Project Report III

May 29, 2018

1 Brief Review

In Milestone 1, we presented our truncation algorithm for the query of finding degree distribution of graphs in the insert-only streaming model. In this report, we extend the truncation mechanism for the insert-delete and sliding window models.

2 Insert-Delete Streaming Model

In the insert-delete or dynamic model of data streams, we can both insert and delete edges of a graph. For a graph $G = (V, e)$, the data stream consists of a sequence of tuples $\{< u, v, \pm 1 >\}, u, v \in V$. $+1$ indicates insertion of the edge (u, v) while -1 indicates deletion of the edge. Note that if we have a stream entry of the form $< u, v, -1 >$ without a preceding matching entry of the form $< u, v, +1 >$, we simply ignore it.

Recall that our truncation algorithm for the inset-only model required 4 passes. In the dynamic model as well, the truncation algorithm requires only 4 passes.

Pass 1

Let us define the set of vertices with degree greater than Θ to be V_Θ.

$$V_\Theta = \{v | v \in V, deg(v) > \Theta\}$$

Now in order to find the vertices of V_Θ, we need to use a frequency estimator that works for insert-delete streams. The count-min sketch can be used for this purpose.

Pass 2 In this pass, for all the nodes in V_Θ we keep track of its adjacent nodes which are also members of V_Θ.

$$v \in V_\Theta, L_v = \{u | (u, v) \in E, u \in V_\Theta\}$$

Pass 3 This is identical to the Pass 3 in our truncation algorithm for the insert-only streaming model except for a single change as follows. If for an incoming

edge, we have $(u, v), u, v \in V_\Theta$ but $u \notin L_v$, this means that the edge (u, v) will be deleted later in the stream. Hence we simply ignore processing that edge.
Pass 4
At the end of the third pass, we have the IDs of all the adjacent vertices of the nodes $v \in V_\Theta$ (the ones marked 1 in L_V and all the adjacent vertices not in V_Θ). Using this, we can delete all the other edges and generate the stream for our truncated graph G_Θ.

3 Sliding Window Model

In this model, the data stream is processed in batches of a fixed size called the window. With the arrival of a new stream entry the oldest entry of the last window is removed and the new entry becomes the most recent member of the current window. Now let us see how can we tweak our truncation algorithm to fit in the sliding window model. Firstly we have to use a heavy-hitter estimation suitable for sliding window models like [1] for our computation of V_Θ in the first pass. From our truncation algorithms for the insert-only stream and insert-delete stream, we see that we maintain a state information about each node-whether it has degree greater than Θ. Let $< u_1, v_1 >$ be the edge that leaves the window and $< u_2, v_2 >$ be the most recent edge that arrives. Thus in the transition phase from window at t-1 to t, the degrees of only these four vertices, $\{u_1, v_1, u_2, v_2\}$ could have changed. Keeping this in mind, we check whether the transition affects the membership of these nodes to V_Θ and carry out a single pass on the data like the third pass to update G_Θ.

4 Number of triangle subgraph queries

One of the basic questions of graph processing especially for social media graphs is counting the number of triangles in a given graph G. In other words, let $G = (V, E)$ be an undirected graph with n vertices and m edges and let t be the number of triangles in G. In the streaming model for graph processing, the goal is to compute an (ϵ, δ)-approximation for t while G is presented to the algorithm as a stream of edges. By sublinear space usage, we mean algorithms that use o(m) bit space, and by stream of edges, we mean a sequence of edges that is an arbitrary permutation of E. In addition to the space usage, we restrict the algorithms to have only O(1) passes over the stream and o(m) per-edge processing time. Here we discuss one such algorithm [2] and show how to extend it to achieve pan-privacy.

Goal: To compute an estimator the total number of triangle subgraphs, T_3.
Algorithm:

1. Let n be the number of vertices of the graph G. Construct a family, $\mathcal{F}$ of uniform ± 1-valued random vectors of length n, which are 12-wise

independent. As indicated in [3], this family can be constructed explicitly using the parity check matrices of BCH codes with only O(log n) bits.

2. Pick a uniformly random vector q from this family, $\mathcal{F}$.

3. As the stream passes, for every edge $(i,j) \in E$ that appears, we compute

$$Z = \Sigma_{(i,j)\in E} q(i)q(j)$$

4. At the end of stream, we define

$$X = \frac{1}{6}Z^3$$

5. Compute E(X). Based on the definition,

$$E(X) = \frac{1}{6}E\Big(\Sigma_{(i,j)\in E}(v(i)v(j))^3\Big)$$

Since $q(i)$ can only take take values ± 1 uniformly at random, $E(q(i)) = 0$. Thus by linearity of expectation and regarding the facts that $E(q^{2k+1}(i)) = 0$ and q(i)'s are 12-wise independent, the terms that have a variable with an odd power are evaluated to zero. Therefore, only the terms in form of $6q^2(i)q^2(j)q^2(k)$ remain. For a given triplet $\{i,j,k\} \in V$, $q^2(i)q^2(j)q^2(k) = 1$ only iff $(q(i)q(j))^2 = (q(j)q(k))^2 = (q(i)q(k))^2 = 1$, This happens only when all the three edges $(i,j),(i,k),(j,k)$ exists, that is the vertices $\{i,j,k\}$ form a triangle. Thus the terms of the form $q^2(i)q^2(j)q^2(k)$ correspond to the triangles and we have,

$$E(X) = \frac{1}{6}(6 \times T_3) = T_3$$

where T_3 is the desired number of triangular subgraphs in G.

Now in order to ensure pan-privacy, essentially we have to make our estimator Z to be differentially private. For this we tweak the algorithm as follow.
We compute a new random variable Y where for each edge that appears in the data stream we include the values $q(i)q(j)$ with probability $\frac{1}{2} + \frac{\epsilon}{4}$ and for all the other edges we include the value $q(i)q(j)$ with probability $\frac{1}{2}$. Thus assuming m is the total number of edges in the stream, we have

$$Y = \sum_{(i,j)\in E} q(i)q(j) * (\frac{1}{2} + \frac{\epsilon}{4}) + \sum_{(i,j)\notin E} v(i)v(j) * \frac{1}{2}$$
$$\Rightarrow Y = Z * (\frac{1}{2} + \frac{\epsilon}{4}) + \sum_{(i,j)\notin E} v(i)v(j) * \frac{1}{2}$$

Now, let $W = \sum_{(i,j)\notin E} q(i)q(j) * \frac{1}{2}$. The total number of edges in a graph with n vertices is $\binom{n}{2}$. If we see m edges appear in the stream, then we must

have $\binom{n}{2}$ edges such that $(i,j) \notin E$. The expected value of $q(i)q(j) = -\frac{1}{2}$. Thus $E(w) = -\frac{1}{4}(\binom{n}{2} - m))$. Hence we have

$$Z = \frac{4}{2+\epsilon} * (Y + \frac{1}{4}(\binom{n}{2} - m))$$

We can use the median-of-means trick to get an (ϵ, δ) approximation on Z. The rest of the algorithm is same as before where $X = \frac{1}{6}Z^3$ is our unbiased estimator for T_3 [using appropriate median-of-means trick].

Since the values $q(i)q(j)$ are obtained from the same random vector q for both the cases when $(i,j) \in E$ and $(i,j) \notin E$, once an edge (i,j) appears in the random variable Y, the distribution of its value $q(i)q(j)$ is identical for both the cases. Thus for differential privacy, we have to ensure that the distribution of whether an edge appears in Y for the two cases should be e^ϵ close to each other. Let D_0 be the distribution of whether $(i,j) \notin E$ appears in Y and D_1 be the corresponding distribution for $(i,j) \in E$.
Claim 3.1. For $\epsilon \leq \frac{1}{2}$, the distributions D_0 and D_1 are $\epsilon-$differentially private. For both cases $b \in \{0,1\}$ where $b = 1$ means an edge (i,j) contributes to Y and $b = 0$ means it does not , it holds that:

$$e^\epsilon \leq \frac{Pr_{D_1}\left[b\right]}{Pr_{D_0}\left[b\right]} \leq e^\epsilon$$

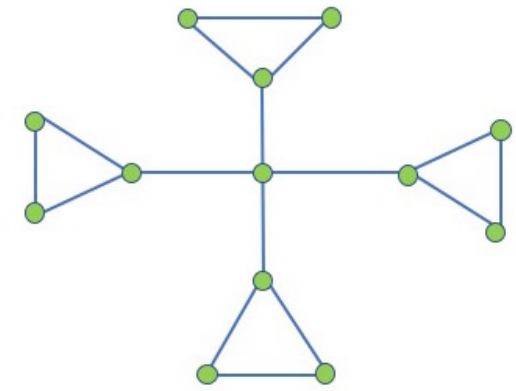
.
Proof of Claim 3.1. The ratio between the probabilities of 1 by D_1 and D_0, *i.e.*, edge (i,j) is present in Y is $\frac{\frac{1}{2}+\frac{\epsilon}{4}}{\frac{1}{2}} = 1 + \frac{\epsilon}{2} \leq e^\epsilon$. The ratio between the probabilities of 0 by D_1 and D_0, i.e, edge (i,j) is not present in Y is $\frac{\frac{1}{2}-\frac{\epsilon}{4}}{\frac{1}{2}} = 1 - \frac{\epsilon}{2} \geq e^\epsilon$ (this inequality holds for $-1 \leq \epsilon \leq 1/2$). Thus we see that the estimator Y is differentially private thereby making our algorithm pan-private.

5 Truncation Algorithm for Querying Connected Components

So the concerned query here is to return the number of connected components in the graph. So there are two steps involved

1. Truncate the original graph G to a Θ-bounded subgraph G_Θ.
2. Compute the sensitivity for the query of counting the number of connected components over this truncated graph G_Θ.

Figure 1:



Note: It is not possible to preserve all the connected components of an arbitrary graph for a given value of Θ. For e.g. in the given graph in Figure. 1, it is impossible to maintain the number of connected components to 1 when $\Theta = 3$. Let $\mathcal{G}$ be the set of all possible $\Theta-$bounded graphs derivable from G. Thus at best we aim to produce a truncated graph,

$$G^*_\Theta = \arg_{G_\Theta \in \mathcal{G}} \min\{\#\ connectedComponenents(G)\}$$

Challenge

1. We need a stable truncation algorithm. For a insert only stream, the graph G can be rendered via $m!$ different streams where m is the total number of edges in G. Thus by a stable truncation algorithm, we mean that the # of connected component of the resulting truncated graph should not depend on the edge order of the streams. Else the resulting sensitivity can be high and graph dependent.

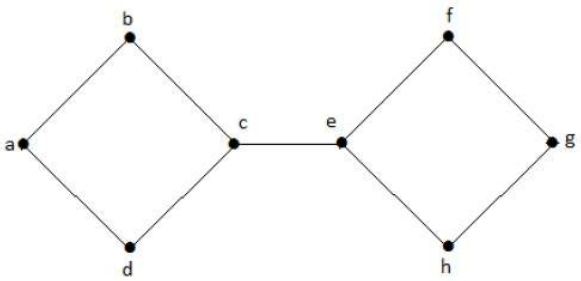
2. As stated above, the best case scenario is to produce G^*_Θ, i.e., the $\Theta-$bounded graph with that preserves maximum possible connected components of the original graph.

First let us define a special type of edge for graphs called bridge.

Definition 1. A bridge is an edge of the graph whose deletion increase the number of connected components.

In Figure 2., the edge $< c, e >$ is a bridge. Recall from our truncation algorithm for computing degree distribution, for a given edge $< u, v >$ iff either of the end points belong to v_Θ, then we include it in our G_Θ . It is only for

Figure 2:



edges of the type $< u, v >\in E, u, v \in V_\Theta$ that we there is uncertainty about whether it will be a part of our truncated graph G_Θ. Hence we should only be considered about this special type of edges, precisely we should not discard any edge $< u, v >\in E, u, v \in V_\Theta$ such that $< u, v >$ is a bridge. Let E' be the set of such edges.

Goal: For every edge $e \in E' = \{< u, v > \mid < u, v >\in E, u, v \in V_\Theta\}$, find whether it is a bridge.

Observation: For a given graph $G = (V, E)$ an edge $< u, v >\in E$ is a bridge iff we can find a subset of vertices $S \subset V$ such that the cut $S, V/S$ has only one edge $< u, v >$. Thus our aim is to attempt to create such a vertex set if possible for all the edges $e \in E'$.

Algorithm:

1. Create a data structure as follows. For each node $v_i \in V$, define a vector $a_i \in \{-1, 0, 1\}^{\binom{n}{2}}$

$$a^i_{\{j,k\}} = \begin{cases} 1 \text{ if } \ i = j < k \text{ and } \{v_j, v_k\} \in E \\ -1 \text{ if } \ j < k = i \text{ and } \{v_j, v_k\} \in E \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

   These vectors then have the useful property that for any subset of nodes $\{v_i\} i \in S$, the non-zero entries of $i \in S, a_i$ correspond exactly to the edges across the cut $(S, V/S)$.

2. Let S be the subset of vertices if it exists such that the cut $(S, V/S)$ has only one edge $< u, v >$. In other words removing edge $< u, v >$ the number of connected components in the resulting graph will be $Components(S)+ Compoenets(V/S) + 1$ where Components(S) means the number of connected components in the subgraph formed by S. Now let us see how we can construct S if it exists. For a given edge $e =< u, v >\in E'$

   (a) Initialise $S = u, S' = v$

   (b) Select a node v_i from S which has not been visited before. Find all the non-zero entries from a_i. These represent all the edges that are incident on v_i. Let this set of vertices be R.

(c) Update $S = S \cup R$. Mark v_i as visited.

(d) Select a node v_j from S' which has not been visited before. Find all the non-zero entries from a_j. These represent all the edges that are incident on v_j. Let this set of vertices be R'.

(e) Update $S' = S' \cup R'$. Mark v_j as visited.

(f) Check if $S' \cup S \neq \emptyset$. If their intersection is non-empty, it means that e is not a bridge and we terminate our algorithm here.

(g) Else check if $S' == V/S$. If this is true, then e is a bridge and terminate the algorithm here. Else go to Step 2 and repeat.

3. Thus once we identify the all the edges $e \in E'$ which are bridges, we simply retain all of them along with the other edges of the form $< u, v >$, $u \notin V_\Theta orv \notin V_\Theta$ in our G_Θ. In this way we can ensure that the resulting truncated graph is actually G^*_Θ. Also note our algorithm is independent of edge order arrival and is hence stable.

Note: The advantage of this algorithm is that the data structure a is used in finding the number of connected components from a graph in streaming model [4]. Thus our truncation algorithm results in no additional space usage.

Bibliography

[1] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Heavy hitters in streams and sliding windows," in proceedings of IEEE INFOCOM, 2016
[2] H. Jowhari, M. Ghodsi, "New streaming algorithms for counting triangles in graphs", in proceedings of COCOON, 2005
[3]N. Alon, Y. Matias, M. Szegedy," The space complexity of approximating the frequency moments", in proceedings of STOC, 1996
[4] Andrew McGregor, "Graph stream algorithms: A survey" in proceeding of SIGMOD, 2014