6.897: CSCI8980 Algorithmic Techniques for Big Data October 4, 2013

Lecture 9

Dr. Barna Saha *Scribe: Shikher Sitoke*

Overview

So far we have studied streaming algorithms where the goal has been to obtain $(1+\epsilon)$-approximate solution in polylogarithmic space (or at least sublinear space in input size n and m) in a single pass. We now introduce graph streaming algorithms where the stream comes in the form of edges of the graph. For graph streaming we use n to denote the number of vertices and m to denote the number of edges. For graphs unfortunately, for all interesting estimates maintaining them in streaming manner requires space that is linear in n. Therefore, we aim towards achieving space complexity that is instead sublinear in m. We allow $O(n)$ space and may allow multiple passes over the graph stream. This is called *semi-streaming*. We start off by discussing two semi-streaming algorithms to count the number of connected components and to determine if a graph is k-edge connected. Then we define α-spanner which approximates graph distance is small space and show how to construct α-spanner in the semi-streaming setting.

1 Introduction to Semi-Streaming Algorithms

For a given graph $G = (V, E)$ we may want to evaluate a lot of interesting properties − number of connected components, connectivity, etc. It is infeasible however to use algorithms which require space sublinear in the number of nodes V. Since $|E|$ could be quadratic in $|V|$, we focus our attention to achieve space restrictions that is sublinear in the number of edges E. Such algorithms are called *semi-streaming algorithms*.

1.1 Number of Connected Components

Our goal is to evaluate the number of connected components for a given graph $G = (V, E)$. We do so by constructing a less dense graph F, which is a spanning forest with the same number of connected components. A connected component of an undirected graph is a subgraph in which any two vertices within the subgraph are connected to each other but disconnected from vertices of another subgraph in the graph. The algorithm for such a construction can be given as:

The intuition with this procedure is that joining two nodes u and v which are already connected would only create a cycle which does not affect the number of disjoint subgraphs we have. The spanning forest F has at most $(n-1)$ edges and we can count the number of connected components in $O(n \log n)$ space.

Algorithm 1 Algorithm to construct F from G

```
Initialize F ← φ
for each edge E(u, v) in G do
  if u and v are not connected in F then
    F ← F ∪ E(u, v)
  end if
end for
return F
```

1.2 k-Edge Connectivity

A graph is said to be k-edge connected if even after removing any $k-1$ edges, it remains connected. Removing the k^{th} edge may make the graph disconnected.

Procedure

We describe an algorithm to come up with a sparser graph H with the same edge connectivity. We start off by maintaining k forests: $F_1, F_2, ..., F_k$. Now we stream one edge at a time from G and for each edge $E(u, v)$, we try to find the smallest forest in which u and v are not connected. If we find such a forest F_i, we add $E(u, v)$ to the forest. If no such forest exists, we ignore the edge. The intuition here is that this edge, even if removed would not affect the connectivity of the graph since u and v were connected in every forest we came across.

Analysis

Formally, for a given graph $G = (V, E)$, we compute $H = F_1 \cup F_2 \cup ... \cup F_k$. H is a subgraph of G and as a result $E(H) \subseteq E(G)$. Our claim is that if G is k-connected then H is also k-connected. If H is not k-connected, G will also not be k-connected.

Consider any cut $(S, V \backslash S)$ which divides the graph into S and its complement $V \backslash S$. Let $E_G(S)$ be the set of edges that cross this cut in G. We consider a similar cut in H that creates a set of edges $E_H(S)$.

Now clearly, since $H \subset G$, $|E_G(S)| \geq |E_H(S)|$

Suppose there exists an edge $(u, v) \in E_G(S)$ which is not present in $E_H(S)$.

So, $|E_G(S)| > |E_H(S)|$. This edge appeared in our stream but we ignored it as every forest $F_1, F_2, ..., F_k$ had a connection from u to v. The two vertices are connected in each of the k disjoint forests. Therefore, in each F_i, $i = 1, 2, .., k$, there is at least one edge between the cut S and V/S

Since the forests are disjoint, $E_H(S) \geq k$ Combining both our arguments, we get

$E_H(S) \geq \min(|E_G(S)|, k)$ Thus G is k connected then for all cuts of G, the number of edges crossing the cut is at least k and so is for H by the above equation. On the otherhand, if G is not k connected then there exists at least one cut for which the number of edges crossing the cut in G is strictly less than k. For this same cut, the number of edges in H crossing the cut will be less than k as H is a subgraph of G. So H will also not be k-edge connected.

2 Spanners

Suppose we need to find out the distance $d_G(u, v)$ between two nodes u and v in a dense graph $G = (V, E)$. We try to give an approximate answer $d_H(u, v)$ by creating a new graph H such that the result returned is within an $\alpha-$ span $\alpha > 0$ of the actual distance.

Formally, a graph H is called an α-spanner of G if

$$d_G(u, v) \leq d_H(u, v) \leq \alpha.d_G(u, v)$$

Algorithm

We stream through all the edges in G and check if the distance between the nodes u and v in the edge encountered $E(u, v)$ is greater than a threshold $2t$. If it is, we add the edge $E(u, v)$ to H else ignore it. Thus, we ensure that any two edges in H are at a distance $\leq 2t$.

Algorithm 2 Algorithm to construct $\alpha - spanner$ of G

```
Initialize H ← φ
t = (α+1)/2
for each edge E(u, v) in G do
  if d_G(u, v) ≥ 2t then
    H ← H ∪ E(u, v)
  end if
end for
return H
```

Analysis

Consider the case where u and v are connected in a path which has a distance $\geq 2t$. i.e $d_H(u, v) \geq 2t$. Our algorithm makes us join u and v with a direct path and thus a cycle in formed in our graph. The length of such a cycle is thus $\geq (2t+1)$. Hence the distance between u and v can only increase at most by a factor of $2t - 1$. Also, all the cycles have length at least $2t + 1$ in H.

We now prove the following theorem

Theorem 1. *If all cycles in H have length $\geq (2t + 1)$, H has $O(n^{1+1/t})$ edges.*

Proof. Total degree of the graph is given by $d = 2 * |E(G)|$

Average degree, $d_{AVG} = 2m/n$, n being the number of edges.

Now, we form a set of edges J by removing all vertices which have a degree $< m/(d_{AVG}/2) = n$. Obviously, J is a non-empty set.

We perform a $BreadthFirstSearch$ from an arbitrary node a.

At the first depth of the BFS, there are atmost $(d_{AVG}/2)^t$ edges.

At the second level, each node then connects to atmost $(\frac{d_{AVG}}{2} - 1)$ nodes since its connection with a has already been taken into account.

So, at the t^{th} level, there are atmost $(\frac{d_{AVG}}{2} - 1)^t$ edges.

Thus,

$$
\begin{aligned}
(\tfrac{d_{AVG}}{2} - 1)^t \leq |J| \leq n & \\
n \quad \geq \quad & (\frac{d_{AVG}}{2} - 1)^t \\
\geq \quad & (\frac{2m}{2n} - 1)^t \\
\geq \quad & (\frac{m}{n} - 1)^t \\
n^{1/t} \quad \geq \quad & \frac{m}{n} - 1 \\
n^{1/t} + 1 \quad \geq \quad & \frac{m}{n} \\
m \quad \leq \quad & n + n^{1+1/t}
\end{aligned}
$$

Thus, all edges in H can be stored in $O(n^{1+1/t})$ □

References

[1] Feigenbaum, Joan and Kannan, Sampath and McGregor, Andrew and Suri, Siddharth and Zhang, Jian, Graph distances in the streaming model: the value of space, SODA '05, 745–754.