CS 395T: Sublinear Algorithms, Fall 2020 November 5, 2020

Lecture 21: More Graph Sketching

Prof. Eric Price *Scribe: Matthew Faw*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Overview

In the last lecture, we began discussing the graph sketching algorithms of [AGM12], where instead of storing the entire graph on n nodes using $O(n^2)$ space, the goal will be to store an $O(n \log^{O(1)}(n))$-space *linear sketch* of the graph. Using this sketch, several relevant properties of the graph can be inferred from the sketch with high probability:

- Sample a random edge from a cut
- Estimate the size of a cut to $1 \pm \epsilon$
- Find a spanning forest
- Find a $(1+\epsilon)$-approximate minimum spanning tree
- Test bipartiteness

All of these are, in fact, based on *one* idea: store a linear sketch (e.g., the sampler sketch) applied to the signed adjacency matrix, $E \in \mathbb{R}^{\binom{n}{2} \times n}$, of the graph. Recall that the entries of E are given as, for any edge e and vertex v in G:

$$E_{e,v} = \begin{cases} 0 & \text{if } e \notin \text{edges}(G) \text{ or } v \notin G \\ \pm 1 & \text{otherwise} \end{cases}$$

where $\sum_{v \in [n]} E_{e,v} = 0$ for each $e \in E$. Each algorithm will use a linear sketch $A \in \mathbb{R}^{m \times \binom{n}{2}}$, and stores $AE \in \mathbb{R}^{m \times n}$ using $O(mn)$ words. The interpretation of AE is that each *vertex* will store a sketch of its neighbors in the graph. If we choose A to be the sampler sketch, then we can sample a random edge for any cut $(S, \bar{S})$ using $O(\log^2(n))$ rows. If we choose A as an AMS-sketch (ℓ_2 sketch), then we can estimate the size of a cut to $(1 \pm \epsilon)$ with high probability using $O\left(\frac{\log n}{\epsilon^2}\right)$ rows.

In this lecture, we will discuss finding a spanning forest, k-connectivity, min-cut estimation, testing bipartiteness, and spectral sparsifiers.

2 Finding a spanning forest

Goal: Find a spanning tree in G, using the sketch AE.

2.1 An (incorrect) first attempt

Idea(Similar to Prim's algorithm): Grow a spanning tree by selecting a node, and repeatedly select a random edge from the unselected edges outgoing from the leaves of the spanning tree.

This idea would require sampling edges only n times, and the random sampling succeeds with high probability, so one might expect that this procedure would produce a spanning tree.

However, it *does not work*, because the cuts are constructed *adaptively*, based on the outcomes of previous random sampling. Thus, at the second round, there are n possibilities for second round, and n^2 for the third round, since which cut you ask about depends on the answers obtained from the sampler sketch matrix so far.

Question: Can we fix this issue by adding more rows to A?

There are 2^n possible cuts, so we need the sampler sketch to fail with probability $\delta = 2^{-n}$, which implies that A needs to have $m = O(\log^2(n)\log(\delta^{-1})) = O(n\log^2 n)$ rows. This implies that AE requires space $O(n^2\log^2 n)$. But storing the *entire* graph requires only n^2 *bits*, so this bound is meaningless.

2.2 A better idea

The previous algorithm was quite similar to Prim's algorithm to find a minimum spanning tree. Let's consider a *different* algorithm for computing a minimum spanning tree.

Borůvka's algorithm: each vertex picks an incident edge, then collapse all connected components to vertices, and repeat.

Note that Borůvka's algorithm repeats $O(\log n)$ times, since each repetition of this process, the number of components shrink by a factor of two.

Question: How can we use this for the graph streaming problem?

We can use this algorithm in the following way:

- **Round** 1: Apply sampler sketch A_1 to all nodes $v \in G$, obtaining n random edges, and connected components $S_1, \ldots, S_k$.
- **Round** 2: Sample for each cut $(S_i, \bar{S}_i)$, with a *fresh* sampler sketch A_2.
- Continue in this fashion, using fresh sampler sketches A_j, $j \in \{3, 4, \ldots, \log n\}$ (note, at each round, we may filter the constructed tree to remove any cycles).

Since we use a fresh sampler sketch at each round, and each sampler sketch requires $O(\log^2 n)$ words, storing every A_jE for $j \in [\log n]$ will require a total of $O(n\log^3(n))$ words of space.

Theorem 1. *$O(n\log^3(n))$ words of space suffice to find a spanning forest in G.*

3 k-connectivity

Warm-up: Solve connectivity – for every pair of vertices u, v, does there exist a path connecting u and v?

We can simply check for this by constructing a spanning forest, since connectivity in the spanning forest is equivalent to connectivity in the underlying graph.

Question: How can we check, for any two vertices u, v, if they are k-connected: do there exist k edge-disjoint $u - v$ paths? How much space will it require?

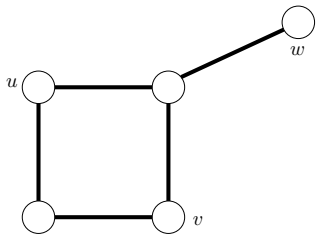

Figure 1: An illustration of k-connectivity. In this figure, u and v are 2-connected, since there are two edge-disjoint paths connecting them. u and w, however, are only 1-connected

Idea: Let's generalize the above warm-up exercise:

1. Find a spanning forest $H^{(1)}$.
2. Delete $H^{(1)}$ from G (recall that we have a linear sketch of G, so this can be done)
3. Find another spanning forest, with a *fresh* A
4. Repeat to produce k total paths from u to v.

Note that this requires a total of $O(kn \log^3 n)$ words of space in total.

Theorem 2. *$O(kn \log^3(n))$ words of space suffice to determine if any pair of nodes u, v is k-connected.*

4 Estimate min-cut

Goal: Estimate value of min-cut to $(1 \pm \epsilon)$, and find a cut that is close to this value.

Now, k-connectivity works great if k is small, but if $k = \frac{n}{100}$, then our algorithm above would require n^2 space.

Question: How can we test if min-cut is $> 2k$ or $< k/2$?

Certainly we can do this in $O(kn \log^3 n)$ words, by k-connectivity.

Idea: Subsample the edges at rate $p \approx \frac{\log n}{k\epsilon^2}$.

The min cut will become $< \frac{1}{2} \frac{(1\pm\epsilon) \log n}{\epsilon^2}$ or $> 2 \frac{(1\pm\epsilon) \log n}{\epsilon^2}$. We can distinguish these two cases by using a Chernoff bound.

Now, Karger's min-cut algorithm says that, if $p > \frac{\log n}{\epsilon^2 n}$, then the min-cut on the subsampled graph will be $(1 \pm \epsilon)pk$. As a result, can test $(1-\epsilon)k$ vs $(1+\epsilon)k$ in $\frac{\log n}{\epsilon^2} n \log^2 n$ space. By testing at $1/\epsilon$ values of k, this would translate to $\frac{\log n}{\epsilon^3} n \log^2 n$ space. However, can achieve $\frac{\log n}{\epsilon^2} n \log^2 n$ by sampling at scales $\frac{1}{2}, \frac{1}{4}, \ldots, \frac{1}{n^2}$.

Theorem 3. *$O\left(\frac{\log n}{\epsilon^2} \log^2 n\right)$ words of space suffice to estimate the min-cut of G to $(1 \pm \epsilon)$.*

5 Testing bipartiteness

Goal: Determine if G is bipartite or not.

Idea: Replace each *vertex* v with two vertices, v_1 and v_2. Replace each edge $u, v \in E$ by (u_1, v_2) and (v_1, u_2). An illustration of this transformation is shown in Figure 2.

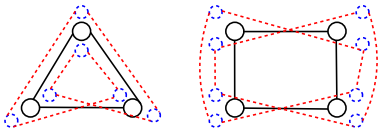

Figure 2: An illustration of the transformation applied to the nodes edges in a triangle and square graph. The original graphs are shown in the black, solid lines, and the tranformed graph is shown in dashed lines. For each node, we create two nodes (a *top* and *bottom*), and connect each top to a corresponding bottom, and each bottom to a top.

Claim: Bipartite and non-bipartite graphs are *different* in the resulting transformation. In fact, each component becomes two components iff the graph is bipartite.

The claim follows by noting that a graph is bipartite iff it has no cycles of odd length. Thus, when the graph is bipartite, any two v_1, v_2 will never be connected. Otherwise, there do exist cycles of odd length, and thus there exist v_1, v_2 that are connected.

Thus, we may check if G is bipartite by:

1. Transforming the graph, as described above
2. Creating a spanning tree on the graph
3. For each pair v_1, v_2 check if v_1 connects to v_2.

Theorem 4. *$O(n \log^3 n)$ words of space suffice to determine if G is bipartite.*

6 Spectral Sparsifiers

6.1 Cut Sparsifiers

Goal: Find a weighted graph H such that, for all sets S,

$$|\text{cut}_H(S)| = (1 \pm \epsilon)|\text{cut}_G(S)|$$

and

$$|\text{edges}(H)| = o\left(\frac{1}{\epsilon^2} n \log^c n\right)$$

for some constant c.

Thus, if one would like to run max-flow/min-cut on G, a dense graph, there are constructions of H for which, if one ran max-flow/min cut on H instead, can still obtain a good approximation of the result on G, and H requires only $O(n\text{polylog}(n))$ space.

6.2 Generalization: Spectral Sparsifier

Goal: For every $x \in \mathbb{R}^n$,

$$\begin{aligned}\text{cost}_H(x) &:= \sum_{e=(u,v)\in H} w(e)(x_u - x_v)^2 \\ &= x^T L x,\end{aligned}$$

where L is the Laplacian.

Note that this is indeed a generalization of Cut Sparsifiers, since if we set $x = \mathbb{1}\{u \in S\}$, then $\text{cost}_H(x) = \sum_e w(e)\mathbb{1}\{e \text{ is on the cut } (S, \bar{S}) = |\text{cut}_H(S)|$.

Note: This cost has an engineering interpretation. Indeed, we can view $\text{cost}_H(x)$ as the power used if we apply voltages x to resistance network $R_e = \frac{1}{w(e)}$.

What is known about spectral sparsifiers?

1. Spectral sparsifiers exists, and are easy to compute offline.
2. Can produce them with $O\left(\frac{n}{\epsilon^2}\right)$ edges. [Easy: $O\left(\frac{n \log n}{\epsilon^2}\right)$ by sampling edges by the inverse of their effective resitance.]
3. Challenging, but true: can compute with a linear sketch in $O\left(\frac{n \log^c(n)}{\epsilon^2}\right)$ space.

We will show an *easier* version of the final point above – restricting ourselves to the *insertion-only* model. We will show this by using *merge and reduce*. We will begin by splitting our stream of G into $G_1, G_2, \ldots$. Now, we may write

$$\begin{aligned}\text{cost}_G(x) &= \sum_{e=(u,v)\in E} w(e)(x_u - x_v)^2 &(1)\\ &= x^T L_G x &(2)\\ &= x^T (\sum_i L_{G_i}) x &(3)\end{aligned}$$

Now, by the properties of spectral sparsifiers, we know that

$$x^T L_{H_i} x = (1 \pm \epsilon') x^T L_{G_i} x$$

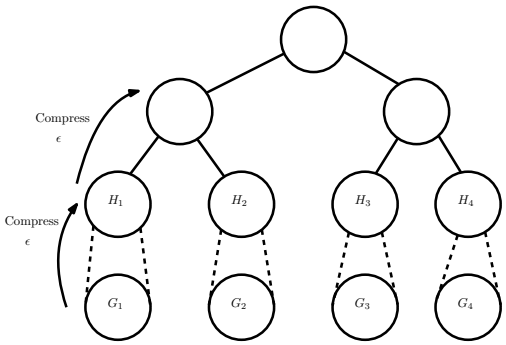

Figure 3: An illustration of the merge and reduce idea. We split the stream into pieces G_i. H_i denote the spectral sparsifiers of the corresponding G_i. Going up each level corresponds to a compression of ϵ

Therefore, after one round of compression, we have:

$$\begin{aligned} x^T L_H &= x^T (\sum_i L_{H_i}) x \\ &= (1 \pm \epsilon') x^T (\sum_i L_{G_i}) x \\ &= (1 \pm \epsilon') x^T L_G x \end{aligned}$$

More generally, in the merge-and-reduce construction, we have that

$$x^T L_H x = (1 \pm \epsilon')^{\log n} x^T L_G x$$

Hence, if we use $\epsilon' = \epsilon / \log n$, then each compressed value takes $\frac{n \log n}{\epsilon'^2} = \frac{n \log^3 n}{\epsilon^2}$ space. Since there are $\log n$ values at a time, this translates to $\frac{n \log^4 n}{\epsilon^2}$ space.

Theorem 5. *$O(n \log^4 n)$ words of space suffice to compute a spectral sparsifier H of G in the insertion-only model.*

References

[AGM12] Ahn, Kook Jin and Guha, Sudipto and McGregor, Andrew. Analyzing graph structure via linear measurements *SODA*, 459–467, 2012