Approaching optimality for solving SDD linear systems[*]

Ioannis Koutis[†] Gary L. Miller Richard Peng[‡]
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213
{ioannis.koutis,glmiller,yangp}@cs.cmu.edu

April 30, 2010

Abstract

We present an algorithm that on input a graph G with n vertices and $m+n-1$ edges and a value k, produces an *incremental sparsifier* $\hat{G}$ with $n-1+m/k$ edges, such that the condition number of G with $\hat{G}$ is bounded above by $\tilde{O}(k\log^2 n)$, with probability $1-p$. The algorithm runs in time

$$\tilde{O}((m\log n + n\log^2 n)\log(1/p)).^{1}$$

As a result, we obtain an algorithm that on input an $n\times n$ symmetric diagonally dominant matrix A with $m+n-1$ non-zero entries and a vector b, computes a vector $\bar{x}$ satisfying $||\bar{x}-A^{+}b||_A < \epsilon||A^{+}b||_A$, in time

$$\tilde{O}(m\log^2 n\log(1/\epsilon)).$$

The solver is based on a recursive application of the incremental sparsifier that produces a hierarchy of graphs which is then used to construct a recursive preconditioned Chebyshev iteration.

1 Introduction

Fast algorithms for solving linear systems and the related problem of finding a few fundamental eigenvectors is possibly one of the most important problems in algorithm design. It has motivated work on fast matrix multiplication methods, graph separators, and more recently graph sparsifiers. For most applications the matrix is sparse and thus one would like algorithms whose run time is efficient in terms of the number of non-zero entries of the matrix. Little is known about how to efficiently solve general sparse systems, $Ax=b$. But substantial progress has been made in the case of symmetric and diagonally dominant (SDD) systems, where $A_{ii}\geq\sum_{j\neq i}|A_{ij}|$. In a seminal work, Spielman and Teng showed that SDD systems can be solved in nearly-linear time [ST04, EEST05, ST06].

Recent research, largely motivated by the Spielman and Teng solver (ST-solver), reveals the power of SDD systems as an algorithmic primitive. The ST-solver is the key subroutine of the fastest known algorithms for a multitude of problems that include: (i) The computation of the first few eigenvectors of the graph Laplacian or normalized Laplacian; the relationship between the first non-trivial (Fiedler) eigenvector and the sparsest cut problem is well known and has been widely used in theory and practice [Fie73, ST96]. (ii) Fast spectral sparsifiers that also act as cut sparsifiers [SS08] (iii) The solution of linear systems derived from elliptic finite elements discretizations of a wide class of partial differential equations [BHV04]. (iv) Generalized lossy flow problems [SD08]. (v) The problem of generating a random spanning

[*]Partially supported by the National Science Foundation under grant number CCF-0635257.

[†]Partially supported by Microsoft Research through the Center for Computational Thinking at CMU

[‡]Partially supported by Natural Sciences and Engineering Research Council of Canada (NSERC) under grant number M-377343-2009.

[1]We use the $\tilde{O}()$ notation to hide a factor of at most $(\log\log n)^4$

tree [KM09]. (vi) Several optimization problems in computer vision [KMT09, KMST09b] and graphics [MP08, JMD+07].

The ST-solver is an *iterative* algorithm that produces a sequence of approximate solutions converging to the actual solution of the input system $Ax = b$. The performance of iterative methods is commonly measured in terms of the time required to reduce by a constant factor an appropriately defined approximation error. Even including recent improvements on some of its components, the time complexity of the ST-solver is at least $O(m \log^{15} n)$. The large exponent in the logarithm is indicative of the fact that the algorithm is quite complicated and lacks practicality. The design of a faster and simpler solver is a challenging open question.

In this paper we present a conceptually simple and possibly practical iterative solver that runs in time $\tilde{O}(m \log^2 n)$. Its main ingredient is a new *incremental* graph sparsification algorithm, which is of independent interest. The paper is organized as follows. In Section 2 we review basic notions and we introduce notation. A reader familiar with very basic notions from spectral graph theory and linear algebra can skip this Section. In Section 3 we discuss the development of SDD solvers, the algorithmic questions it motivated, and the progress on them, with an emphasis on the graph sparsification problem. The presentation is self-contained and with a minimum level of technical details, but the reader can always consult Section 2 for basic definitions. In Section 4 we present a high level description of our approach and discuss implications of our solver for the graph sparsification problem. The incremental sparsifier is presented and analyzed in Sections 5 and 6. In Section 7 we explain how it can be used to construct the solver.

2 Preliminaries

In this Section, we briefly recall background facts about weighted graph Laplacians. For more details, we refer the reader to [RG97] and [BH03]. Throughout the paper, we discuss connected graphs with positive edge weights. We use n and m to denote $|V|$ and $|E|$.

A matrix A is positive semi-definite if for any vector x, $x^T Ax \geq 0$. For such semi-definite matrices A, we can also define the A-norm as follows:

$$||x||_A^2 = x^T Ax.$$

Fix an arbitrary numbering of the edges of a graph G. Let $w_{i,j}$ denote the weight of the edge (i,j) The Laplacian L_G of G is the matrix defined by: (i) $L_G(i,j) = -w_{i,j}$. (ii) $L_G(i,i) = \sum_{i \neq j} w_{i,j}$. It can be checked that for any vector x, we have

$$x^T L_G x = \sum_{u,v \in E} (x_u - x_v)^2 w_{uv}.$$

It follows that L_G is positive semi-definite and L_G-norm is a valid norm.

We also define a partial order $\preceq$ on semi-definite matrices, where $A \preceq B$ if $B - A$ is positive semi-definite. This is equivalent to $x^T Bx \geq x^T Ax$ for all x. We will say that a graph H *κ-approximates* a graph G if

$$L_H \preceq L_G \preceq \kappa L_H.$$

By the definition of $\preceq$ from above, this is equivalent to $x^T L_H x \leq x^T L_G x \leq \kappa x^T L_H x$ for all vectors x. This implies that the *condition number* of the pair (L_G, L_H) is upper bounded by κ. The condition number is an algebraically motivated notion; upper bounds on it are used to measure the of iterative numerical algorithms.

3 Prior work on SDD solvers and related graph theoretic problems

Symmetric diagonally dominant systems are linear-time reducible to linear systems whose matrix is the Laplacian of a weighted graph via a construction known as double cover which only doubles the number of non-zero entries in the system [GMZ95, Gre96]. The one-to-one correspondence between graphs and their Laplacians allows us to focus on weighted graphs, and interchangeably use the words graph and Laplacian.

In a ground-breaking approach, Vaidya [Vai91] proposed the use of spectral graph-theoretic properties for the design of provably good graph *preconditioners*, i.e. graphs that -in some sense- approximate the input graph, but yet are somehow easier to solve. Many authors built upon the ideas of Vaidya, to develop *combinatorial preconditioning*, an area on the border of numerical linear algebra and spectral graph theory [BGH$^+$05]. The work in the present paper as well as the Spielman and Teng solver is based on this approach. It is worth noting that combinatorial preconditioning is only one of the rich connections between combinatorics and linear algebra [Chu97, RG97].

Vaidya originally proposed the construction of a preconditioner for a given graph, based on a maximum weight spanning tree of the graph and its subsequent augmentation with graph edges. This yielded the first non-trivial results, an $O((dn)^{1.75})$ time algorithm for maximum degree d graphs, and an $O((dn)^{1.2})$ algorithm for maximum degree d planar graphs [Jos97].

Later, Boman and Hendrickson [BH03] made the crucial observation that the notion of *stretch* (see Section 6 for a definition) is crucial for the construction of a good spanning tree preconditioner; they showed that if the non-tree edges have average stretch s over a spanning tree, the spanning tree is an $O(sm)$-approximation of the graph. Armed with this observation and the low-stretch of Alon et al. [AKPW95], Spielman and Teng [ST03] presented a solver running in time $O(m^{1.31})$.

The utility of low-stretch trees in SDD solvers motivated further research on the topic. Elkin et al. [EEST05] gave a $O(m \log^2 n)$ time algorithm for the computation of spanning trees with total stretch $\tilde{O}(m \log^2 n)$. More recently, Abraham et. al. presented a nearly tight construction of low-stretch trees [ABN08], giving an $O(m \log n + n \log^2 n)$ time algorithm that on input a graph G produces a spanning tree of total stretch $\tilde{O}(m \log n)$. The algorithm of [EEST05] is a basic component of the ST-solver. While the algorithm of [ABN08] didn't improve the ST-solver, it is indispensable to our upper bound.

The major new notion introduced by Spielman and Teng [ST04] in their nearly-linear time algorithm was that of a *spectral sparsifier*, i.e. a graph with a nearly-linear number of edges that α-approximates a given graph for a constant α. Before the introduction of spectral sparsifiers, Benczúr and Karger [BK96] had presented an $O(m \log^3 n)$ algorithm for the construction of a *cut-preserving* sparsifier with $O(n \log n)$ edges. A good spectral sparsifier is a also a good cut-preserving sparsifier, but the opposite is not necessarily true.

The ST-solver [ST04] consists of a number of major algorithmic components. The base routine is a local partitioning algorithm which is the main subroutine of a global nearly-linear time partitioning algorithm. The partitioning algorithm is used as a subroutine in the construction of the spectral sparsifier. Finally, the spectral sparsifier is combined with the $O(m \log^2 n)$ total stretch spanning trees of [EEST05] to produce a $(k, O(k \log^c n))$ *ultrasparsifier*, i.e. a graph $\hat{G}$ with $n - 1 + (n/k)$ edges which $O(k \log^c n)$-approximates the given graph, for some $c > 25$. The bottleneck in the complexity of the ST-solver lies in the running time of the ultra-sparsification algorithm and the approximation quality of the ultrasparsifier.

In the special case of planar graphs the ST-solver runs in time $\tilde{O}(n \log^2 n)$. An asymptotically optimal linear work algorithm for planar graphs was given in [KM07]. The key observation in [KM07] was that despite the fact that planar graphs don't necessarily have spanning trees of average stretch less than $O(\log n)$, they still have $(k, ck \log k)$ ultrasparsifiers for a large enough constant c; they can be obtained by finding ultrasparsifiers for constant size subgraphs that contain most of theedges of the graph, and conceding the rest of the edges in the global ultrasparsifier. In addition, a more practical approach to the construction of constant-approximation preconditioners for the case of graphs of bounded average degree was given in [KM08]. To this day, the only known improvement for the general case was obtained by Andersen et.al [ACL06] who presented a faster and more effective local partitioning routine that can replace the core routine of Spielman and Teng, improving the complexity of the sparsifier and the solver.

Significant progress has been made on the spectral graph sparsification problem. Spielman and Srivastava [SS08] showed how to construct a much stronger spectral sparsifier with $O(n \log n)$ edges, by sampling edges with probabilities proportional to their effective resistance, if the graph is viewed as an electrical network. While the algorithm is conceptually simple and attractive, its fastest known implementation still relies on the ST-solver. Leaving the envelope of nearly-linear time algorithms Batson, Spielman and Srivastava [BSS09] presented a polynomial time algorithm for the construction of a "twice-Ramanujan" spectral sparsifier with a nearly optimal linear number of edges. Finally, Kolla et al. [KMST09a] gave a polynomial time algorithm for the construction of a nearly-optimal $(k, \tilde{O}(k \log n))$ ultrasparsifier.

4 Our contribution

In an effort to design a faster sparsification algorithm, we ask: when and why the much simpler faster cut-preserving sparsifier of [BK96] fails to work as a spectral sparsifier? Perhaps the essential example is that of the cycle and the line graph; while the two graphs have roughly the same cuts, their condition number is $O(n)$. The missing edge has a stretch of $O(n)$ through the rest of the graph, and thus it has high effective resistance; the effective resistance-based algorithm of Spielman and Srivastava would have kept this edge. It is then natural to try to design a sparsification algorithm that avoids precisely to generate a graph whose "missing" edges have a high stretch over the rest of the original graph.

This line of reasoning leads us to a conceptually simple sparsification algorithm: find a low-stretch spanning tree with a total stretch of $O(m \log n)$. Scale it up by a factor of k so the total stretch is $O(m \log n/k)$ and add the scaled up version to the sparsifier. Then over-sample the rest of the edges with probability proportional to their stretch over the scaled up tree, taking $\tilde{O}(m \log^2 n/k)$ samples. In Sections 5 and 6 we analyze a slight variation of this idea and we show that while it doesn't produce an ultrasparsifier, it produces what we call an *incremental sparsifier* which is a graph with $n-1+m/k$ edges that $\tilde{O}(k \log^2 n)$-approximates the given graph [2]. Our proof relies on the machinery developed by Spielman and Srivastava [SS08].

As we explain in Section 7 the incremental sparsifier is all we need to design a solver that runs in the claimed time. Precisely, we prove the following.

Theorem 4.1 *On input an $n \times n$ symmetric diagonally dominant matrix A with m non-zero entries and a vector b, a vector $\bar{x}$ satisfying $||\bar{x} - A^+ b||_A < \epsilon ||A^+ b||_A$, can be computed in expected time $\tilde{O}(m \log^2 n \log(1/\epsilon))$.*

4.1 Implications for the graph sparsification problem

The only known nearly-linear time algorithm that produces a spectral sparsifier with $O(n \log n)$ edges is due to Spielman and Srivastava [SS08], and it is based on $O(\log n)$ calls to a SDD linear system solver. Our solver brings the running time of the Spielman and Srivastava algorithm to $\tilde{O}(m \log^3 n)$. It is interesting that this algebraic approach matches up to $\log \log n$ factors the running bound of the purely combinatorial algorithm of Benczúr and Karger [BK96] for the computation of the (much weaker) cut-preserving sparsifier.

Sparsifying once with the Spielman and Srivastava algorithm and then applying our incremental sparsifier gives a $(k, O(k \log^3 n))$ ultrasparsifier that runs in $\tilde{O}(m \log^3 n)$ randomized time. Within the envelope of nearly-linear time algorithms, this becomes the best known ultrasparsification algorithm in terms of both its quality and its running time. Our guarantee on the quality of the ultrasparsifier is off by a factor of $O(\log^2 n)$ comparing to the ultrasparsifier presented in [KMST09a]. In the special case where the input graph has $O(n)$ edges, our incremental sparsifier is a $(k, O(k \log^2 n))$ ultrasparsifier.

[2]In the latest version of their paper [ST06], Spielman and Teng also construct and use an incremental sparsifier, but they still use the term ultrasparsifier for it.

5 Sparsification by Oversampling

In this section we revisit a sampling scheme proposed by Spielman and Srivastava for sparsifying a graph, [SS08]. Consider the following general sampling scheme:

```
SAMPLE
Input: Graph $G = (V, E, w)$, $p' : E \to \mathbb{R}^+$, integer $q$.
Output: Graph $G' = (V, E', w')$.

• $t := \sum_e p'_e$
• $p_e := p'_e / t$
• $G' := (V, E', w')$ with $E' = \emptyset$
• FOR $q$ times
•       Pick edge $e$ in $G$ with probability $p_e$
•       Add $e$ to $E'$ with weight $w'_e = w_e / p_e$
• ENDFOR
• For all $e \in E'$, let $w_{e'} := w_e / q$
• RETURN $G'$
```

Figure 1: The sampling Algorithm

Spielman and Srivastava pick $p'_e = w_e R_e$ where R_e is the effective resistance of e in G. This choice returns a spectral sparsifier. Calculating good approximations to the effective resistances seems to be at least as hard as solving a system, but as we will see in Section 6, it is easier to compute numbers $p'_e \geq (w_e R_e)$. The following Theorem considers a sampling scheme based on numbers with this property.

Theorem 5.1 ***(Sampling higher than effective resistance)*** *Given $G = (V, E, w)$, let $p'_e \geq w_e R_e$ for each edge $e \in E$ and $\xi \in \Omega(1/n)$. Let $t = \sum_e p'_e$ and $q = C_s t \log t \log(1/\xi)$, where C_s is a constant independent from G. Then if $G' = \textsc{Sample}(G, p', q)$, we have*

$$G \preceq 2G' \preceq 3G$$

with probability at least $1 - \xi$.

The proof follows closely that Spielman and Srivastava [SS08], with only a minor difference in one calculation. Let us first review some necessary lemmas.

If we assign arbitrary orientations on the edges, then we can define the incidence matrix $\Gamma \in \Re^{m \times n}$ as follows:

$$\Gamma_{e,u} = \begin{cases} -1 & \text{if u is the head of e} \\ 1 & \text{if u is the tail of e} \\ 0 & \text{otherwise} \end{cases}$$

Let W be the diagonal matrix containing edge weights, then $W^{1/2}$ is a real positive diagonal matrix as well since all edge weights are positive. The Laplacian L can be written in terms of W and Γ as

$$L = \Gamma^T W \Gamma = \Gamma^T W^{1/2} W^{1/2} \Gamma.$$

Algorithm $\textsc{Sample}$ forms a new graph by multiplying each edge e by a nonnegative number s_e. If $\mathbf{S}$ is the diagonal matrix with $S(e,e) = s_e$, the Laplacian of the new graph can be seen to be equal to

$$\tilde{L} = \Gamma^T W \Gamma = \Gamma^T W^{1/2} \mathbf{S} W^{1/2} \Gamma.$$

Let L^+ denote the Moore-Penrose of L, i.e. the unique matrix sharing with L its null space, and acting as the inverse of L in its range. The key to the proofs of [SS08] is the $m \times m$ matrix

$$\Pi = W^{1/2} \Gamma L^+ \Gamma^T W^{1/2},$$

for which the following lemmas are proved.

Lemma 5.2 *(Lemma 3i in [SS08])* Π *is a projection matrix, i.e.* $\Pi^2 = \Pi$.

Lemma 5.3 *(Lemma 4 in [SS08])*

$$(1 - ||\Pi\Pi - \Pi S \Pi||_2) L \preceq \tilde{L} \preceq (1 + ||\Pi\Pi - \Pi S \Pi||_2) L.$$

We also use Lemma 5.4 below, which is Theorem 3.1 from Rudelson & Vershynin [RV07]. The first part of the Lemma was also used as Lemma 5 in [SS08] in a similar way.

Lemma 5.4 *Let p be a probability distribution over $\Omega \subseteq R^d$ such that* $\sup_{y \in \Omega} ||y||_2 \leq M$ *and* $||\mathbb{E}_p(yy^T)||_2 \leq 1$. *Let $y_1 \dots y_q$ be independent samples drawn from p, and let*

$$a := CM \sqrt{\frac{\log q}{q}}.$$

Then:

1.
$$\mathbb{E}||\frac{1}{q}\sum_{i=1}^{q} y_i y_i^T - \mathbb{E}(yy^T)||_2 \leq a.$$

2.
$$Pr[||\frac{1}{q}\sum_{i=1}^{q} y_i y_i^T - \mathbb{E}(yy^T)||_2 > x] \leq \frac{2}{e^{cx^2/a^2}}.$$

Here C and c are fixed constants.

Proof (of Theorem 5.1) The algorithm draws samples $y_1, \dots, y_q$ from

$$y = \frac{1}{\sqrt{p_e}} \Pi(\cdot, e) \text{ with probability } p_e.$$

In this way we ensure $E(yy^T) = \Pi\Pi = \Pi$. We also have $||\Pi||_2 \leq 1$ as it is a projection matrix. So, the conditions of Lemma 5.4 are satisfied. The fact that Π is a projection matrix also gives $\Pi(:,e)^T \Pi(:,e) = (\Pi\Pi)(e,e) = \Pi(e,e)$, which we use to bound M as follows.

$$M = \sup_e \frac{1}{\sqrt{p_e}} ||\Pi(:,e)||_2 = \sup_e \frac{1}{\sqrt{p_e}} \sqrt{\Pi(e,e)} = \sup_e \frac{1}{\sqrt{p_e}} \sqrt{w_e R_e} = \sup_e \sqrt{\frac{t}{p'_e}} \sqrt{w_e R_e} \leq \sqrt{t}. \tag{5.1}$$

The last inequality follows from the assumption about the p'_e. Recall now that we have $\log(1/\xi) \leq \log n$ by assumption, $t \geq \sum_e w_e R_e$ by construction, and $\sum_e w_e R_e = n-1$ by Lemma 3 in [SS08]. Combining these facts and setting $q = c_S t \log t \log(1/\xi)$ for a proper constant c_S, part 1 of Lemma 5.4 gives

$$a \leq \sqrt{\frac{4}{c \log(2/\xi)}}.$$

Now substituting $x = \frac{1}{2}$ into part 2 of Lemma 5.4, we get

$$Pr[||\frac{1}{q}\sum_{i=1}^{q} y_i y_i^T - E(yy^T)||_2 > 1/2] \leq \frac{2}{e^{(c/4)/a^2}} \leq \frac{2}{e^{(c/4)/(4/c\log 2/\xi)}} \leq \xi.$$

It follows that with probability at least $1-\xi$ we have

$$||\frac{1}{q}\sum_{i=1}^{q} y_i y_i^T - E(yy^T)||_2 \leq 1/2,$$

which implies $||\Pi S \Pi - \Pi||_2 \leq 1/2$. The theorem then follows by Lemma 5.3. ■

Note. The upper bound for M in inequality 5.1 is in fact the only place where our proof differs from that of [SS08]. In their case the last inequality is replaced by an exact inequality, which is possible because the exact values for $w_e R_e$ are used. In our case, by using inexact values we get a weaker upper bound which reflects in the density (depending on m, not n) of the incremental sparsifier. It is however enough for the solver.

6 Incremental Sparsifier

Consider a spanning tree T of $G = (V, E, w)$. Up until now we have been thinking of the weights as conductors. We now invert the weights and view them as resistors. Let $w'(e) = 1/w(e)$. Let the unique path connecting the endpoints of e consists of edges $e_1 \ldots e_k$, the stretch of e by T is defined to be

$$\frac{\sum_{i=1}^{k} w'(e_i)}{w'(e)} = stretch_T(e).$$

But $\sum_{i=1}^{k} 1/w(e_i) = R(T)_e$ the effective resistance of e in T. Thus $stretch_T(e) = w_e R(T)_e$. By Rayleigh's monotonicity law [DS00], we have $R(T)_e \geq R_e$, so $stretch_T(e) \geq w_e R_e$. Our algorithm will based on the construction of a low-stretch tree, with the guarantees provided by the following result of Abraham, Bartal, and Neiman [ABN08].

Theorem 6.1 *Given a graph $G = (V, E, w')$, LowStretchTree(G) in time $O(m \log n + n \log^2 n)$, outputs a spanning tree T of G satisfying $\sum_{e \in E} = O(m \log n \cdot \log\log n \cdot (\log\log\log n)^3)$.*

Our key idea is to scale up the low stretch tree by a factor of κ, incurring a condition number of κ but allowing us to sample the non-tree edges aggressively by the upper bounds on their effective resistances given by the tree. The details are given in algorithm IncrementalSparsify given in Figure 2.

Theorem 6.2 *Given a graph G with n vertices, m edges and any values $\kappa < m$, $\xi \in \Omega(1/n)$, IncrementalSparsify finds H with $n - 1 + \tilde{O}((m/\kappa)\log^2 n \log(1/\xi))$ edges, such that $G \preceq H \preceq 3\kappa G$ with probability at least $1-\xi$, in $O(n \log^2 n \log(1/\xi) + m \log n + m \log^3 n \log(1/\xi)/\kappa)$ time.*

Proof Since the weight of an edge is increased by at most a factor of κ, we have $G \preceq G' \preceq \kappa G$. Furthermore, the effective resistance along the tree of each non-tree edge decreases by a factor of κ. So

```
IncrementalSparsify
Input: Graph $G$, reals $\kappa, \xi$.
Output: Graph $H$

• $T \leftarrow$ LowStretchTree(G)
• Let $T'$ be $T$ scaled by a factor of $\kappa$
• Let $G'$ be the graph obtained from $G$ by replacing $T$ with $T'$
• FOR $e \in E$
•     Calculate $stretch_{T'}(e)$
• ENDFOR
• $H \leftarrow$ Sample($G'$, $stretch_{T'}$, $1/2\xi$)
• RETURN $2H$
```

Figure 2: Incremental Sparsifier

we set $p'_e = 1$ if $e \in T$ and $stretch_T(e)/\kappa$ otherwise, and invoke Sample to get a graph H such that with probability at least $1 - 1/\xi$, we get

$$G \preceq G' \preceq H \preceq 3G' \preceq 3\kappa G.$$

We first bound the number of non-tree edges. Let $t' = \sum_{e \notin T} stretch_{T'}(e)$, with $t' = \tilde{O}(m \log n/\kappa)$. Then for the number t used in Sample we have $t = t' + n - 1$ and $q = C_s t \log t \log(1/\xi)/\kappa$ is the number of edges sampled in the call of Sample. Let X_i be a random variable which is 1 if the i^{th} edge picked by Sample is a non-tree and 0 otherwise. The total number of non-tree edges sampled is the random variable $X = \sum_{i=1}^{q} X_i$, and its expected value can be calculated using the fact $Pr(X_i = 1) = t'/t$:

$$E[X] = q\frac{t'}{t} = t'\frac{C_s t \log t \log(1/\xi)}{\kappa t} = \tilde{O}((m/\kappa)\log^2 n \log(1/\xi)).$$

A standard form of Chernoff's inequality is:

$$Pr[X > (1+\delta)E[X]] < \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^{E[X]}.$$

Letting $\delta = 2$, and using the assumption $k < m$, we get $Pr(X > 3E[X]) < (e^2/27)^{E[X]} < 1/n^c$, for any constant c. Hence, the probability that IncrementalSparsify succeeds, with respect to both the number of non-tree edges and the condition number, is at least $1 - \xi$.

We now turn to the running time claim. The standard $O(m \log n)$ algorithm for computing least common ancestor allows us to calculate the stretch of all m edges in $O(m \log n)$ time. To compute the sample efficiently, we can assign each edge an interval on the unit interval $[0, 1]$ with length corresponding to its probability such that no two intervals overlap. At each sampling iteration, pick a random value in $[0, 1]$ and binary search to find the interval that contains it in $O(\log n)$ time. There are $O(t \log t \log(1/\xi))$ iterations of sampling, so the total time to compute the samples is $\tilde{O}(n \log^2 n \log(1/\xi) + m \log^3 n \log(1/\xi)/\kappa)$. ■

7 Solver Using Incremental Sparsifier

The solver of Spielman and Teng [ST06] works by: (i) building a chain $\mathcal{C}$ of graphs that satisfies a list of requirements, (ii) using $\mathcal{C}$ along with the b side of the system as input to a recursive preconditioned

Chebyshev method that produces an approximate solution of the system. Our approach differs only in the way we build the chain $\mathcal{C}$. We state without proof or details a Lemma listing the requirements for the chain. For details, we refer the reader to [ST06]. Before we proceed, we review GreedyElimination, a key procedure for the construction of the chain.

```
GreedyElimination
Input: Weighted graph $G = (V, E, w)$
Output: Weighted graph $\hat{G} = (\hat{V}, \hat{E}, \hat{w})$

• $\hat{G} := G$.
• UNTIL there are nodes of degree 1 or 2 in $\hat{G}$
•       Greedily remove all degree 1 nodes
•       If $v$ is a degree node with adjacent edges $e_1$ and $e_2$,
...       replace $e_1, e_2$ by an edge of weight $(1/w(e_1) + 1/w(e_2))^{-1}$
```

We will make use of the following (adapted) Lemma from Spielman and Teng [ST06].

Lemma 7.1 ***(Chain requirements)*** *Let A be a graph, and assume we are given a sequence of graphs $\{A = A_1, B_1, A_2, \ldots, A_d\}$ such that*

- *A_i has n_i nodes and $m_i + n_i - 1$ edges.*
- *$A_i \preceq B_i \preceq \kappa(n_i) A_i$, where κ is a fixed monotonic function.*
- *$A_{i+1} =$ GreedyElimination(B_i).*
- *$m_i / m_{i+1} \geq c\sqrt{\kappa(n_i)}$, for some constant c.*
- *m_d is some fixed constant c'.*

Then a vector $\bar{x}$ such that $||\bar{x} - L_A^+ b||_A < \epsilon ||L_A^+ b||_A$ can be computed in $O(m_d^3 m \sqrt{\kappa(n)} \log(1/\epsilon))$ time.

Spielman and Teng take B_i to be an ultrasparsifier of A_i. We take B_i to be the incremental sparsifier we constructed in Section 6. Let us now formally state the algorithm for building the chain of graphs.

```
BuildChain
Input: Graph $A$.
Output: Chain of graphs $\{A = A_1, B_1, A_2, \ldots, A_d\}$.

• Let $A_1 = A$.
• While $m_i > (\log\log n)^{1/3}$ do:
•       Let $\kappa = k'\tilde{O}(\log^2 n_i \log(1/p))$, where $k' = \tilde{O}(\log^2 n_i)$
•       If $m_i > \log n$ then $\xi := \log n$ else $\xi := \log\log n$
•       $B_i :=$ IncrementalSparsify$(A_i, \kappa, p/(2\xi))$
•       $A_{i+1} :=$ GreedyElimination$(B_i)$
•       if $|A_{i+1}| > |B_i|/k'$
•           return FAILURE
•       $i := i + 1$.
```

We now show that the chain constructed by BuildChain satisfies the requirements of Lemma 7.1.

Lemma 7.2 *Given a graph A,* BuildChain(A) *produces a chain that satisfies the requirements of Lemma 7.1, with probability at least $1-p$. The algorithm runs in expected time $\tilde{O}((m\log n + n\log^2 n)\log(1/p))$.*

Proof The second requirement of Lemma 7.1 is satisfied by construction. The call to IncrementalSparsify is set to construct an incremental sparsifier B_i with at most $n_i - 1 + m_i/k'$ edges, that $\tilde{O}(k'\log^2 n_i)$ approximates A_i. This happens with probability at least $1 - p/2\log n$ if $n_i > \log n$ and $1 - p/2\log\log n)$ otherwise. Note that since A_i is not reducible by GreedyElimination we get that $m_i > 2n_i$. Hence A_i has at least $2m_i$ edges. A key property of GreedyElimination is that if G is a graph with $n-1+j$ edges, GreedyElimination(G) has at most $2j-2$ vertices and $3j-3$ edges [ST06]. Hence GreedyElimination(B_i) has at most $4m_i/k'$ edges. It follows that $m_i/m_{i+1} \geq k'/2$. Thus taking $k' = \tilde{O}(\log^2 n_i)$ satisfies the other two requirements when $m_i > (\log\log n_i)^{1/3}$. The probability that the requirements hold for all i is at least

$$(1 - p/(2\log n))^{\log n}(1-p/(2\log\log n))^{\log\log n} > (1-p/2)^2 > 1-p.$$

Finally note that each call to IncrementalSparsify takes $\tilde{O}((m_i\log^2 n)\log(1/p))$ time. Since m_i decreases geometrically with i, the claim about the running time follows. ■

Combining Lemmas 7.1 and 7.2 proves our main Theorem.

Theorem 7.3 *On input an $n\times n$ symmetric diagonally dominant matrix A with mu non-zero entries and a vector b, a vector $\bar{x}$ satisfying $||\bar{x} - A^+b||_A < \epsilon ||A^+b||_A$, can be computed in expected time $\tilde{O}(m\log^2 n)\log(1/\epsilon))$.*

8 Comments / Extensions

Unraveling the analysis of our bound for the condition number of the incremental sparsifier, it can been that one $\log n$ factor is due to the number of samples required by the Rudelson and Vershynin theorem. The second $\log n$ factor is due to the average stretch of the low-stretch tree.

It is quite possible that the low-stretch construction and perhaps the associated lower bound can be bypassed -at least for some graphs- by a simpler approach similar to that of [KM07]. Consider for example the case of unweighted graphs. With a simple ball-growing procedure we can concede in our incremental sparsifier a $1/\log n$ fraction of the edges, while keeping within clusters of diameters $O(\log^2 n)$ the rest of the edges. The design of low-stretch trees may be simplified within the small diameter clusters. This diameter-restricted local sparsification is a natural idea to pursue, at least in an actual implementation of the algorithm.

References

[ABN08] Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *49th Annual IEEE Symposium on Foundations of Computer Science*, pages 781–790, 2008. 3, 6

[ACL06] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 475–486, Washington, DC, USA, 2006. IEEE Computer Society. 3

[AKPW95] Noga Alon, Richard Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM J. Comput.*, 24(1):78–100, 1995. 3

[Axe94] Owe Axelsson. *Iterative Solution Methods*. Cambridge University Press, New York, NY, 1994.

[BGH$^+$05] Marshall Bern, John R. Gilbert, Bruce Hendrickson, Nhat Nguyen, and Sivan Toledo. Support-graph preconditioners. *SIAM J. Matrix Anal. Appl.*, 27:930–951, 2005. 3

[BH03] Erik G. Boman and Bruce Hendrickson. Support theory for preconditioning. *SIAM J. Matrix Anal. Appl.*, 25(3):694–717, 2003. 2, 3

[BHV04] Erik G. Boman, Bruce Hendrickson, and Stephen A. Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. *CoRR*, cs.NA/0407022, 2004. 1

[BK96] András A. Benczúr and David R. Karger. Approximating s-t Minimum Cuts in $\tilde{O}(n^2)$ time Time. In *STOC*, pages 47–55, 1996. 3, 4, 4.1

[BSS09] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 255–262, 2009. 3

[Chu97] F.R.K. Chung. *Spectral Graph Theory*, volume 92 of *Regional Conference Series in Mathematics*. American Mathematical Society, 1997. 3

[Dem97] James W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

[DS00] Peter G. Doyle and J. Laurie Snell. Random walks and electric networks, 2000. 6

[EEST05] Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 494–503, 2005. 1, 3

[Fie73] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Math. J.*, 23(98):298–305, 1973. 1

[Geo73] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10:345–363, 1973.

[GMZ95] K.D. Gremban, Gary L. Miller, and M. Zagha. Performance evaluation of a parallel preconditioner. In *9th International Parallel Processing Symposium*, pages 65–69, Santa Barbara, April 1995. IEEE. 3

[Gre96] Keith Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. PhD thesis, Carnegie Mellon University, Pittsburgh, October 1996. CMU CS Tech Report CMU-CS-96-123. 3

[JMD$^+$07] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. *ACM Trans. Graph.*, 26(3):71, 2007. 1

[Jos97] Anil Joshi. *Topics in Optimization and Sparse Linear Systems*. PhD thesis, University of Illinois at Urbana Champaing, 1997. 3

[KM07] Ioannis Koutis and Gary L. Miller. A linear work, $O(n^{1/6})$ time, parallel algorithm for solving planar Laplacians. In *Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA 2007)*, 2007. 3, 8

[KM08] Ioannis Koutis and Gary L. Miller. Graph partitioning into isolated, high conductance clusters: Theory, computation and applications to preconditioning. In *Symposiun on Parallel Algorithms and Architectures (SPAA)*, 2008. 3

[KM09] Jonathan A. Kelner and Aleksander Madry. Faster generation of random spanning trees. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:13–21, 2009. 1

[KMST09a] Alexandra Kolla, Yury Makarychev, Amin Saberi, and Shanghua Teng. Subgraph sparsification and nearly optimal ultrasparsifiers. *CoRR*, abs/0912.1623, 2009. 3, 4.1

[KMST09b] Ioannis Koutis, Gary L. Miller, Ali Sinop, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. Technical report, CMU, 2009. 1

[KMT09] Ioannis Koutis, Gary L. Miller, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. In *International Symposium of Visual Computing*, pages 1067–1078, 2009. 1

[LRT79] R.J. Lipton, D. Rose, and R.E. Tarjan. Generalized nested dissection. *SIAM Journal of Numerical Analysis*, 16:346–358, 1979.

[MP08] James McCann and Nancy S. Pollard. Real-time gradient-domain painting. *ACM Trans. Graph.*, 27(3):1–7, 2008. 1

[RG97] Gordon Royle and Chris Godsil. *Algebraic Graph Theory*. Graduate Texts in Mathematics. Springer Verlag, 1997. 2, 3

[RV07] Mark Rudelson and Roman Vershynin. Sampling from large matrices: An approach through geometric functional analysis. *J. ACM*, 54(4):21, 2007. 5

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

[SD08] Daniel A. Spielman and Samuel I. Daitch. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, May 2008. 1

[SS08] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances, 2008. 1, 3, 4, 4.1, 5, 5, 5.2, 5.3, 5, 5

[ST96] Daniel A. Spielman and Shang-Hua Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *FOCS*, pages 96–105, 1996. 1

[ST03] Daniel A. Spielman and Shang-Hua Teng. Solving Sparse, Symmetric, Diagonally-Dominant Linear Systems in Time $0(m^{1.31})$. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 416. IEEE Computer Society, 2003. 3

[ST04] Daniel A. Spielman and Shang-Hua 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*, pages 81–90, June 2004. 1, 3

[ST06] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2006. 1, 2, 7, 7

[Vai91] P.M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. A talk based on this manuscript, October 1991. 3