CMPUT 675: Algorithms for Streaming and Big Data **Fall 2019**

Lecture 15 (Oct 28, 2019): Sparsification, Connectivity, and Min-Cut

Lecturer: Mohammad R. Salavatipour *Scribe: Ramin Mousavi*

15.1 Introduction

We continue on the graph streaming algorithms. The material of this lecture is based on a survey by McGregor [M14]. In this lecture, we talk about cut and spectral sparsifiers. These sparsifiers reduce the number of edges of the graph while "approximately" preserving the size of all cuts and the spectrum of the Laplacian matrix. Thus, we can speed up many graph algorithms that their running time depends on the number of edges of the underlying graph at the cost of $1 \pm \epsilon$ approximation of the solution. We use sparsification to reduce the time of answering question like whether a graph is k-edge connected or what is the size of the minimum cut in a graph.

15.2 Graph Sparsification

First, we need some background on spectral graph theory.

Given an $n \times n$ matrix A, we say $v \in \mathbb{R}^n$ is an eigenvector of A if $Av = \lambda v$ for some scalar λ. We call λ an eigenvalue of A.

Given a graph $G = (V, E)$ where $|V| = n$ and $|E| = m$. Let $w \in \mathbb{R}^m_+$ be the vector of weights on the edges. The adjacency matrix A_G is an $n \times n$ matrix defined as follows:

$$(A_G)_{i,j} = \begin{cases} w_{ij}, & \text{if } (i,j) \in E \\ 0, & \text{otherwise.} \end{cases}$$

Define the degree matrix D_G as follows:

$$(D_G)_{i,j} = \begin{cases} \sum\limits_{k:(i,k)\in E}^{n} w_{ik}, & \text{if } i = j \\ 0, & \text{otherwise.} \end{cases}$$

The Laplacian L_G of the graph G is $L_G = D_G - A_G$. Note that we have

$$(L_G)_{i,j} = \begin{cases} \sum\limits_{k:(i,k)\in E}^{n} w_{ik}, & \text{if } i = j \\ -w_{ij}, & \text{if } i \neq j. \end{cases}$$

Claim 1 *For a vector $x \in \mathbb{R}^n$, we have $x^T L_G x = \sum\limits_{(u,v)\in E} w_{uv}(x_u - x_v)^2$.*

Proof. Consider a graph $G_{u,v}$ with two vertices u and v and an edge between them. Then,

$$L_{G_{u,v}} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

Let $x_{u,v}^T = [x_u,\ x_v]$ be the vector in $\mathbb{R}^2$. Then, it is easy to see that $x_{u,v}^T L_{G_{u,v}} x_{u,v} = (x_u - x_v)^2$. Also, it is easy to check that $L_G = \sum_{(u,v\in E)} w_{uv} L_{G_{u,v}}$. Thus, we can write

$$x^T L_G x = \sum_{(u,v)\in E} w_{u,v} x_{u,v}^T L_{G_{u,v}} x_{u,v} = \sum_{(u,v)\in E} w_{uv}(x_u - x_v)^2.$$

■

We say that a matrix M is positive semidefinite (psd) if for all vectors x we have $x^T M x \geq 0$. By Claim 1, L_G is psd for any graph G.

A cut in a graph $G = (V, E)$ is a partition of V into S and $\bar{S} := V \setminus S$. We define the size of a cut $(S, \bar{S})$ to be $E_G(S, \bar{S}) = \sum_{\substack{i\in S \\ j\in \bar{S} \\ (i,j)\in E}} w_{ij}$. Now we are ready to formalize the definition of a cut sparsifier of a graph.

Definition 1 (Cut sparsification) *A subgraph H of G, with possibly different edge weights, is a $(1 \pm \epsilon)$-cut sparsifier if*

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

Spielman and Teng [ST11] introduced the more general notion of spectral sparsification than cut sparsification.

Definition 2 (Spectral sparsification) *A subgraph H of G, with possibly different edge weights, is a $(1 \pm \epsilon)$-spectral sparsifier if*

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

Note that $(1 \pm \epsilon)$-spectral sparsification implies $(1 \pm \epsilon)$-cut sparsification. Because, for any cut $(S, \bar{S})$, let χ_S be the characteristic vector of S. Then, $E_G(S, \bar{S}) = \chi_S^T L_G \chi_S$.

In our construction of spectral sparsifier for graphs in the streaming model, we use the following result as a black box.

Theorem 1 (Batson, Spielman, and Srivastava [BSS12]) *Given $\epsilon > 0$, there is a deterministic algorithm to find $(1 \pm \epsilon)$-spectral sparsifier using $O(\frac{n}{\epsilon^2})$ edges.*

We use the following simple facts in our algorithm. These facts can be driven directly from the definition of spectral sparsifiers.

Claim 2 *Let G_1 and G_2 be two graphs on the same set of vertices, and let H_1 and H_2 be α-spectral sparsifiers for G_1 and G_2, respectively. Then, $H_1 \cup H_2$ is an α-spectral sparsifier for $G_1 \cup G_2$.*

Claim 3 *Let H_1 be an α-spectral sparsifier for G, and let H_2 be a β-spectral sparsifier for H_1. Then, H_2 is an $\alpha \cdot \beta$-spectral sparsifier for G.*

Here is a **"Merge and Reduce"** technique for constructing the spectral sparsifier of a graph in the streaming model where edges only can be inserted and not to be deleted: The algorithm is based on hierarchical partitioning of the stream. Let $G = (V, E)$ be a graph that the edges coming in a stream. Let $n := |V|$, $m := |E|$. We partition the stream of edges into $t = \frac{m}{l}$ segments of length $l := O(\frac{n}{\gamma^2})$, for $\gamma > 0$ which will be determined later. For simplicity, we assume that t is a power of 2. Let G_i^0 be the graph corresponding to the i-th segment of edges. For $j \in \{1, ..., \log t\}$ and $i \in \{1, ..., \frac{t}{2^j}\}$, we define

$$G_i^j := G_{2i-1}^{j-1} \cup G_{2i}^{j-1}.$$

Let A be the $(1 \pm \gamma)$-spectral sparsification algorithm given in Theorem 1. Let $H_i^0 := G_i^0$. Then, inductively define H_i^j as follows:

$$\forall j \geq 1 : H_i^j := A(H_{2i-1}^{j-1} \cup H_{2i}^{j-1}).$$

Note that by Claim 2 and 3, we conclude that $H_1^{\log t}$ is a $(1 \pm \gamma)^{\log t}$-spectral sparsifier for G. Also notice that each time we merge two graphs, we sparsify the merged graph in order to keep the space "small". Setting γ to be $\frac{\epsilon}{\log n}$, we get that $H_1^{\log t}$ is a $(1 \pm \epsilon)$-spectral sparsifier for G.

We can view this procedure as a binary tree. Each node H_{2i-1}^{j-1} (except the root) will be replaced by H_i^j as soon as its sibling, H_{2i}^{j-1}, has been computed. Therefore, in each level of the tree, we only keep two graphs of size $l = O(\frac{n}{\gamma^2})$ (by Theorem 1). Hence, at any given time, the space we use is at most $2 \cdot l \cdot \log t = O(\frac{n \log^3 n}{\epsilon^2})$.

15.3 k-edge Connectivity

A graph G is k-edge connected if each cut $(S, \bar{S})$ has size at least k, i.e., $E_G(S, \bar{S}) \geq k$. In this section, we are interested in answering the question that whether a graph is k-edge connected in the dynamic model where edges might be deleted.

The algorithm relies on a simple fact about connectivity.

Lemma 1 *Let $G = (V, E)$ be a graph, and let F_1 be a spanning forest of G [1]. Inductively, define $F_i = (V, E \setminus \bigcup_{j=1}^{i-1} E(F_j))$ for $2 \leq i \leq k$ be a spanning forest of G that does not have any edges from previous spanning forests. Then, G is k-edge connected if and only if $\bigcup_{i=1}^{k} F_i$ is k-edge connected. We call $\bigcup_{i=1}^{k} F_i$ the **skeleton** of G.*

Proof. Suppose $\bigcup_{i=1}^{k} F_i$ is k-edge connected. Since the edges of F_is are distinct, that implies that every cut of G has size at least k and hence G is k-edge connected.

Now suppose G is k-edge connected. Consider a cut $\delta(S) = \{(u, v) \in E : u \in S,\ v \in \bar{S}\}$. If each F_i has a cut edge in $\delta(S)$, then since F_is are distinct, we conclude that $|\delta(S)| \geq k$. So suppose that F_i does not have any edge in $\delta(S)$. This means that $\delta(S) \setminus \bigcup_{j=1}^{i-1} E(F_j) = \emptyset$ which implies $|\bigcup_{j=1}^{i-1} E(F_j)| = |\delta(S)| \geq k$. Hence, $\bigcup_{i=1}^{k} F_i$ is k-edge connected. ■

Recall that in the algorithm in the last lecture for connectivity, we construct a sketch of a graph which then a spanning forest can be obtained from. Let A be algorithm to get this sketch. The algorithm for k-edge connectivity is as follows:

[1] Here by spanning forest, we mean a subgraph of G that is a forest with minimum number of connected components.

k-edge connectivity

1. In a single pass compute $A^1(G), ..., A^k(G)$, k pair wise independent sketches of G. Note that $A^i(G)$ is the sketch we construct for the connectivity (spanning forest) problem in the last lecture, i.e., having the sketch $A^i(G)$, we can construct a spanning forest for G.
2. In the post processing, construct the following k spanning forests: Let F_1 be the spanning forest obtained from $A^1(G)$. Then, let F_i be the spanning forest obtained from $A^i(G - F_1 - ... - F_{i-1}) = A^i(G) - A^i(F_1) - ... - A^i(F_{i-1})$. In other words, first remove the edges of $F_1, ..., F_{i-1}$ from the sketch $A^i(G)$ and then construct a spanning forest based on the resulting sketch.
3. Return "Yes" if $\bigcup_{i=1}^{k} F_i$ is k-edge connected and "No" otherwise.

Note that by the way of construction, F_is have distinct edges. The correctness of the algorithm follows from Lemma 1.

Recall from the last lecture that each $A^i(G)$ requires $O(n \cdot \text{polylog}\, n)$. We are using k many such sketches; hence, the total space we use is $O(k \cdot n \cdot \text{polylog}\, n)$.

Note that using Karger's minimum cut algorithm, the running time of computing the minimum cut of $\bigcup_{i=1}^{k}$ is $O(k \cdot n \cdot \log k \cdot n)$. If we wanted to compute the minimum cut of G directly, the running time would be $O(n^2 \log n)$ in the dense graphs that we have $\Omega(n^2)$ edges.

15.4 Estimating Min-Cut

In this section, we use the following edge sparsification to estimate the size of the minimum cut in a graph in the streaming model.

Edge sparsification

1. Sample each edge e with probability p_e.
2. Weight each sampled edge e by $\frac{1}{p_e}$.

Note that with the above sparsification, the expected value of the size of each cut will be preserved. Let λ be the size of the minimum cut and let λ_e be the size of the minimum cut that separates the endpoints of e. For some constant c_1, Karger [K94] showed the following result.

Theorem 2 *If $p_e \geq \min\{1, c_1 \lambda^{-1} \epsilon^{-2} \log n\}$ in the above algorithm, then the resulting graph is a $(1 \pm \epsilon)$-cut sparsifier with high probability.*

Fung et al. [FHHP11] strengthened Karger's result by showing that for some constant c_2, the sampling probability could only depend on λ_e, i.e., $p_e \geq \min\{1, c_2 \lambda_e^{-1} \epsilon^{-2} \log n\}$.

Finally, Spielman and Srivistava [SS11] showed that if $p_e \geq \min\{1, c_3 r_e \epsilon^{-2} \log n\}$ where r_e is the effective resistance of edge e and c_3 is a constant, then the resulting graph is $(1 \pm \epsilon)$-spectral sparsifier.

Now, we use Theorem 2 to give an estimation on the min-cut.

Min-cut estimator

1. Let $k = 3c_1\epsilon^{-2}\log n$.
2. Let G_i be the graph obtained from G by including an edge with probability $\frac{1}{2^i}$, and let H_i be a k-skeleton of G_i obtained from k-edge connectivity algorithm, see Lemma 1 for the definition of k-skeleton.
3. Let $j = \min\{i : \ mincut(H_i) < k\}$.
4. Return $2^j \cdot mincut(H_j)$.

Let q be the sampling probability in Theorem 2, i.e., $q = \min\{1, c_1\lambda^{-1}\epsilon^{-2}\log n\}$.

Lemma 2 *With high probability, for $i \leq \lfloor \log \frac{1}{q} \rfloor$, we have $2^i \cdot mincut(H_i)$ is a $(1\pm\epsilon)$ approximation of minimum cut of G.*

Proof. If the $mincut(G_i) < k$, then $mincut(H_i) = mincut(G_i)$. By Theorem 2, we know that G_i is $\frac{1}{2^i}(1 \pm \epsilon)$-cut sparsifier of G (note that in the sampling G_i of G we did not adjust the weight as in the algorithm Edge sparsification and that is why we have the factor $\frac{1}{2^i}$ in the estimation); hence, $2^i \cdot mincut(H_i)$ is $(1 \pm \epsilon)$ approximation of G.

It remains to show that with high probability $mincut(G_i) < k$. Let $i = \lfloor \log \frac{1}{q} \rfloor$, then we have

$$\mathbb{E}[mincut(G_i)] \leq \frac{1}{2^i}\lambda \leq 2q\lambda \leq 2c_1\epsilon^{-2}\log n.$$

Using Chernoff bound (note that we sample each edge independently), we get that with high probability $mincut(G_i) \leq 3c_1\epsilon^{-2}\log n = k$, as desired. ■

By the above lemma, we conclude that the Min-Cut estimator terminates by sampling at most $\log \frac{1}{q}$ many subgraphs of G. For each subgraph, we are computing a k-skeleton which uses $O(k \cdot n \cdot \text{polylog}\, n) = O(n \cdot \text{polylog}\, n)$; hence, we the total space used in the algorithm is $O(n\,\text{polylog}\, n)$.

Note that we used k-skeleton because the running time of computing the minimum cut on a sparse graph is much less than a dense graph, please see the discussion at the end of Section 15.3.

References

ST11 D. A. Spielman and S.-H. Teng, Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981-1025, 2011.

BSS12 J. D. Batson, D. A. Spielman, and N. Srivastava, Twice-Ramanujan sparsifier. *SIAM J. Comput.*, 41(6):1704-1721, 2012.

M14 A. McGregor, Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43.1 (2014): 9-20.

FHHP11 W. S. Fung, R. Hariharan, N. J. A. Harvey, and D. Panigrahi, A general framework for graph sparsification. *In ACM Symposium on Theory of Computing*, pages 71-80, 2011.

K94 D. R. Karger, Random sampling in cut, flow, and network design problems. *In ACM Symposium on Theory of Computing*, pages 648-657, 1994.

SS11 D. A. Spielman and N. Srivastava, Graph sparsification by effective resistances. *SIAM J. Comput*, 40(6):1913-1926, 2011