Project Report II

Amrita Roy Chowdhury
Andrew Brockmann

May 11, 2018

1 Problem Setting

In this project, we are considering differential privacy for graph processing in the streaming setting. Here we consider the insert-only streaming model. To be more precise, we define a stream of a graph G to be an infinite sequence of tuples of the form $< u, v >$ where u, v belong to the node sets and $< u, v >$ represent an edge of the graph G in question.

2 Privacy Definition

2.1 Classical Graph Processing Setting

Let us first define node-differential privacy for graphs in the classical setting.

Given a privacy parameter $\epsilon > 0$, an algorithm A is ϵ node differentially private if for all neighbor graphs G and $G'$and for all sets S of possible outputs produced by A:

$$Pr[A(G) \subseteq S] \leq e^{\epsilon} \cdot Pr[A(G) \subseteq S] \tag{1}$$

where two graphs are called neighbors if one can be obtained from the other by removing a node and its adjacent edges.

2.2 General Streaming Setting

For the general streaming setting two streams, one with and the other without all information of a particular individual are said to be neighbors even though the data of an individual are interleaved arbitrarily with other data in the stream. User-level privacy in this setting is defined such that two such neighboring streams will produce very similar distributions on states and outputs.

2.3 Graph Processing in Streaming Setting

Given our definition of the streams in the section 1, treating nodes as users, the user level privacy of the general streaming setting becomes equivalent to node-differential privacy for graphs. Two streams S and S' are said to be neighboring if

$$|set(S') - set(S)| = \{< u, v_i >, v_I \in V\} \tag{2}$$
$$|set(S) - set(S')| = \emptyset \tag{3}$$

where $set()$ is the set representation of a given stream S. In other words the graph generated by the streams S and S' are node-DP neighbors according to our definition in section 2.1. However there is a conceptual difference between the two settings, general streams and graph streams. Let $\mathcal{S}$ be a general stream and $\mathcal{S}'$ be a neighboring stream with additional information about user u. Both has the same information about all the users other than u. However, for graphs, adding the edges incident on u also changes the information about some other nodes (the nodes sharing edges with u).

In the general streaming setting, also affects the information about other users,Note that Graph algorithms can be broadly classified into two types-

1. **Numeric results** These consist of numeric responses to q query like degree distribution of a graph, the number of subgraphs present, connectivity etc
2. **Graph results** These consist of synthetic or subgraph responses like MST, cut sparsifiers etc.

The truncation algorithm designed has to be specifically tailored to the query being asked so that we can minimize its sensitivity. First we focus on queries related to the degree distribution of graphs.

3 Truncation in Classical Setting

Even in the classical setting, achieving node-DP is problematic. The problem, roughly, is that node-private algorithms must be robust to the insertion of a new node in the graph, but the properties of a sparse graph can be altered dramatically by the insertion of a well-connected node. For example, for common graph statistics – the number of edges, the frequency of a particular subgraph – the change can overwhelm the value of the statistic in sparse graphs. This is so because removing a node, in the worst case can remove n-1 edges [n is the total number of nodes] thereby giving rise to high sensitivity. The main idea to tackle this is to "project" the input graph onto the set of graphs with maximum degree below a certain threshold. The benefits of this approach are two-fold. First, node privacy is easier to achieve in bounded- degree graphs since the insertion of one node affects only a relatively small part of the graph. Technically, the sensitivity of a given query function may be much lower when the function is restricted to graphs of a given degree. Second, for realistic networks this transformation loses relatively little information when the degree threshold is chosen carefully.

4 Truncation in Streaming Setting

Note that the tuples in our streams are devoid of a timestamp. Such a definition is a much stronger requirement than one having a timestamp. It is so because for timestamped streams we can have a weaker definition of neighbors as follows. Let S' be the stream with the edges incident on a particular node u and S be the stream without them. S and S' are said to be neighbors if the edges incident on u are interleaved in any arbitrary manner provided the edges that are common to both the graphs ,i.e., the ones which are not incident on u appear in the same order in both the streams. However, in our definition we relax this restriction and the common edges can now also appear in arbitrary order. Although this guarantees a much stronger privacy definition, it introduces a new challenge for the truncation algorithm. Now we need to have stable truncation algorithm for the graph steams where stability means that the resulting truncated graph is independent of the order of edge arrivals in the stream. It is so because, otherwise the sensitivity can be very high. To have an intuition, if we do not have control over the number of edges in the resulting truncated graph (i.e., if the algorithm is not stable and outputs different truncations based on the edge arrival order), then for computing the sensitivity we have to compute the difference in the number of edges between these two graphs- the maximal Θ-bounded graph with the lowest number of edges and the maximum truncate graph which is graph dependent. The answer is clearly dependent on the graph in question and will most likely be very high.

In the existing literature for classical graph processing, the truncation algorithm proposed by Day et al in [1] has the best performance (least sensitivity)$(2\Theta+1)$. In [1] the algorithm requires a stable ordering of all edges in an input graph G, denoted by $\Lambda(G)$. We say that a graph edge ordering Λ is stable if and only if given two neighboring graphs G = (V,E) and $G' = (V', E')$ that differ by only a node , $\Lambda(G)$ and $\Lambda(G')$ are consistent, in the sense that if two edges appear both in G and G', their relative ordering are the same in $\Lambda(G)$ and $\Lambda(G')$ where . The projection first forms a graph with all nodes in G but no edges, and then keeps inserting edges from Λ following the ordering whenever inserting an edge e = (u, v) will not cause the degree of either u or v larger than Θ. Such stabling edge ordering can be easily obtained in practice. For example, if G is abstracted from a data source where each node v has a unique id id(v) (e.g., the account id in social network data), and these node ids can be totally ordered by an ordering $<$ (e.g., $<$ can be alphabetical ordering), then each edge (u, v) can be represented as a pair (id(u), id(v)) if id(u)$<$ id(v) and (id(v), id(u)) otherwise, and the lexicographical ordering of edges is stable. It is to be noted that this algorithm produces only a maximal truncation which is not necessarily a maxima.

To translate this to the streaming setting, we will first have to establish a stable ordering on the edges of the graph. Any ordering would constitute a sort requiring $O(e \log e)$ operations, so we would need atleast $\log e$ passes where e is the size of the stream. But we present a truncation algorithm, that requires only 4 passes over the data stream.

5 Proposed Truncation Algorithm

First let us define maximum truncation.

Definition 5.1. For a given graph G, maximum truncation is the Θ bounded truncated graph G^*_Θ such that it has the maximum number of edges out of all possible Θ bounded truncated graphs of G. In other words this graph retains the most number of information about the original graph G with respect to queries related to degree statistics. Formally,

$$\mathcal{G}^*_\Theta = arg \max_{G_\Theta \in \mathcal{G}} \{E(G_\Theta)\} \tag{4}$$

where $\mathcal{G}$ is the set of all possible Θ-bounded graphs derivable from G and $E(G)$ returns the number of edges in a given graph G.

Our proposed algorithm has four passes as discussed below.

First Pass

Firstly, we have to identify the nodes with degree greater than Θ . This can be done in one pass by using the standard heavy hitter sketching algorithms like the count-min algorithm. Thus at the end of this pass, we have the IDs of all the nodes with degree greater than Θ. Let N_Θ denote the set of such nodes. Space required is $O(|N_\Theta| \log t)$ where t is the number of nodes.

Second Pass

In this pass for each of the nodes $u \in N_\Theta$, we compute the number of its adjacent vertices that also belong to N_Θ. Formally, we compute the following count for each such node

$$c_u = |\{< u, v > | v \in N_\Theta\}|, u \in N_\Theta \tag{5}$$

This requires space $O(|N_\Theta| log^2 t)$ where t is the number of nodes.

Third Pass

In this pass we decide the edges that are to be truncated from the graph and the method is given by Algorithm 1.

Algorithm 1 Third Pass

```
Input:Edge $< u, v >$
Initialize:$E_u = 0$ /*The observed degree till now for vertex u*/
    $L_u =$ /*The list of all adjacent vertices of u that belong to $N_\Theta$
that we have observed so far in the stream, sorted according to $c_u$. Vertices
which are to be included in are marked 1. */
IF$u \notin N_\Theta$ and $v \notin N_\Theta$
    add edge to $G_\Theta$
ELSEIF $u \in N_\Theta$ and $v \notin N_\Theta$
    add edge to $G_\Theta$
    IF$E_u < \Theta$
        $E_u++$
    ELSE
        Let $x = \arg\max_{x \in M} c_x, M = \{x \in L_u | x.mark=1\}$
        Unmark x
        Call $FindReplacement(x)$
    ENDIF
ELSE
    IF $E_u < \Theta$ and $E_v < \Theta$
        Add v to $L_u$ in correct sorted order and set v.mark=1. $E_u++$
        Add u to $L_v$ in correct sorted order and set u.mark=1. $E_v++$
    ELSEIF$E_u < \Theta$ and $\Theta - E_u \geq c_u - |L_u|$
        Add v to $L_u$ in correct sorted order and set v.mark=1, $E_u++$
        Add u to $L_v$ in correct sorted order and set u.mark=1
        IF$E_v == \Theta$
            Let $x = \arg\max_{x \in M} c_x, M = \{x \in L_u | x.mark=1\}$
            Set x.mark=2
            Call $FindReplacement(x)$
        ENDIF
    ELSE
        Add v to $L_u$ in correct sorted order and set v.mark=0.
        Add u to $L_v$ in correct sorted order and set u.mark=0.
    ENDIF
ENDIF
PROCEDURE FindReplacement(vertex u)
    $M = \{x.mark = 0 | x \in L_u\}$
    WHILE ($M \neq \emptyset$)
        Let $x = \arg\min_{x \in M} c_x$
        IF$E_x < \Theta$
            Set x.mark=1 in $L_u$
            Set u.mark=1 in $L_x$ $E_x++$
        ELSE
            $M = M - x$
        ENDIF
    ENDWHILE
    IF($M == \emptyset$)
        $E_u--$
    ENDIF
END PROCEDURE
```

Fourth Pass

As seen in Algorithm 1, edges incident on atleast one node with original degree less than Θ is included in G_Θ. At the end of the third pass, we have the IDs of all the adjacent vertices of the nodes $u \in N_\Theta$ (the ones marked 1 in L_u and all the adjacent vertices not in N_Θ). Using this, we can delete all the other edges and generate the stream for our truncated graph G_Θ.

Theorem 5.1. *For a given graph G the above protocol gives a maximum truncation G^*_Θ*

Proof. The maximum truncation conceptually should include all the edges of every node with degree less than Θ and contain atmost Θ edges for all other nodes. Note that there are three types of edges for any graph G broadly.

1. $< u, v >$ where $u, v \notin N_\Theta$ These edges should be definitely included in G^*_Θ because they can never cause any conflict. Our algorithm does just that.
2. $< u, v >$ where $u \in N_\Theta$ and $v \notin N_\Theta$ From node u's perspective, it has only degree less than Θ, hence all of its nodes should be added into G_Θ. This concurs with our algorithm.
3. $< u, v >$ where $u, v \in N_\Theta$ Finally, for these type of edges, we should aim to retain the maximum number of them such that the truncated degree in G_Θ for u and v is as close to Θ as possible. Let us look into this case in more detail.

From Algorithm 1, we see that we only delete or disregard an edge in two stages. First is in steps $19 - 23$. But notice that, here we add an edge in steps $17 - 18$ before deleting one. So, in case we don't find any replacement in the edge $< x, v >$, the total number of edges of the graph does not decrease.But in case we find a replacement, the total number actually increases. We disregard an edge in steps 25-26. This can occur under two circumstances, one when both u and v have degree Θ in G_Θ constructed so far. Since, we have already reached our goal, disregarding that edge is fine. The other case is when v has degree Θ and u has $c_u - |L_u| > E_u - \theta$ possible edges left to choose from to increase its degree. Note that, in the event when, u has limited edges to choose from so that its degree becomes Θ, we always select every edge (steps 17-18) that is incident on it. Thus we can see that for all the nodes , the algorithm selects the maximum number of edges possible, thereby resulting in the maximum truncation. □

Theorem 5.2. *The sensitivity for computing degree distribution with the aforementioned truncation scheme is* 2Θ

Proof. We have established that the proposed truncation algorithm results in the maximum truncated graph for a given stream S. Now for a neighboring stream S', which has additional edges incident on a node u, the total number of extra edges that can be added is G'_θ is at most Θ. Hence since each edge affects the degree of two nodes, we can have a maximum change of 2Θ in the degree histogram. □

Note that selecting edges according to step 32 in Algorithm 1, is to support the intuitive heuristic that lesser the value of c_u lesser is the possibility of a conflict.

The space complexity is $O(|N_\Theta|c * log^t)$ where t is the number of nodes in G and $c* = \max\{c_u | u \in N_\Theta\}$. Note that $O(|N_\Theta| logt)$ is the minimum space required for any truncation algorithm information theoretically as in the least we will have to identify these nodes. In practice, $c*$ cannot be very large, hence our space requirement is practical.

6 Milestone 2

Things to be done:

1. So far we have only considered insert only streams that is where new edges can only arrive. The next step is to consider insert-delete streams (where edges can be deleted as well) and sliding-window model and extend the truncation algorithm accordingly.

2. Collectors of confidential data, such as governmental agencies, hospitals, or search engine providers, can be pressured to permit data to be used for purposes other than that for which they were collected. To support the data curators, Dwork et al [2] introduced the notion of pan privacy; roughly speaking, these algorithms retain their privacy properties even if their internal state becomes visible to an adversary. In this model, an algorithm moves through a sequence of internal states and produces a (possibly unbounded) sequence of outputs. We assume that the adversary can only observe internal states and outputs; the adversary cannot see the data in the stream (although it may have auxiliary knowledge about some of these data). We can model this through the assumption that in a single atomic step the algorithm can read a datum, compute, and change the internal state. Some sort of granularity assumption is essential, since otherwise the adversary could learn the raw data and privacy is impossible. Thus by definition it is impossible to achieve pan privacy, in the truncation step because the output of it is a subgraph. Hence, we have to assume a bigger granularity window where a single atomic step the algorithm can read a datum, compute the truncated graph, and change the internal state for a specific query. Using this assumption, we propose to give a pan-private algorithm for computing the degree distributions and #triangle subgraphs queries.

3. Propose a truncation algorithm for the query of number of connected components. The intuition here is to retain all the bridge edges of the original graph G in the truncated G_Θ.

References

[1] Wei-Yen Day, Ninghui Li, and Min Lyu. Publishing graph degree distribution with node differential privacy. In *In Proceedings of SIGMOD*, SIGMOD '16, 2016.

[2] Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N. Rothblum, and Sergey Yekhanin. Pan-private streaming algorithms. In *In Proceedings of ICS*, 2010.