CS 395T: Sublinear Algorithms Fall 2016

Lecture 9 — Sept 22, 2016

Prof. Eric Price *Scribe: David Liau*

1 Overview

In this lecture we would like to discuss the problem of graph sketching, specifically the use of algorithms that utilized limited number of linear measurements on the graph to determine properties of it. The main algorithm to be learned in the course was introduced by Ahn, Guha, and McGregor in 2012 [AGM12]. To get an intuition on how to deal with graph sketching problem, topics will be covered as follows.

1. Define Graph Sketching problem
2. Find a random edge across a cut on the graph
3. Find the number of edges in a cut
4. Find a spanning forest
5. Test for properties

For detail in l_0 sampling algorithm, the basic paper provided are:

1. Review of Count-Sketch algorithm originates in a paper by Charikar, Chen, and Farach-Colton. [CCF02]
2. Finishing an improved version of Count-Sktech introduced by Minton and Price. [MP14]

Many of the materials covered are from the surveys provided by Horlescu and McGregor[SIG14].

2 Preliminaries for Graph Sketching

A graph can be formally defined as a set of pairs (V, E), where

- V is a nonemtpy set defined by all the vertices in a graph
- E is a collection of two-element subsets of V, which represents edges inside a graph

A cut C is defined as a partition of the vertices of a graph into two disjoint subsets.(ie.,S, S^C). One of the ways which we are familiar to is representing the graph G explicitly by the associating adjacency matrix. However, to ease up the discuss today we would use incidence matrix instead.

Given a undirected graph with n vertices, an incidence matrix Z is a $R^{m\times n}$ matrix where $m = C_2^n$. Edges are associate to the row elements of the incidence matrix. To be more precise, given an edge e across vertices a and b, an element of some matrix M is defined as,

$$M_{e,w} = \begin{cases} 1 & \text{if } w = a\,, \\ -1 & \text{if } w = b\,, \\ 0 & otherwise\,. \end{cases} \tag{1}$$

Define matrix $D_E \in R^{m\times m}$ to be diagonal and,

$$\Big(D_E\Big)_e = \begin{cases} 1 & \text{if } e \in E\,, \\ 0 & otherwise\,. \end{cases} \tag{2}$$

The matrix $D_E \cdot M$ would give us the desired incidence matrix.

Let $A \in R^{s\times m}$, $s \leq m$, be a linear sketch. The product of $A \cdot D_E \cdot M$ is the information to be stored. Now given the stored information, we would like complete the following tasks,

- Find spanning tree or forest
- Test whether the graph is bipartite or not

3 Application of l_0 Sampling Algorithm

For instance, we can design A to give answer to

- a random edge crossing some cut $C = (S, S^c)$ on the graph
- the number of edges crossing some cut $C = (S, S^c)$ on the graph

To solve the problem, recall the l_0 sampling problem from which the following lemma is provided.

Lemma 1. *There exist a random matrix $Q \in R^{d\times m}$ with $d = O(log^2 m)$ such that for any vector $r \in R^m$, Qr will be some support of r with high probability.*

Proof. The proof is straight forward based on the previous results from class. One can check for fundamental results and details about the problem. □

To sample a random edge crossing some cut $C = (S, S^c)$ on the graph we can choose the matrix A to be the l_0 sampling matrix. To calculate the number of edges crossing some cut C on the graph, we should choose the matrix A to be the l_2 sampling matrix given that the summation over a subset of vertices in an incidence matrix give us the number of edges crossing the cut.

4 Spanning Forest

Given the results from above, we are capable of sampling a random edge crossing given cuts. Now let us try to utilize this tool to find a spanning tree/forest on the graph.

Naive Method —First Try

1. Choose a random cut S_0 where S_0 contains only one vertex u.
2. Use the algorithm above to sample a edge on the cut, which would give us the vertex v on the other end of the edge.
3. Update S_n as $S_n = S_{n-1} \cup \{v\}$, back to the previous step.

If the algorithm worked, how about repeatedly sample the same cut and getting all the edges crossing the cut in the end? The methods are tempting at the first place, but actually it some flaws in it. Thus they do not work.

1. Loss of Independence: The first issue is the simple observation that if we repeatedly update and query S, the random neighbors returned will not be independent. More precisely, it will depend on the matrix A chosen at the first place.
2. Sketch Updates must be Non-Adaptive: The second issue is more insidious. Suppose we query the edge crossing cut C and are returned node u. Can we somehow get the sketch S^c to yield an additional neighbor of v? One idea would be to update S^c by removing u from the neighborhood of v and then query the updated sketch of S^c. However, this clearly does not work because otherwise we could repeat the process multiple times and return all neighbors of v from the $O(log^2 nlog\delta^{-1})$ size sketch! The issue is that we may not adaptively update the data being sketched based on the sketch itself.[AGM12]

The correct algorithm is illustrated as figure 1 and summarized as below:

1. Pick a l_0 sampling matrix A
2. Sample the edge from all vertices using the l_0 sampling algorithm. The algorithm will return a neighbor(if any) for each vertex. Make the connected vertices a set which also creates a cut and a subgraph associate to it.
3. Pick another l_0 sampling matrix A ▷ For Independence
4. Sample the edge from each cut associates to the sets.
5. If any of the resulting subgraphs is connected, group them to form a new set.
6. Return to step 3 until there is only one component to deal with.

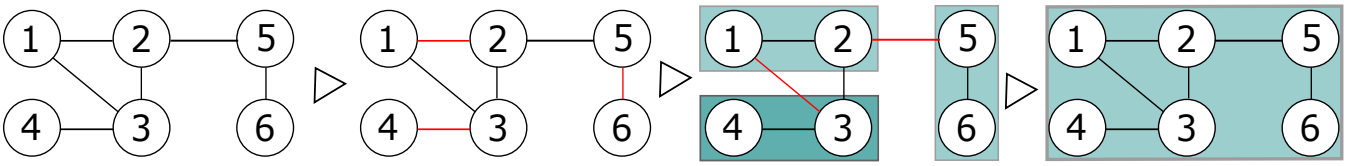

Figure 1: The algorithm stops since the subgraph is actually the whole graph.

5 Bipartitness problem

First of all, lets review the basic definition of a bipartite graph.

Definition. *A graph is bipartite if we can divide its vertices into two sets such that every edge connects from a set to another.*

Or in other words, the vertices in each set is disconnected from one another.

Now the problem is that we have to find way to test bipartitness of the graph using the l_0 sampling algorithm. This can actually be done by a special construction of a new graph G' from the original graph G. The construction is stated as below:

1. For every edge $e = (u, v)$, duplicate vertices to have $u^{(1)}, u^{(2)}, v^{(1)}, v^{(2)}$.
2. Connect $u^{(1)}, v^{(2)}$ and $u^{(1)}, v^{(2)}$ to have two edges. This step would create a bipartite double cover from the original graph.

The relationship between bipartite double cover and the original graph would give us the following lemma:

Lemma 2. *The original graph G is bipartite if and only if the bipartite double cover G' has twice more connected component than G.*

Proof. We know that G is bipartite if and only if G contains no odd length cycle. Let u be a node in some odd cycle of G(assume G is not bipartite). Then the cycle in G corresponds to a path from $u^{(1)}$ to $u^{(2)}$ in G'. Since G is connected there exists a path from any v to u in G. So there is a path from any v to $u^{(1)}$ to $u^{(2)}$ in G'. Since there is also a path from $u^{(1)}$ to $u^{(2)}$ in G', then G' is connected. Now lets assume there is only one connected component in G'. If G' has one connected component then there is path from $u^{(1)}$ to $u^{(2)}$ in G'. Then there must exist an odd length cycle in G. Thus G cannot be bipartite. □

Now that we know the bipartite double cover G' would have twice more connected component than G, we can utilize the property to test bipartitness of the graph. More specifically, since we have duplicate the vertices, the two set of $\{u^{(1)}, v^{(2)}, ...\}$ and $\{u^{(2)}, v^{(1)}, ...\}$ must be disconnected if the original graph is bipartite.

6 Further reading

For detailed proof on complexity and discussion about graph sketching algorithms which guarantees the use of $O(\log^c n)$ spaces, the paper from Ahn,Guha,and McGregor[ACM12][AGM12] are

highly recommended. Also for general results, McGregor[SIG14] and Horlescu also have some light tutorials online.

References

[CCF02] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Automata, Languages and Programming*, 693–703, Springer, 2012.

[MP14] Gregory T Minton and Eric Price. Improved Concentration Bounds for Count-Sketch. *SODA*, 669–686, 2014.

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. *Proceedings of the 31st symposium on Principles of Database Systems*, 5–14, ACM, 2012.

[ACM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing Graph Structure via Linear Measurements. *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*,459–467,2012.

[SIG14] Andrew McGregor. Graph Stream Algorithms: A Survey*. *SIGMOD Rec.*, 43(1):920, 2014.