CS 15-859: Algorithms for Big Data Fall 2022

Lecture Graph Streaming Part 2 — December 18, 2022

Prof. David Woodruff *Scribe: Sixiang Gao*

1 Insertion-Deletion Graph Streams

Previously we discussed algorithms for the insertion-only graph streaming model. We now consider graph streams where undirected edges can be either added or deleted. Formally, the input is a sequence of updates

$$S = \langle a_1, a_2, \cdots \rangle, a_i = (e_i, \Delta_i),$$

where $\Delta_i \in \{-1, 1\}$ and at the end of a stream we have for any e, $f_e = \sum_{i:e_i=e} \Delta_i \in \{0, 1\}$ representing the multiplicity of edge e in our graph. That is to say, we have a simple graph at the end of the stream.

A generic approach to problems in this insertion-deletion streaming model is the use of **linear sketch** matrices. For example, let $\mathbf{f} \in \{0,1\}^{\binom{n}{2}}$ denote the vector of indicators of edges for the stream. We maintain a sketch matrix A, drawn from some distribution of matrices $\mathcal{D}_{s\times\binom{n}{2}}$. Along the stream we store $\mathbf{f}' := A\mathbf{f}$, a shorter vector than $\mathbf{f}$, and update it as

$$\mathbf{f}' \leftarrow \mathbf{f}' + \Delta \cdot A(\mathbf{i}^e)$$

where $\mathbf{i}^e$ is the vector whose entry is 1 at index e and 0 elsewhere. By linearity (a.k.a. homomorphism of vector spaces), we always store $A(\mathbf{f})$ at any point of the stream. Generally, the goal of using sketches is to

- Infer nontrivial information about the original vector,
- And use lower space than storing the entire vector itself.

Some graph sketches use a vector of sketches whose concatenation is the overall sketch. Specifically, we may have

$$Af = A1f_1 \circ A_2 f_2 \circ \cdots \circ A_n f_n,$$

where each f_i is a length $\binom{n}{2}$ vector sketched by A_i. Next, we look at specific examples of graph streaming problems and their sketches.

1.1 Connectivity

The problem is to determine if a graph has just one connected component, and we may reduce this problem to the task of finding a spanning forest of a graph. To do this, we first describe a generic spanning forest algorithm and adapt it to our case of sketching algorithms.

To output a spanning forest of a graph (not from streaming), we can engage in a $O(\log n)$ stage process. At first, we initialize a set of n supernodes as singleton sets of the nodes. At each stage, for each supernode, we collapse an edge between this supernode and another supernode (if exists). After $O(\log n)$ stages, we then have found a spanning forest.

Definition. Given unweighted graph $G = (V, E)$, define $n \times \binom{n}{2}$ matrix A_G such that

$$(A_G)_{i,(j,k)} = \begin{cases} 1, & i = j, (v_j, v_k) \in E \\ -1, & j = k, (v_j, v_k) \in E \\ 0, & \text{otherwise} \end{cases}$$

Immediately from definition we have

Lemma 1. *Let $E_S = E(S, V \setminus S)$ be the set of edges across the cut $(S, V \setminus S)$. Then, $|E_S| = l_0(x)$, where $x = \sum_{v_i \in S} a_i$. Further, for $x \in \{-1, 0, 1\}^{\binom{n}{2}}$, $|x_{(j,k)}| = 1$ iff $(v_j, v_k) \in S$.*

Now we build a sketched based algorithm for this problem. For the t stages of coalescing super nodes, we associate $t = O(\log n)$ sketch matrices $S_1, \cdots, S_t$. For each vector $a_i = (A_G)_i$, we sketch it t times with the sketch matrices. Then in total we use $O(nt \log^2 n) = \widetilde{O}(n)$ space.

In the first stage, to sample an edge incident on v_i with constant success probability, we use sketch $S_1(a_i)$ and apply l_0 sampling technique from [4]. At the next step, we use sketch $S_2(a_i) + S_2(a_j)$ to l_0 sample an edge between supernodes. Note that reusing S_1 would not work here. It is not difficult to see correctness of this algorithm and the fact that we need only $t = O(\log n)$ many sketches.

1.2 Bipartite

To check whether or not a graph is bipartite, it suffices to show the number of connected components in the double cover of G is exactly twice the number of connected components in G. The double cove of a graph G consists of double the number of vertices labeled u_1, u_2 for each u. Connect (u_1, v_2) and u_2, v_1 iff (u, v) is an edge in the original graph.

Now, it is clear that the algorithm to find a spanning forest from above can be used to find the number of connected components, which also determines if a graph is bipartite.

1.3 k-Connectivity

The above idea can be extended more generally to the k-connectivity problem, where the graph is said to be k-connected unless there is a cut with less than k edges. We need the following lemma.

Lemma 2. *Given a graph $G = (V, E)$, for $i \in [k]$, let F_i be a spanning forest of $(V, E \setminus \bigcup_{j<i} F_j)$. Then $(V, F_1 \cup F_2 \cup \cdots \cup F_k)$ is k-edge-connected iff $G = (V, E)$ is at least $k - edge - connected$.*

Proof. The forward direction is clear. So we assume G is k-edge-connected. Let $E' = F_1 \cup F_2 \cup \cdots \cup F_k$. Consider a cut $(S, V \setminus S)$. Let $E_S \subseteq E$ be the set of edges crossing this cut. Similarly, let $E'_S \subseteq E'$ be the set of edges in E' that cross this cut. It suffices to show $|E'_S| \geq k$. Suppose there exists some i such that $F_i \cap E'_S = \varnothing$ (otherwise we are done since the F_i's are pairwise edge-disjoint). Then $E_S \cap \bigcup_{j<i} F_j = E_S$, which means $|E'_S| = |E_S| \geq k$. ■

If disjoint F_i has the property that $\bigcup F_i$ contains at least $\min(k, |E_S|)$ edges across a cut S, then we call this set of edges a k-skeleton. Using the spanning forest as k-skeleton gives us an easy design of a k-pass algorithm. However, a one-pass algorithm also exists for the k-connectivity problem.

To do this, start with k independent instantiations $I_1, \cdots, I_k$ of the spanning forest algorithm. For each $i \in [k]$, use the sketches for I_i to find a spanning forest F_i of $(V, E \setminus (F_1 \cup \cdots F_{i-1}))$. Update sketches for $I_{i+1}, \cdots, I_k$ by delting all edges from F_i. Since instantiations are independent, updating them as claimed is valid. Summarizing the above, we have a one-pass $\widetilde{O}(kn)$-space algorithm for k-connectivity.

1.4 Min-Cut

We make a few definitions.

Definition. A weighted graph $H = (V, E_0, w_0)$ is a sparsifier for a graph $G = (V, E, w)$ if for every cut, the cut value in H is within a $(1 \pm \epsilon)$ factor of the cut value in G.

Definition. A generic sparsification algorithm samples each edge e with probability p_e and weigths each sampled edge e by $1/p_e$.

A useful result due to Karger [6] is as follows.

Lemma 3. *For some constant c_1, if $p_e \geq q := \min\{1, c_1 \lambda^{-1} \epsilon^{-2} \log n\}$, where λ is the size of the minimum cut of the graph, then the resulting graph is a cut sparsifier with high probability.*

We now show how to estimate the minimum cut λ of a graph stream. To do this, we use the algorithm for constructing k-skeletons from k-connectivity. In addition, we make use of Karger's lemma.

Specifically, let G_i be the subsampled graph that induces each edge with probability $1/2^i$. Let $H_i = \text{skeleton}_k(G_i)$ be a k-skeleton of G_i, where $k = 3c_1\epsilon^{-2} \log n$. Then, for

$$j = \min\{i : \mathit{mincut}(H_i) < k\},$$

we claim

$$2^j \mathit{mincut}(H_j) = (1 \pm \epsilon)\lambda.$$

For $i \leq \lfloor \log_2 1/q \rfloor$, Karger's lemma implies that all cuts are approximately preserved, and

$$2^i \cdot \mathit{mincut}(H_i) = (1 \pm \epsilon)\mathit{mincut}(G_i)$$

Meanwhile, for $i = \lfloor \log_2 1/q \rfloor$, $\mathbb{E}[\mathit{mincut}(H_i)] \leq 2^{-1}\lambda \leq 2q\lambda \leq 2c_1\epsilon^{-2} \log n$. Hence, by Chernoff, $\mathit{mincut}(H_i) < k$ with high probability and thus $j \leq \log_n 1/q$ with high probability. Our claim then follows.

1.5 Sparsification

To construct a sparsifier, the basic idea is to sample edges with probability $q_e = \min\{1, t/\lambda_e\}$. For fuller understanding of this technique, the validity of this construction relies on results from [5] [7] [1].

2 Sliding Windows

We now consider a new graph streaming model that is distinct from the insertion model or the insertion-deletion model. Here, we consider an infinite stream of added edges $\langle e_1, e_2, \cdots \rangle$. Howeve,r at time t we only consider the graph whose edge set consists of the last w edges,

$$\{e_t - w + 1, \cdots, e_t\}.$$

We call these the active edges and only consider the case when $w \geq n$.

2.1 Connectivity

Definition. The stored edges F is said to have the **Recent Edges Property** if for every cut $(U, V \setminus U)$, the stored edges F contains the most recent $\min(k, \lambda(U))$ edges across the cut where $\lambda(U)$ denotes the total number of edges from $\{e_1, \cdots, e_t\}$ that cross the cut.

Lemma 4. *To test k-connectiity, it is sufficient to maintain a set of edges F along with time of arrival toe(e) for $e \in F$, where F satisfies the Recdent Edges Property.*

Proof. We can tell if the graph on the active edges is k-connected by checking if F would be k-connected once we remove all edges $e \in F$ where $\text{toe}(e) \leq t - L$, where L is the size of the active set. This is because if there are at least k edges among the last L edges across a cut, F will include the k most recent of them. ■

Then we may have the following simple algorithm that maintains a set with the above property. We maintain k disjoint sets of edges $F_1, F_2, \cdots, F_k$ where each F_i is acyclic. Initially, the k sets are all empty and on seeing edge e in the stream, we update the sets as follows.

- Define the sequence $f_0, f_1, \cdots$ where $f_0 = \{e\}$ and for each $i \geq 1$, f_i consists of the oldest edge in a cycle in $F_i \cup f_{i-1}$ if such a cycle exists and $\varnothing$ otherwise. Since each F_i acyclic, there will be at most one cycle in each $F_i cup f_{i-1}$.
- For $i \in [k]$, $F_i := (F_i \cup f_{i-1}) \setminus f_i$.

It then remains to show that the unions of the F_i's indeed has the recent edges property. To do this, fix some $i \in [k]$ and some cut $(U, V \setminus U)$. Observe that youngest edge $y \in F_i$ crossing the cut is never removed from F_i since its removal would require it to be the oldes edge in some cycle C, which cannot happen since there must be an even number of edges in C that cross the cut (so there's at least another edge, which is older, that crosses the cut). It follows that F_1 contains the youngest edge crossing any cut, and by induction the ith youngest edge crossing any cut is contained in $\bigcup_{j \leq i} F_i$. This is true because this edge cannot leave $\bigcup_{j \leq i} F_j$. That is, for the ith youngest edge to leave F_i, there would have to be a younger crossing edge in F_i, but, inductively any such edge is contained in $\bigcup_{j<i} F_j$.

2.2 Matchings

We focus on the unweighted case and get a $(3 + \epsilon)$ approximation. The technique makes use of smooth histograms from [2] and is described in detail from [3].

3 Conclusion

A large body of research has been devoted to graph streaming algorithms in the insertion-only model, the insertion-deletion model, as well as the sliding window model. However, there are some other directions that may deserve more attention. Here is a list.

- Directed graph streams. So far we have been operating solely in the undirected case. Connectivity, for example, in the directed case, may be fundamentally different from connectivity in the undirected case.
- Communication complexity. Suppose a graph in its adjacency matrix representation is split among n parties, then we may investigate what communication problems naturally have low communication complexity.
- Stream ordering. The idea is we wanted our algorithms to work for the worst case inputs in the worst case order, but if we relax the assumption that the order of the inputs is worst case and instead have it randomized, we may find better streaming algorithms for the same problems that were considered difficult.

References

[1] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. “Graph sketches: sparsification, spanners, and subgraphs”. In: *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*. 2012, pp. 5–14.

[2] Vladimir Braverman and Rafail Ostrovsky. “Smooth histograms for sliding windows”. In: *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS’07)*. IEEE. 2007, pp. 283–293.

[3] Michael S Crouch, Andrew McGregor, and Daniel Stubbs. “Dynamic graphs in the sliding-window model”. In: *European Symposium on Algorithms*. Springer. 2013, pp. 337–348.

[4] Gereon Frahling, Piotr Indyk, and Christian Sohler. “Sampling in dynamic data streams and applications”. In: *Proceedings of the twenty-first annual symposium on Computational geometry*. 2005, pp. 142–149.

[5] Wai Shing Fung et al. “A general framework for graph sparsification”. In: *Proceedings of the forty-third annual ACM symposium on Theory of computing*. 2011, pp. 71–80.

[6] David R Karger. “Random sampling in cut, flow, and network design problems”. In: *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*. 1994, pp. 648–657.

[7] Daniel A Spielman and Nikhil Srivastava. “Graph sparsification by effective resistances”. In: *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 2008, pp. 563–568.