in $O(\log n)$ steps.

For the running time, note that as there are at most $O(n)$ recursive calls, the total number of vertices per level of the recursion is $O(n)$. The running time on each level are also dominated by the calls to SCHURSPARSE, which comes out to

$$\tilde{O}\left(|V_1(i)|^2 \frac{n}{\delta' |V_1(i)|}\right) = \tilde{O}\left(|V_1(i)| n \delta^{-2}\right),$$

and once again sums to $\tilde{O}(n^2\delta^{-2})$. We note that this running time can also be obtained from more standard analyses of recursive algorithms, specifically applying guess-and-check to a running time recurrence of the form of:

$$T(n,\delta) = T(\theta n, \theta\delta) + T((1-\theta)n + 1, (1-\theta)\delta) + \tilde{O}(n^2\delta^{-1}).$$

[7]This Lemma only applies when the matrices are fixed with respect to the randomness used in the invocations of SCHURSPARSE mentioned in the Lemma. In other words, it only applies when the result of running SCHURSPARSE on each of these $\boldsymbol{L}(i)$ matrices is independent of the result of running it on the other matrices. This is why the Lemma only immediately bounds error within a level of the recursion—where this independence holds—rather than for the entire algorithm.

[8]namely, the event that all the leverage score estimation calls to Lemma 7 from SCHURSPARSE succeed

Correctness. As shown in the running time analysis, our recursion tree has depth at most $O(\log n)$, and there are at most $O(n)$ total vertices at any given level. We associate each level of the recursion in our algorithm with the list of matrices which are given as input to the calls making up that level of recursion. For any level in our recursion, consider the product of $\det_+$ applied to each of these matrices. We refer to this quantity for level j as q_j. Notice that q_0 is the determinant we wish to compute and $q_{\#\text{ levels}-1}$ is what our algorithm actually outputs. As such, it suffices to prove that for any j, $q_j = (1 \pm \frac{\delta}{\#\text{ levels}})q_{j-1}$ with probability of failure at most $\frac{1}{10 \cdot \#\text{ levels}}$. However, by the fact that we set $\delta' = \Theta(\delta^2/\log^3 n)$ in the top level of recursion with sufficiently small constants, this immediately follows from Lemma 25.

A minor technical issue is that Lemma 25 only gives guarantees conditioned on a WHP event. However, we only need to invoke this Lemma a logarithmic number of times, so we can absorb this polynomially small failure probability into the our total failure probability without issue.

Standard boosting techniques—such as running $O(\log n)$ independent instances and taking the medians of the estimates—give our desired with high probability statement. □

It remains to bound the variances per level of the recursion.

Proof. (Of Lemma 25) As a result of Fact 8

$$\prod_{i=1}^{k} \det_+ \left(\boldsymbol{L}(i)\right) = \prod_{i=1}^{k} \det\left(\boldsymbol{L}(i)_{[V_2(i),V_2(i)]}\right) \det_+ \left(\textsc{Sc}\left(\boldsymbol{L}(i), V_1(i)\right)\right).$$

Consequently, it suffices to show that with probability at least $1 - p$

$$\prod_{i=1}^{k} \det_+ \left(\textsc{Sc}\left(\boldsymbol{L}(i), V_1(i)\right)\right) = \left(1 \pm O\left(\sqrt{\delta/p}\right)\right) \prod_{i=1}^{k} \det_+ \left(\boldsymbol{L}^{V_1(i)}\right).$$

Recall that $\boldsymbol{L}^{V_1(i)}$ denotes the random variable that is the approximate Schur complement generated through the call to $\textsc{SchurSparse}(\boldsymbol{L}(i), V_1(i), \beta_i\delta)$.

Using the fact that our calls to SCHURSPARSE are independent along with the assumption of $\sum_{i=1}^{k} \beta_i = O(1)$, we can apply the guarantees of Theorem 24 to obtain

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{L}^{V_1(1)} \ldots \boldsymbol{L}^{V_1(k)}} \left[\prod_{i=1}^{k} \det_+ \left(\boldsymbol{L}^{V_1(1)}\right)\right] \\
&\quad = \prod_{i=1}^{k} \mathbb{E}_{\boldsymbol{L}^{V_1(i)}} \left[\det_+ \boldsymbol{L}^{V_1(i)}\right] \\
&\quad = \left(1 \pm O\left(\delta\right)\right) \prod_{i=1}^{k} \det_+ \left(\textsc{Sc}\left(\boldsymbol{L}(i), V_1(i)\right)\right),
\end{aligned}$$

and

$$\begin{aligned}
&\frac{\mathbb{E}_{\boldsymbol{L}^{V_1(1)} \ldots \boldsymbol{L}^{V_1(k)}} \left[\prod_{i=1}^{k} \det_+ \left(\boldsymbol{L}^{V_1(i)}\right)^2\right]}{\mathbb{E}_{\boldsymbol{L}^{V_1(1)} \ldots \boldsymbol{L}^{V_1(k)}} \left[\prod_{i=1}^{k} \det_+ \left(\boldsymbol{L}^{V_1(i)}\right)^2\right]} \\
&\quad = \prod_{i=1}^{k} \frac{\mathbb{E}_{\boldsymbol{L}^{V_1(i)}} \left[\det_+ \left(\boldsymbol{L}^{V_1(i)}\right)^2\right]}{\mathbb{E}_{\boldsymbol{L}^{V_1(i)}} \left[\det_+ \left(\boldsymbol{L}^{V_1(i)}\right)\right]^2} \\
&\quad \leq \prod_{i=1}^{k} \exp\left(O\left(\beta_i\delta\right)\right) \leq \exp\left(O\left(\delta\right)\right).
\end{aligned}$$

By assumption δ is small, so we can approximate $\exp\left(O(\delta)\right)$ with $1 + O(\delta)$, which with bound above gives

$$\begin{aligned}
&\mathbf{Var}_{\boldsymbol{L}^{V_1(1)} \ldots \boldsymbol{L}^{V_1(k)}} \left[\prod_{i=1}^{k} \det_+ \left(\boldsymbol{L}^{V_1(i)}\right)\right] \\
&\quad \leq O\left(\delta\right) \mathbb{E}_{\boldsymbol{L}^{V_1(1)} \ldots \boldsymbol{L}^{V_1(k)}} \left[\prod_{i=1}^{k} \det_+ \left(\boldsymbol{L}^{V_1(i)}\right)\right]^2,
\end{aligned}$$

Then applying the approximation on $\mathbb{E}\left[\prod_{i=1}^{k} \det_+ \left(\textsc{SchurSparse}(\boldsymbol{L}(i), V_1(i), \beta_i\delta)\right)\right]$ gives

$$\begin{aligned}
&\mathbf{Var}_{\boldsymbol{L}^{V_1(1)} \ldots \boldsymbol{L}^{V_1(k)}} \left[\prod_{i=1}^{k} \det_+ \left(\boldsymbol{L}^{V_1(i)}\right)\right] \\
&\quad \leq O\left(\delta\right) \left(\prod_{i=1}^{k} \det_+ \left(\textsc{Sc}\left(\boldsymbol{L}(i), V_1(i)\right)\right)\right)^2.
\end{aligned}$$

At which point we can apply Chebyshev's inequality to obtain our desired result.

□

VI. Acknowledgment

David Durfee was supported by NSF Grant No. 1718533. John Peebles was supported by the NSF Graduate Research Fellowship under Grant No. 1122374, and by the NSF under Grant No. 1065125 and 1565235. Richard Peng was supported by NSF Grant No. 1718533. Anup B. Rao was supported by the NSF under Grant CCF-1563838; most of this research was done while Anup B. Rao was at Georgia Tech.

References

[1] G. Kirchhoff, "Über die auflösung der gliechungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird," in *Poggendorgs Ann. Phys. Chem.*, 1847, pp. 497–508.

[2] D. A. Spielman and S.-H. Teng, "Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems," *SIAM Journal on Matrix Analysis and Applications*, vol. 35, no. 3, pp. 835–885, 2014, available at http://arxiv.org/abs/cs/0607105.

[3] N. Goyal, L. Rademacher, and S. Vempala, "Expanders via random spanning trees," in *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '09. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2009, pp. 576–585. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496770.1496834

[4] A. Asadpour, M. X. Goemans, A. Madry, S. O. Gharan, and A. 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*, ser. SODA '10. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2010, pp. 379–389. [Online]. Available: http://dl.acm.org/citation.cfm?id=1873601.1873633

[5] W. S. Fung, R. Hariharan, N. J. Harvey, and D. Panigrahi, "A general framework for graph sparsification," in *Proceedings of the forty-third annual ACM symposium on Theory of computing*. ACM, 2011, pp. 71–80, https://arxiv.org/abs/1004.4080.

[6] C. Boutsidis, P. Drineas, P. Kambadur, and A. Zouzias, "A randomized algorithm for approximating the log determinant of a symmetric positive definite matrix," *CoRR*, vol. abs/1503.00374, 2015, available at: http://arxiv.org/abs/1503.00374.

[7] T. Hunter, A. E. Alaoui, and A. M. Bayen, "Computing the log-determinant of symmetric, diagonally dominant matrices in near-linear time," *CoRR*, vol. abs/1408.1693, 2014, available at: http://arxiv.org/abs/1408.1693.

[8] I. Han, D. Malioutov, and J. Shin, "Large-scale log-determinant computation through stochastic chebyshev expansions." in *ICML*, 2015, pp. 908–917, available at: https://arxiv.org/abs/1606.00942.

[9] J. Kelner and A. Madry, "Faster generation of random spanning trees," in *Proceedings of the 50th annual Symposium on Foundations of Computer Science, FOCS 2009*, 2009, pp. 13–21, available at https://arxiv.org/abs/0908.1448.

[10] A. Madry, D. Straszak, and J. 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*, 2015, pp. 2019–2036, available at http://arxiv.org/pdf/1501.00267v1.pdf.

[11] D. Durfee, R. Kyng, J. Peebles, A. B. Rao, and S. Sachdeva, "Sampling random spanning trees faster than matrix multiplication," *CoRR*, vol. abs/1611.07451, 2016.

[12] S. Janson, "The numbers of spanning trees, hamilton cycles and perfect matchings in a random graph," *Combinatorics, Probability and Computing*, vol. 3, no. 01, pp. 97–126, 1994.

[13] D. A. Spielman and N. Srivastava, "Graph sparsification by effective resistances," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1913–1926, 2011.

[14] J. A. Tropp, "User-friendly tail bounds for sums of random matrices," *Found. Comput. Math.*, vol. 12, no. 4, pp. 389–434, Aug. 2012, available at http://arxiv.org/abs/1004.4389. [Online]. Available: http://dx.doi.org/10.1007/s10208-011-9099-z

[15] M. B. Cohen and R. Peng, "ℓ_p row sampling by Lewis weights," in *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, ser. STOC '15. New York, NY, USA: ACM, 2015, pp. 183–192, available at http://arxiv.org/abs/1412.0588. [Online]. Available: http://doi.acm.org/10.1145/2746539.2746567

[16] M. B. Cohen, "Nearly tight oblivious subspace embeddings by trace inequalities," in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2016, pp. 278–287.

[17] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, "Sparsification—a technique for speeding up dynamic graph algorithms," *J. ACM*, vol. 44, no. 5, pp. 669–696, Sep. 1997. [Online]. Available: http://doi.acm.org/10.1145/265910.265914

[18] A. A. Benczúr and D. R. Karger, "Approximating s-t minimum cuts in Õ(n2) time," in *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: ACM, 1996, pp. 47–55. [Online]. Available: http://doi.acm.org/10.1145/237814.237827

[19] D. A. Spielman and S.-H. Teng, "Spectral sparsification of graphs," *SIAM J. Comput.*, vol. 40, no. 4, pp. 981–1025, Jul. 2011. [Online]. Available: http://dx.doi.org/10.1137/08074489X

[20] M. B. Cohen, J. A. Kelner, J. Peebles, R. Peng, A. Rao, A. Sidford, and A. Vladu, "Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs," 2017, accepted to *STOC 2017*. Preprint available at https://arxiv.org/abs/1611.00755.

[21] W. Baur and V. Strassen, "The complexity of partial derivatives," *Theoretical Computer Science*, vol. 22, no. 3, pp. 317 – 330, 1983. [Online]. Available: http://www.sciencedirect.com/science/article/pii/030439758390110X

[22] I. C. F. Ipsen and D. J. Lee, "Determinant approximations," 2011.

[23] T. Hunter, A. E. Alaoui, and A. M. Bayen, "Computing the log-determinant of symmetric, diagonally dominant matrices in near-linear time," *CoRR*, vol. abs/1408.1693, 2014. [Online]. Available: http://arxiv.org/abs/1408.1693

[24] C. Boutsidis, P. Drineas, P. Kambadur, and A. Zouzias, "A randomized algorithm for approximating the log determinant of a symmetric positive definite matrix," *CoRR*, vol. abs/1503.00374, 2015. [Online]. Available: http://arxiv.org/abs/1503.00374

[25] A. Broder, "Generating random spanning trees," in *Proceedings of the 30th annual Symposium on Foundations of Computer Science, FOCS 1989*, 1989, pp. 442–447.

[26] D. Aldous, "The random walk construction of uniform spanning trees and uniform labelled trees," in *SIAM Journal on Discrete Mathematics*, 1990, pp. 450–465.

[27] A. Guenoche, "Random spanning tree," *Journal of Algorithms*, vol. 4, no. 3, pp. 214–220, 1983.

[28] V. G. Kulkarni, "Generating random combinatorial objects," *Journal of Algorithms*, vol. 11, no. 2, pp. 185–207, 1990.

[29] C. J. Colbourn, W. J. Myrvold, and E. Neufeld, "Two algorithms for unranking arborescences," *Journal of Algorithms*, vol. 20, no. 2, pp. 268–281, 1996.

[30] N. J. A. Harvey and K. Xu, "Generating random spanning trees via fast matrix multiplication," in *LATIN 2016: Theoretical Informatics*, vol. 9644, 2016, pp. 522–535.

[31] V. V. Williams, "Multiplying matrices faster than coppersmith-winograd," in *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '12. New York, NY, USA: ACM, 2012, pp. 887–898.

[32] N. K. Vishnoi, "Lx = b laplacian solvers and their algorithmic applications," 2012.

[33] R. Peng and D. A. Spielman, "An efficient parallel solver for SDD linear systems," in *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, ser. STOC '14. New York, NY, USA: ACM, 2014, pp. 333–342, available at http://arxiv.org/abs/1311.3286. [Online]. Available: http://doi.acm.org/10.1145/2591796.2591832

[34] D. Cheng, Y. Cheng, Y. Liu, R. Peng, and S. Teng, "Efficient sampling for Gaussian graphical models via spectral sparsification," *Proceedings of The 28th Conference on Learning Theory*, pp. 364–390, 2015, available at http://jmlr.org/proceedings/papers/v40/Cheng15.pdf.

[35] R. Kyng, Y. T. Lee, R. Peng, S. Sachdeva, and D. A. Spielman, "Sparsified cholesky and multigrid solvers for connection laplacians," in *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 2016, pp. 842–850, available at http://arxiv.org/abs/1512.01892.

[36] R. Burton and R. Pemantle, "Local characteristics, entropy and limit theorems for spanning trees and domino tilings via transfer-impedances," *The Annals of Probability*, pp. 1329–1371, 1993.