Improved Spectral Sparsification and Numerical Algorithms for SDD Matrices

Ioannis Koutis[1], Alex Levin[2], and Richard Peng[3]

1 **Computer Science Department, University of Puerto Rico, Río Piedras**
ioannis.koutis@upr.edu
2 **Department of Mathematics, Massachusetts Institute of Technology**
levin@mit.edu
3 **School of Computer Science, Carnegie Mellon University**
yangp@cs.cmu.edu

Abstract

We present three spectral sparsification algorithms that, on input a graph G with n vertices and m edges, return a graph H with n vertices and $O(n \log n/\epsilon^2)$ edges that provides a strong approximation of G. Namely, for all vectors x and any $\epsilon > 0$, we have

$$(1-\epsilon)x^T L_G x \leq x^T L_H x \leq (1+\epsilon)x^T L_G x,$$

where L_G and L_H are the Laplacians of the two graphs. The first algorithm is a simple modification of the fastest known algorithm and runs in $\tilde{O}(m \log^2 n)$ time, an $O(\log n)$ factor faster than before. The second algorithm runs in $\tilde{O}(m \log n)$ time and generates a sparsifier with $\tilde{O}(n \log^3 n)$ edges. The third algorithm applies to graphs where $m > n \log^5 n$ and runs in $\tilde{O}(m \log_{m/n \log^5 n} n)$ time. In the range where $m > n^{1+r}$ for some constant r this becomes $\tilde{O}(m)$. The improved sparsification algorithms are employed to accelerate linear system solvers and algorithms for computing fundamental eigenvectors of dense SDD matrices.

1998 ACM Subject Classification G.2.2 [Discrete Mathematics]: Graph Theory—graph algorithms; G.3 [Probability and Statistics]: Probabilistic algorithms (including Monte Carlo)

Keywords and phrases Spectral sparsification, linear system solving

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.266

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 [10] 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.[1]

Given that even the existence of cut-preserving sparsifiers is not immediately clear, the result of Benczúr and Karger was indeed very surprising; they 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

[1] A matrix A is SDD if for all i, $A_{ii} \geq \sum_{j \neq i} |A_{ij}|$.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 266–277
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

graph can be computed by a randomized algorithm that runs in $O(m \log^3 n)$ time, 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 that of Benczúr and Karger's cut-preserving sparsifiers. However, recent progress that we review below allows for the construction of spectral sparsifiers with $O(n \log n/\epsilon^2)$ edges in $\tilde{O}(m \log^3 n \log(1/\epsilon))$ time.[2]

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 provides a strong incentive for speeding up the construction of sparsifiers even further.

This problem was recently 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 $O(m + n \log n)$ 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 $O(m + n \log^4 n/\epsilon^2)$ 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 to the result of Fung *et al.* [5] possible for spectral sparsification? We answer the question in the affirmative. We first show that a slight modification of the known algorithm can improve the run time to $\tilde{O}(m \log^2 n \log(1/\epsilon))$. This nearly matches the most general result of [5]. We present two additional sparsification algorithms. The first 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 $\tilde{O}(m/\log^2 n)$ edges in $O(m \log_{m/(n \log^5 n)} n)$ time. As in the cut-preserving case, transitivity then allows us to re-sparsify these sparsifiers with the fastest general-case algorithm in order to get a sparsifier with $O(n \log n/\epsilon^2)$ edges.

Applications in numerical algorithms. Sparsification can be used to accelerate the computation of an approximate Fiedler eigenvector of a (normalized) graph Laplacian [11], and more generally of the first non-trivial eigenvector of an SDD matrix L. The approximate eigenvector is a normalized vector x such that $x^T Lx$ is within $1 \pm \epsilon$ of the eigenvalue λ_2. More concretely, by applying the simple inverse power method analyzed in [11] to the sparsifier with $O(n \log^3 n/\epsilon^2)$ edges one can obtain a $1 \pm \epsilon$ approximation of its eigenvector in $O(n \log^5 n \log(1/\epsilon)/\epsilon^2)$ time. However, sparsification preserves the eigenvalues within $1 \pm \epsilon$ and so the computed approximation is a $1 \pm 3\epsilon$ approximation for the dense graph. This implies overall that the Fiedler eigenvector of a graph with $m > n \log^3 n$ can be computed in $O(m \log n + n \log^5 n \log(1/\epsilon)/\epsilon^2)$ time. The previously fastest known algorithm runs in time $O(m \log^2 n \log(1/\epsilon))$. 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] We use the $\tilde{O}()$ notation to hide $\log \log n$ factors.

In addition, the $1 \pm \epsilon$ sparsifiers we obtain can be applied in a standard way as preconditioners for SDD linear systems, giving us a faster solver for these systems. In particular, for the case when m, the number of non-zero entries in the matrix of the system, is greater than n^{1+r} for some small constant r, we can show that our solver approximates a solution with relative error δ in time $\tilde{O}(m \log(1/\delta))$. The previously best known algorithm [8] runs in time $\tilde{O}(m \log n \log(1/\delta))$.

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 [9]. 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 by solving $O(\log n)$ SDD systems; their approach involves characterizing the effective resistances as the squared lengths of vectors and then applying the Johnson-Lindenstrauss (JL) theorem [2]. 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 $\tilde{O}(m \log^2 n)$ 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 [7, 8], which runs in time $\tilde{O}(m \log n \log(1/\delta))$. 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 a fixed precision from the SDD solver is actually sufficient for sparsification. This decreases the running time to $\tilde{O}(m \log^2 n/\epsilon^2)$. This improvement is included in all our subsequent algorithms.

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

In order to speed up the algorithm further, we need to break the central bottleneck, which comes from having to solve $O(\log n)$ linear systems each of which takes $\tilde{O}(m \log n)$ time. We improve the running time of this step by allowing for cruder, but more easily-computable, approximations of the effective resistances. It was shown in [7] that if we estimate the effective resistances, the Spielman-Srivastava scheme still goes through, but we may need to sample more edges to compensate for the loss of accuracy.

[3] Laplacian matrices are SDD.

In particular, we estimate the effective resistances by using a *spine-heavy* approximation to G. This is a graph that has an extremely good low stretch spanning tree. In [8] it was shown that linear equations in Laplacians of spine-heavy graphs can be solved in $\tilde{O}(m \log(1/\delta))$ time. Further any graph can be easily transformed into a spine-heavy approximation while distorting the effective resistances by at most an $\tilde{O}(\log^2 n)$ factor. Using this spine-heavy approximation in order to quickly estimate effective resistances, and then sampling with respect to these estimates, allows us to get a sparsifier with $O(n \log^3 n/\epsilon^2)$ edges in $\tilde{O}(m \log n)$ time. The details are given in Section 5.

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

Several more obstacles needs to be circumvented for an even faster algorithm. Even assuming a computationally free SDD solver, estimating the effective resistances via the Johnson-Lindenstrauss projection requires operating on m vectors of dimension $O(\log n)$, which is too expensive. This forces us to try to decrease (hopefully down to a constant) the dimension of the projections. Of course this introduces higher distortions in the estimates for the effective resistances, but as we noted above the algorithm can compensate by taking more samples. The second key to our result comes into play here: transitivity. We observe that it is enough to produce a sparsifier with $m' = O(m/\log^2 n)$ edges since we can then run our slightly slower algorithm in time $\tilde{O}(m' \log^2 n) = \tilde{O}(m)$ and get the final sparsifier. This trick allows us to reduce the dimension of the JL projection to a constant, for large enough m. The details are given in Section 6.

However to get these severely distorted estimates for the effective resistances, it is not enough to just take our $O(m \log^2 n)$ algorithm and replace the JL projection by a constant-dimensional one. The remaining bottleneck is the running time of the solver; its construction requires at the minimum the computation of a low-stretch tree which takes $\tilde{O}(m \log n)$ time [1]. The solver steps after the construction of the low-stretch tree take $\tilde{O}(m)$ time on a spine-heavy graph. This implies that we would be able to sparsify in $\tilde{O}(m)$ time if the computation of the low-stretch tree were not an issue.

To solve this problem, we show that every graph can be decomposed into graphs of diameter $O(\log n)$ with relatively few edges between the pieces. Spanning trees with $O(\log n)$ average stretch can be easily computed for each of these pieces, and thus we sparsify them separately and then put the results together. The details are given in Section 7.

3 Background on spectral graph theory

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. 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 $x^T Ax \leq x^T Bx$ 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 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

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

▶ **Definition 2.** Given a graph G, we say that a (sparser) graph H is a $1 \pm \epsilon$ **spectral sparsifier** of G if

$$(1-\epsilon)G \preceq H \preceq (1+\epsilon)G. \tag{2}$$

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

$$R^G(i,j) = (\chi_i - \chi_j)^T L_G^+ (\chi_i - \chi_j) \tag{3}$$

The above equation allows us to apply (1) and see that

$$G \preceq H \preceq \kappa G \Rightarrow (1/\kappa) R^G(e) \leq R^H(e) \leq R^G(e). \tag{4}$$

The definition of the effective resistance for (i, j) in (3) shows directly that it can be computed by solving the system $L_G x = (\chi_i - \chi_j)$. In light of this, (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 trees, spine-heavy graphs and SDD solvers

Let T be a spanning tree of G. For any edge $e = (i, j)$ of G, there is a unique path $e_1, e_2, \ldots, e_\nu$ between i and j along edges of T. We say that the *stretch of e in T* is $\text{stretch}_T(e) := w_e \sum_{i=1}^{\nu} w_{e_i}^{-1}$, i.e. the weight of e multiplied by the sum of inverse weights of tree edges on the path from i to j. We denote by $\text{stretch}_T(G)$ the sum of stretches in T of all edges of G, i.e. $\text{stretch}_T(G) = \sum_{e \in G} \text{stretch}_T(e)$.

It is known that every graph G has a spanning tree T with $\text{stretch}_T(G) = \tilde{O}(m \log n)$, known as a **low-stretch tree**. The tree can computed in time $\tilde{O}(m \log n)$ [1, 8]. 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 $\tilde{O}(\log^2 n)$-approximates it by computing a low-stretch tree and then scaling up the weights of tree edges in G by the $\tilde{O}(\log^2 n)$ factor. This is summarized in the following lemma.

▶ **Lemma 3.** *Every graph G with n vertices is $\tilde{O}(\log^2 n)$-approximated by a spine-heavy graph H. 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 [8].

▶ **Lemma 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 $\tilde{O}(m \log n \log(1/\delta))$ 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 $\tilde{O}(m \log(1/\delta))$ time.

4 The Spielman-Srivastava sampling scheme

Spielman and Srivastava [9] give the following simple algorithm for producing a $1 \pm \epsilon$ sparsifier of a graph G: For each i from 1 to $N = O(n \log n/\epsilon^2)$, we sample an edge e of G from the probability distribution $\boldsymbol{p}$ assigning e a probability p_e proportional to $q_e = w_e R^G(e)$. If we select edge e, we add it to the sparsifier with weight $w_e/(Np_e)$.

This scheme produces a $1 \pm \epsilon$ sparsifier with high probability. An analysis is given in [9], and a different perspective can be found in Srivastava's dissertation [13].

For the efficient implementation of their algorithm Spielman and Srivastava 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 $W^{1/2} B L^+$.

This new expression still involves the solution of a linear system with L. The natural idea is to replace L with an approximation $\tilde{L}$ satisfying the properties described in Lemma 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 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.

5 The first two sparsification algorithms

5.1 The $\tilde{O}(m \log^2 n)$ 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 $\tilde{O}(m \log^3 n \log(1/\epsilon))$.

The following lemma shows that in fact it is enough to take take δ to be a constant. Furthermore, our proof significantly simplifies the corresponding analysis of [9].

▶ **Lemma 5.** *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 (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:

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

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 6.** *There is a $1 \pm \epsilon$ sparsification algorithm that runs in time $\tilde{O}(m \log^2 n \log(1/\epsilon))$.*

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

In [7] it was proven 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. Specifically, for $\alpha > 1$, if the probabilities with which we sample all edges are at least $1/\alpha$ of the true values, then we have to take α times as many samples. This is formalized in the following lemma.

▶ **Lemma 7.** *Suppose that we run the Spielman-Srivastava algorithm and sample edges with probabilities proportional to $\tilde{q}_e$ such that $(1/\alpha)q_e \leq \tilde{q}_e \leq q_e$ for all edges e. Then, taking α times as many samples gets us a $1 \pm \epsilon$ sparsifier with the same high probability guarantee as the Spielman-Srivastava algorithm run with probabilities proportional to q_e.*

We are now ready to state our second theorem.

▶ **Theorem 8.** *There is a $1 \pm \epsilon$ sparsification algorithm for graphs with $m > n \log^3 n$ edges that runs in time $\tilde{O}(m \log n \log(1/\epsilon))$. The output sparsifier contains $\tilde{O}(n \log^3 n/\epsilon^2)$ edges.*

Proof. Given the input graph G we construct a spine-heavy graph H that $\tilde{O}(\log^2 n)$-approximates G. The construction can be done in time $\tilde{O}(m \log n)$, by Lemma 3. We 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 $\tilde{O}(m \log n \log(1/\epsilon))$ time on H, by Lemma 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 5. Then, by (2) the calculated approximate effective resistances satisfy

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

Finally we let $\tilde{q}_e = w_e \hat{R}^H(i,j)$ for all edges $e = (i,j)$ of G and sample the edges of G with probabilities proportional to $\tilde{q}_e$. By Lemma 7 we see that we get a $1 \pm \epsilon$ sparsifier with $\tilde{O}(n \log^3 n/\epsilon^2)$ edges. ◀

6 Effective resistances via very-low dimensional projections

With the improvement of the last section, all three steps of the Spielman-Srivastava algorithm take $\tilde{O}(m \log n)$ time; our goal now is to reduce this to $\tilde{O}(m)$. The extra logarithm in the current implementation is due to the dimension $k = O(\log n)$ of the projection matrix Q, and we address this issue here.

It is worth noting that once we have a sparsifier H with $O(\frac{m}{\log^2 n})$ edges such that

$$\left(1 - \frac{\epsilon}{2}\right) G \preceq H \preceq \left(1 + \frac{\epsilon}{2}\right) G,$$

we can afford to fully $(1 \pm \frac{\epsilon}{2})$-sparsify that H using our $\tilde{O}(m \log^2 n)$ algorithm. The sparsifier of H (with $O(n \log n/\epsilon^2)$ edges) will then be a $1 \pm \epsilon$-sparsifier for G.

Since we can take more samples, we are able to underestimate probabilities more aggressively by decreasing the dimension we project onto, and still get a good approximation to G with high probability. In order to show that we do not underestimate effective resistances by too much, we need a more detailed understanding of the relationship between the dimension k and the approximation guarantee. This is provided by the version of the Johnson-Lindenstrauss theorem stated as Lemma 7 of [6]:

▶ **Lemma 9.** *Let u be a unit vector in $\mathbb{R}^\nu$. For any given positive integers k, let $U_1, \ldots, U_k$ be random vectors chosen independently from the ν-dimensional Gaussian distribution $N^\nu(0,1)$. For $X_i = u^T U_i$, define $W = W(u) = (X_1, \ldots, X_k)$ and $L = L(u) = \|W\|^2$. Then for any $\beta > 1$:*

1. $E(L) = k$,
2. $\Pr[L \geq \beta k] < O(k)\exp(-\frac{k}{2}(\beta - (1 + \ln \beta))$,
3. $\Pr[L \leq k/\beta] < O(k)\exp(-\frac{k}{2}(\beta^{-1} - (1 - \ln \beta))$.

Following standard analysis of the Johnson-Lindenstrauss theorem, we see that this lemma essentially gives us the probability of increasing or decreasing sizes of a given vector by a certain factor when we multiply the vector by a random matrix of Gaussian entries.[4] Roughly, the third part states that for a given small constant $r \ll 1$, the probability of underestimating distances (and hence effective resistances in our application) by an n^r factor is around $O(n^{-rk/2})$. By setting k sufficiently large and applying a union bound, we obtain that with high probability all estimates are at least $\Omega(n^{-r})$ of the true quantities required by the Spielman-Srivastava algorithm.

Combining this with the fact that weight times effective resistance is upper bounded by 1, one can show by concentration of measure theorems that the normalizing factor (i.e. the weighted sum of the estimated effective resistances) stays within a constant factor of its true value with high probability. Therefore, with high probability we underestimate the edge selection probabilities by at most a factor of $O(n^r)$. The number of samples we need to take as a result is $n^{1+r}\log n$. As long as this is smaller than $m/\log^2 n$ we can sparsify in $\tilde{O}(m)$ time. This shows that as long as m is big enough relative to n, we can sparsify in linear time, as we claimed in the introduction. We formalize this argument below.

▶ **Lemma 10.** *There is an algorithm that, on input a graph G with n vertices, m edges, a low-stretch spanning tree for G with total stretch $\tilde{O}(m \log n)$, and a parameter t, generates a $1 \pm \epsilon$ sparsifier with $\tilde{O}(\frac{m}{t}\log n/\epsilon^2)$ edges in $\tilde{O}(m \log_{\frac{m}{3t\log^2 n}} n \log(1/\epsilon))$ time.*

Proof. We first construct in $O(m)$ time the spine-heavy graph G' that $\tilde{O}(\log^2 n)$-approximates G. We then apply the Spielman-Srivastava sampling scheme in order to estimate the effective resistances in G'.

Invoking Part 3 of Lemma 9 with $\beta = \frac{m}{nt\log^2 n}$ shows us that when we project onto k dimensions, the probability of underestimating by a factor of β is at most:

$$O(k)\exp\left(\frac{k}{2}(1 - \beta^{-1} - \ln\beta)\right) \leq O(k)\exp\left(\frac{k}{2}(1 - \ln\beta)\right) \leq O(k)(3/\beta)^{\frac{k}{2}}$$

where the first inequality follows from $k/2 \geq 0$ and $1 - \beta^{-1} \leq 1$. So when $(3/\beta)^{\frac{k}{2}} = n^{-d}$, taking a union bound over all $m \leq n^2$ edges gives that no edge's effective resistance is underestimated by more than a factor of β. The requirement on k imposed by this is:

$$\begin{aligned} O(k)(3/\beta)^{\frac{k}{2}} &\leq n^{-d} \\ k &\geq 2d\log_{\beta/3} n + \log_{\beta/3} k + O(1) \end{aligned}$$

Setting d to be some constant and taking the value of β as before we see that taking $k = O(\log_{\frac{m}{3nt\log^2 n}} n)$ will give us the required high probability claims.

[4] This is a minor difference from previous parts, where we use matrices entries randomly chosen in $\pm 1/\sqrt{k}$

This shows that projecting in order to estimate effective resistances and using these to estimate edge selection probabilities will give us values that are at least an $nt \log^2 n/m$ factor of the true value (for β as above). Following the proof of Lemma 5 we can see that using an approximate solver introduces a small multiplicative error. Using the fact that G' is a graph that $O(\log^2 n)$-approximates G, we see that this method produces approximates probabilities in G that are at least a factor of $\frac{nt}{m}$ of the true values.

Consider sampling with these estimated probabilities. Then, by the discussion at the beginning of Section 5.2 with $\alpha = m/(nt)$, we see that to sparsify we need to take $O(\frac{m}{t} \log n\epsilon^{-2})$ samples.

The running time of this process is dominated by amount of time it takes to do k solves in $L_{G'}$, namely $O(km \log(1/\epsilon))$ by Lemma 4. For the choice of k as before this is $\tilde{O}(m \log_{\frac{m}{3t \log^2 n}} n \log(1/\epsilon))$, as required. ◀

▶ **Theorem 11.** *Given a graph G with n vertices, m edges such that $m > n \log^5 n$, and a low-stretch spanning tree with stretch $\tilde{O}(m \log n)$, we can generate a $1 \pm \epsilon$-sparsifier H of G with $O(n \log n/\epsilon^2)$ edges in $\tilde{O}(m \log_{\frac{m}{n \log^5 n}} n \log(1/\epsilon))$ time.*

Proof. Applying Lemma 10 with $t = O(\log^3 n)$ gives a graph with $\tilde{O}(\frac{m}{\log^2 n})$ edges that is a $1 \pm \epsilon$-sparsifier. This graph can in turn be sparsified in $\tilde{O}(\frac{m}{\log^2 n} \log^2 n) = \tilde{O}(m)$ time, by Theorem 8. ◀

7 Improved sparsification via graph decompositions

Theorem 11 reveals that the computation of the low-stretch tree of the input graph is the final bottleneck on our way to getting the faster algorithms. In order to solve this problem, we no longer compute a low-stretch spanning tree for the entire graph. Instead, we decompose the graph into subgraphs for which we can trivially find low-stretch spanning trees and we sparsify each subgraph separately. The decomposition is based on the following simple fact about low diameter graphs:

▶ **Lemma 12.** *Given an unweighted graph with n vertices, m edges, and diameter $O(\log n)$, finding a breadth-first search (BFS) tree in $O(m)$ time gives low stretch spanning tree with average stretch $O(\log n)$.*

We can now apply low diameter decomposition to extend this to arbitrary undirected graphs losing an extra factor of $\log \log n$. The variant of low diameter decomposition that we use can be best described using the following lemma (see, e.g., [14, Lemma 4]).

▶ **Lemma 13.** *Given an undirected, unweighted graph with n vertices and m edges, we can partition it into pieces of $O(\log n)$ diameter so that at most $m/2$ edges are between the pieces.*

Applying this $O(\log \log n)$ times and sparsifying the edges between pieces each time gives the claim for arbitrary unweighted graphs:

▶ **Theorem 14.** *Given an undirected, unweighted graph G with n vertices and m edges such that $m > \Omega(n \log^4 n)$, we can output a sparsifier H with $\tilde{O}(n \log n/\epsilon^2)$ edges in $\tilde{O}(m \log_{\frac{m}{3n \log^4 n}} n \log(1/\epsilon))$ time.*

Proof. We create $G_1, \ldots G_l$ where $l = 4 \log \log n$ as follows. Given $G_1 \ldots G_i$, we partition $E(G) \setminus E(G_1) \ldots \setminus E(G_i)$ into low diameter pieces using Lemma 13 and let G_{i+1} be edges

with both endpoints in the same piece that's not in some G_j with $j \leq i$. Applying guarantees of Lemma 13 inductively gives $|E(G_i)| \leq 2^{-i}E(G) = 2^{-i}m$, and specifically $|E(G_l)| \leq \frac{m}{\log^2 n}$. Therefore G_l can be sparsified to H_l via the slower algorithm in time $\tilde{O}(m \log(1/\epsilon))$.

We now turn our attention to $G_1 \ldots G_{l-1}$. If G_i contains less than $O(m/(\log^2 n/4 \log\log n))$ edges, it can be left unsparsified. Otherwise, since a low-stretch tree can be obtained trivially, we can sparsify it by means of Lemma 10. Concretely, by letting $t = \log^2 n$ we get graphs $H_1, \ldots, H_{l-1}$ (the $1 \pm \epsilon$-sparsifiers of the corresponding G_i) such that

$$(1-\epsilon)G_i \preceq \hat{H}_i \preceq (1+\epsilon)G_i,$$

in total time $\tilde{O}(m \log_{\frac{m}{n \log^4 n}} n \log(1/\epsilon))$. Letting $\hat{H} = H_l + \sum_{i<l} H_i$ gives a sparsifier with $\tilde{O}(\frac{m}{\log n}/\epsilon^2)$ edges, which can in turn be sparsified in $\tilde{O}(m)$ time to generate H with $O(n \log n/\epsilon^2)$ edges. ◀

For weighted graphs, we partition edges by weights into buckets and sparsify each subgraph. Combining this type of partition with the lemmas above gives a sparsifier with $O(\log n)$ loss in edge count.

▶ **Theorem 15.** *Given a graph G with n vertices, m edges such that $m > \Omega(n \log^5 n)$ we can compute in $\tilde{O}(m \log_{\frac{m}{3n \log^5 n}} n \log(1/\epsilon))$ time a sparsifier for it with $\tilde{O}(n \log n/\epsilon^2)$ edges.*

Proof. By Section 10.2 of [12], edges whose endpoints are connected by a path with weights that are larger by a factor of n^3 can be discarded without significant changes to the spectral structure of the graph. Then grouping the edges by weights into buckets containing edges with weights $[(1+\epsilon)^i W_{\min}, (1+\epsilon)^{i+1} W_{\min}]$ gives a partition of G into $G_1, \ldots, G_l$ such that $|V(G_1)| + \cdots + |V(G_l)| \leq O(n \log n \epsilon^{-1})$. By Lemma 10 with $t = \log^2 n$, each G_i where $|E(G_i)| \geq |V(G_i)|\frac{m}{n \log^3 n}$ can be sparsified to a graph with $|V(G_i)|\frac{m}{n \log^3 n}$ edges in $\tilde{O}(\log_{\frac{m}{n}} n(|E(G_i)| + |V(G_i)| \log n))$ time. The total running time of this part is $\tilde{O}(\log_{\frac{m}{n}} n(m + n \log^2 n \epsilon^{-1}))$. Then the total number of edges remaining is at most $\frac{m}{n \log^3 n} \sum_i |V(G_i)| \leq \frac{m}{\log^2 n}$. This graph can in turn be sparsified in $\tilde{O}(m)$ time to give a sparsifier with $O(n \log n/\epsilon^2)$ edges. ◀

8 Final Remarks

We remark that the $\tilde{O}(m)$ sparsification algorithm of this paper relies 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. Given that it may be impossible to achieve this, it also makes sense to ask for algorithms that compute very sparse κ-approximations for small κ. Such algorithms could play a significant role in the development of more practical SDD solvers.

Finally, the possibility of a linear time sparsification algorithm for graphs with $m = O(n \log^c n)$ edges is left open, and we believe it poses an interesting open problem.

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

References

1 Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. *CoRR*, abs/0808.2017, 2008.

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.

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.

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

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.

6 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

7 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.

8 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.

9 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.

10 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.

11 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.

12 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *CoRR*, abs/0808.4134, 2008.

13 Nikhil Srivastava. Spectral sparsification and restricted invertibility, 2010. PhD Thesis, Yale University.

14 Luca Trevisan. Approximation algorithms for unique games. In *FOCS '05: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, pages 5–34, 2005.