6.897: CSCI8980 Algorithmic Techniques for Big Data October 10, 2013

Lecture 11

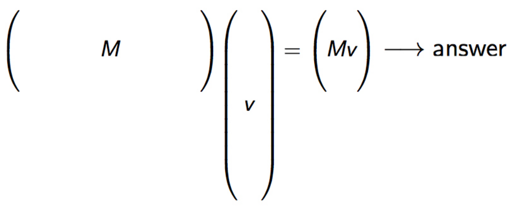
Dr. Barna Saha *Scribe: Koshy P Rajan*

Overview

In the previous lecture we studied how graph sparsifiers can be used to approximate a graph to a sparse graph by approximating the cuts in the original graph. The first approach was to cut approximation which included sampling all edges with uniform probability and another algorithm was discussed that samples the edge with higher probability if the edge participates in the cut. In this lecture, we study about sketching graphs. We can monitor the connectivity of dynamic stream graphs where edges can be inserted and deleted. All the previous works in this problem has assumed that edges can only be inserted into the graph and not deleted.

1 Sketches

Sketches encode data as vectors and uses linear projections to compress the data while preserving properties. Random linear projection ($M : R^n \to R^k$) preserves properties of any ($v \in R^n$) with high probability where $k \ll n$.



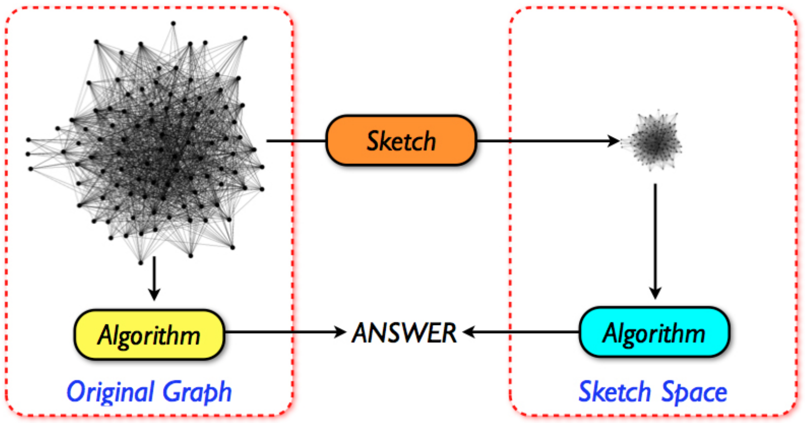
There is already an extensive theory on linear sketches with connections to compressed sensing, metric embedding, etc. It is a widely applicable technique due to its linearity and is suitable for stream processing. There are also many positive results such as distinct elements, entropy, estimating norms, fitting histograms, heavy hitters, etc.

Sketching method has two advantages: (i) it handles both insertion and deletion (ii) it does distributed computing.

2 Connectivity

Theorem: Can monitor connectivity of dynamic graph streams where edges are both inserted and deleted with O($nlog^3n$)-size sketch.

Plan: Sketch data and emulate connectivity algorithm in sketch space.



2.1 Basic Connectivity Algorithm

Algorithm 1 Spanning-Forest

```
For each node: pick incident edge
For each connected components: pick incident edge
Repeat until no edges between connected components.
```

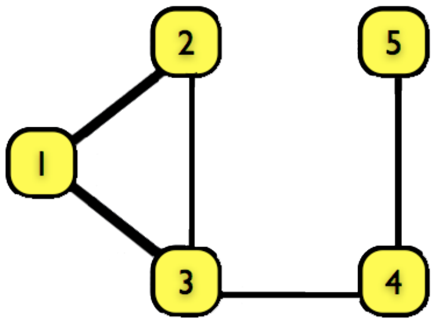
After log n rounds of this algorithm, these edges will include the spanning forest of the graph.

2.2 Graph Representation

Encode the neighborhood as the vector a_i, where a_i be the i^{th} row of signed vertex-edge matrix. For node i, let a_i be vector indexed by node pairs. Non- zero entries $a_i[i, j] = 1$ if $j > i$ and $a_i[i, j]=$ -1 if $j < i$.

Example:

Consider a stream of edges inserts and deletions, e.g., add(1; 2); add(1; 4); add(2; 3); add(1; 3); add(4; 5); add(3; 4); del(1; 4) would result in the following graph



$$\begin{array}{cccccccccccc}
 & (1,2) & (1,3) & (1,4) & (1,5) & (2,3) & (2,4) & (2,5) & (3,4) & (3,5) & (4,5) & \\
a_1 = (& 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\
a_2 = (& -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 &) \\
a_3 = (& 0 & -1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 &) \\
a_4 = (& 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 &) \\
a_5 = (& 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 &)
\end{array}$$

For the above graph, a_1 is a vector indexed by all node paths and the non zero entries to this vector are the incident edges to node 1. For a_2, the first value is -1 since $j < i$. This is done for all the nodes of the graph.

For any subset of nodes $S \subset V$,

$$support(\sum_{i \in S} a_i) \quad = \quad E(S, V \setminus S)$$

2.3 l_0 Sampling

There exist a sketch C when applied to any a (support vector like a_1), will return an entry e which belongs to support(a) with probability 9/10.

$Ca = e \in support(a)$

This works only when the non-zero entries in support vector is (+1, -1). We need to find at least one non-zero entry in the support vector. Let Ca_1, Ca_2 - - - Ca_n be sketches for l_0 sampling. We can post-process each sketch to find incident edge on each node. Suppose we found edges that are connected, e.g., S = (a_1, a_2, a_3), we can find an edge $e \in E(S)$ without taking another pass by using the property of linearity of sketches by just adding them $Ca_1 + Ca_2 + Ca_3 = C(a_1 + a_2 + a_3) \rightarrow e \in E(S)$

2.4 Sketch and Compute on sketches

For a given connected component(corresponding to a set of node S) we can find an incident edge to another connected component. If we need to find an edge across any cut S,

Sketch: Apply log n sketches C_i to each a_j

Case 1: One sketch for entire stream.

If a node 1 has a single neighbor at any particular point, we can keep a sketch B for that node

$$B \quad = \quad \sum_{(a, x_b, v): v \in \pm 1} v x_b$$

Algorithm 2 Algorithm

```
Use $C_1 a_j$ to get incident edge on each node $j$
For $i = 2$ to $t$:
    To get incident edge on supernode $S \subset V$ use:
```

$$\sum_{j \in S} C_i a_j \quad = \quad C_i(\sum_{j \in S} a_j) \rightarrow e \in support(\sum_{j \in S} a_j) = E(S, V \setminus S)$$

where x_b is the value of the neighbor. But this will not work if the node has multiple neighbors.

Case 2: One sketch for each sub stream.

Here the goal is to divide into sub stream with high probability that one sub stream has exactly 1 neighbor. This is achieved using hashing.

Consider hash function $g : [n] \rightarrow [n]$ and taken from a pair-wise independent family of hash function. Create a new hash function $h(x) = lsb(g(x))$. Suppose g(1) = 4, then h(1) = lsb(g(1)) = lsb(4) = 3 (4 in binary representation is 100 and the position of first '1' is 3)

Prob(h(x) = t) $\rightarrow$ Prob(lsb(g(x)) = t) = $1/2^t$

Once the sub stream is created, we apply the same method as Case 1.

B_1	(1,x)	h(x) = 1
B_2	(1,x)	h(x) = 2
-		
-		
-		
-		
B_{logn}	(1,n)	h(x) = log n

Each hash is again mapped to a hash bucket of 60 entries. This is because each h(x) can have more than one entries.

3 Analysis

Lets consider node 1. The number of neighbors of node 1 ranges from $2^0 \leq |neighbor(1)| \leq 2^j$

Consider the sub stream B_t: t =j-2

Prob(a particular neighbor gets mapped to B_t) = $1/2^t$

E[number of neighbors] = $\frac{2^j}{2^t} = \frac{2^j}{2^{j-2}} = 4$

There will be at least one t with less value like 4. Also, after applying hash bucket, there will be at least one entry in the bucket with only 1 element.

But there is one issue because this value need not be a neighbor. It can be the sum of neighbors. In order to ensure that it is a neighbor, we keep a F_2 sketch for every sub stream and if the F_2 value is 1 and if its corresponding value in the bucket is of 1 element, then it is the neighbor.

3.1 Space Complexity

Total space requirement for 1 node = O(log^2n)

Total space requirement(words) for n nodes = O($nlog^2n$)

Total space requirement(bits) for n nodes = O($nlog^3n$)

References

[1] Kook Jin Ahn, Sudipto Guha and Andrew McGregor. Analyzing Graph Structure via Linear Measurements.

[2] http://people.cs.umass.edu/ mcgregor/stocworkshop/mcgregor.pdf

[3] http://people.cs.umass.edu/ mcgregor/slides/epit-2.pdf