CS369G: Algorithmic Techniques for Big Data Spring 2015-2016

Lecture 12: Graph Stream Algorithms

Prof. Moses Charikar *Scribe: Nolan Skochdopole*

1 Overview

In this lecture, we continue our discussion of graph streaming algorithms. In particular, we establish some space lower bounds for approximating the all-pairs shortest paths problem. We define cut and spectral sparsifiers and go over an algorithm to compute a spectral sparsifier of a graph G in a streaming environment. Finally, we go over a couple streaming algorithms for counting triangles. Note that throughout this lecture, we are assuming all streaming is just edge additions (no edge deletions).

2 Space lower bound for shortest paths

Last time we saw a streaming algorithm for the all-pairs shortest paths problem, which gave an α-approximation (i.e. $d(u,v) \leq \hat{d}(u,v) \leq \alpha d(u,v)$, $\forall u, v \in V$). The algorithm keeps a sketch of the graph in the form of a subgraph. When $\alpha = 2k-1$, the sketch has space $O(n^{1+1/k})$, which we proved last class. We will now show that this space is essentially tight, even for sketches/data structures that aren't necessarily spanners (subgraphs of the original graph). That is, any sketch that gives an α-approximation to all-pairs shortest paths requires space $\Omega(n^{1+1/k})$.

2.1 Exact all-pairs shortest paths

We will first consider solving the problem exactly, which corresponds to $k = 1$ (we assume unweighted graphs, but analysis for weighted graphs should be similar). On n nodes, there are $2^{\binom{n}{2}}$ labeled graphs, each with distinct sets of distances. To see this fact, consider that any pair of distinct labeled graphs on n nodes differ on at least one edge, say (u,v); then, in one graph, $d(u,v) = 1$ and in the other graph, $d(u,v) > 1$. Therefore, if we want to distinguish exactly between any of these graphs, we need a sketch with $\log 2^{\binom{n}{2}} = \binom{n}{2}$ bits.

2.2 Lower bound for general k

We will use a similar argument for the space lower bound for $k > 1$, and we will make use of the following fact.

Fact 1. *There exists a graph with $n^{1+1/k}$ edges and no cycles of length $\leq c \cdot k$ for some constant c.*

We omit the proof of this fact, but it can be done with the probabilistic method[1]. Suppose we

[1] Erdos' girth conjecture states that there exists a graph on $\Omega(n^{1+1/k})$ edges and no cycles of length $\leq 2k+2$

wished to approximate all-pairs shortest paths for this graph. This graph has $2^{n^{1+1/k}}$ subgraphs. We must be able to distinguish between any two of these subgraphs; every pair of subgraphs must be different on at least one edge, say (u, v). In one subgraph, $d(u, v) = 1$ and in the other that does not contain that edge, $d(u, v) > ck - 1$ because[2] the original graph has no cycles of length $\leq ck$. Therefore, to exactly distinguish between any of these subgraphs, we need $\log(2^{n^{1+1/k}}) = n^{1+1/k}$ bits, and the space required for the α-approximation for all-pairs shortest path is $\Omega(n^{1+1/k})$.

3 Approximating cuts via sparsifiers

A cut of a graph G is a partition of the vertices into two disjoint sets, S and $\bar{S}$ $(= V \setminus S)$. We are interested in the weight of the edges of G going between these two sets, called the cut-set, i.e. we want to know the value given by $E_G(S, \bar{S}) = \sum_{(u,v)\in E | u\in S, v\in \bar{S}} w_{uv}$, or at least approximate this for any cut in G. One such method to approximate cuts is to find a suitable cut sparsifier[1].

Definition 2. *A (weighted) subgraph H of G is called cut sparsifier of G if:*

$$E_G(S, \bar{S}) \leq E_H(S, \bar{S}) \leq (1 + \epsilon)E_G(S, \bar{S}),\ \forall S \subseteq V$$

3.1 Notation

For a weighted graph $G(V, E)$ we define the Laplacian of the graph, L_G, to be an $n \times n$ matrix with the following entries[3]:

$$L_G(i, j) = -w_{ij} \text{ if } i \neq j$$
$$L_G(i, i) = \sum_j w_{ij}$$

Definition 3. *A (weighted) subgraph H of G is called a spectral sparsifier[2] if:*

$$x^T L_G x \leq x^T L_H x \leq (1 + \epsilon)x^T L_G x,\ \forall x \in \mathbb{R}^n$$

Note that spectral sparsifers are more general than cut sparsifers and were originally motivated by work on fast solvers for symmetric diagonally dominant systems.

Observation 4. $x^T L_G x = \sum_{(i,j)\in E} (x_i - x_j)^2 w_{ij}$.

We can easily see this observation by viewing the graph Laplacian as the the sum of edge Laplacians:

$$L_G = \sum_{(i,j)\in E} L_{(i,j)} = \sum_{(i,j)\in E} w_{ij}(e_i - e_j)(e_i - e_j)^T$$
$$\Rightarrow x^T L_G x = \sum_{(i,j)\in E} w_{ij} x^T (e_i - e_j)(e_i - e_j)^T x = \sum_{(i,j)\in E} w_{ij}(x_i - x_j)^2$$

[2]Given the Erdos girth conjecture, we approximate c as 2 to recover the fact that there is a distance on any pair of subgraphs that differs by $> 2k - 1 = \alpha$

[3]Note that if $(i, j) \notin E$, we suppose $w_{ij} = 0$.

where we have used e_i to denote the ith standard basis vector for $\mathbb{R}^n$.

Notice that when $x \in \{0,1\}^n$, $x^T L_G x = E_G(S, \bar{S})$ where $S = \{i | x_i = 1\}$. That is, the quadratic form with the Laplacian gives cut sizes on these x vectors. Therefore, all spectral sparsifiers are cut sparsifiers. However, the other direction does not hold.

3.2 Computing spectral sparsifiers in a non-streaming environment

If we think of graphs as electrical networks, we may compute effective resistances for every edge. Spielman and Srivastava [3] showed that if we sample edges proportional to effective resistances, we may get a $(1+\epsilon)$-spectral sparsifer with $O(\epsilon^{-2} n \log n)$ edges. Note that this result is a randomized construction. Intuitively, this construction should make sense; if any edge is part of a cut with very few edges in it, its effective resistance will be higher and so we will have a higher chance to sample it.

Batson, Spielman and Srivastava [4] later developed a deterministic algorithm to give $(1+\epsilon)$-spectral sparsifers with $O(\epsilon^{-2} n)$ edges. We will use these results as black boxes.

3.3 Computing spectral sparsifiers in a streaming environment

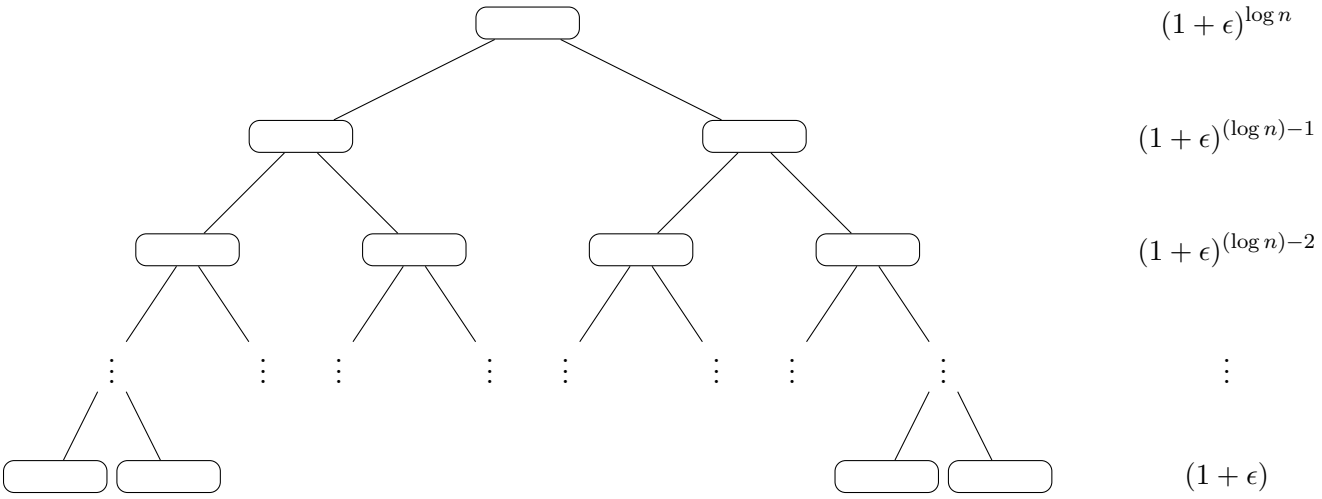
We will use the following facts about compositions of sparsifiers.

Fact 5. *Let H_1 be a sparsifier of G_1 and H_2 be a sparsifier of G_2, then $H_1 \cup H_2$ is a sparsifier of $G_1 \cup G_2$.*

Fact 6. *Let H_1 be an α-sparsifer of G and H_2 a β-sparsifier of H_1, then H_2 is an $\alpha\beta$-sparsifier of G.*

The proofs of these facts are easy to see by the linearity of Laplacians and definition of sparsifiers.

To compute a spectral sparsifier of a graph in a streaming environment, consider processing the stream in chunks, calculating sparsifiers for each chunk. We can then combine those sparsifiers to a new sparsifier, and we can continue to do this in a binary-tree structure until we are left with one final sparsifier. Please see the below figure for a representation of the sparsifier computations.



Note that the numbers on the right in the above figure represent the level of approximation at that level assuming all sparsifiers are $(1+\epsilon)$ originally. To have a useful final sparsifier, we need to set $\epsilon = \frac{\delta}{\log n}$ (to give about a $(1+\delta)$-approximation in the end).

Each sparsifier needs $O(\epsilon^{-2} n \log^2 n)$ edges if we use the deterministic algorithm of Batson, Spielman and Srivastava. There are $\log n$ levels of the sparsifier computation tree. And notice that at most one of the sparsifiers at each level of the tree is waiting for its sibling in order to be sparsified[4]. Therefore, we have at most $\log n$ sparsifiers at any one time, for a total space of $O(\epsilon^{-2} n \log^3 n)$. Also, note that we can try to form a different tree structure (other than binary) and analyze the tradeoffs between space and time of the algorithm with different structures.

If we want to construct and maintain spectral sparsifiers for streaming environments in which edges can arrive and leave, we can do so in $O(n \text{ polylog } n)$ space [5], but we will not go into details here.

4 Counting subgraphs

One common problem people are interested in is counting the number of a certain subgraph within a graph. We will focus on counting triangles, which is related to the clustering coefficient among other things. The clustering coefficient is given by: $\frac{1}{n}\sum_{v \in V} \frac{T_3(v)}{\binom{\deg(v)}{n}}$, where $T_3(v)$ is the number of triangles containing v.

Note that $\Omega(n^2)$ space is required to distinguish between cases of 0 and 1 triangles. Typically we assume T_3 is large enough to approximate it so we do not need to worry about this limit.

Think of constructing a vector x of length $\binom{n}{3}$ for the graph, such that $x_T = |\{\text{edges in } T \cup G\}|$ for all $T = \{i, j, k\}$ triplets. That is, the vector keeps track of the number of edges in G for every possible triangle. Then to compute the number of triangles of G, we simply need to compute the number of coordinates of x with entries equal to 3.

4.1 Counting triangles with frequency moments

Of course, we do not want to keep this vector, it is too large. So to approximate the number of triangles we can use sketches for frequency moments. Recall that $F_k = \sum_T x_T^k$ and that we can approximate F_0, F_1 and F_2 in a small amount of space.

It is easy to check that $T_3 = F_0 - 1.5F_1 + 0.5F_2$ gives the number of triangles (number of coordinates of x with value of 3)[5]. This sketch has very small space, but may require too much time. Each new edge updates n^2 coordinates of the vector x, so many updates per edge are needed. If we disregard the update times, we want to analyze what sort of space we need to guarantee a $(1+\epsilon)$ approximation to triangle counting. Note that we are still assuming T_3 is large enough so that we do not run into bad lower bounds.

[4] Whenever we have two siblings in the tree, we will immediately sparsify them to save space.

[5] This is just a polynomial that has roots at 0, 1 and 2 and evaluates to 1 at 3.

Suppose we have $(1+\gamma)$-approximations for F_0, F_1 and F_2. Then it's not hard to check the following:

$$\begin{aligned}|\hat{T_3} - T_3| &\leq \gamma(F_0 + 1.5F_1 + 0.5F_2) \\ &\leq 8\gamma \cdot m \cdot n\end{aligned}$$

Thus we want $\frac{8\gamma mn}{T_3} \approx \epsilon$, i.e. $\gamma \approx \frac{\epsilon T_3}{mn}$. This requirement gives us space $O(\gamma^{-2}) = O\left(\frac{1}{\epsilon^2}\left(\frac{mn}{T_3}\right)^2\right)$.

This space is not great if the number of triangles is small, but can be good if the number of triangles is large.

4.2 Counting triangles via sampling x

We can sample from non-zero coordinates and estimate the number of non-zero coordinates that are 3. The number of non-zero coordinates is mn, so $\frac{T_3}{mn}$ is the fraction of triangles. If we sample $O\left(\frac{mn}{T_3} \cdot \frac{\text{poly}\log n}{\epsilon^2}\right)$ non-zero coordinates of x, we can get an estimate of T_3. Note that we do not know T_3, but the space and number of samples of these algorithms depend on it, so in order for these algorithms to make sense we need some good guarantee for T_3, such as a good lower bound.

References

[1] A. Benczur, D. Karger, Randomized Approximation Schemes for Cuts and Flows in Capacitated Graphs, *SIAM Journal on Computing*, 44(2), pp.290-319, 2015.

[2] D. A. Spielman, S. Teng, Spectral Sparsification of Graphs, *SIAM Journal on Computing*, 40(4), pp.981-1025, 2011.

[3] D. A. Spielman, N. Srivastava, Graph Sparsification by Effective Resistances, *SIAM Journal on Computing*, 40(6), pp.1913-1926, 2011.

[4] J. Batson, D. A. Spielman, N. Srivastava, Twice-Ramanujan Sparsifiers, *SIAM Review*, 56(2), pp.315-334, 2014.

[5] M. Kapralov, Y. T. Lee, C. Musco, C. Musco, A. Sidford, Single pass spectral sparsification in dynamic streams, *Foundations of Computer Science (FOCS)*, pp.561–570, 2014.