Faster spectral sparsification and numerical algorithms for SDD matrices

Ioannis Koutis
CSD-UPRRP
ioannis.koutis@upr.edu

Alex Levin
MATH-MIT
levin@mit.edu

Richard Peng
CSD-CMU
yangp@cs.cmu.edu

September 27, 2012

Abstract

We study algorithms for spectral graph sparsification. The input is a graph G with n vertices and m edges, and the output is a sparse graph $\tilde{G}$ that approximates G in an algebraic sense. Concretely, for all vectors x and any $\epsilon > 0$, $\tilde{G}$ satisfies

$$(1-\epsilon)x^T L_G x \leq x^T L_{\tilde{G}} x \leq (1+\epsilon)x^T L_G x,$$

where L_G and $L_{\tilde{G}}$ are the Laplacians of G and $\tilde{G}$ respectively.

We show that the fastest known algorithm for computing a sparsifier with $O(n \log n/\epsilon^2)$ edges can actually run in $\tilde{O}(m \log^2 n)$ time[1], an $O(\log n)$ factor faster than before. We also present faster sparsification algorithms for slightly dense graphs. Specifically, we give an algorithm that runs in $\tilde{O}(m \log n)$ time and generates a sparsifier with $\tilde{O}(n \log^3 n/\epsilon^2)$ edges. We also give an $\tilde{O}(m)$ time algorithm for graphs with more than $n \log^5 n (\log\log n)^3$ edges of polynomially bounded weights, and an $O(m)$ algorithm for unweighted graphs with more than $n \log^8 n (\log\log n)^3$ edges. The improved sparsification algorithms are employed to accelerate linear system solvers and algorithms for computing fundamental eigenvectors of slightly dense SDD matrices.

1 Introduction

The efficient transformation of dense instances of graph problems to nearly equivalent sparse instances is a very powerful tool in algorithm design. The idea, widely known as *graph sparsification*, was originally introduced by Benczúr and Karger [3] in the context of cut problems. Spielman and Teng [13] generalized the *cut-preserving sparsifiers* of Benczúr and Karger to the more powerful *spectral sparsifiers*, which preserve in an algebraic sense the Laplacian matrix of the dense graph. The main motivation of spectral sparsifiers was the design of nearly-linear time algorithms for the solution of symmetric diagonally dominant (SDD) linear systems. A matrix A is SDD if it is symmetric and for all i, $A_{ii} \geq \sum_{j \neq i} |A_{ij}|$.

Benczúr and Karger proved that, for arbitrary ϵ, cuts can be preserved within a factor of $1 \pm \epsilon$ by a graph with $O(n \log n/\epsilon^2)$ edges. This graph can be computed by a randomized algorithm that runs in $O(m \log^3 n)$ time[2], where m is the number of edges in the dense graph. Spielman and Teng gave the first construction of spectral sparsifiers, but the edge count of these objects was several log factors bigger than

[1]We use the $\tilde{O}()$ notation to hide one $\log\log n$ factor which in most cases it is due to the guarantees of the best currently known algorithm for computing low-stretch trees [1]. In the main part of the paper we provide a more accurate accounting of the running time.

[2]All sparsification algorithms in this paper are randomized with a probability failure inversely proportional to n. They consist of a preprocessing phase followed by the generation of the sparsifier which in general can be performed in time proportional to the number of edges in it (e.g. $O(n \log n/\epsilon^2)$). For the sake of conciseness our running time statements will include only the time for preprocessing and will omit the failure probability.

that of Benczúr and Karger's cut-preserving sparsifiers. However, recent progress that we review below allows now for the construction of spectral sparsifiers with $O(n \log n/\epsilon^2)$ edges in $\tilde{O}(m \log^3 n)$ time.

Sparsification can be employed to immediately accelerate algorithms for numerous problems. In several cases and depending on the density of the instance, the sparsification routine dominates the running time of the sparsifier-enhanced algorithm. This is a strong incentive for speeding up the construction of sparsifiers even further.

This problem was undertaken in the context of cut-preserving sparsifiers by Fung *et al.* [5]. Improving upon the work of Benczúr and Karger, they proved that there is an $O(m \log^2 n)$ time algorithm that computes a sparsifier with $O(n \log n/\epsilon^2)$ edges. This stands as the fastest known algorithm with this sparsity guarantee for general graphs. However, Fung *et al.* also showed that we can do even better on slightly more dense graphs. More concretely, they proved that there is an $O(m)$ time algorithm that computes a sparsifier with $O(n \log^2 n/\epsilon^2)$ edges. Note that by transitivity, a combination of the two algorithms can produce a graph with $O(n \log n/\epsilon^2)$ edges in $O(m + n \log^4 n)$ time. In other words, there is a linear time sparsification algorithm for graphs with more than $n \log^4 n$ edges.

This leads us to the main question we address in this paper: Is something analogous possible for spectral sparsification? We answer the question in the affirmative. We first show that a slight modification of the Spielman-Srivastava algorithm [12] can improve the run time to $\tilde{O}(m \log^2 n)$. This nearly matches the general case algorithm of [5]. We present three additional sparsification algorithms. The first is a variation of the Spielman-Srivastava algorithm that generates a sparsifier with $\tilde{O}(n \log^3 n/\epsilon^2)$ edges in $\tilde{O}(m \log n)$ time. The second produces a sparsifier with $\tilde{O}(n \log n/\epsilon^2)$ edges in $\tilde{O}(m)$ time, assuming the input has more than $n \log^5 n (\log \log n)^3$ edges whose weights are bounded by a polynomial in n. The third produces a sparsifier with $O(n \log n/\epsilon^2)$ edges in $O(m)$ time assuming the input is unweighted and has more than $n \log^8 n (\log \log n)^3$ edges.

Applications in numerical algorithms

The $(1 \pm \epsilon)$-sparsifiers we obtain can be employed in a standard way as preconditioners for SDD linear systems, giving us faster solvers for slightly dense graphs: (i) an $\tilde{O}(m)$ time solver for systems with more than $n \log^5 n (\log \log n)^3$ non-zero entries and (ii) an $O(m)$ time solver for Laplacians of unweighted graphs with more than $n \log^8 n (\log \log n)^3$ non-zero entries. The best previously known algorithm [11] runs in $\tilde{O}(m \log n \log(1/\delta))$ time.

In addition, our sparsification algorithms accelerate the computation of an approximate Fiedler eigenvector of a graph Laplacian L_G. An $(1+\epsilon)$-approximate eigenvector is a unit norm vector x such that $x^T L_G x$ is within a factor $1+\epsilon$ from the eigenvalue λ_2 of L_G. The algorithm consists of two steps: (i) computing a spectral sparsifier $\tilde{G}$ that $(1 \pm \epsilon/2)$-approximates the input graph G. (ii) computing a $(1+\epsilon/3)$-approximate eigenvector of $\tilde{G}$; this will automatically be an $(1 \pm \epsilon)$-approximate eigenvector of the (more) dense input graph because the spectral sparsification step preserves the eigenvalues of G within $1 \pm \epsilon/2$. Hence combining our sparsification algorithms with the inverse power method [14] (which consists of solving $O(\log n \log(1/\epsilon))$ systems in $L_{\tilde{G}}$) gives an approximate eigenvector in $\tilde{O}(m + n \log^5 n \log(1/\epsilon)/\epsilon^2)$ time. The fastest previously known algorithm runs in time $\tilde{O}(m \log^2 n \log(1/\epsilon))$. The same result applies to the computation of the Fiedler eigenvector of a normalized Laplacian $D^{-1/2} L_G D^{-1/2}$; applying the inverse power method on $D^{-1/2} L_{\tilde{G}} D^{-1/2}$ gives the required eigenvector.

We note here that one practical application of eigenvectors is in partitioning algorithms; the analysis of Cheeger's inequality [4] tells us how to turn an approximate Fiedler vector into a partition. Hence, we give an improvement to the running time of a fundamental graph partitioning algorithm. Finally we note that the computation of additional eigenvectors can be performed in the same amount of time (per vector) by restricting the action of the matrix to the complement of the subspace spanned by the previously computed eigenvectors.

2 Overview of our techniques

2.1 Brief background on spectral sparsification

The first algorithm for edge-efficient spectral sparsifiers was given by Spielman and Srivastava [12]. Their algorithm produces a sparsifier with $O(n \log n/\epsilon^2)$ edges in a very elegant way: it samples edges with replacement. The probability of sampling an edge is proportional to its weight multiplied by its effective resistance in the resistive electrical network associated with the given graph.

Computing the effective resistance of a given edge requires—almost by definition—the solution of a linear system on the graph Laplacian.[3] However, Spielman and Srivastava also provided a way of estimating *all* m effective resistances, via solving $O(\log n)$ SDD linear systems. This holds under the assumption that the SDD solver is direct, i.e. it outputs an exact solution. The use of a nearly-linear time *iterative* solver that computes approximate solutions introduces an additional source of imprecision; Spielman and Srivastava showed that solving the systems up to an inverse polynomial precision is sufficient for sparsification. This brings the running time of their algorithm to $\tilde{O}(m \log^{c+2} n)$, where c is the constant appearing in the running time of the SDD solver.

2.2 The $\tilde{O}(m \log^2 n)$ time algorithm

While the work of Spielman and Srivastava did not improve the running time of the SDD solver, it proved to be a decisive step towards the fast SDD solver of Koutis, Miller, and Peng [10, 11], which runs in time $\tilde{O}(m \log n \log(1/\delta))$, where δ is the desired precision. Using this solver in the Spielman and Srivastava sparsification sampling scheme immediately yields an $\tilde{O}(m \log^3 n/\epsilon^2)$ time algorithm. This brings us to the first contribution of this paper, a tighter analysis of the Spielman and Srivastava algorithm. In Section 5 we show that solving the systems up to *fixed* precision is actually sufficient for sparsification. This decreases the running time to $\tilde{O}(m \log^2 n/\epsilon^2)$.

2.3 Faster algorithms: The main idea

To get our two faster algorithms, we will trade accuracy in the computation of effective resistances for speed. The idea is to transform the input graph G into another graph H where effective resistances can be computed faster while still providing good bounds for the true effective resistances in G. These approximate effective resistances can still be used for sparsification at the expense of additional sampling [10] that yields slightly more dense sparsifiers. These sparsifiers can be re-sparsified to $O(n \log n/\epsilon^2)$ edges by applying the fast general-case algorithm.

2.4 The $\tilde{O}(m \log n)$ time algorithm

The $\tilde{O}(m \log n)$ time algorithm is based on the observation that the Spielman-Srivastava scheme can be implemented to run in $\tilde{O}(m \log n)$ time on a *spine-heavy* approximation H of G. The spine-heavy graph H is derived in $\tilde{O}(m \log n)$ time from G by computing a low-stretch tree of G and scaling it up by a $\tilde{O}(\log^2 n)$ factor. In [11] it was shown that linear systems involving the Laplacian of H can be solved in $\tilde{O}(m)$ time, enabling the faster implementation of the Spielman-Srivastava scheme on H. At the same time the effective resistances in H are at most a $\tilde{O}(\log^2 n)$ factor smaller than those in G. Sampling with respect to these estimates, allows us to get a sparsifier $\tilde{G}'$ with $O(n \log^3 n/\epsilon^2)$ edges. Re-sparsifying $\tilde{G}'$ gives a sparsifier $\tilde{G}$ with $O(n \log n/\epsilon^2)$ edges in $\tilde{O}(n \log^5 n)$ time. The details are given in Section 5.

2.5 The $\tilde{O}(m)$ and $O(m)$ time algorithms

The starting point for our fastest algorithm is an $O(m)$ time algorithm for computing an *approximate sparsifier*, i.e. a sparse approximation for the input graph, but of moderate quality. We will actually do this by sparsifying a graph H that is a κ-approximation of G. Then we observe that with some additional

[3]Laplacian matrices are SDD.

work we can 'leverage' the approximate sparsifier to compute a sparsifier for G as well. Indeed, let $\tilde{H}$ be a sparsifier of H with $O(n \log n)$ edges. Then we generate a low-stretch spanning tree T of $\tilde{H}$ in $\tilde{O}(n \log^2 n)$ time and approximate the effective resistances of G over T in $O(m)$ time. We will be able to claim that these approximate values are enough to generate a sparsifier $\tilde{G}'$ for G with $\tilde{O}(n\kappa \log^3 n)$ edges. Finally from $\tilde{G}'$ we can compute a sparsifier $\tilde{G}$ with $O(n \log n/\epsilon^2)$ edges in $O(\kappa n \log^5 n (\log\log n)^2)$ time using our first algorithm. The fact that we can tightly estimate the resistances in G using a tree T is a departure from the Spielman-Srivastava scheme and may be of independent interest.

We will derive our $O(m)$ algorithm via a single application of the above 'leveraging' idea for $\kappa = \tilde{O}(\log^3 n)$. To improve performance for sparser graphs we will progressively sparsify a sequence of $t = O(\log\log n)$ graphs $H = H_0, H_1, \ldots, H_t = G$, such that H_i is a 2-approximation of H_{i+1}: given the sparsifier for H_i we can construct the sparsifier for H_{i+1} within the claimed time. The details are given in Section 6.

3 Background on spectral graph theory and sparsification

3.1 The graph Laplacian and its pseudoinverse

Let $G = (V, E, w)$ be an undirected weighted graph on n vertices, which we identify with the integers $\{1, 2, \ldots, n\}$, and m edges, where the weight of edge e is given by w_e. Without loss of generality we will assume that minimum weight is 1. We will also assume that matrices are represented as adjacency lists.

The Laplacian of G is denoted by L_G. It is a symmetric $n \times n$ matrix with zero row and column sums, where the (i, j) off-diagonal entry is given by $-w_{(i,j)}$ if (i, j) is an edge of G and 0 otherwise. The ith diagonal entry is given by the weighted degree of vertex i.

If G is a connected graph, then L_G is a matrix of rank $n - 1$, with its kernel spanned by $\mathbf{1}$ (the vector of all 1's). We let L_G^+ denote the Moore-Penrose pseudoinverse of L_G; this is a matrix that acts as the inverse of L_G on $(\ker L_G)^\perp$, and satisfies $L_G^+ L_G = L_G L_G^+ = I_{n-1}$, where I_{n-1} is the projection onto the $(n-1)$-dimensional image of L_G.

Given the one-to-one correspondence of graphs and their Laplacians we will often apply algebraic notation to graphs, with the obvious meaning.

3.2 Spectral approximation and sparsification

In this paper we concentrate on symmetric diagonally dominant matrices. For two matrices A and B of the same dimension, we write $A \preceq B$ if $x^T A x \leq x^T B x$ for all vectors x. For two graphs G and H, we write $G \preceq H$ if the Laplacians satisfy $L_G \preceq L_H$.

Definition 3.1 *We say that a graph H is a κ-approximation of a graph G if $G \preceq H \preceq \kappa G$.*

It is not hard to show that if H is a graph that κ-approximates a graph G then we have

$$\frac{1}{\kappa} L_G^+ \preceq L_H^+ \preceq L_G^+ \tag{3.1}$$

Definition 3.2 *Given a graph G, we say that a (sparser) graph H is a $1 \pm \epsilon$* **spectral sparsifier** *of G if*

$$(1-\epsilon) G \preceq H \preceq (1+\epsilon) G. \tag{3.2}$$

It is easy to see that if H is a $1 \pm \epsilon$ spectral sparsifier of G then $\frac{1}{1-\epsilon} H$ is a graph that $\frac{1+\epsilon}{1-\epsilon}$-approximates G. By the definition, it is also easy to verify **transitivity**. If G_1 is a $1 \pm \epsilon_1$ sparsifier of G and G_2 is a $1 \pm \epsilon_2$ of G_1 then G_2 is a $(1 \pm \epsilon_1)(1 \pm \epsilon_2)$ sparsifier of G.

3.3 Graphs as resistive electrical networks

We can consider our graph G as an electrical network of nodes (vertices) and wires (edges), where edge e has resistivity of w_e^{-1} Ohms.

In this context it is very useful to give another definition of the Laplacian L_G, in terms of its incidence matrix B_G. To define B_G, fix an arbitrary orientation for each edge in G. For a vertex i let χ_i be its $(n \times 1)$ characteristic vector, with a 1 at the ith entry and 0's everywhere else. Let $e = (i, j)$ be an edge and define $b_e = \chi_i - \chi_j$. Then B_G is the $m \times n$ matrix whose eth row is the vector b_e. Let W_G be the $m \times m$ diagonal matrix whose eth diagonal entry is w_e. With these definitions, it is easy to verify that

$$L_G = B_G^T W_G B_G = \sum_{e \in G} w_e b_e b_e^T.$$

For notational convenience, we will drop the subscripts on L_G, B_G, and W_G when the graph we are dealing with is clear from context.

Going back to the electrical analogy, the *effective resistance* between vertices i and j, denoted by $R^G(i, j)$ or $R^G(e)$ when (i, j) is an edge e, is the voltage difference that has to be applied between i and j in order to drive one unit of external current between the two vertices. Algebraically it is given by

$$R^G(i, j) = (\chi_i - \chi_j)^T L_G^+ (\chi_i - \chi_j) \tag{3.3}$$

The above equation allows us to apply (3.1) and see that

$$G \preceq H \preceq \kappa G \Rightarrow (1/\kappa) R^G(e) \leq R^H(e) \leq R^G(e). \tag{3.4}$$

The definition of the effective resistance for (i, j) in (3.3) shows directly that it can be computed by solving the system $L_G x = (\chi_i - \chi_j)$. In light of this, (3.4) will be of *central importance* in our proofs. Informally, it states that if H is a κ-approximation of G, then the effective resistance of any edge in G can be approximated by the effective resistance of the same edge in H, which can be done by solving the system $L_H x = (\chi_i - \chi_j)$. This will allow us to construct special approximations H for which solving with L_H is easier than with L_G.

3.4 Low-stretch subgraphs, spine-heavy graphs and SDD solvers

Let S be a graph on the same vertex set with a graph G. Let $e = (i, j)$ be an edge of G. If p is a path $e_1, e_2, \ldots, e_\nu$ between i and j in S we say that the *stretch of e over p* is $\text{stretch}_p(e) := w_e \sum_{i=1}^{\nu} w_{e_i}^{-1}$, i.e. the weight of e multiplied by the sum of inverse weights of tree edges on the path from i to j. If $\mathcal{P}(e)$ is the set of all paths between i and j in S we define

$$\text{stretch}_S(e) = \min_{p \in \mathcal{P}(e)} \text{stretch}_p(e).$$

We will use the term *stretch of e over S* for $\text{stretch}_S(e)$ The definition is simpler when T is a tree. In this case there is a unique path between the endpoints of e. We denote by $\text{stretch}_S(G)$ the sum of stretches in S of all edges of G, i.e.

$$\text{stretch}_S(G) = \sum_{e \in G} \text{stretch}_S(e).$$

It is known that every graph G has a spanning tree T with $\text{stretch}_T(G) = O(m \log n \log\log n)$, known as a **low-stretch tree**. The tree can computed in $O(m \log n \log\log n)$ time [1]. Because these guarantees are still open to improvement we will state our results with respect to two parameters: We will denote by $\mathcal{T}_m$ the time required for computing a low-stretch tree on a graph with m edges, and by s_f the factor in excess of $O(m \log n)$ in $\text{stretch}_T(G)$ provided by the $O(\mathcal{T}_m)$ time algorithm. That is, as noted above, the best current guarantees are $s_f = O(\log \log n)$ and $\mathcal{T}_m = O(m \log n \log \log n)$.

We call a graph **spine-heavy** if it has a spanning tree with $\text{stretch}_T(G) = O(m/\log n)$. Given a graph G we can compute a spine-heavy graph H that $O(s_f \log^2 n)$-approximates it by computing a low-stretch tree and then scaling up the weights of tree edges in G by the $O(s_f \log^2 n)$ factor. This is summarized in the following lemma.

Lemma 3.3 *For every graph G with n vertices there is a spine-heavy graph H that $O(s_f \log^2 n)$-approximates G. The graph H can be constructed in time dominated by the computation of a low-stretch tree for G.*

Finally we state a lemma that summarizes the recent work on fast SDD solvers [11].

Lemma 3.4 *Let A be an SDD matrix. There is a symmetric operator $\tilde{A}_\delta$ such that*

$$(1-\delta)A \preceq \tilde{A}_\delta \preceq (1+\delta)A$$

and that for any vector b, the vector $\tilde{A}_\delta^+ b$ can be evaluated in $O(\mathcal{T}_m + s_f m \log n \log(1/\delta))$ time. Moreover, if A is the Laplacian of a spine-heavy graph and its low-stretch tree is given, then $\tilde{A}_\delta^+ b$ can be evaluated in $O(m \log(1/\delta))$ time.

3.5 Sampling for sparsification

In a remarkable work, Spielman and Srivastava [12] analyzed a spectral sparsification algorithm based on a simple sampling procedure. The procedure will be central in our algorithms and we review it here. It takes as input a weighted graph G and frequencies p'_e for each edge e. These frequencies are normalized to probabilities p_e summing to 1. It then picks in q rounds exactly q *samples* which are weighted copies of the edges. The probability that given edge e is picked in a given round is p_e. The weight of the corresponding sample is set so that the expected weight of the edge e after sampling is equal to its actual weight in the input graph. The details are given in the following pseudocode.

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

1: $t := \sum_e p'_e$
2: $\hat{t} := \sum_{e \in E - \hat{E}} p'_e$
3: $q := C_s t \log t/\epsilon^2$   (* $C_S$ is an explicitly known constant *)
4: $p_e := p'_e/t$
5: $G' := (V, \mathcal{L}, w')$ with $\mathcal{L} = \emptyset$
6: for $q$ times do
7:     Sample one $e \in E$ with probability of picking $e$ being $p_e$
8:     Add sample of $e$, $l$ to $\mathcal{L}_e$ with weight $w'_l = w_e/(p_e q)$
9: end for
10: For all $l \in \mathcal{L}$, let $w'_l := w'_l/q$
11: return $G'$
```

Spielman and Srivastava analyzed the case when $p'_e = w_e R^G(e)$, where $R^G(e)$ is the effective resistance of e in G. The following generalization characterizes the quality of G' as a spectral sparsifier for G. It is shown in [8] and it was originally proved with a weaker success guarantee in [10].

Theorem 3.5 ***(Oversampling)*** *Let $G = (V, E, w)$ be a graph. Assuming that $p'_e \geq w_e R_G(e)$ for each edge $e \in E$ the graph $G' = \textsc{Sample}(G, p')$ is a $(1 \pm \epsilon)$ sparsifier of G with probability at least $1 - 1/n^2$.*

4 The general case: An $\tilde{O}(m \log^2 n)$ time algorithm

As we discussed above, Spielman and Srivastava [12] use the Sample algorithm with $p'_e = w_e R^G(e)$. For the efficient implementation of their algorithm they first obtain a different expression for the effective resistance, via a simple algebraic manipulation:

$$\begin{aligned} R^G(i,j) &= (\chi_i - \chi_j)^T L^+ (\chi_i - \chi_j) \\ &= (\chi_i - \chi_j)^T L^+ L L^+ (\chi_i - \chi_j) \\ &= (\chi_i - \chi_j)^T L^+ B^T W^{1/2} W^{1/2} B L^+ (\chi_i - \chi_j) \\ &= \|W^{1/2} B L^+ (\chi_i - \chi_j)\|^2 \end{aligned}$$

The advantage of this definition is that it expresses the effective resistance as the squared Euclidean distance of two points, given by the ith and jth column of the matrix $W^{1/2}BL^+$. This new expression still involves the solution of a linear system with L. The natural idea is to replace L with an approximation $\tilde{L}_\delta$ satisfying the properties described in Lemma 3.4. So instead of $R^G(i,j)$ we compute the quantities $\hat{R}^G(i,j) = \|W^{1/2} B \tilde{L}^+_\delta (\chi_i - \chi_j)\|^2$.

Of course, there are still m systems to be solved. To work around this hurdle, Spielman and Srivastava observe that projecting the vectors to an $O(\log n)$-dimensional space preserves the Euclidean distances within a factor of $1 \pm \epsilon/8$, by the Johnson- Lindenstrauss theorem. Algebraically this amounts to computing the quantities $\|QW^{1/2} B \tilde{L}^+_\delta (\chi_i - \chi_j)\|^2$, where Q is a properly defined random matrix of dimension $k \times m$ for $k = O(\log n)$. The authors invoke the result of Achlioptas [2], which states that one can use a matrix Q each of whose entries is randomly chosen in $\{\pm 1/\sqrt{k}\}$.

The construction of the sparsifiers can can thus be broken up into three steps.

1. Compute $QW^{1/2}B$. This takes time $O(km)$, since B has only two non-zero entries per row.
2. Apply the linear operator $\tilde{L}^+_\delta$ to the k columns of the matrix $(QW^{1/2}B)^T$, using Lemma 3.4. This gives the matrix $Z = QW^{1/2}B\tilde{L}^+_\delta$.
3. Compute all the (approximate) effective resistances (time $O(km)$) via the square norm of the differences between columns of the matrix Z. Then sample the edges.

4.1 The $\tilde{O}(m \log^2 n)$ time algorithm

Spielman and Srivastava prove that the approximations $\hat{R}^G(i,j)$ can be used to obtain the sparsifier if they satisfy

$$(1 - \epsilon/4) R^G(i,j) \leq \hat{R}^G(i,j) \leq (1 + \epsilon/4) R^G(i,j).$$

Then they show that this can be satisfied if δ, the accuracy guarantee of the linear system solver, is taken to be an *inverse polynomial* in n. Thus their algorithm is dominated by the second step (the applications of $\tilde{L}^+_\delta$) and takes time $O(\mathcal{T}_m + s_f m \log^3 n \log(1/\epsilon))$.

The following lemma shows that in fact it is enough to take δ to be a constant. Furthermore, our proof significantly simplifies the corresponding analysis of [12].

Lemma 4.1 *For a given ϵ, if $\tilde{L}$ satisfies $(1-\delta)L \preceq \tilde{L} \preceq (1+\delta)L$ where $\delta = \epsilon/8$, then the approximate effective resistance values $\hat{R}^G(u,v) = \|W^{1/2} B \tilde{L}^+ (\chi_u - \chi_v)\|^2$ satisfy:*

$$(1 - \epsilon) R^G(u,v) \leq \hat{R}^G(u,v) \leq (1 + \epsilon) R^G(u,v).$$

Proof. We only show the first half of the inequality, as the other half follows similarly. Since L and $\tilde{L}$ have the same null space, by (3.1) the given condition is equivalent to:

$$\frac{1}{1+\delta}L^+ \preceq \tilde{L}^+ \preceq \frac{1}{1-\delta}L.$$

Since $\frac{1}{1+\delta}L^+ \preceq \tilde{L}^+$, we have

$$\begin{aligned} R^G(u,v) &= (\chi_u - \chi_v)^T L^+ (\chi_u - \chi_v) \\ &\leq (1+\delta)(\chi_u - \chi_v)^T \tilde{L}^+ (\chi_u - \chi_v) \\ &= (1+\delta)(\chi_u - \chi_v)^T \tilde{L}^+ \tilde{L} \tilde{L}^+ (\chi_u - \chi_v). \end{aligned}$$

Applying the fact that $\tilde{L} \preceq (1+\delta)L$ to the vector $\tilde{L}^+(\chi_u - \chi_v)$ in turn gives:

$$\begin{aligned} R^G(u,v) &\leq (1+\delta)^2 (\chi_u - \chi_v)^T \tilde{L}^+ L \tilde{L}^+ (\chi_u - \chi_v) \\ &= (1+\delta)^2 \|W^{1/2} B \tilde{L}^+ (\chi_u - \chi_v)\|^2 = (1+\delta)^2 \hat{R}^G(u,v) \end{aligned}$$

The rest of the proof follows from $\frac{1}{(1+\delta)^2} \leq 1 - \epsilon/4$ by choice of δ. □

This proves our first theorem.

Theorem 4.2 *There is a $1 \pm \epsilon$ sparsification algorithm that runs in $O(\mathcal{T}_m + s_f m \log^2 n \log(1/\epsilon))$ time.*

5 The $\tilde{O}(m \log n)$ time algorithm

Informally, the oversampling Theorem 3.5 states that if we use estimates to the effective resistances, rather than the true values, the Spielman-Srivastava scheme still works; but in order to produce the sparsifier we have to compensate by taking more samples. We exploit this in our second Theorem.

Theorem 5.1 *There is a $(1 \pm \epsilon)$-sparsification algorithm that runs in $O(\mathcal{T}_m + m \log n \log(1/\epsilon))$ time and returns a sparsifier with $O(s_f n \log^3 n/\epsilon^2)$ edges. As a result, we can compute an $(1 \pm \epsilon)$-sparsifier with $O(n \log n/\epsilon^2)$ edges in $O(\mathcal{T}_m + m \log n \log(1/\epsilon) + s_f^2 n \log^5 n)$ time.*

Proof. Given the input graph G we construct a spine-heavy graph H that $O(s_f \log^2 n)$-approximates G. The construction can be done in $O(\mathcal{T}_m)$ time, by Lemma 3.3. We then run the Spielman-Srivastava scheme (Section 4) on H to approximate the effective resistances $R^H(i,j)$ within a factor of $1 \pm \epsilon$. Step 2 of the Spielman-Srivastava scheme runs in $O(m \log n \log(1/\epsilon))$ time on H, by Lemma 3.4. We adjust the approximate effective resistances in H down by a factor of $1+\epsilon$ to accommodate for the upper side of the error in Lemma 4.1. Then, by (3.2) the calculated approximate effective resistances satisfy

$$\frac{1}{O(s_f \log^2 n)} R^G(i,j) \leq \hat{R}^H(i,j) \leq R^G(i,j).$$

So Theorem 3.5 applies if we take $p'_e = O(s_f \log^2 n) w_e \hat{R}^H(i,j)$. We have

$$\sum_e p'_e = O(s_f \log^2 n) \sum_e w_e \hat{R}^H(i,j) \leq O(s_f \log^2 n) \sum_e w_e R^G(i,j) = O(s_f n \log^2 n).$$

The last equality follows from the fact that $\sum_e w_e R^G(i,j) = n-1$ for any graph G (e.g. see [12]). Hence the total number of samples we need to take in order to produce an $(1 \pm \epsilon)$-sparsifier is $O(s_f n \log^3 n/\epsilon^2)$. The second sparsifier is computed by re-sparsifying with the general case algorithm (and appropriate settings for ϵ). □

6 The $\tilde{O}(m)$ and $O(m)$ time algorithms

6.1 The base approximate sparsifier

We will show that for every graph G we can in $O(m)$ time compute an $O(s_f \log^4 n)$-approximation with $O(n \log n)$ edges, i.e. an *approximate sparsifier* H of G. To find the approximate sparsifier we will first identify an $O(s_f \log^4 n)$-approximation H of G, which in turn can be sparsified in $O(m)$ time.

Consider for the moment the following Lemma.

Lemma 6.1 *Let G be a graph with n vertices and $m > 2n$ edges. Let T be a low-stretch tree of G and let $H = G + O(s_f^2 \log^4 n)T$. Given T, a 4-approximation of H with $O(n \log n)$ edges can be computed in $O(m + s_f n \log^2 n)$ time.*

Proof. In [10] it was shown that applying Sample on H, with p'_e taken to be the stretch of e over the tree T, produces a graph I with $n + m/(s_f \log^2 n)$ edges such that I is a 2-approximation of H. With the general case algorithm we can – in time $O(m + s_f n \log^2 n)$ – compute a 2-approximation $\tilde{I}$ of I with $O(n \log n)$ edges. By transitivity $\tilde{I}$ is a 4-approximation of H. □

The graph H in the above Lemma comes close to our requirements. The main problem lies in the speed of the low-stretch tree computation: the fastest known algorithm for computing a low-stretch tree requires $O(s_f m \log n)$ time, and we would like to compute H faster. To work around this hurdle we will replace the low-stretch with a low-stretch subgraph S of G which can be computed in $O(m)$ time. For this, we will use *spanners*. A t-spanner of a graph G is a subgraph S of G that preserves distances within a factor of t. It follows that if G is *unweighted* and S is a $\log n$-spanner, then for every edge e of G there is a path of length $O(\log n)$ in S. That is for each e, we have

$$\text{stretch}_S(e) = O(\log n).$$

We will use the following Lemma from [7].

Lemma 6.2 *Given an unweighted graph G, a $O(\log n)$-spanner S with with $O(n)$ edges can be computed in $O(m)$ time.*

We then get the following.

Lemma 6.3 *Let G be an unweighted graph graph with n vertices. We can compute a $O(s_f \log^4 n)$-approximation H of G and a 4-approximation of H with $O(n \log n)$ edges in in $O(m + s_f n \log^3 n)$ time. The graph H is of the form $G + O(s_f \log^4 n)S$ where S is a spanner of G.*

Proof. Let S be the spanner of G provided by Lemma 6.2. The proof mirrors that of Lemma 6.1, with some minor changes. First notice that the graph S gets scaled up by a factor s_f smaller than the tree in Lemma 6.1. This reflects the fact that the average stretch over S is s_f times smaller than over T. To compute the required sparsifier, we first find a graph I with $O(n \log n + m/(s_f \log^2 n))$ edges which is a 2-approximation of H. The only difference is that we now compute I by applying Sample with uniform frequencies p'_e, because the stretch of all edges is bounded above by $O(\log n)$. Then using our general case algorithm we can –in time $O(m + s_f n \log^3 n)$– compute a 2-approximation $\tilde{I}$ of I with $O(n \log n)$ edges. By transitivity $\tilde{I}$ is a 4-approximation of H. □

6.2 Leveraging an approximate sparsifier

The purpose of this section is to show that provided an approximate sparsifier for a graph G we can produce efficiently a sparsifier for G. So far we have been using only low-stretch *subgraphs* of G to get approximations to the effective resistances in G. A key to our fastest algorithm is the realization that we can find low-stretch trees that are not necessarily subtrees of the given graph G; the total stretch will actually be only a near-linear function of n. This is based on the following Lemma.

Lemma 6.4 *Let $H' \preceq H$. Then for any tree T*

$$\mathrm{stretch}_T(H') \leq \mathrm{stretch}_T(H).$$

Proof. Let A^+ denote the Moore-Penrose pseudoinverse of a matrix A and $\lambda_i(A)$ denote the i^{th} largest eigenvalue of A. By Spielman and Woo [15] we know that for any graph G

$$\mathrm{stretch}_T(G) = \mathrm{trace}(L_G L_T^+) = \sum_i \lambda_i(L_G L_T^+)$$

Because L_G and L_T have the same null space (the constant vector), we can write

$$\lambda_i(L_G L_T^+) = \lambda_i(L_T^{+/2} L_G L_T^{+/2}).$$

Notice now that we have

$$H' \preceq H \Rightarrow (L_T^{+/2} L_{H'} L_T^{+/2}) \preceq (L_T^{+/2} L_H L_T^{+/2}).$$

This follows easily by definition. It is also easy to prove (see for example [9], Ch. 6.1) that

$$A \preceq B \Rightarrow \lambda_i(A) \leq \lambda_i(B).$$

Hence

$$\begin{aligned}
\lambda_i(L_T^{+/2} L_{H'} L_T^{+/2}) &\leq \lambda_i(L_T^{+/2} L_H L_T^{+/2}) \Rightarrow \\
\mathrm{trace}(L_T^{+/2} L_{H'} L_T^{+/2}) &\leq \mathrm{trace}(L_T^{+/2} L_H L_T^{+/2}) \Rightarrow \\
\mathrm{stretch}_T(H') &\leq \mathrm{stretch}_T(H).
\end{aligned}$$

□

We are now ready to prove the main Lemma in this subsection. To avoid confusion we will use m_H to denote the number of edges of a graph H.

Lemma 6.5 ***(Leveraging)*** *Let H be a κ-approximation of H'. Suppose we are given $\tilde{H}'$, a 4-approximation of H' with $O(n \log n)$ edges. Then we can construct a $(1 \pm \epsilon)$-approximation of H with $O(s_f \kappa n \log^3 n)$ edges in $O(m_H + \mathcal{T}_{n \log n})$ time. We can also construct a $(1 \pm \epsilon)$-sparsifier of H with $O(n \log n)$ edges in $O(m_H + \mathcal{T}_{n \log n} + s_f^2 \kappa n \log^5 n)$ time.*

Proof. We compute a low-stretch spanning tree T of $\tilde{H}'$ in $O(\mathcal{T}_{n \log n})$ time. Because $\tilde{H}'$ has $O(n \log n)$ edges we have

$$\mathrm{stretch}_T(\tilde{H}') = O(s_f n \log^2 n).$$

We then compute in $O(m_H)$ time the effective resistance $R^T(e)$ of each edge e of H over T. This can be done in $O(m_H)$ time using off-line LCA algorithms by Gabow and Tarjan[16, 6]. Since $\tilde{H}' \preceq 4H'$ and

$H' \preceq H$ we can apply (3.2) twice and get

$$R^H(e) \le R^{H'}(e) \le 4R^{\tilde{H}'}(e).$$

By Rayleigh's monotonicity theorem (e.g. see [10]) we get that:

$$R^{\tilde{H}'}(e) \le R^T(e).$$

Hence setting $p'_e = 4w_e R^T(e)$ in Theorem 3.5 allows us to sparsify H. To get a $(1 \pm \epsilon)$-approximation of H, the number of samples we need to take is $O(q \log q/\epsilon^2)$ where $q = \sum_e 4w_e R^T(e) = \text{stretch}_T(H)$. Since $H' \preceq \tilde{H}'$ and $H \preceq \kappa H'$, we apply Lemma 6.4 twice and we get that

$$\text{stretch}_T(H) \le \kappa \cdot \text{stretch}_T(H') \le \kappa \cdot \text{stretch}_T(\tilde{H}') = O(s_f \kappa n \log^2 n).$$

This proves the first claim. The second claim follows via picking the appropriate ϵ and re-sparsifying the first sparsifier with the general case sparsification algorithm. □

6.3 Sparsifiers for unweighted graphs

We are now ready to prove our main claims.

Theorem 6.6 *Given an unweighted graph G we can compute an $(1 \pm \epsilon)$-spectral sparsifier of G with $O(n \log n/\epsilon^2)$ edges in $O((m + \mathcal{T}_{n \log n} + s_f^2 n \log^5 n) \log\log n)$ time.*

Proof. Let $H = G + O(s_f \log^3 n)S$ be the graph provided by Lemma 6.3. We construct a sequence of graphs,

$$H = H_0, H_1, \dots, H_t = G$$

where $H_i = G + O(s_f \log^4 n)S/2^i$, for some appropriate $t = O(\log\log n)$. Notice that all graphs H_i have m edges, so this takes $O(m \log\log n)$ time. For $i = 0, \dots, t-1$, let $\tilde{H}_i$ denote a 4-approximation of H_i with $O(n \log n)$ edges. By Lemma 6.3 we can compute $\tilde{H}_0$ in $O(m)$ time. Provided now that we have $\tilde{H}_j$ we can apply Lemma 6.5 (with $\kappa = 2$, $\epsilon = 1/2$ and a proper scaling of the $(1 \pm \epsilon)$-sparsifier) to get a 4-approximation $\tilde{H}_{j+1}$ in $O(m + s_f^2 n \log^5 n)$ time. From H_{t-1} we produce a $(1 \pm \epsilon)$-sparsifier of $H_t = G$ again by Lemma 6.5, using the desired value for ϵ. Because we apply the algorithm of Lemma 6.5 $O(\log\log n)$ times, we get the claimed running time. □

Theorem 6.7 *Given an unweighted graph G we can compute an $(1 \pm \epsilon)$-spectral sparsifier of G with $O(n \log n/\epsilon^2)$ edges in $O(m + \mathcal{T}_{n \log n} + s_f^3 n \log^8 n)$ time.*

Proof. Let $H = G + O(s_f \log^3 n)S$ be constructed as in Lemma 6.3. Notice the smaller scaling factor relative to the Lemma. Similarly to Lemma 6.3 we compute a graph I which 2-approximates H and has $O(n \log n + m/(s_f \log n))$ edges; this is slightly more dense than the graph I in the Lemma, due to the smaller scale-up factor. We then use our second algorithm to sparsify I and get a 4-approximation H' of H with $O(n \log n)$ edges in $O(m + s_f^2 n \log^6 n)$ time. Because H is an $O(s_f \log^3 n)$-approximation of G we can 'leverage' its sparsifier H' to compute a $(1 \pm \epsilon)$-sparsifier of G via Lemma 6.5 (with $\kappa = O(s_f \log^3 n)$) in $O(m + \mathcal{T}_{n \log n} + s_f^3 n \log^8 n)$ time. □

6.4 The weighted case: sparsification via decompositions

We first prove the following Lemma.

Lemma 6.8 *For any graph G with polynomially bounded weights we can compute in $O(m \log\log n)$ time a low-stretch subgraph S with $O(n \log n)$ edges such that for all edges e of G, we have*

$$\text{stretch}_S(e) = O(\log n).$$

Proof. Recall our assumption that the graph weights are polynomially bounded. In $O(m \log\log n)$ time we can decompose G into $O(\log n)$ edge-disjoint graphs G_j. Suppose G_j consists of m_j edges with weights in $[2^j, 2^{j+1})$, for $j = 0 \ldots O(\log n)$. We then round the weights in G_j to the nearest power of 2 and find a spanner S_j using Lemma 6.2. Finally we let $S = \sum_j S_j$. This computation can be performed in $\sum_j O(m_j) = O(m)$ time. Now let e be an edge in G_j. We have

$$\text{stretch}_S(e) \leq \text{stretch}_{S_j}(e) = O(\log n).$$

□

This Lemma is enough to generalize Theorem 6.6 to weighted graphs with polynomially bounded weights. All we need to do is to replace the spanner S in the proof with the low-stretch subgraph provided by Lemma 6.8.

7 Final Remarks

We remark that the fastest sparsification algorithms of this paper rely crucially on graph decompositions. However it seems natural to conjecture that decompositions are not necessary, and that the same upper bound can be obtained via straightforward sampling scheme. We believe that this is an interesting question that would potentially lead to a deeper understanding of low-stretch subgraph computations.

On the other hand the original algorithm of Spielman and Teng remains the only known combinatorial sparsification algorithm that does not rely on solving systems. Designing a spectral sparsification algorithm that does not depend on a linear system solver and that outputs a very sparse graph with $O(n \log n)$ or $O(n \log^2 n)$ edges is a challenging open problem. Since achieving this may prove to be difficult, an alternate approach could be algorithms that compute very sparse κ-approximations for small values of κ. Such algorithms could play a significant role in the development of more practical SDD solvers.

8 Acknowledgments

We would like to thank Jonathan Kelner and Gary Miller for useful discussions and the anonymous referees for carefully reading an earlier version of this paper. Ioannis Koutis is supported by NSF CAREER award CCF-1149048. Alex Levin is supported by a National Science Foundation graduate fellowship. Richard Peng was at Microsoft Research New England for part of this work and is supported by a Microsoft Research Fellowship. He is also partially supported by the National Science Foundation under grant number CCF-1018463.

References

[1] Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. In Howard J. Karloff and Toniann Pitassi, editors, *STOC*, pages 395–406. ACM, 2012. 1, 3.4

[2] Dimitris Achlioptas. Database-friendly random projections. In *PODS '01: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 274–281, 2001. 4

[3] András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $O(n^2)$ time. In *STOC '96: Proceedings of the Twenty-Eighth Annual ACM symposium on Theory of Computing*, pages 47–55, 1996. 1

[4] Fan Chung. Random walks and local cuts in graphs. *Linear Algebra and its applications*, 423(1):22–32, 2007. 1

[5] Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *STOC '11: Proceedings of the 43rd ACM Symposium on Theory of Computing*, pages 71–80, 2011. 1

[6] Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 246–251, New York, NY, USA, 1983. ACM. 6.2

[7] Eran Halperin and Uri Zwick. Linear time deterministic algorithm for computing spanners for unweighted graphs. 1996. manuscript. 6.1

[8] Jonathan A. Kelner and Alex Levin. Spectral sparsification in the semi-streaming setting. *Theory of Computing Systems*. To appear. 3.5

[9] Ioannis Koutis. *Combinatorial and algebraic tools for optimal multilevel algorithms*. PhD thesis, Carnegie Mellon University, Pittsburgh, May 2007. CMU CS Tech Report CMU-CS-07-131. 6.2

[10] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD systems. In *FOCS '10: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, 2010. 2.2, 2.3, 3.5, 6.1, 6.2

[11] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-$m \log n$ solver for SDD linear systems. In *FOCS '11: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, 2011. 1, 2.2, 2.4, 3.4

[12] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *STOC '08: Proceedings of the 40th Annual ACM symposium on Theory of Computing*, pages 563–568, 2008. 1, 2.1, 3.5, 4, 4.1, 5

[13] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC '04: Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 81–90, 2004. 1

[14] 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

[15] Daniel A. Spielman and Jaeoh Woo. A note on preconditioning by low-stretch spanning trees. *CoRR*, abs/0903.2816, 2009. 6.2

[16] Robert Endre Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715, 1979. 6.2