Spectral Sparsification in the Semi-Streaming Setting

Jonathan Kelner, Alex Levin

Spectral Sparsification in the Semi-Streaming Setting

Jonathan A. Kelner[1] and Alex Levin[2]

1 **Department of Mathematics**
Massachusetts Institute of Technology
Cambridge, MA 02139
kelner@mit.edu

2 **Department of Mathematics**
Massachusetts Institute of Technology
Cambridge, MA 02139
levin@mit.edu

Abstract

Let G be a graph with n vertices and m edges. A sparsifier of G is a sparse graph on the same vertex set approximating G in some natural way. It allows us to say useful things about G while considering much fewer than m edges. The strongest commonly-used notion of sparsification is spectral sparsification; H is a spectral sparsifier of G if the quadratic forms induced by the Laplacians of G and H approximate one another well. This notion is strictly stronger than the earlier concept of combinatorial sparsification.

In this paper, we consider a semi-streaming setting, where we have only $\tilde{O}(n)$ storage space, and we thus cannot keep all of G. In this case, maintaining a sparsifier instead gives us a useful approximation to G, allowing us to answer certain questions about the original graph without storing all of it. In this paper, we introduce an algorithm for constructing a spectral sparsifier of G with $O(n \log n/\epsilon^2)$ edges (where ϵ is a parameter measuring the quality of the sparsifier), taking $\tilde{O}(m)$ time and requiring only one pass over G. In addition, our algorithm has the property that it maintains at all times a valid sparsifier for the subgraph of G that we have received.

Our algorithm is natural and conceptually simple. As we read edges of G, we add them to the sparsifier H. Whenever H gets too big, we resparsify it in $\tilde{O}(n)$ time. Adding edges to a graph changes the structure of its sparsifier's restriction to the already existing edges. It would thus seem that the above procedure would cause errors to compound each time that we resparsify, and that we should need to either retain significantly more information or reexamine previously discarded edges in order to construct the new sparsifier. However, we show how to use the information contained in H to perform this resparsification using only the edges retained by earlier steps in nearly linear time.

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 Algorithms and data structures, graph algorithms, spectral graph theory, sub-linear space algorithms, spectral sparsification

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.440

1 Introduction

Graph sparsification is the task of finding a sparse approximation to a given (possibly weighted) graph G. Specifically, we seek a weighted graph H on the same vertex set as G

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 440–451

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

but with fewer edges, which approximates G in some respect. This enables one to approximately solve problems on G while performing calculations on a much simpler graph, which often gives substantially faster running times. This problem has been extensively studied by many researchers, and it has recently become a very active area of investigation as a result of its connections with numerous topics, including the construction of fast solvers for symmetric diagonally dominant linear systems (see, e.g., [11, 5]).

Benczúr and Karger [3] introduced the notion of combinatorial sparsification and proved that any graph G on n vertices has a combinatorial sparsifier H with $O(n \log n/\epsilon^2)$ edges. This H preserves the total weight of the edges crossing any cut of G up to a multiplicative $1 \pm \epsilon$ factor. (Note that this condition will require H to be weighted even if G is unweighted.) Moreover, the authors provided an efficient algorithm for computing H by sampling the edges of G according to certain very carefully chosen probabilities.

Recently, Spielman and Teng [10] defined the notion of spectral sparsification. A graph H is a $1 \pm \epsilon$ spectral sparsifier of G if it preserves the quadratic form induced by the graph Laplacian to within a multiplicative $1 \pm \epsilon$ factor. This definition is strictly stronger than combinatorial sparsification. In a remarkable paper, Spielman and Srivastava [9] showed that by sampling each edge with probability proportional to its effective resistance, adding it in with a certain weight, and taking $O(n \log n/\epsilon^2)$ independent samples in this manner, one obtains a $1 \pm \epsilon$ spectral sparsifier of G with high probability.[1] Furthermore, Spielman and Srivastava showed how to approximate *all* of the effective resistances to within a $1 \pm \eta$ factor in time $\tilde{O}(m/\eta^2)$, where m is the number of edges of G; using these approximate values allows one to construct a sparsifier with only a constant blowup in the number of edges. This leads to a nearly linear (specifically $\tilde{O}(m/\epsilon^2)$) time algorithm for computing spectral sparsifiers.

Having a sparsifier allows us to approximately answer numerous algorithmic questions about a dense graph without doing as laborious a calculation. For example, we can approximate the values of cuts, the effective resistances between pairs of vertices, and many properties of the behavior of random walks on the graph. As such, sparsifiers can be used to give faster approximate algorithms to numerous problems; Benczúr and Karger did this in their paper. Additionally, in a setting where we do not have room to store a given dense graph in its entirety, or perhaps can only access it through slow memory, a sparsifier, which might be small enough to store in fast memory, can be a useful proxy.

This, along with the growing need to process extremely large graphs, leads us to ask if we can also construct the sparsifier for a graph G even if we have much less work space than it would take to represent G. In particular, we consider the semi-streaming model, where we have $\tilde{O}(n)$ memory available. The algorithm of [9] requires access to the entire graph G, and thus, does not conform to the model. In this paper, we show how to build a sparsifier almost as quickly as with the original Spielman-Srivastava algorithm, while using at most $\tilde{O}(n)$ space and taking only one pass over G.

In our analysis, we will consider a more general dynamic setting, where we start with a graph G and its sparsifier H, and, as G is constantly updated, we want to maintain a $1 \pm \epsilon$ approximation to the current graph. Specifically, we consider the case of adding edges to G. (Setting the initial graphs G and H to be empty graphs on the vertex set V, we obtain the semi-streaming case discussed above; for details, see Section 3.7.) It is not hard to see that as

[1] Spielman and Srivastava actually showed that this holds with probability at least 1/2 using Rudelson's sampling theorem [7], but a straightforward application of a stronger concentration of measure result [12, Corollary 4] yields the high probability claim.

we add edges to G, by adding in those same edges to H, we get the desired approximation of G. Unfortunately, as we keep doing this, our sparsifier will contain increasingly many edges and may eventually become too large for our needs. Thus we will need to resample, to produce a sparsifier of smaller size. We show how to periodically do this resampling very fast, leading to amortized nearly constant update time per edge added to G. The resampling algorithm relies on two main insights:

1. As we add new edges to G to produce a graph G', the effective resistances of the edges of G do not increase, and thus, neither does their probability of being selected for a sparsifier. Thus, if we can compute their new probabilities, we can rejection sample the edges in H and also appropriately sample the new edges to produce edges selected with the probability distributions from G', and hence a sparsifier of G'. Thus, we need not consider all the probabilities in G', but only those of edges in H and the added edges.
2. Since H with the new edges well-approximates G', we can use it to quickly estimate the effective resistances for the edges we need; this estimate turns out to be good enough.

On a high level, the key idea of our construction is that the original sparsifier already contains a great deal of information, which we can reuse to save time instead of building a sparsifier from scratch.

In addition to being a generalization of the semi-streaming problem, the dynamic setting has numerous applications. As [2] points out, by maintaining a sparsifier of a changing graph at all times, we will be able to perform certain calculations quickly whenever we need to know something about the current graph. Furthermore, in some cases, it might be that we have a family of graphs we can choose from, and all of them are variants on some base graph G (e.g., G with extra edges). In that instance, by sparsifying G, and using our procedure, one can produce sparsifiers of the other graphs relatively quickly.

Related work

The problem of graph sparsification in the semi-streaming setting was introduced by Ahn and Guha [2], and it was then further studied by Goel, Kapralov, and Khanna [4] (the latter of which is concurrent to and independent of the present paper). Ahn and Guha constructed combinatorial sparsifiers in the semi-streaming model. However, while the space complexity of their algorithm was $\tilde{O}(n)$, the running time was $\tilde{O}(mn)$, which is often too slow when the graphs are large. This is remedied by the present work, as well as by Goel, Kapralov, and Khanna, who obtain results that are similar to ours when one aims to construct combinatorial sparsifiers.

However, the graphs that we produce obey the strictly stronger constraints imposed by spectral sparsification. To our knowledge, ours is the first work to do this in the semi-streaming setting.

Furthermore, we believe that our algorithm is conceptually cleaner and simpler than that of [4], and our techniques are quite different from theirs. The algorithm set forth by Goel *et al.* inherently requires a logarithmic number of passes through the data, and they maintain a multi-level collection of graphs and partitions of graphs. They then, using an ingenious construction and careful analysis, find a way to implement this in a single pass. This results in a graph that has logarithmically more edges than necessary, which they then clean up at the end.

Our algorithm, on the other hand, operates inherently in a single pass. We simply add edges to our graph until it becomes large. When this occurs, we replace our graph with a sparser version still preserving the approximation guarantee and continue. By taking advantage of the stronger notion of sparsification that we are employing, and properly sparsifying

and analyzing the probabilities, we are able to show that this simple algorithm produces the desired sparsifiers while requiring a nearly constant amount of amortized work per edge and maintaining at all times a graph with $\tilde{O}(n/\epsilon^2)$ edges.

Acknowledgments This work was partially supported by NSF grant CCF-0843915. The second author is supported by a National Science Foundation graduate fellowship.

2 Background

Throughout, let $G = (V, E)$ be a graph on n vertices $V = \{1, \ldots, n\}$ and m edges. The Laplacian of G is given by $L_G := A_G - D_G$, where A_G is the adjacency matrix of G and D_G is the diagonal matrix whose i^{th} diagonal entry is the degree of vertex i. Let L_G^+ be the (Moore-Penrose) pseudoinverse of L_G.

For $i \in V$, we denote by χ_i the $n \times 1$ characteristic vector of i (having a 1 at the ith position and 0's everywhere else). For an edge e of G with endpoints $i, j \in V$, we let b_e be the $n \times 1$ vector $\chi_i - \chi_j$ (where the choice of i or j to have the positive sign is made arbitrarily). Further, denote by B the $m \times n$ edge-vertex incidence matrix, with rows indexed by the edges and columns indexed by the vertices, and whose e^{th} row is b_e. It is a standard fact that the Laplacian can be decomposed as $L_G = \sum_{e \in G} b_e b_e^T$, or, equivalently, that $L_G = B^T B$.

Finally, we discuss some notions from electrical network theory that will be relevant. Consider the graph as an electric network of nodes (vertices) and wires (edges), where each edge has resistance of 1. The *effective resistance* between vertices i and j is the voltage difference that would be induced between i and j if one unit of current were put in i and taken out from j. It can be shown that this is precisely equal to $(\chi_i - \chi_j)^T L_G^+ (\chi_i - \chi_j)$. For an edge e of G, the effective resistance of e, denoted by R_e, is defined to be the effective resistance between the endpoints of e. Namely, we have $R_e = b_e^T L_G^+ b_e$. It is a standard fact that if G is connected, we have $\sum_{e \in G} R_e = n - 1$. The sum is $n - c - 1$ if G has c connected components.

2.1 Spectral sparsifiers

Fix notation as above. In what follows, we will write $A \preceq B$ for matrices A and B when $B - A$ is positive semidefinite.

▶ **Definition 1.** A $1 \pm \epsilon$ spectral sparsifier of G is a possibly weighted graph H such that the edge set of H is contained in that of G and

$$(1 - \epsilon) L_G \preceq L_H \preceq (1 + \epsilon) L_G. \tag{1}$$

In other words, for all $x \in \mathbb{R}^n$, we have

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

Note that we have approximations for pseudoinverses as well. If H is a $1 \pm \epsilon$ sparsifier of G, it is the case that

$$\frac{1}{1 + \epsilon} L_G^+ \preceq L_H^+ \preceq \frac{1}{1 - \epsilon} L_G^+. \tag{2}$$

In particular, because the effective resistance between i and j is given by evaluating the quadratic form defined by the Laplacian pseudoinverse at $\chi_i - \chi_j$, we see that the effective resistances between any two nodes in G and H are the same up to a $1/(1 \pm \epsilon)$ factor.

Algorithm 1 Sparsify

Input: $G = (V, E)$

Set H to be the empty graph on V.

for $i = 1$ to $N = O(n \log n/\epsilon^2)$ **do**

Sample edge $e \in G$ with probability p_e proportional to R_e, and add it in with weight $1/(Np_e)$ to H

end for

In [9], Spielman and Srivastava considered Algorithm 1 for generating a sparsifier of G. They proved:

▶ **Theorem 2.** *Fix $\epsilon \geq 1/\sqrt{n}$. Then, Algorithm 1 produces a $1 \pm \epsilon$ sparsifier of G with high probability.*

Additionally, the authors were able to show how to approximate all the effective resistances in time $\tilde{O}(m/\epsilon^2)$. Up to log factors, this is optimal, since it takes $\Omega(m)$ time to even write down all of the effective resistances. This fast way of estimating resistances gives a nearly-linear time (in m) algorithm for sparsifying G.[2]

Notation

Before proceeding, we make a few remarks about notation. Let G be a graph with n vertices. Let Γ be another graph on the same vertex set as G. Then, $G + \Gamma$ is the graph given by adding the weights of the edges of Γ to the corresponding edges of G. In this paper, for the most part G and Γ will be unweighted graphs, and Γ will be edge-disjoint from G. In this case, $G + \Gamma$ represents the graph we get when we add the edges of Γ to G. The definition agrees with the previous one if we regard missing edges as having a weight of 0, and those that are in the graph as having a weight of 1.

For an edge e not in G, we denote $G + e$ the graph obtained by adding e to G.

For convenience, we define $q = q(n, \epsilon) = O(\log n/\epsilon^2)$ to be the quantity such that taking nq samples in the above algorithm (using exactly correct probabilities) gives us a $1 \pm \epsilon$ sparsifier H of G with probability at least $1 - n^{-d}$ (for some desired constant d determined at the outset).

Finally, we note that as prescribed by [9], when we add an edge to H multiple times, we just sum up the weights. In our presentation, it will be convenient to think instead of keeping parallel edges, so that our sparsifiers will have have exactly nq edges.

3 The dynamic update algorithm

Throughout, for notational convenience, we will consider the setting of adding new edges to an unweighted graph G without adding new vertices. It is straightforward to generalize to the case where we add vertices, or where the graph is weighted and we may increase the weights of existing edges as well as add new ones, provided that the weights are polynomially bounded.

[2] Throughout, we will use the terms "nearly constant" or "nearly linear" to mean constant or linear, up to poly-logarithmic factors. This terminology is fairly standard.

3.1 Setup

Initially, we assume that we are given access to the exact effective resistances when we need them to sample. We will later relax this requirement.

Suppose that G is a graph on n vertices and H is a $1 \pm \epsilon$ sparsifier of G with nq edges. (If G is the initial graph, we assume that H was produced using the sampling procedure in [9], given the correct effective resistances.) Let e be an edge not in G; then it is clear that $H + e$ is a $1 \pm \epsilon$ sparsifier of $G + e$. Indeed, we have

$$L_{G+e} = L_G + b_e b_e^T, \qquad L_{H+e} = L_H + b_e b_e^T,$$

whence the desired statement follows.

As we add edges to G, we can add those same edges to H, until the sparsifier gets too large, forcing us to resample. In this work, we say that this happens when it is of size Cnq for some constant $C > 1$ that we can choose at will.

We will formalize this situation as follows. Let $G = (V, E)$ be a graph, and let H be its $1 \pm \epsilon$ sparsifier with nq edges. Let Γ represent the added edges (i.e., it is a graph on V with edges exactly those that are added to G) such that $H_+ := H + \Gamma$ has Cnq edges. (Note that H_+ is a $1 \pm \epsilon$ sparsifier of $G' := G + \Gamma$.)

Because H_+ is large, we want to construct a sparsifier H' of G' of nq edges. We call this procedure *resparsification*. We would like this resparsification to take much less time than it would take to sample from scratch, namely $\tilde{O}(m/\epsilon^2)$. Sparsifying G' from scratch gives us an average update time of $\tilde{O}(m/n)$ per operation, which is $\tilde{O}(n)$ when G' is dense. We want a $\tilde{O}(1)$ amortized time instead. The key insight is to use the information already contained in H, which will allow us to sample edges from the correct distribution in time $\tilde{O}(n/\epsilon^2)$, leading to the desired bound.

The main observation is that when we add a new edge to G, the effective resistances of the other edges cannot increase. This is more or less clear from the physical model, and it can be proven rigorously as in [6, Lecture 9]. Further, the sum of the effective resistances of all of the edges cannot decrease. (If adding the edge reduces the number of connected components, this quantity increases, otherwise it stays the same.) Thus, the probabilities of choosing the edges in the sparsification procedure of [9] cannot increase.

In what follows, we let R_e (resp. R'_e) be the effective resistances across edge e in G (resp. G'), and p_e (resp. p'_e) be the probability of selecting those edges (i.e. $p_e = R_e / \sum_{f \in G} R_f$, and similarly for p'_e).

Consider Algorithm 2, which details the resparsification step in our simplified context.

Algorithm 2 Resparsification (knowing the correct probabilities)

Input: H, Γ %G, *the original graph, is not given*
Output: H', a $1 \pm \epsilon$ sparsifier of G' with nq edges

1: Compute the effective resistances in G of all the edges of H, and from there the probabilities of selecting them. % *We will show how to do this later*
2: Compute the effective resistances in G' of all the edges of H_+ (i.e. all the edges of H and Γ) and the probabilities of selecting them.
3: **for all** edges e of H **do**
4: With probability p'_e/p_e, add e to H' with weight $1/(nqp'_e)$
5: Otherwise, pick an edge f of Γ with probability $p'_f / \left(\sum_{g \in \Gamma} p'_g\right)$ and add it to H' with weight $1/(nqp'_f)$
6: **end for**

Note that the sampling procedure is well-defined, since $p'_e \leq p_e$. Further, we claim that all the edges are selected with exactly the correct probabilities, giving us a proper sample, so that H' is a $1 \pm \epsilon$ sparsifier of G' with high probability. Indeed, imagine selecting one edge for H' using a three-step process:

1. Select an edge from G with probability p_e.
2. Keep it with probability p'_e/p_e.
3. If you reject in Step 2, pick an edge f of Γ with probability $p'_f / \left(\sum_{g \in \Gamma} p'_g\right)$.

Note that you select edge $e \in G$ with probability p'_e, and the overall probability of rejecting in the second step is $1 - \sum_{e \in G} p'_e = \sum_{g \in \Gamma} p'_g$, hence you select f with probability p'_f, as it should be. By going through edges in H as in the algorithm and accepting them with the desired probability, we are effectively reusing the randomness we used to construct H. This, in turn, allows us to save time by only considering the probabilities for the Cnq edges of H_+, rather than for all the edges of G'.

The heart of algorithm then becomes correctly estimating the R_e and R'_e and from them, the p_e and p'_e.

3.2 Estimating effective resistances

Unfortunately, we are not able to exactly compute the effective resistances (and hence selection probabilities) quickly enough, so we will have to estimate them. We know that if our estimate of each probability is guaranteed to be at least $1/\alpha$ of the correct value (for some fixed $\alpha > 1$), then αnq samples are enough to get a $1 \pm \epsilon$ sparsifier with high probability (see [8, Corollary 6], where this fact is proven implicitly). We will be able to provide this guarantee for $\alpha \ll 2$, so if we let H have $2nq$ edges (and take $2nq$ samples for computing H'), the same claims about the sparsification quality of H and H' will hold.

With that in mind, we note that our method is closely related to the one in [9] for calculating the effective resistances quickly. We first recapitulate the ideas in that paper, and then we describe our modifications.

Recall that

$$R_e = b_e^T L_G^+ b_e = b_e^T L_G^+ L_G L_G^+ b_e = b_e^T L_G^+ B^T B L_G^+ b_e = \left\| B L_G^+ b_e \right\|^2.$$

Thus, if $e = (i, j)$, then R_e is the squared distance between the ith and jth columns of the $m \times n$ matrix BL_G^+. To compute this would be too slow, so Spielman and Srivastava make the following approximations:

1. Since the effective resistances are given by distances between $n \times 1$ vectors, we can apply the Johnson-Lindenstrauss Theorem to randomly project the vectors onto a smaller dimensional subspace such that all the distances stay roughly the same with high probability [1].

   Specifically, for $k = O(\log n/\epsilon^2)$, if we choose a $k \times m$ matrix Q each of whose entries is $\pm 1/\sqrt{k}$ uniformly at random, then, with high probability over the choice of Q, we know that for every edge $e = (i, j)$ of G, the squared distance between the ith and jth column of the $k \times n$ matrix $Z = QBL_G^+$ approximates R_e to within a $1 \pm \epsilon$ factor. Furthermore, computing QB takes $O(km) = \tilde{O}(m/\epsilon^2)$ time, since B has only $2m$ entries. (We remark that the ϵ in the above can be the same as the parameter measuring the quality of the sparsifier.)
2. Computing L_G^+ is costly, so instead, we will approximate it by approximately solving linear systems in L_G. Each solution gives us an approximation of a row of Z, and takes $O(m \log(1/\delta))$ time (since L_G has $O(m)$ entries), using the Spielman-Teng linear system

solver for symmetric diagonally dominant systems [11]. Here, δ is a parameter for the quality of approximation, which can be appropriately set. Since there are k rows, we need to solve $k \ll n$ linear systems. Let $\tilde{Z}$ be the matrix we obtain in this fashion.

Spielman and Srivastava show that if Z encodes all the effective resistances to within a $1 \pm \epsilon$ factor (which happens with high probability over the choice of projection matrix Q), then for

$$\delta \leq \frac{\epsilon}{3}\sqrt{\frac{2(1-\epsilon)}{(1+\epsilon)n^3}} \tag{3}$$

we have that $\tilde{Z}$ encodes the effective resistances to within a $(1 \pm \epsilon)^2$ factor [8, Lemma 9]. The running time of one instance of the linear system solver is thus $\tilde{O}(m \log(1/\epsilon))$, and since there are at most k of them, the total time for computing $\tilde{Z}$ is $\tilde{O}(m/\epsilon^2)$.

For our purposes, we need to compute a matrix encoding approximations to effective resistances in faster time, namely $\tilde{O}(n/\epsilon^2)$. Now, because we already have sparsifiers (H and H_+) for G and G' of size $2nq$ and $2Cnq$ respectively, we can use their Laplacians in the place of L_G for estimating the R_e and R'_e respectively. With high probability, $Q_H W_H^{1/2} B_H L_H^+$ (resp. $Q_{H_+} W_{H_+}^{1/2} B_{H_+} L_{H_+}^+$) encodes the effective resistances of edges in H (resp. H_+) to within $1 \pm \epsilon$. Here, B_H is the edge-vertex incidence matrix of H, such that $L_H = B_H^T W_H B_H$ for W_H the matrix whose diagonal entries are the weighted degrees of H; Q_H is a $k \times 2nq$ matrix with entries uniform from $\pm 1/\sqrt{k}$. The definitions of B_{H_+}, W_{H_+}, and Q_{H_+} are similar.

We can compute the $Q_H W_H^{1/2} B_H$ in $O(kqn) = \tilde{O}(n/\epsilon^2)$ time, and similarly for the matrix $Q_{H_+} W_{H_+}^{1/2} B_{H_+}$, up to a factor of C.

Thus, by running the linear system solver, we obtain matrices, call them $\tilde{Z}_H$ and $\tilde{Z}_{H_+}$, encoding the approximations to effective resistances in H and H_+.

Because the effective resistances in H and H_+ are within a $1/(1 \pm \epsilon)$ factor of those of G, in this manner, we approximate the effective resistances in G to within a factor of $(1 \pm 2\epsilon)^3$. Furthermore, solving the linear systems now takes $\tilde{O}(n/\epsilon^2)$ time, since the Laplacians have $\tilde{O}(n/\epsilon^2)$ entries (we are absorbing the constant C into the big-$\tilde{O}$ notation).

3.3 An alternate sparsification algorithm

We would now like to use the approximate effective resistances to run a variant of Algorithm 2, which simulates the random process in Algorithm 1. For technical reasons, in this setting, it is useful to consider the following variant of Algorithm 1, and simulate it instead:

Algorithm 3 Alternative sparsify

Input: G

Output: H, a $1 \pm \epsilon$ sparsifier of G (with high probability)

```
for all edges e of G do
    for i from 1 to N := O(n log^2 n/ε^2) do      %Run this loop implicitly
        With probability p_e = R_e/(n−1) add e to H with weight 1/(Np_e)      %R_e
        is the effective resistance of e in G.
    end for
end for
return H
```

This algorithm produces a $1 \pm \epsilon$ sparsifier H of G with high probability. Moreover, the number of edges in H is tightly concentrated around $O(N)$. If we underestimate each p_e by at most an α factor, and use these estimates to run the algorithm, we will have to increase N by the same α factor to get the high probability claim.

The inner loop is run implicitly; for each edge, given p_e, it is easy to determine how many samples of the edge will be taken, as this follows a binomial distribution. In particular, once we know the probability, the total running time of the algorithm is proportional to the total number of edges we select, which is $O(N)$ with high probability.

Note that the sparsifiers we get as a result have $O(\log n)$ more edges than the sparsifiers produced by Algorithm 1. We need the extra factor for a technical reason, and from now on, will assume that all the sparsifiers we deal with are of size $O(n \log^2 n/\epsilon^2)$.

As before, if we add multiple copies of an edge to H, we treat them as parallel edges in what follows.

3.4 Putting it all together

Now we are ready to show the final algorithm. So, let G be a graph and H its $1 \pm \epsilon$ sparsifier generated according to Algorithm 3, with $2N$ rather than N steps in the inner loop, where, from now on, $N := O(n \log^2 n/\epsilon^2)$. The sparsifier H has $O(N)$ edges with high probability. Let the $\tilde{p}_e$ be the estimates of the probabilities of edges in G, which were used to generate H. As before, Γ will represent the new edges (of which there will now be $O(n \log^2 n/\epsilon^2)$), $G' := G + \Gamma$, and $H_+ := H + \Gamma$. Denote by $\tilde{p}'_e$ the probabilities of edges in G', computed using H_+, as described previously.

Consider Algorithm 4.

Algorithm 4 Resparsification

Input: H, Γ, as well as the $\tilde{p}_e$ for every edge $e \in H$.

Output: H', a $1 \pm \epsilon$ sparsifier of G' with $O(n \log^2 n/\epsilon^2)$ edges with high probability, as well as $\tilde{p}'_e$ for every edge $e \in H'$.

1: Estimate the effective resistances in G' of all the edges of H_+.
2: For $e \in H_+$, let $\tilde{p}'_e = \tilde{R}'_e/(n-1)$ %*Good approximation to true* p_e
3: **for** each edge e of H **do**
4: $\tilde{p}'_e \leftarrow \min(\tilde{p}_e, \tilde{p}'_e)$
5: **end for**
6: **for all** edges e of H **do**
7: Keep e with probability $\tilde{p}'_e/\tilde{p}_e$ and add it to H' with weight $1/(2\tilde{p}'_e N)$.
8: **end for**
9: **for all** edges e of Γ **do**
10: **for** i from 1 to $2N$ **do** %*Do this loop implicitly*
11: With probability $\tilde{p}'_e$ put e into H' with weight $1/(2\tilde{p}'_e N)$
12: **end for**
13: **end for**
14: **return** H' and the $\tilde{p}'_e$ for $e \in H'$.

It is not hard to see that this algorithm simulates the random process for sparsifying G' using Algorithm 3. (Again, we reuse the randomness used to generate H.) We can see that if we keep adding edges and resparsifying, we will put edges into the resulting sparsifier with exactly the desired probability, up to the modification in Step 4. The probability is over the randomness of the entire algorithm up to the current step.

We remark that we need Step 4 since approximation errors might cause the estimate of the probability of an edge to go up after we have added Γ, even though the true probabilities should go down. For rejection sampling to simulate the proper probability distributions, the probabilities have to be non-increasing. We show that the change does not in fact hurt our construction.

Indeed, suppose p_e and p'_e are the true probabilities of e in G and G' respectively, and assume that $\tilde{p}_e \geq p_e/\alpha$ and $\tilde{p}'_e \geq p'_e/\alpha$ for some α (which is much smaller than 2). We must have $p_e \geq p'_e$, hence $\tilde{p}_e \geq p_e/\alpha \geq p'_e/\alpha$, and hence $\min(\tilde{p}_e, \tilde{p}'_e)$ is at least as big as p'_e/α. Therefore, the samples we take suffice to guarantee that H' will be a $1 \pm \epsilon$ sparsifier with high probability.

Computing the matrices encoding the effective resistances takes $\tilde{O}(n/\epsilon^2)$ times. We only need to compute $\tilde{O}(n/\epsilon^2)$ effective resistances (since we do this only for edges in H and those in H_+). Sampling also takes $\tilde{O}(n/\epsilon^2)$ time. Since we resparsify every $\tilde{O}(n/\epsilon^2)$ steps, we conclude that the update procedure takes $\tilde{O}(1)$ steps per added edge.

This procedure works to give resparsified graphs with the correct approximation guarantee with high probability as long as the previous resparsified graph was a $1 \pm \epsilon$ sparsifier. (Note that by construction, the previous graph will always have edges drawn from a probability distribution that is close to correct; we need it to be a good sparsifier so that we can use it to quickly obtain good approximations to effective resistances.) We can union bound the probability of failure of any one of the resparsification steps (where we can include the event that the sparsifier ends up too big as a failure). The other potential place for error is when we apply the Johnson-Lindenstrauss theorem, but again, we have a high probability guarantee there, and are free to increase the number of rows k in the projection matrices Q_H and Q_{H_+} by a constant factor. Thus, by picking N and k to have a large enough constants at the outset, we can perform this procedure any desired polynomial number of times and be guaranteed that we always maintain a sparsifier with high probability.

By keeping careful track of the running times of the construction, we can prove:

▶ **Theorem 3.** *Our dynamic update algorithm takes* $O(\log^4 n (\log\log n)^3 \log(1/\epsilon)/\epsilon^2)$ *operations per added edge.*

Proof. The bottleneck in the algorithm is solving the linear systems in the resparsification step. Using the recent results of Koutis, Miller, and Peng [5], which give the best asymptotics known to date, the solution to each linear system for the Laplacian of a graph with $O(n \log^2 n/\epsilon^2)$ edges can be approximated in time $O(n \log^4 n (\log\log n)^3 \log(1/\delta))$. Solving $O(\log n/\epsilon^2)$ linear systems with δ as in (3), requires time $O(n \log^6 n (\log\log n)^3 \log(1/\epsilon)/\epsilon^4)$. Since we resparsify after adding $O(n \log^2 n/\epsilon^2)$ edges, the amortized cost is

$$O(\log^4 n (\log\log n)^3 \log(1/\epsilon)/\epsilon^2)$$

per added edge, as claimed. ◀

3.5 Error-forgetfulness of the construction

Before concluding this section, we note one interesting property of our construction in Algorithm 4. Using H and H_+, which are approximations to G and G' respectively, we obtain estimates on effective resistances, which are slightly worse than those we would get had we used the full graphs G and G' (but allow us to do the computation much faster). Despite the approximations that we make, by taking twice as many samples as we would have needed had we known the true probabilities, we once again obtain a high-quality sparsifier (with

high probability), allowing us to make the approximation all over. In other words, because we take enough samples, and do so intelligently, the errors we make in approximating the effective resistances do not propagate; the procedure has no memory for the approximations we made in the past.

Compare this to a more naïve approach to the problem of resparsifying. If we have G, G', H and H_+, defined as before, it is tempting to use Algorithm 1 to sparsify H_+ directly to a graph of size nq. Unfortunately, the resulting graph $\bar{H}$ is a $1 \pm \epsilon$ approximation of H_+, which is a $1 \pm \epsilon$ approximation of G', so $\bar{H}$ is only guaranteed to be a $(1 \pm \epsilon)^2 \approx 1 \pm 2\epsilon$ sparsifier of G'. In other words, the error propagates.

3.6 Straightforward generalizations

It is easy to generalize the above construction to the following cases. First, the construction goes through almost directly for the case of weighted graphs, where we are allowed to add weighted edges. For example, the probability of selecting an edge becomes the weight of that edge times its effective resistance. The weights with which we add sampled edges depend on their weights in G, so in order to do this properly, we should store the weights of the edges in the current sparsifier.

We can also consider operations where we increase the weight of an edge e of G by some amount w. In this case, we imagine adding an edge parallel to e and with weight w to G, and proceed as before (we add e' with weight w to H, and resparsify after some number of steps). The reason for considering parallel edges here is that while increasing the weight of an edge decreases the probabilities of other edges, it may increase the probability of that edge, which can stymie our construction. If we instead add an independent copy of the edge, all the arguments go through.

The only thing we have to be careful about is that in the weighted case, the value of δ in (3) depends on the ratio of the maximum weight to the minimum weight in the graph. If this is always bounded by some polynomial in n, then we need to add at most a factor of $\log n$ to the running time of the linear system solver, and hence of the overall algorithm.

Secondly, we can envision adding vertices as well as edges to G. Adding a vertex and connecting it by an edge to some existing vertex does not affect the effective resistances of the other edges, and it does not increase the number of connected components in the graph. Hence, once again, the probability of existing edges can only decrease, and we can use the same arguments. Here, by adding vertices, we increase the number of times we need to sample in the inner loop of Algorithm 3 in order to get a $1 \pm \epsilon$ approximation guarantee. If we have an upper bound on the number of vertices we will end up with, we can ensure that we take enough samples from the outset.

3.7 The semi-streaming setting

The dynamic algorithm described above goes through almost unchanged in the semi-streaming case (where we start with the empty graph). After adding the first $2CN$ edges (where $N = O(n \log^2 n/\epsilon^2)$), we use Algorithm 4 (with H set to the empty graph and Γ set to the current graph, and $2N$ iterations in the inner loop), giving us a $1 \pm \epsilon$ approximation to the current graph, containing of $2N$ edges in expectation. The number of edges is in fact tightly concentrated around this expectation. Then we continue as before, adding edges and resparsifying every $2CN$ steps.

For our algorithm to be valid in the semi-streaming model, we only need to prove that it requires $\tilde{O}(n/\epsilon^2)$ work space. But this is immediate, since, with high probability, we will

only deal with graphs of $\tilde{O}(n/\epsilon^2)$ edges throughout the run.

If we would like to end up with a sparsifier containing $O(n \log n/\epsilon^2)$ edges, we can run Algorithm 1 on the output, which will change the final error guarantee from $1 \pm \epsilon$ to $(1 \pm \epsilon)^2$.

4 Conclusions and future work

We have presented an algorithm for maintaining a sparsifier of a growing graph, such that the average time is $\tilde{O}(1)$ for each added edge. The main idea is a resampling procedure that uses information in the existing sparsifier to construct a new one very quickly. Our construction is robust and holds relatively unchanged for several natural variants. An interesting question left open by our work is whether similar results could be obtained in a dynamic model that permits the removal of edges as well. While this is somewhat unnatural in the semi-streaming setting, it is a very reasonable goal in the dynamic setting where one aims to maintain a sparsifier for a graph that is changing over time.

References

1. 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, New York, NY, USA, 2001. ACM.
2. Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part II*, ICALP '09, pages 328–338, Berlin, Heidelberg, 2009. Springer-Verlag.
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, New York, NY, USA, 1996. ACM.
4. A. Goel, M. Kapralov, and S. Khanna. Graph sparsification via refinement sampling. arXiv: 1004.4915 [cs.DS].
5. 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.
6. Gregory F. Lawler and Lester N. Coyle. *Lectures on contemporary probability*, volume 2 of *Student Mathematical Library*. American Mathematical Society, Providence, RI, 1999.
7. M. Rudelson. Random vectors in the isotropic position. *J. Funct. Anal.*, 164(1):60–72, 1999.
8. Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. arXiv: 0803.0929v4 [cs.DS].
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, New York, NY, USA, 2008. ACM.
10. Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. arXiv: 0808.4134 [cs.DS].
11. 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 Thirty-Sixth Annual ACM Symposium on Theory of Computing*, pages 81–90, New York, NY, USA, 2004. ACM.
12. Roman Vershynin. A note on sums of independent random matrices after Ahlswede-Winter. `http://www.umich.edu/~romanv/teaching/reading-group/ahlswede-winter.pdf`.