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

Lecture 13 (Oct 21, 2019): Graph Stream Algorithms

Lecturer: Mohammad R. Salavatipour *Scribe: Brandon Fuller*

13.1 Graph Streams

In the graph streaming model, our tokens consist of edges of the graph. We will only be dealing with undirected and unweighted graphs. The number of vertices of the graph is $n = |V|$ and the number of edges is $m = |E|$. In these models, m is generally much larger than n. Most algorithms that we will deal with will use $\Omega(n)$ space. Below is a list of some common problems.

- Connectivity
- Distance Queries or Shortest path
- Triangle Counting
- Bipartite
- Cuts / Partitions of Vertices

13.1.1 Connectivity

We start with an easy graph streaming problem which is connectivity. Precisely we want to know whether the given graph is connected. Unlike other problems that we have solved, this is just a yes or no answer. The basic idea is to maintain a collection of trees that form a spanning forest as the edges come, we merge to components if the end-points of the new edge belong to different components. We use a union-find (or disjoint-set) data structure which maintains what elements are in the same components. If you ever have one component, then you know it is a connected graph. This is basically like maintaining a spanning forest at any point in time.

Connectivity

```
1. $UF \leftarrow$ union-find with $n$ elements for each vertex of $G$
2. $c \leftarrow 0$
3. While there is still another token $(u, v)$ do:
     • If $u$ not connected to $v$ in $UF$:
         – Connect $u$ and $v$ in $UF$
         – $c \leftarrow c + 1$
4. Return yes if $c = n - 1$, otherwise return no
```

Clearly this answers the question exactly as we can add at most $n - 1$ edges, and once we have done so we must have connected the whole graph. Since union-find uses $O(n)$ entries (or words) of space we know that this algorithm uses $O(n \log n)$ bits. Depending on which union-find we use, the time complexity can be $O(n \log n)$, or $O(n\alpha(n))$ (where α is the inverse Ackermann function, which is essentially constant).

13.1.2 Shortest Path Estimation

We now will present an algorithm and analysis to estimate the shortest path from a stream of edges. Let $d_G(u,v)$ denote the length of the shortest u, v path in the graph G. We say a subgraph H, $E(H) \subseteq E(G)$ is a **t-spanner** ($t \geq 1$) subgraph if $\forall u, v \in V$, $d_G(u,v) \leq d_H(u,v) \leq t d_G(u,v)$. Our algorithm is going to maintain a t-spanner subgraph which we will use to estimate the shortest paths in the original graph.

Shortest Path Estimator

```
1. $V(H) \leftarrow V$, $E(H) \leftarrow \emptyset$
2. While there is still another token $(u,v)$ do:
    • If $d_H(u,v) > t$ then:
        – $E(H) \leftarrow E(H) \cup \{(u,v)\}$
3. For each $u, v$ return $d_H(u,v)$ as an estimate for $d_G(u,v)$
```

We will first prove that this graph H is indeed a t-spanner and then we will show the space complexity for maintaining this subgraph.

Lemma 1 *The output of the Shortest Path Estimator Algorithm is a t-spanner subgraph of G.*

Proof. Suppose that $u = u_0, u_1, \ldots, u_k = v$ is a shortest path from u to v in G. When edge (u_i, u_{i+1}) is given from the stream, either $d_H(u_i, u_{i+1}) > t$ in which case we add it, or $d_H(u_i, u_{i+1}) \leq t$ in which case we do not add it. In either case, after that edge is processed, $d_H(u_i, u_{i+1}) \leq t$. Thus after the algorithm

$$d_H(u,v) \leq \sum_{i=0}^{k-1} d_H(u_i, u_{i+1}) \leq tk = t d_G(u,v)$$

Also, it is clear that $d_G(u,v) \leq d_H(u,v)$ since H is a subgraph. ■

The **girth** of a graph G is the size of the shortest cycle in G. We use the following lemma and theorem to prove the space complexity.

Lemma 2 *After the algorithm has finished, the girth of H is at least $t + 2$.*

Proof. Suppose towards a contradiction that there is a cycle $C = v_1 v_2 \ldots v_k$ that has size $k \leq t + 1$. Suppose that the edge (v_i, v_{i+1}) was the last edge of C which was added to H. Then at the point before it was added, $d_H(v_i, v_{i+1}) \leq k - 1 \leq t$ which contradicts adding it. ■

Theorem 1 *A graph with girth $2t + 1$ has at most $O(n^{1+\frac{1}{t}})$ edges.*

Proof. Suppose G is a $2t + 1$-girth graph and let $d = \frac{2m}{n}$ be the average degree of vertices. Iteratively delete vertices with degree strictly less than $\frac{d}{2}$. Let F be the resulting graph of this process. We see that if we deleted

n vertices, this means we would have to have deleted strictly less than $n\frac{d}{2} = n\frac{2m}{n} = m$ edges but this would be a contradiction. Thus, F is non-empty and has min-degree $\delta \geq \frac{d}{2}$. If we ran a BFS from a node in F up to t levels, we must have no cycles and thus $(\frac{d}{2} - 1)^t \leq n$. Which, after substituting $\frac{2m}{n}$ in for d and solving for m, we get $m \leq n^{1+\frac{1}{t}} + n$. ■

We now present the following theorem (without proof) which we use to explain our space complexity.

Theorem 2 *There are graphs with girth $\Omega(t)$ and have $\Omega(n^{t+\frac{1}{t}})$ edges.*

Such a graph would have $2^{\Omega(n^{1+\frac{1}{t}})}$ subgraphs. Our estimator would have to be able to distinguish between them and thus we need $\log(2^{\Omega(n^{1+\frac{1}{t}})}) = \Omega(n^{1+\frac{1}{t}})$ bits of space for a t-estimator.

13.1.3 Triangle Counting

A **triangle** in a graph is any three vertices that are all adjacent. In this section, our goal is to estimate T, the number of triangles in a graph. An example would be, given a friendship network, how often do two friends have a common friend.

Let x be a vector of dimension $\binom{n}{3}$ where x_S is the number of edges among the vertices in the set S, where $|S| = 3$. Then T is just the number of coordinates in x with $x_S = 3$. We note that $F_p(x) = \sum_S x_S^p = 1^pT_1 + 2^pT_2 + 3^pT_3$ where T_i is the number of elements in x with value i.

Claim 1 $T = F_0 - 1.5F_1 + 0.5F_2$.

Proof. For a set S, if $x_S = 0$ then the amount that it contributes to the RHS is 0. If $x_S = 1$ then the amount that it contributes is $1 - 1.5 \cdot 1 + 0.5 \cdot 1 = 0$. Similarly if $x_S = 2$ then it contributes $1 - 1.5 \cdot 2 + 0.5 \cdot 2^2 = 0$. Finally, if $x_S = 3$ then it contributes $1 - 1.5 \cdot 3 + 0.5 \cdot 3^2 = 1$. ■

Suppose we have γ-estimators for F_0, F_1, F_2, say $\hat{F}_0, \hat{F}_1, \hat{F}_2$. Specifically, $|F_i - \hat{F}_i| \leq \gamma F_i$. Then if we have the estimator $\hat{T} = \hat{F}_0 - 1.5\hat{F}_1 + 0.5\hat{F}_2$.

Claim 2 $\max(F_0, \frac{F_2}{9}) \leq F_1 < mn$

Proof. Obviously, $F_0 \leq F_1$ so we need to show $\frac{F_2}{9} \leq F_1$. This can be seen since the largest possible value for each element in the vector x is 3. $F_1 < mn$ since each edge contributes to less than n elements of x. ■

This finally gives $|\hat{T} - T| \leq \gamma(F_0 - 1.5F_1 + 0.5F_2) \leq 4\gamma mn$. If we want an ϵ-estimator then we need $\frac{4\gamma mn}{T} \leq \epsilon$ which means $\gamma \approx \frac{T\epsilon}{4mn}$ which only really works if T is large. This would also require $O(\gamma^{-2}\log n) = O((\frac{mn}{\epsilon t})^2 \log n)$ space where t is a lower bound for the number of triangles.

References

BKS02 Z. Bar-Yossef, R. Kumar, and D. Sivakumar, Reductions in streaming algorithms, with an application to counting triangles in graphs. *In Proc. of the 2002 Annual ACM-SIAM Symp. on Discrete Algorithms*, 623-632, 2002.

FKMSZ05 J. Feigenbaum, S. Kannan, A. McGregor, S.Suri, and J. Zhang, On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3): 207-216, 2005.

M14 A. McGregor, Graph stream algorithms: A survey. *SIGMOD Record*, 43: 9-20, 2014.