Solving SDD Linear Systems in Nearly $m \log^{1/2} n$ Time

Michael B. Cohen
M.I.T.[*]
micohen@mit.edu

Rasmus Kyng
Yale University[†]
rasmus.kyng@yale.edu

Gary L. Miller
Carnegie Mellon University[†]
glmiller@cs.cmu.edu

Jakub W. Pachocki
Carnegie Mellon University[‡]
pachocki@cs.cmu.edu

Richard Peng
M.I.T.[§]
rpeng@mit.edu

Anup B. Rao
Yale University[‖]
anup.rao@yale.edu

Shen Chen Xu
Carnegie Mellon University[**]
shenchex@cs.cmu.edu

ABSTRACT

We show an algorithm for solving symmetric diagonally dominant (SDD) linear systems with m non-zero entries to a relative error of ϵ in $O(m \log^{1/2} n \log \log^c n \log(1/\epsilon))$ time. Our approach follows the recursive preconditioning framework, which aims to reduce graphs to trees using iterative methods. We improve two key components of this framework: random sampling and tree embeddings. Both of these components are used in a variety of other algorithms, and our approach also extends to the dual problem of computing electrical flows.

We show that preconditioners constructed by random sampling can perform well without meeting the standard requirements of iterative methods. In the graph setting, this leads to ultra-sparsifiers that have optimal behavior in expectation.

The improved running time makes previous low stretch embedding algorithms the running time bottleneck in this framework. In our analysis, we relax the requirement of these embeddings to snowflake spaces. We then obtain a two-pass approach algorithm for constructing optimal embeddings in snowflake spaces that runs in $O(m \log \log n)$ time. This algorithm is also readily parallelizable.

Categories and Subject Descriptors

G.1.3 [**NUMERICAL ANALYSIS**]: Numerical Linear Algebra—*Sparse, structured, and very large systems (direct and iterative methods)*

General Terms

Algorithms,Theory

Keywords

combinatorial preconditioning, SDD linear systems, random matrices, tree embeddings

1. INTRODUCTION

Recently, solvers for linear systems in graph Laplacians have emerged as a powerful primitive for designing efficient algorithms. The ability to solve such systems in nearly-linear time [38] led to a variety of efficient algorithms [10, 17, 13, 22, 31, 12, 11, 29, 28] as well as studies of more efficient solver algorithms [25, 26, 23, 27, 33]. For an undirected graph $G = (V, E, \boldsymbol{w})$, with n vertices and m edges, the graph Laplacian is given by the $n \times n$ matrix with $O(m)$ non-zero entries:

$$\forall u \neq v: \quad \boldsymbol{L}_{G,uv} = -\boldsymbol{w}_{uv},$$
$$\forall u: \quad \boldsymbol{L}_{G,uu} = \sum_{v \neq u} \boldsymbol{w}_{uv}.$$

Solving linear systems in these matrices in turn leads to solutions to linear systems in symmetric diagonally dominant matrices [19] and symmetric M matrices [17]. This reduction also holds for approximate solvers [38], and in the presence of fixed point round-off errors [23]. As a result, algorithmic studies of this problem usually work with graph Laplacians.

The close connection between graph Laplacians and graphs allows the use of combinatorial methods to bound the convergence of numerical methods. This was first systematically studied by Vaidya [43], and played a central role in the de-

[†]Partially supported by AFOSR Award FA9550-12-1-0175.
[‖]Partially supported by NSF grant CCF-1111257.
[**]Supported in part under NSF grant CCF-1018463
[*]Part of this work was done while at CMU under NSF grants CCF-1018463 and CCF-1065106
[†]Supported in part under NSF grants CCF-1018463 and CCF-1065106
[‡]Supported in part under NSF grant CCF-1018463
[§]Part of this work was done while at CMU under NSF grant CCF-1018463 and by a Microsoft Research PhD Fellowship

STOC '14, May 31 - June 03 2014, New York, NY, USA

http://dx.doi.org/10.1145/2591796.2591833.

velopment of nearly-linear time solvers [9, 37, 38]. These algorithms led to a sophisticated framework consisting of:

1. Recursive preconditioning that reduces a solve to several ones on smaller, but similar graphs.
2. Ultra-sparsifiers that lead to such smaller graphs.
3. Tree embeddings that form backbones of ultra-sparsifiers.

Koutis et al. [25] gave a simpler construction of ultra-sparsifiers. Instead of using the trees to reroute the graph, they use the distortions of edges in the trees, known as stretch, to directly sample the graph. This gave a direct connection between the total stretch of off tree edges and the size of the problem. Existing results on finding low-stretch spanning trees [3, 18, 1, 2] then led to a running time of about $m \log^2 n$.

This approach of sampling by off-tree stretch has been extended in two directions: a more refined analysis in conjunction with the recursive preconditioning framework gives a running time of about $m \log n$ under exact arithmetic [26]. This overhead of $\log n$ is in fact the square root of a product of two log factors: one from tree embedding, and one from spectral sparsification. The square-root is in some sense natural: preconditioned Chebyshev iteration, which is at the heart of the recursive conditioning framework, requires about $\sqrt{\kappa}$ iterations to correct a multiplicative distortion of κ. This is also reflected by the fact that the nearly-optimal *ultra-sparsifiers* by Kolla et al. [24] shows the existence of solver circuits of size about $m \log^{1/2} n$ for any graph.

Kelner et al. [23] proposed a combinatorial approach for solvers based on repeatedly updating off-tree cycles. It replaces the iterative method with data structures, leading to a simple algorithm that's stable under fixed precision arithmetic. Improvements by Lee and Sidford [27] led to an algorithm that makes about $m \log^{1/2} n$ data structure calls. The faster convergence is obtained by showing that small steps made by sampling edges with probabilities proportional to stretch reduces the total error by a good amount *in expectation*. On the other hand, each of these steps updates a (possibly long) path in a tree, which with the data structures gives a $\log n$ factor overhead. One data structure that supports such operations is Top Trees [4], which are based on repeatedly eliminating low degree vertices. Such operations also underlie recursive preconditioning, leading to the hope that these two frameworks can be combined.

In this paper, we give an algorithm that removes both the overhead of spectral sparsification and data structure calls, leading to the following result:

Theorem 1.1. *Given a graph G with m edges, a vector $\boldsymbol{b} = \boldsymbol{L}_G \overline{\boldsymbol{x}}$, and any error $\epsilon > 0$, we can find w.h.p. a vector $\boldsymbol{x}$ such that*

$$\|\overline{\boldsymbol{x}} - \boldsymbol{x}\|_{\boldsymbol{L}_G} \leq \epsilon \|\overline{\boldsymbol{x}}\|_{\boldsymbol{L}_G},$$

in expected $O(m \log^{1/2} n \log\log^{3+\delta} n \log(\frac{1}{\epsilon}))$ time for any constant $\delta > 0$.

In Section 3, we give an overview of the recursive preconditioning framework and the modifications that we make. There were two main challenges that we addressed in order to obtain the faster algorithm. They cover fairly disjoint directions that are of independent interest, and are addressed in more details in two manuscripts on arXiv [15, 16]. We will only describe the overall algorithm and state key theorems here.

In [15], we prove a multi-edge version of the expected error reduction bound from [23]. It can also be viewed as a probabilistic version of matrix Chernoff bound that holds in expectation. This is discussed in detail in Section 4.

A common component in all the solver algorithms is a tree used to generate the sampling probabilities. The low running time of our algorithm precludes us from directly using existing tree constructions. In fact, the decomposition schemes in all efficient constructions of low-stretch spanning trees lead to a running time of about $m \log n$. Instead, we show that discounting stretch as in snowflake spaces suffices for the recursive preconditioning algorithm. This discounting allows us to treat some edge classes with coarser granularity. It leads to a two-pass approach for constructing these embeddings that runs in $O(m \log\log n)$ time. An outline of this algorithm is in Section 5, and it's described in detail in [16].

2. BACKGROUND

We begin by stating some of our key notations. Many of the tools that we use and modify have been studied before. As a result, our presentation combines (and as a result omits) several sets of notations that originated independently.

We will use standard linear algebraic notations. As graph Laplacians are not of full rank, we will use the pseudo-inverse of $\boldsymbol{L}_G$, $\boldsymbol{L}_G^\dagger$, to denote the inverse on its rank space. A symmetric matrix $\boldsymbol{A}$ is positive semi-definite if $\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \geq 0$ for all vectors $\boldsymbol{x}$. For such matrices, we can define the $\boldsymbol{A}$ norm of $\boldsymbol{x}$ as $\|\boldsymbol{x}\|_{\boldsymbol{A}} \stackrel{\text{def}}{=} \sqrt{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}$. We will describe matrix approximations using the Loewner ordering of matrices. For matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, we let $\boldsymbol{A} \preceq \boldsymbol{B}$ if $\boldsymbol{B} - \boldsymbol{A}$ is positive semidefinite.

A graph Laplacian can be written as a sum of rank-one matrices, one per edge. We will denote these matrices by $\boldsymbol{Y}_1 \ldots \boldsymbol{Y}_m$ and let $\boldsymbol{Y}_i = \boldsymbol{v}_i \boldsymbol{v}_i^T$. Graph approximations can be obtained by sampling and rescaling these matrices. Here a crucial quantity is the effective resistance which in its more general form is known as statistical leverage score. These quantities have wide applications in sampling the rows of matrices [14, 36, 42]. We will use a generalization of these measured w.r.t. a different matrix, specifically the graph Laplacian of a tree. For a matrix $\boldsymbol{X}$, the statistical leverage score of $\boldsymbol{Y}_i$ w.r.t. $\boldsymbol{X}$ is

$$\overline{\boldsymbol{\tau}}_{\boldsymbol{X}}(\boldsymbol{Y}_i) \stackrel{\text{def}}{=} \text{Tr}\left(\boldsymbol{X}^\dagger \boldsymbol{Y}_i\right).$$

Given any upper bound of these scores, $\boldsymbol{\tau}$, we can approximate a sum of positive semi-definite matrices by sampling them with probabilities proportional to $\boldsymbol{\tau}$, and rescaling them to maintain expectation. Pseudocode of this process is given in Figure 1. It is common in most random constructions of matrix approximations [34, 36, 42].

The combinatorial components of our algorithm rely on the connection between leverage scores and stretches of edges first observed in [40]. Stretch is more conveniently defined in terms of lengths, which are inverses of the weights, $\boldsymbol{l}(e) \stackrel{\text{def}}{=} \frac{1}{\boldsymbol{w}(e)}$. For a tree $T = (V_T, E_T, \boldsymbol{l}_T)$, the stretch of an edge $e = uv$ with respect to T is

$$\textbf{STR}_T(e) \stackrel{\text{def}}{=} \frac{\boldsymbol{l}_T(u, v)}{\boldsymbol{l}(e)} = \overline{\boldsymbol{\tau}}_{\boldsymbol{L}_T}(\boldsymbol{L}_e).$$

$\boldsymbol{Z}$ = SAMPLE($\{\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_m\}, \boldsymbol{X}, \boldsymbol{\tau}, \delta$), where $\boldsymbol{Y}_i = \boldsymbol{v}_i \boldsymbol{v}_i^T$ are rank one matrices, $\boldsymbol{\tau}_i$ are upper bounds of leverage scores, $\boldsymbol{\tau}_i \geq \overline{\boldsymbol{\tau}}_i$ for all i, and $\delta < 1$ is an arbitrary parameter.

1. Initialize $\boldsymbol{Z}$ to $\boldsymbol{X}$.
2. Let s be $\sum_{i=1}^{m} \boldsymbol{\tau}_i$ and $t = \delta^{-1} s$.
3. Pick an integer r uniformly at random in the interval $[t, 2t-1]$.
4. For $j = 1 \ldots r$
   (a) Sample entry i_j with probability proportional to $\boldsymbol{\tau}_{i_j}$.
   (b) $\boldsymbol{Z} \leftarrow \boldsymbol{Z} + \frac{\delta}{\boldsymbol{\tau}_{i_j}} Y_{i_j}$.
5. Return $\boldsymbol{Z}$.

Figure 1: Sampling Algorithm

3. OVERVIEW OF THE ALGORITHM

Our algorithm is recursive, but works on a different random sample at each recursive call. To analyze it, we inductively prove convergence towards solutions at the levels of the recursion. Here it is convenient to define an abstract graph Laplacian solver routine.

Definition 3.1. A routine SOLVER($\cdot$) is said to be a *Laplacian solver* when it takes as input $(G, T, \boldsymbol{\tau}, \boldsymbol{b}, \epsilon)$, where G is a graph, T a spanning tree of this graph, and $\boldsymbol{\tau}$ upper bounds on the combinatorial stretch of the off-tree edges of G w.r.t. T, and the routine returns as output a vector $\boldsymbol{x}$ such that

$$\left\| \boldsymbol{x} - \boldsymbol{L}_G^{\dagger} \boldsymbol{b} \right\|_{\boldsymbol{L}_G} \leq \epsilon \left\| \boldsymbol{L}_G^{\dagger} \boldsymbol{b} \right\|_{\boldsymbol{L}_G}.$$

As we will use total off tree stretch to bound graph sizes, it is convenient to view inputs as graph-tree tuples containing the graph G, the tree T, and upper bounds on stretches of off-tree edges $\boldsymbol{\tau}$. The recursive preconditioning framework aims to transfers solutions between a sequence of sparsifiers known as a solver chain [39]. It generates graph preconditioners, called *ultra-sparsifiers*, by sampling a number of edges to supplement a carefully chosen spanning tree. A small sample size allows reductions to the problem size by eliminating off tree edges. This can be interpreted as partial Cholesky factorization and its guarantees can be stated as:

Lemma 3.2. *There is a routine* ELIMINATE&SOLVE *that for a graph-tree tuple $(H, T, \boldsymbol{\tau})$ with n vertices and m' off-tree edges, input vector $\boldsymbol{b}$ and a Laplacian solver* SOLVER, *ϵ and returns a vector $\boldsymbol{x}$ such that*

$$\|\overline{\boldsymbol{x}} - \boldsymbol{x}\|_{\boldsymbol{L}_H} \leq \epsilon \|\overline{\boldsymbol{x}}\|_{\boldsymbol{L}_H}.$$

Furthermore, ELIMINATE&SOLVE$(H, T, \tau,$ SOLVER$, \boldsymbol{b}, \epsilon)$ *performs $O(n + m')$ operations plus one call to* SOLVER *with a graph-tree tuple $(H', T', \boldsymbol{\tau}')$ with $O(m')$ vertices and edges, the same bounds for the stretch of off-tree edges, and the same error tolerance ϵ.*

A proof of it can be found in Appendix C of [32]. This suggests the goal of reducing off tree edges, which in turn allows us to work with smaller problems. An observation central to the Koutis et al. construction of preconditioners [25, 26] is that the total off-tree stretch serves as a proxy for the number of off-tree edges. As SAMPLE will sample edges with high stretch, as well as tree edges, we need to modify its construction bounding both the number of off-tree edges, and the total off-tree ℓ_p-stretch. Pseudocode of this modified algorithm for generating a preconditioner is given in Figure 2.

$(H, T') =$ RANDPRECON$(G, T, \boldsymbol{\tau})$, where G is a graph, T is a tree, $\boldsymbol{\tau}$ are upper bounds of the stretches of edges in G w.r.t. T.

1. Let $\boldsymbol{X} = \boldsymbol{L}_T$, $\boldsymbol{Y} = \boldsymbol{L}_G$, $\boldsymbol{Y}_i$ be the rank-1 matrix corresponding to each edge.
2. Set $\hat{\boldsymbol{\tau}}$ to be the same as $\boldsymbol{\tau}$ for non tree-edges, and 1 for all tree edges.
3. Repeat
   (a) $\boldsymbol{Z}$ = SAMPLE$\left(\boldsymbol{Y}, \boldsymbol{X}, \hat{\boldsymbol{\tau}}, \frac{1}{10}\right)$.
   (b) Set
      i. H be the edges corresponding to $\boldsymbol{Z}$, and
      ii. T' be the tree in H with the same combinatorial edges as T, and
      iii. $\boldsymbol{\tau}'$ to be 10 times the number of times each off-tree edge is sampled.
4. Until the number of off-tree edges in H is at most $4800 \|\boldsymbol{\tau}\|_p^p$, and $\|\boldsymbol{\tau}'\|_p^p \leq 480 \|\boldsymbol{\tau}\|_p^p$.
5. Return $(H, T', \boldsymbol{\tau}')$.

Figure 2: Pseudocode for Generating a Randomized Preconditioner

Off-tree stretch is useful as a measure for problem size because perturbing the tree by a factor of κ decreases this parameter by a factor of κ. Iterative methods in turn allows us to solve the problem by solving $\sqrt{\kappa}$ instances of the original. This leads to a recurrence of the form

$$T(m) = \sqrt{\kappa} \left(m + T(m/\kappa)\right),$$

which solves to $O(m)$. Of course, this is the ideal situation, and our algorithm will be slower due to distortions from the sampling, as well as the initial sum of $\boldsymbol{\tau}$ being larger than m. We also need one additional modification. In Section 5, we show that it is easier to produce a tree where $\|\boldsymbol{\tau}\|_p^p$ is small for some constant $1/2 < p < 1$. It can be checked that this quantity is also a proxy for problem size.

Lemma 3.3. *For any parameter p,* RANDPRECON$(G, T, \boldsymbol{\tau})$ *runs in expected $O(m + \|\boldsymbol{\tau}\|_p^p)$ time and produces a graph-tree tuple $(H, T, \boldsymbol{\tau}')$ such that*

1. *the number of off-tree edges in H is at most $O(\|\boldsymbol{\tau}\|_p^p)$, and*
2. *$\|\boldsymbol{\tau}'\|_p^p \leq O(\|\boldsymbol{\tau}\|_p^p)$.*

To complete the picture, we need to show that this smaller graph Laplacian, $\boldsymbol{L}_H$, can be used to solve linear systems in

$\boldsymbol{L}_G$. For large distortions, we will use Chebyshev iteration, which can be described as follows:

Lemma 3.4. *Given matrices $\boldsymbol{A}, \boldsymbol{B}$ such that $\boldsymbol{A} \preceq \boldsymbol{B} \preceq \kappa \boldsymbol{A}$ for some constant $\kappa > 0$, along with error $\epsilon > 0$ and a routine $\textsc{Solve}_{\boldsymbol{B}}$ such that for any vector $\boldsymbol{b}' = \boldsymbol{B}\overline{\boldsymbol{x}}'$ we have*

$$\left\|\textsc{Solve}_{\boldsymbol{B}}(\boldsymbol{b}') - \overline{\boldsymbol{x}}'\right\|_{\boldsymbol{B}} \leq \frac{\epsilon^4}{30\kappa^4} \left\|\overline{\boldsymbol{x}}'\right\|_{\boldsymbol{B}};$$

preconditioned Chebyshev iteration gives a routine $\textsc{Solve}_{\boldsymbol{A}}(\boldsymbol{b}) = \textsc{PreconCheby}(\boldsymbol{A}, \boldsymbol{B}, \textsc{Solve}_{\boldsymbol{B}}, \boldsymbol{b})$, such that in the exact arithmetic model, for any vector $\boldsymbol{b} = \boldsymbol{A}\overline{\boldsymbol{x}}$,

- $\left\|\textsc{Solve}_{\boldsymbol{A}}(\boldsymbol{b}) - \overline{\boldsymbol{x}}\right\|_{\boldsymbol{A}} \leq \epsilon \left\|\overline{\boldsymbol{x}}\right\|_{\boldsymbol{A}}$, *and*
- $\textsc{Solve}_{\boldsymbol{A}}(\boldsymbol{b})$ *takes $O(\sqrt{\kappa}\log(1/\epsilon))$ iterations, each consisting of one call to $\textsc{Solve}_{\boldsymbol{B}}$ and a matrix-vector multiplication using $\boldsymbol{A}$.*

The following guarantee can be shown for such a preconditioner using matrix Chernoff bounds [34, 21, 42]. It can also be derived from spectral sparsification by effective resistance [36].

Lemma 3.5. *There exists a constant c such that for any graph-tree tuple $(G, T, \boldsymbol{\tau})$, $H = \textsc{RandPrecon}(G, T, \boldsymbol{\tau})$ satisfies*

$$\frac{1}{c\log n}\boldsymbol{L}_G \preceq \boldsymbol{L}_H \preceq c\log n \boldsymbol{L}_G$$

with high probability.

This means that if we increase the tree by a factor of κ, we can obtain a graph H' with $\|\boldsymbol{\tau}'\|_p^p \leq \frac{1}{\kappa^p}\|\boldsymbol{\tau}\|_p$, and:

$$\boldsymbol{L}_G \preceq \boldsymbol{L}_{H'} \preceq O(\log^2 n\kappa)\boldsymbol{L}_G.$$

Therefore $O(\log n\sqrt{\kappa})$ recursive calls to a Laplacian solver for $\boldsymbol{L}_{H'}$ produces a good answer. When $1/2 < p < 1$, setting κ to a sufficiently large polylog factor then gives a nearly-linear algorithm. This is a crude version of the algorithm underlying the previous fastest graph Laplacian solver algorithm [26], which runs in about $m\log n$ time.

To obtain a faster algorithm, note that if the approximation ration between G and H from Lemma 3.5 is $O(1)$, we would only need $O(\sqrt{\kappa})$ iterations to solve a problem whose size is smaller by a factor of κ^p. Setting κ to a constant would then lead to a $O(\|\boldsymbol{\tau}\|_p^p + m)$ time algorithm. Such optimal ultra-sparsifiers were constructed by Kolla et al. [24] by building upon the nearly-optimal spectral sparsifiers by Batson et al. [7]. However, the current fastest algorithm for constructing such sparsifiers by Zouzias [44] takes cubic time. Finding nearly-linear time algorithms for constructing such sparsifiers was listed as an important open question in the article by Batson et al. [8].

To work around this issue, it's worth observing that the randomized descent algorithms by Kelner et al. [23] and Lee and Sidford [27] give bounds without this distortion. From the preconditioner setting, this can be viewed as $\boldsymbol{L}_H^\dagger$ behaving in expectation as $\boldsymbol{L}_G^\dagger$ from the purpose of iterative methods. Although we believe such behavior is the case for preconditioned Chebyshev iterations, it is much easier to prove such a bound for preconditioned Richardson iteration, or iterative refinement. As a result, we view the graph with T scaled up as another intermediate state, and analyze the iterative methods with large and small number of calls separately. This simplification in analysis introduces yet another procedure. The call structure of our procedures is given in Figure 3

$\boldsymbol{x} = \textsc{RandRichardson}(G, T, \boldsymbol{\tau}, \textsc{Solver}, \boldsymbol{b}, \epsilon)$, where G is a graph, T is a tree, $\boldsymbol{\tau}$ are upper bounds of the stretches of edges of G w.r.t. T, $\boldsymbol{b}$ is the vector to be solved, and ϵ is the target error.

1. Set $\epsilon_1 = \frac{1}{320c_s\log n}$ and $t = O\left(\log\left(\epsilon^{-1}\log n\right)\right)$.
2. Let $\boldsymbol{Z}$ be the linear operator corresponding to the solver given in Lemma 3.7.
3. Repeat
   (a) $\boldsymbol{x}_0 = 0$.
   (b) For $i = 1 \ldots t$
      i. $(H_i, T_i, \boldsymbol{\tau}_i) = \textsc{RandPrecon}(G, T, \boldsymbol{\tau})$.
      ii. $\boldsymbol{r}_i = \boldsymbol{L}_G\boldsymbol{x}_{i-1} - \boldsymbol{b}$.
      iii. $\boldsymbol{y}_i = \textsc{Eliminate\&Solve}(H_i, T_i, \boldsymbol{\tau}_i, \textsc{Solver}, \boldsymbol{r}_i, \epsilon_1)$.
      iv. $\boldsymbol{x}_i = \boldsymbol{x}_{i-1} - \frac{1}{10}\boldsymbol{y}_i$.
4. Until $\|\boldsymbol{Z}(\boldsymbol{L}_G\boldsymbol{x}_t - \boldsymbol{b})\|_{\boldsymbol{L}_G} \leq \frac{\epsilon}{c_Z\log^4 n}\|\boldsymbol{Z}\boldsymbol{b}\|_{\boldsymbol{L}_G}$.
5. Return $\boldsymbol{x}_t$.

Figure 4: Randomized Richardson Iteration

Our use of randomized preconditioners relies on two facts, the first being that each step of standard Richardson iteration makes significant progress in expectation.

Lemma 3.6. *For any pair of vectors $\boldsymbol{x}$ and $\boldsymbol{b} = \boldsymbol{L}_G\overline{\boldsymbol{x}}$, we have*

$$\mathbb{E}_H\left[\left\|\overline{\boldsymbol{x}} - \left(\boldsymbol{x} - \frac{1}{10}\boldsymbol{L}_H^\dagger(\boldsymbol{L}_G\boldsymbol{x} - \boldsymbol{b})\right)\right\|_{\boldsymbol{L}_G}\right] \leq \left(1 - \frac{1}{160}\right)\|\overline{\boldsymbol{x}} - \boldsymbol{x}\|_{\boldsymbol{L}_G}.$$

We can also check whether the error has been reduced by sufficient amounts using coarser solvers.

Lemma 3.7. *There exist a constant c_Z such that given a graph-tree tuple $(G, T, \boldsymbol{\tau})$, we can construct with high probability a linear operator $\boldsymbol{Z}$ such that under exact arithmetic*

1. $\boldsymbol{L}_G^\dagger \preceq \boldsymbol{Z} \preceq c_Z\log^4 n\boldsymbol{L}_G^\dagger$, *and*
2. *given any vector $\boldsymbol{b}$, $\boldsymbol{Z}\boldsymbol{b}$ can be evaluated in $O(m + \|\boldsymbol{\tau}\|_p^p)$ time where p is any constant $> 1/2$.*

This gives us additional control over the process, and allows us to produce a Las Vegas algorithm that w.h.p. produces a solution meeting the guarantees. Pseudocode of such an algorithm is given in Figure 4

Lemma 3.8. *Given a Laplacian solver $\textsc{Solver}$, any graph-tree pair (G, T), bounds on stretch $\boldsymbol{\tau}$, vector $\boldsymbol{b} = \boldsymbol{L}_G\overline{\boldsymbol{x}}$ and*

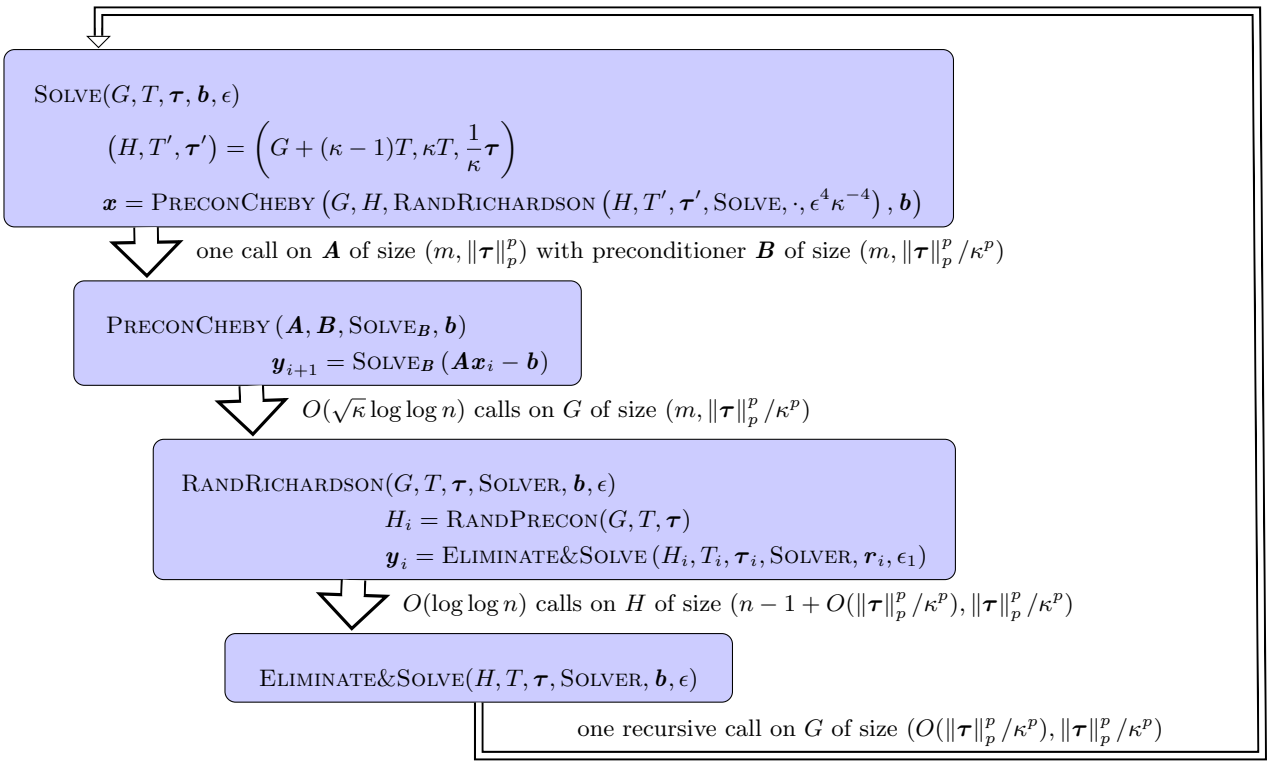

Figure 3: Workflow of recursive solve algorithm with arrows indicating function calls, and the sizes given as (number of edges, and total ℓ_p-stretch). On an input graph with m edges and stretch upper bounds $\boldsymbol{\tau}$, Solve makes $O(\sqrt{\kappa}\log\log n\log(1/\epsilon))$ recursive calls with graphs of size $(O(\|\boldsymbol{\tau}\|_p^p/\kappa^p), \|\boldsymbol{\tau}\|_p^p/\kappa^p)$

error $\epsilon > 0$, RANDRICHARDSON$(G, T, \boldsymbol{\tau},$ SOLVER$, \boldsymbol{b}, \epsilon)$ *returns with high probability a vector $\boldsymbol{x}$ such that*

$$\|\boldsymbol{x} - \overline{\boldsymbol{x}}\|_{\boldsymbol{L}_G} \leq \epsilon \|\overline{\boldsymbol{x}}\|_{\boldsymbol{L}_G},$$

and the algorithm takes an expected $O(\log(\epsilon^{-1}) + \log\log n))$ iterations. Each iteration consists of one call to SOLVER *on a graph with $O(\|\boldsymbol{\tau}\|_p^p)$ edges and error $\frac{1}{O(\log n)}$, plus an overhead of $O(m + \|\boldsymbol{\tau}\|_p^p)$ operations.*

Combining Lemma 3.8 with Chebyshev iteration as described in Lemma 3.4 leads to a Laplacian solver that follows the workflow of Figure 3. It picks the scaling factor κ based on the given values of $\|\boldsymbol{\tau}\|_p^p$, m, and p in order to optimize running time. Pseudocode of this algorithm is given in Figure 5.

Lemma 3.9. *Given a parameter $1/2 < p < 1$ and a graph-tree tuple $(G, T, \boldsymbol{\tau})$ with m edges such that $\|\boldsymbol{\tau}\|_p^p \leq m\log^p n$. For any vector $\boldsymbol{b} = \boldsymbol{L}_G\overline{\boldsymbol{x}}$,* SOLVE$(G, T, \boldsymbol{\tau}, \boldsymbol{b}, \frac{1}{320c_s\log n})$ *returns w.h.p. a vector $\boldsymbol{x}$ such that*

$$\|\overline{\boldsymbol{x}} - \boldsymbol{x}\|_{\boldsymbol{L}_G} \leq \frac{1}{320c_s\log n}\|\boldsymbol{x}\|_{\boldsymbol{L}_G},$$

and its expected running time is

$$O\left(m\left(\frac{\|\boldsymbol{\tau}\|_p^p}{m}\right)^{\frac{1}{2p}}\log\log^{2+\frac{2}{2p-1}} n\right).$$

Proof. The proof is by induction on graph size. As our induction hypothesis, we assume the lemma to be true for all graphs of size $m' < m$. The choice of κ gives

$$\|\boldsymbol{\tau}'\|_p^p \leq \frac{m}{c^p\log\log^{2+\frac{2}{2p-1}} n}.$$

The guarantees of randomized Richardson iteration from Lemma 3.8 gives that all the randomized preconditioners have both off-tree edge count and off-tree stretch bounded by $O(\|\boldsymbol{\tau}'\|_p^p) = O\left(\frac{m}{c^p\log\log^{2+\frac{2}{2p-1}} n}\right)$.

$\boldsymbol{x} =$ SOLVE$(G, T, \boldsymbol{\tau}, \boldsymbol{b}, \epsilon)$, where G is a graph, T is a tree, $\boldsymbol{\tau}$ are upper bounds of the stretches of edges in G w.r.t. T, $\boldsymbol{b}$ is the vector to be solved, and ϵ is the goal error.

1. Set $\kappa = c(\log\log n)^{4/(2p-1)}\left(\frac{\|\boldsymbol{\tau}\|_p^p}{m}\right)^{1/p}$ for an appropriate constant c (dependent on p).
2. Let $(H, T', \boldsymbol{\tau}')$ be the graph-tree tuple with T scaled up by a factor of κ, $\boldsymbol{\tau}$ scaled down by a factor of κ.
3. $$\boldsymbol{x} = \text{PRECONCHEBY}\left(G, H, \text{RANDRICHARDSON}\left(H, T', \boldsymbol{\tau}', \text{SOLVE}, \epsilon^4\kappa^{-4}\right), \boldsymbol{b}\right).$$
4. Return $\boldsymbol{x}$

Figure 5: Recursive Solver

An appropriate choice of c makes both of these values strictly less than m, and this allows us to apply the inductive hypothesis on the graphs obtained from the randomized preconditioners by ELIMINATE&SOLVE.

As κ is bounded by $c\log^2 n$ and ϵ is set to $\frac{1}{320c_s \log n}$, the expected cost of the recursive calls made by RANDRICHARDSON is $O(m \log\log n)$. Combining this with the iteration count in PRECONCHEBY of

$$O(\sqrt{\kappa}\log(1/\epsilon)) = O\left((\log\log n)^{\frac{2}{2p-1}}\left(\frac{\|\boldsymbol{\tau}\|_p^p}{m}\right)^{\frac{1}{2p}}\log\log n\right)$$

gives the inductive hypothesis. □

It remains to pick a tree with small value of $\|\boldsymbol{\tau}\|_p^p$. The state of the art low-stretch spanning tree algorithm due to Abraham and Neiman gives a tree with $\|\boldsymbol{\tau}\|_p^p \leq O(m\log n)$ for any $p < 1$ in $O(m \log n \log\log n)$ time. To reduce the preprocessing cost, we use the low ℓ_p-stretch embeddable trees from [16]. The guarantees of these trees can be summarized as follows:

Lemma 3.10. *Given a graph $\hat{G}$ with n vertices, m edges, and any constant $0 < p < 1$, we can construct a graph-tree tuple $(G, T, \boldsymbol{\tau})$ and associated bounds on stretches of edges $\boldsymbol{\tau}$ such that*

1. *The construction takes $O(m \log\log n \log\log\log n)$ time in the RAM model*
2. *G has at most $2n$ vertices and $n + m$ edges, and*
3. *$\|\boldsymbol{\tau}\|_p^p \leq O(m \log n)$, and*
4. *there is a $|V_{\hat{G}}| \times |V_G|$ matrix $\boldsymbol{\Pi}$ with one 1 in each row and zeros everywhere else such that:*

$$\frac{1}{2}\boldsymbol{L}_{\hat{G}}^{\dagger} \preceq \boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T \preceq \boldsymbol{L}_{\hat{G}}^{\dagger}.$$

Note that $\boldsymbol{\Pi}$ maps some vertices of G to unique vertices of $\hat{G}$, and $\boldsymbol{\Pi}^T$ maps each vertex of $\hat{G}$ to a unique vertex in G. The spectral guarantees given in Part 4 allow the solver for $\boldsymbol{L}_G$ to be converted to a solver for $\boldsymbol{L}_{\hat{G}}$ while preserving the error quality.

Fact 3.11. *Let $\boldsymbol{\Pi}$ and $\boldsymbol{\Pi}_1$ be the two projection matrices defined in Lemma 3.10 Part 4. For a vector $\hat{\boldsymbol{b}}$, if $\boldsymbol{x}$ is a vector such that*

$$\left\|\boldsymbol{x} - \boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}}\right\|_{\boldsymbol{L}_G} \leq \epsilon\left\|\boldsymbol{L}_G^{\dagger}\Pi^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}}\right\|_{\boldsymbol{L}_G},$$

for some $\epsilon > 0$. Then the vector $\hat{\boldsymbol{x}} = \boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{x}$ satisfies

$$\left\|\hat{\boldsymbol{x}} - \boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}}\right\|_{\left(\boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\right)^{\dagger}} \leq \epsilon\left\|\boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}}\right\|_{\left(\boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\right)^{\dagger}}.$$

Therefore, a good solution to $\boldsymbol{L}_G\boldsymbol{x} = \boldsymbol{\Pi}^T\hat{\boldsymbol{b}}$ also leads to a good solution to $\boldsymbol{L}_{\hat{G}}\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}$. The constant relative error can in turn be corrected using preconditioned Richardson iteration described in Section 3. For the rest of our presentation, we will focus on solving linear systems in settings where we know small bounds to $\|\boldsymbol{\tau}\|_p^p$.

To prove Theorem 1.1, we first invoke SOLVE with ϵ set to a constant. Following an analysis identical to the proof of Lemma 3.9, at the top level each iteration of PRECONCHEBY will require $O(m \log\log n)$ time, but now only

$$O(\sqrt{\kappa}\log(1/\epsilon)) = O\left((\log\log n)^{\frac{2}{2p-1}}\left(\frac{\|\boldsymbol{\tau}\|_p^p}{m}\right)^{\frac{1}{2p}}\right)$$

iterations are necessary. Setting p arbitrarily close to 1 means that for any constant $\delta > 0$ and relative error ϵ, there is a solver for $\boldsymbol{L}_G$ that runs in $O(m\log^{1/2} n \log\log^{3+\delta} n)$ time. This error can be reduced using another iterative method. We will use Richardson iteration at this outer loop, while transferring solutions and errors to the original graph using the guarantees of the embeddable tree given in Lemma 3.10.

Proof. (of Theorem 1.1) Using Fact 3.11 on the solver described above for $\boldsymbol{L}_G$ gives a solver for $(\boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T)^{\dagger}$ with relative error $\frac{1}{5}$. This condition and Lemma 3.10 Part 4 then allows us to invoke the above Lemma with $\boldsymbol{A} = \boldsymbol{L}_{\hat{G}}$ and $\boldsymbol{B} = \boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T$. Incorporating the $O(\log(\frac{1}{\epsilon}))$ iteration count then gives the overall result. □

4. PRECONDITIONING IN EXPECTATION

Here we describe the guarantee that we can give for the algorithm SAMPLE. When δ is set to $\frac{1}{O(\log n)}$, the algorithm is identical to random sampling routines for constructing spectral sparsifiers [36]. The preconditioner produced contains $\boldsymbol{X}$ plus $O(s \log n)$ of the matrices $\boldsymbol{Y}_i$'s, where $s = \sum \boldsymbol{\tau}_i$. If we let the original matrix be $\boldsymbol{Y} \stackrel{\text{def}}{=} \boldsymbol{X} + \sum_{i=1}^m \boldsymbol{Y}_i$, matrix Chernoff bounds such as the ones by Tropp [42] gives $\frac{1}{2}\boldsymbol{Y} \preceq \boldsymbol{Z} \preceq 2\boldsymbol{Y}$. The Kolla et al. [24] result can be viewed as finding $\boldsymbol{Z}$ consisting of only $O(s)$ of the matrices, and $\boldsymbol{Y} \preceq \boldsymbol{Z} \preceq O(1)\boldsymbol{Y}$, albeit in cubic time. These spectral guarantees allows us to show that one step of preconditioned Richardson iteration using $\boldsymbol{Z}$ as a preconditioner makes good progress. Such a statement is Lemma 3.6 without the expectation.

We will show that running SAMPLE with δ set to a constant gives a preconditioner that makes similar progress in expectation. This is formalized in Lemma 3.6, and can be proven by bounding the first and second moments of $\boldsymbol{Z}^{-1}$ w.r.t. $\boldsymbol{Y}$. These bounds are at the core of our result. By normalization and restricting operators to their column spaces, it suffices to show the following about sampling vectors in the isotropic position.

Lemma 4.1. *Suppose $\boldsymbol{R}_i = \boldsymbol{u}_i\boldsymbol{u}_i^T$ are rank one matrices with $\sum_{j=1}^m \boldsymbol{u}_j\boldsymbol{u}_j^T = \boldsymbol{I}$, $\boldsymbol{S}$ is a positive definite matrix satisfying $\boldsymbol{S} \preceq \boldsymbol{I}$ and $\boldsymbol{\tau}_1 \ldots \boldsymbol{\tau}_m$ are values that satisfy $\boldsymbol{\tau}_i \geq \mathrm{Tr}\left(\boldsymbol{S}^{-1}\boldsymbol{R}_i\right)$, and $0 < \delta < 1$ is an arbitrary parameter. Then the matrix $\boldsymbol{W} = \text{SAMPLE}(\boldsymbol{R}_1 \ldots \boldsymbol{R}_m, \boldsymbol{S}, \boldsymbol{\tau}_1 \ldots \boldsymbol{\tau}_m, \delta)$ satisfies:*

1. *$\mathbb{E}_{r,i_1\ldots i_r}\left[\boldsymbol{x}^T\boldsymbol{W}^{-1}\boldsymbol{x}\right] \geq \frac{1}{3}\boldsymbol{x}^T\boldsymbol{x}$, and*
2. *$\mathbb{E}_{r,i_1\ldots i_r}\left[\boldsymbol{x}^T\boldsymbol{W}^{-1}\boldsymbol{x}\right] \leq \frac{1}{1-2\delta}\boldsymbol{x}^T\boldsymbol{x}$, and*
3. *$\mathbb{E}_{r,i_1\ldots i_r}\left[\boldsymbol{x}^T\boldsymbol{W}^{-2}\boldsymbol{x}\right] \leq \frac{1}{1-3\delta}\boldsymbol{x}^T\boldsymbol{x}$.*

To analyze the SAMPLE algorithm, it will be helpful to keep track of its intermediate steps. We define $\boldsymbol{W}_0$ to be the initial value of the sample sum matrix $\boldsymbol{W}$. This corresponds to the initial value of $\boldsymbol{Z}$ from line 2 in the pseudocode of

figure 1, and $\boldsymbol{W}_0 = \boldsymbol{S}$. We define $\boldsymbol{W}_j$ to be the value of $\boldsymbol{W}$ after j samples, $\boldsymbol{W}_{j+1} = \boldsymbol{W}_j + \frac{\delta}{\boldsymbol{\tau}_{i_{j+1}}} \boldsymbol{u}_{i_{j+1}} \boldsymbol{u}_{i_{j+1}}^T$. Our proof also relies on the definition of Harmonic sums, and various convexity properties related to it.

$$\text{HrmSum}\,(x, y) \stackrel{\text{def}}{=} \frac{1}{1/x + 1/y}.$$

Some of the facts that we use involving it and matrices are given in Figure 6. Facts 1, 2, and 3 in Figure 6 are standard results, while the remaining are proven in [15]. With these facts, Part 1 follows readily from a generalization of the arithmetic mean (AM) - harmonic mean (HM) inequality for matrices [35]. In remainder of this section, we give a brief justification of Parts 2 and 3.

1. Given positive definite matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ where $\boldsymbol{A} \preceq \boldsymbol{B}$, we have $\boldsymbol{B}^{-1} \preceq \boldsymbol{A}^{-1}$.

2. Let $w_1, \ldots, w_r$ be positive numbers such that $\sum_1^r w_i = 1$, and $\boldsymbol{M}_1, \ldots, \boldsymbol{M}_r$ be positive definite matrices. Then
$$\left(w_1 \boldsymbol{M}_1^{-1} + \ldots + w_r \boldsymbol{M}_r^{-1}\right)^{-1} \preceq w_1 \boldsymbol{M}_1 + \ldots + w_r \boldsymbol{M}_r.$$

3. Sherman-Morrison formula:
$$(\boldsymbol{A} + \boldsymbol{u}\boldsymbol{u}^T)^{-1} = \boldsymbol{A}^{-1} - \frac{\boldsymbol{A}^{-1}\boldsymbol{u}\boldsymbol{u}^T\boldsymbol{A}^{-1}}{1 + \boldsymbol{u}^T\boldsymbol{A}^{-1}\boldsymbol{u}}.$$

4. Let us define $\text{HrmSum}\,(x, y) \stackrel{\text{def}}{=} \frac{1}{1/x + 1/y}$. Then if X is a positive random variable and $\alpha > 0$ is a constant, then
$$\mathbb{E}_X\left[\text{HrmSum}\,(X, \alpha)\right] \leq \text{HrmSum}\,(\mathbb{E}\left[X\right], \alpha).$$

5. For any unit vector $\boldsymbol{v}$, positive definite matrix $\boldsymbol{A}$, and scalar $\alpha > 0$
$$\boldsymbol{v}^T\left(\boldsymbol{A} + \alpha \boldsymbol{I}\right)^{-1}\boldsymbol{v} \leq \text{HrmSum}\left(\boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{v}, 1/\alpha\right).$$

6. Let $\boldsymbol{A}$ be a positive definite matrix and $\boldsymbol{u}$ be any unit vector. Then
$$-\frac{1}{t}\boldsymbol{u}^T\boldsymbol{A}^{-2}\boldsymbol{u} \leq \text{HrmSum}\left(\boldsymbol{u}^T\boldsymbol{A}^{-1}\boldsymbol{u}, t\right) - \boldsymbol{u}^T\boldsymbol{A}^{-1}\boldsymbol{u}.$$

Figure 6: Matrix Facts

The Sherman-Morrison formula given in Fact 3 gives

$$\begin{aligned}
&\mathbb{E}_{i_j}\left[(\boldsymbol{W}_j + \boldsymbol{u}_j\boldsymbol{u}_j^T)^{-1}\middle|\boldsymbol{W}_j\right] \\
&\quad\preceq \mathbb{E}_{i_j}\left[\boldsymbol{W}_j^{-1} - \frac{\boldsymbol{W}_j^{-1}\boldsymbol{u}_j\boldsymbol{u}_j^T\boldsymbol{W}_j^{-1}}{1+\delta}\middle|\boldsymbol{W}_j\right] \\
&\quad\preceq \boldsymbol{W}_j^{-1} - (1-\delta)\boldsymbol{W}_j^{-1}\mathbb{E}_{i_j}\left[\boldsymbol{u}_j\boldsymbol{u}_j^T\right]\boldsymbol{W}_j^{-1} \\
&\quad\preceq \boldsymbol{W}_j^{-1} - \frac{(1-\delta)}{t}\boldsymbol{W}_j^{-2}. \quad (1)
\end{aligned}$$

If we combine the above with Fact 6, we find that for any unit vector $\boldsymbol{v}$

$$\begin{aligned}
&\boldsymbol{v}^T\mathbb{E}_{i_{j+1}}\left[(\boldsymbol{W}_j + \boldsymbol{u}_{j+1}\boldsymbol{u}_{j+1}^T)^{-1}\middle|\boldsymbol{W}_j\right]\boldsymbol{v} \\
&\qquad\leq \delta\boldsymbol{v}^T\boldsymbol{W}_j^{-1}\boldsymbol{v} + (1-\delta)\,\text{HrmSum}\left(\boldsymbol{v}^T\boldsymbol{W}_j^{-1}\boldsymbol{v}, t\right).
\end{aligned}$$

If we now include the choice of $\boldsymbol{W}_j$ in the expectation and apply Fact 4 with $X = \boldsymbol{v}^T\boldsymbol{W}_j^{-1}\boldsymbol{v}$ and $\alpha = t$, we then get

$$\begin{aligned}
\mathbb{E}_{i_1,\ldots,i_{j+1}}\left[\boldsymbol{v}^T\boldsymbol{W}_{j+1}^{-1}\boldsymbol{v}\right] &\leq \delta\mathbb{E}_{i_1,\ldots,i_j}\left[\boldsymbol{v}^T\boldsymbol{W}_j^{-1}\boldsymbol{v}\right] \\
&\quad + (1-\delta)\,\text{HrmSum}\left(\mathbb{E}_{i_1,\ldots,i_j}\left[\boldsymbol{v}^T\boldsymbol{W}_j^{-1}\boldsymbol{v}\right], t\right).
\end{aligned}$$

For convenience, we define $E_j := \mathbb{E}_{i_1,\ldots,i_j}\left[\boldsymbol{v}^T\boldsymbol{W}_j^{-1}\boldsymbol{v}\right]$. Now the previous inequality can be written as

$$E_{i+1} \leq \delta E_i + (1-\delta)\,\text{HrmSum}\,(E_i, t)$$

Since we start with $\boldsymbol{W}_0 = \boldsymbol{S}$, we have $\boldsymbol{W}_j \succeq \boldsymbol{S}$. Thus, by Fact 1

$$\boldsymbol{W}_j^{-1} \preceq \boldsymbol{S}^{-1} = \boldsymbol{S}^{-1}\sum_i \boldsymbol{R}_i.$$

So $\text{Tr}\left(\boldsymbol{W}_j^{-1}\right) \leq \sum_{i=1}^m \boldsymbol{\tau}_i = t\delta$, and hence $E_j \leq t\delta < t$.

This lets us write:

$$\begin{aligned}
E_{j+1} &\leq \delta E_j + \frac{1-\delta}{\frac{1}{E_j} + \frac{1}{t}} \\
&\leq \frac{1}{\left(\frac{1}{E_j} + \frac{1}{t}\right)\left(1 - \frac{\delta E_j}{t}\right)} \\
&\leq \frac{1}{1/E_j + (1-2\delta)/t}.
\end{aligned}$$

So

$$\frac{1}{E_{j+1}} \geq \frac{1}{E_j} + (1-2\delta)/t.$$

Then it follows by induction that after t steps

$$\frac{1}{E_j} \geq (1-2\delta).$$

Thus we have proven

$$\mathbb{E}_{i_1,\ldots,i_t}\left[\boldsymbol{v}^T\boldsymbol{W}_t^{-1}\boldsymbol{v}\right] \leq \frac{1}{1-2\delta}.$$

Additionally, for any integer $r \geq t$, $\boldsymbol{W}_r \succeq \boldsymbol{W}_t$, so fact 1 gives $\boldsymbol{W}_r^{-1} \preceq \boldsymbol{W}_t^{-1}$. This means that with r chosen uniformly at random in the interval $[t, 2t-1]$, we have

$$\mathbb{E}_{r,i_1,\ldots,i_r}\left[\boldsymbol{v}^T\boldsymbol{W}_r^{-1}\boldsymbol{v}\right] \leq \frac{1}{1-2\delta}.$$

Thus we have shown Part 2 of Lemma 4.1.

Using the spectral inequality fomr Fact 1, taking expectation over $\boldsymbol{W}_j$, and telescoping gives

$$\begin{aligned}
&\mathbb{E}_{i_1,\ldots,i_{2t}}\left[\boldsymbol{v}^T\boldsymbol{W}_{2t-1}^{-1}\boldsymbol{v}\right] - \mathbb{E}_{i_1,\ldots,i_t}\left[\boldsymbol{v}^T\boldsymbol{W}_t{}^{-1}\boldsymbol{v}\right] \\
&\qquad\leq \sum_{j=t}^{2t-1}\mathbb{E}_{i_1,\ldots,i_j}\left[\frac{-(1-\delta)}{t}\boldsymbol{v}^T\boldsymbol{W}_j^{-2}\boldsymbol{v}\right].
\end{aligned}$$

So

$$\frac{1}{t}\sum_{j=t}^{2t-1}\mathbb{E}_{i_1,\ldots,i_j}\left[\boldsymbol{v}^T\boldsymbol{W}_j^{-2}\boldsymbol{v}\right] \leq \frac{1}{1-\delta}\mathbb{E}_{i_1,\ldots,i_t}\left[\boldsymbol{v}^T\boldsymbol{W}_t^{-1}\boldsymbol{v}\right] \leq \frac{1}{(1-2\delta)(1-\delta)} < \frac{1}{1-3\delta}.$$

This implies that for an integer r chosen uniformly at random in the interval $[t, 2t-1]$, we have

$$\mathbb{E}_{r,i_1,\ldots,i_r}\left[\boldsymbol{v}^T\boldsymbol{W}_r^{-2}\boldsymbol{v}\right] \leq \frac{1}{1-3\delta}.$$

Which completes the proof of Part 3 of Lemma 4.1.

5. STRETCHING STRETCH

In this section we briefly describe how to remove the tree construction obstacle from obtaining faster solver algorithms. More details on this algorithm can be found in [16]. We introduce two modifications to the definition of low stretch spanning trees which greatly simplify the construction. First, we allow additional vertices in the tree, leading to a Steiner tree. that we require to be *embeddable* into the original graph. Secondly, we discount the cost of high-stretch edges in ways that more accurately reflect how these trees are used. This discounting of stretch has been studied as embeddings in snowflake space [5, 30], where distance is discounted to some exponent $p < 1$. We will use the notation $\mathbf{STR}_T^p(e)$ to denote this discounting of stretch.

Our algorithm draws upon works by Alon et al. [3] and Bartal [6] on low stretch embeddings. The tree construction by Alon et al. is based on a bottom-up decomposition which runs in linear time, and it can be shown that this scheme achieves polylog stretch under ℓ_p stretch. On the other hand, Bartal's decomposition is a top-down scheme runs in $O(m \log n)$ time, but it leads to an expected $O(\log^p n)$ ℓ_p-stretch per edge. In much the same spirit as [6], we define a certain decomposition of the graph

Definition 5.1. Let $G = (V, E)$ be a connected graph, we say that a sequence of forests $\boldsymbol{B} = (B_0, B_1, \ldots, B_t)$ is a *Bartal decomposition* of G if

1. B_0 is a spanning tree of G and B_t is an empty graph.
2. For any $i \leq t$, B_i is a subgraph of G in the weighted sense.
3. For any $u, v \in V$ and $i < t$, if u and v are connected in B_{i+1}, then they are also connected in B_i.
4. There is a sequence $(d_0, d_1, \ldots, d_t)$ of diameter bounds, such that for any $i \leq t$, any connected component of B_i has diameter at most d_i.

The definition of the stretch can be naturally extended: we simply define the stretch of an edge e w.r.t. to a Bartal decomposition to be $d_i/l(e)$ if the endpoints of e is separated on level i. Furthermore, a Bartal decomposition is embeddable if the union of the B_is is embeddable. By first pre-computing a crude decomposition à la Alon et al. in linear time, and using it as a guide for speeding up the Bartal decomposition construction, we obtain a two-pass algorithm which efficiently produces a Bartal decomposition with good l_p stretch:

Lemma 5.2. *There is a routine that for any graph G and a constant $p < 1$, produces in expected $O(m \log\log n)$ time in the RAM model an implicit representation of a Bartal decomposition $\boldsymbol{B}$ with expected size $O(m \log\log n)$ such that with high probability for any edge e, $\mathbb{E}_{\boldsymbol{B}}\left[\boldsymbol{STR}_{\boldsymbol{B}}^p(e)\right] \leq O(\log^p n)$.*

To satisfy the embeddability requirement, we make crucial use of our definition of l_p stretches, and obtain embeddability by exploiting different moments:

Lemma 5.3. *For constants $0 < p < q < 1$ and a given graph G, we can construct another graph G' in linear time, such that from a Bartal decomposition $\boldsymbol{B}'$ of G' we can obtain an* embeddable *Bartal decomposition $\boldsymbol{B}$ of G in linear time. Furthermore, the ℓ_p-stretch of an edge e in G w.r.t. $\boldsymbol{B}$ and its ℓ_q-stretch in G' w.r.t. $\boldsymbol{B}'$ are related by*

$$\boldsymbol{STR}_{\boldsymbol{B}}^p(e) = O\left(\boldsymbol{STR}_{\boldsymbol{B}'}^q(e)\right).$$

Finally, we can extract our Steiner tree from the implicit representation of an Bartal decomposition with same guarantees on stretches up to a constant factor.

Lemma 5.4. *Given an embeddable Bartal decomposition $\boldsymbol{B}$ of a graph $G = (V, E)$, we can construct in $O(m \log\log n)$ time an embeddable tree T with $O(n)$ vertices containing V such that for any edge e we have*

$$\boldsymbol{STR}_T(e) = O(\boldsymbol{STR}_{\boldsymbol{B}}(e)).$$

We also show the folklore result that embeddings are sufficient for guarantees on the linear operators.

Lemma 5.5. *Let G and H be graphs such that G is a subgraph of H in the weighted sense and $H \setminus G$ is embeddable in G. Furthermore, let the graph Laplacians of G and H be $\boldsymbol{L}_G$ and $\boldsymbol{L}_H$ respectively. Also, let $\boldsymbol{\Pi}$ be the $|V_G| \times |V_H|$ matrix with one 1 in each row at the position that vertex corresponds to in H and 0 everywhere else, and $\boldsymbol{\Pi}_1$ the orthogonal projection operator onto the part of $\Re^{V_G}$ that's orthogonal to the all-ones vector. Then we have:*

$$\frac{1}{2}\boldsymbol{L}_G^\dagger \preceq \boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_H^\dagger\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T \preceq \boldsymbol{L}_G^\dagger.$$

The above lemmas together gives us Lemma 3.10. The various stages of our algorithm are shown in Figure 7 Although our running time of $O(m \log\log n)$ is in the RAM model, this dependency only occurs in the implicit computation of the decomposition in Lemma 5.2. Here the use of RAM model algorithms occur in two places: approximately bucketing the edge weights, and a shortest path algorithm due to Han and Thorup [20, 41]. The use of RAM-model shortest path routines can be removed with an overhead of $O(\log\log n)$ by bucketing the edges as in [26, 2]. The bucketing process can also work with approximate edge weights. If all edge lengths are between 1 and Δ, this can be done in $O(m \log(\log \Delta))$ time in the pointer machine model, which is $O(m \log\log m)$ when $\Delta \leq m^{\mathbf{poly}(\log m)}$. We suspect that there are pointer machine algorithms without even this mild dependence on Δ, and perhaps even algorithms that improve on the runtime of $O(m \log\log n)$. Less speculatively, we also believe that our two-stage approach of combining bottom-up and top-down schemes can be applied with the decomposition scheme of [2] to generate actual spanning trees (as opposed to merely embeddable Steiner trees) with low ℓ_p-stretch. However, we do not have a rigorous analysis of this approach, which would presumably require a careful interplay with the radius-bounding arguments in that paper.

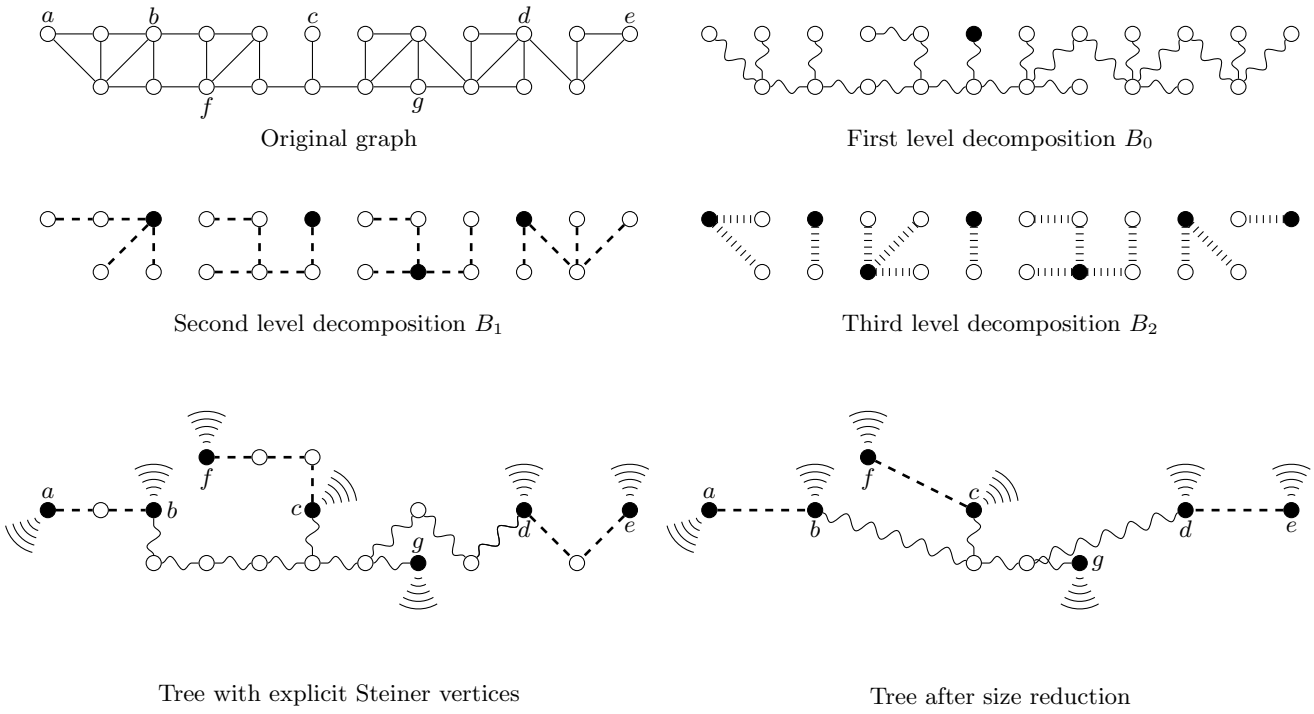

Figure 7: Bartal decomposition and the tree produced for a particular graph

Acknowledgements

We thank the anonymous reviewers for their comments and suggestions, and Jon Kelner and Dan Spielman for very helpful discussions.

6. REFERENCES

[1] I. Abraham, Y. Bartal, and O. Neiman. Nearly tight low stretch spanning trees. In *Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '08, pages 781–790, Washington, DC, USA, 2008. IEEE Computer Society.

[2] I. Abraham and O. Neiman. Using petal decompositions to build a low stretch spanning tree. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 395–406, New York, NY, USA, 2012. ACM.

[3] N. Alon, R. Karp, D. Peleg, and D. West. A graph-theoretic game and its application to the k-server problem. *SIAM J. Comput.*, 24(1):78–100, 1995.

[4] S. Alstrup, J. Holm, K. D. Lichtenberg, and M. Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, Oct. 2005.

[5] P. Assouad. Plongements lipschitziens dans $\{\{r\}\}^n$. *Bulletin de la Société Mathématique de France*, 111:429–448, 1983.

[6] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 184–193, 1996.

[7] J. Batson, D. A. Spielman, and N. Srivastava. Twice-Ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

[8] J. Batson, D. A. Spielman, N. Srivastava, and S.-H. Teng. Spectral sparsification of graphs: theory and algorithms. *Commun. ACM*, 56(8):87–94, Aug. 2013.

[9] E. Boman and B. Hendrickson. On spanning tree preconditioners. Manuscript, Sandia National Lab., 2001.

[10] E. G. Boman, B. Hendrickson, and S. A. Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. *SIAM J. Numerical Analysis*, 46(6):3264–3284, 2008.

[11] H. H. Chin, A. Madry, G. L. Miller, and R. Peng. Runtime guarantees for regression problems. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 269–282, New York, NY, USA, 2013. ACM.

[12] H. H. Chin and G. L. Miller. Applications of spectral algorithms for image processing tasks. 2012.

[13] P. Christiano, J. A. Kelner, A. Madry, D. A. Spielman, and S.-H. Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 273–282, New York, NY, USA, 2011. ACM.

[14] K. L. Clarkson. Subgradient and sampling algorithms for l1 regression. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '05, pages 257–266, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[15] M. B. Cohen, R. Kyng, J. W. Pachocki, R. Peng, and A. Rao. Preconditioning in expectation. *CoRR*,

abs/1401.6236, 2014.

[16] M. B. Cohen, G. L. Miller, J. W. Pachocki, R. Peng, and S. C. Xu. Stretching stretch. *CoRR*, abs/1401.2454, 2014.

[17] S. I. Daitch and D. A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 451–460, New York, NY, USA, 2008. ACM.

[18] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 38(2):608–628, 2008.

[19] K. D. Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. PhD thesis, Carnegie Mellon University, 1996.

[20] Y. Han. Deterministic sorting in o(nlog logn) time and linear space. *J. Algorithms*, 50(1):96–105, Jan. 2004.

[21] N. Harvey. C&O 750: Randomized algorithms, winter 2011, lecture 11 notes. 2011.

[22] J. A. Kelner, G. L. Miller, and R. Peng. Faster approximate multicommodity flow using quadratically coupled flows. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 1–18, New York, NY, USA, 2012. ACM.

[23] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, STOC '13, pages 911–920, New York, NY, USA, 2013. ACM.

[24] A. Kolla, Y. Makarychev, A. Saberi, and S.-H. Teng. Subgraph sparsification and nearly optimal ultrasparsifiers. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 57–66, New York, NY, USA, 2010. ACM.

[25] I. Koutis, G. L. Miller, and R. Peng. Approaching optimality for solving SDD linear systems. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 235–244, Washington, DC, USA, 2010. IEEE Computer Society.

[26] I. Koutis, G. L. Miller, and R. Peng. A nearly-m log n time solver for SDD linear systems. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 590–598, Washington, DC, USA, 2011. IEEE Computer Society.

[27] Y. T. Lee and A. Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. *CoRR*, abs/1305.1922, 2013.

[28] A. Madry. Navigating central path with electrical flows: from flows to matchings, and back. *CoRR*, abs/1307.2205, 2013.

[29] G. L. Miller and R. Peng. Approximate maximum flow on separable undirected graphs. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1151–1170. SIAM, 2013.

[30] A. Naor and O. Neiman. Assouad's theorem with dimension independent of the snowflaking. *arXiv preprint arXiv:1012.2307*, 2010.

[31] L. Orecchia, S. Sachdeva, and N. K. Vishnoi. Approximating the exponential, the Lanczos method and an $\tilde{O}(m)$-time spectral algorithm for balanced separator. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 1141–1160, New York, NY, USA, 2012. ACM.

[32] R. Peng. *Algorithm Design Using Spectral Graph Theory*. PhD thesis, Carnegie Mellon University, Pittsburgh, August 2013. CMU CS Tech Report CMU-CS-13-121.

[33] R. Peng and D. A. Spielman. An efficient parallel solver for SDD linear systems. *CoRR*, abs/1311.3286, 2013.

[34] M. Rudelson and R. Vershynin. Sampling from large matrices: An approach through geometric functional analysis. *J. ACM*, 54(4):21, 2007.

[35] M. Sagae and K. Tanabe. Upper and lower bounds for the arithmetic-geometric-harmonic means of positive definite matrices. *Linear and Multilinear Algebra*, 37(4):279–282, 1994.

[36] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 563–568, 2008.

[37] D. A. Spielman and S.-H. Teng. Solving sparse, symmetric, diagonally-dominant linear systems in time $O(m^{1.31})$. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '03, pages 416–, Washington, DC, USA, 2003. IEEE Computer Society.

[38] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 81–90, June 2004.

[39] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2008. Available at `http://www.arxiv.org/abs/cs.NA/0607105`. Submitted to SIMAX.

[40] D. A. Spielman and J. Woo. A note on preconditioning by low-stretch spanning trees. *CoRR*, abs/0903.2816, 2009.

[41] M. Thorup. Floats, integers, and single source shortest paths. *Journal of Algorithms*, 35:189–201, 2000.

[42] J. A. Tropp. User-friendly tail bounds for sums of random matrices. *Found. Comput. Math.*, 12(4):389–434, Aug. 2012.

[43] P. M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. A talk based on this manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991.

[44] A. Zouzias. A matrix hyperbolic cosine algorithm and applications. In *Proceedings of the 39th international colloquium conference on Automata, Languages, and Programming - Volume Part I*, ICALP'12, pages 846–858, Berlin, Heidelberg, 2012. Springer-Verlag.