CS 395T: Sublinear Algorithms Fall 2014

Lecture 6 — Sept 16, 2014

Prof. Eric Price *Scribe: Taewan Kim*

1 Overview

In the last lecture we covered

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]
3. Fast recovery time Algorithm in $O(\log^c n)$ developed by Gilbert, Li, Porat and Strauss. [GLPS12]

In this lecture we will discuss a new topic: Graph Sketching. 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, sub-topics will be covered as follows.

1. Define Graph Sketching problem
2. Warm up problem: with only insertions of elements
3. Warm up problem: with insertions/deletions
4. Warm up problem: as a linear sketch

2. Preliminaries for Graph Sketching

2 Preliminaries for Graph Sketching

Graph is composed of sets of vertices and edges which can be found easily around us such as friendship in social network. When a given graph is dense, it is a natural question to have a sketch of the graph which preserves the structural characteristics of the original graph.

Streaming model for the graph is usually defined by the stream of edges, and the number of nodes are given as n. Also the stream of edges can have both additions and deletions. You can't solve many problems with $o(n)$ space, so the "semistreaming" model allows $O(n \log^c n)$ space. A number of graph problems can be solved in the semistreaming model, using a remarkable technique due to [AGM12]. These include:

- Find spanning tree

- $(1+\epsilon)$-approximation of minimum spanning tree (MST)
- Test whether the graph is bipartite
- Estimate the size of cuts
- Sample uniform edge from cut

2.1 Warm Up Problem

Before introducing graph sketching algorithm directly, let's consider a simpler problem that will be used in the graph problems.

Given Stream is composed of insertions/deletions of elements. We want to sample a random element. If there are k distinct elements, output each element with probability $\approx \frac{1}{k}$

Goal $O(\log^c n)$ space

Case 1 Insertions only

We can solve this problem using a simple hash function

1. $h : [n] \to [0,1]$ uniformly random
2. Store a if $h(a)$ is minimum

This simple algorithm can output an element with probability $\approx \frac{1}{k}$, since the hash function h doesn't care about the input stream. So any element with minimum $h(a)$ value can be stored uniformly random. However this algorithm can not deal with a case when deletion is included in the stream.

Case 2 Insertions and deletions

- Can assume we that know k to factor 2
- Can output FAIL with $\frac{1}{2}$ probability
- Use S: $\frac{1}{k}$ fraction of coordinates
    - Before: In distinct elements algorithm, we checked "$|S \bigcap stream| > 0$?"
    - Now: Record $S \bigcap stream$ (if not > 1 element.)
      Finally: output it

To fully build an algorithm for the case with both insertions and deletions, some algorithms for sparse recovery are needed to check the number of distinct elements k in the stream.

2.2 Sparse Recovery for $k = 1$

Goal of this section is to introduce algorithms which can check the number of nonzero elements in a vector. The first step of the algorithm is to find the index of nonzero elements, i.e. support of the verctor. Following statements describes the problem to consider in formal statements.

Stream Input: $(i, \alpha) \Rightarrow$ Update: $x_i \to x_i + \alpha$

Guarantee At end, $\exists\, i^* \ s.t.\ x_j = 0 \ \ (\forall j \neq i^*)$ and $x_{i^*} \neq 0$

Goal Find i^*

Choose matrix $A \in \mathbb{R}^{o(1) \times n}$ such that Ax gives i^* via some algorithm, i.e. $Ax \Rightarrow i^*$.

So, the basic goal of the problem is to recover the index of nonzero element given that there is only one nonzero element. To develop an algorithm, let's start with an simple case when the nonzero element has value 1.

Simple case $x_{i^*} = 1$

Consider a vector $v = [1, 2, 3, 4, \cdots, n]$

$$\langle v, x \rangle = v_{i^*} = i^* \quad \Rightarrow \quad \text{gives } i^*$$

By defining $A = v = [1, 2, 3, 4, \cdots, n]^T$ we can easily recover the index i^*. Now let's consider more general case.

General case $x_{i^*} = \alpha$

Consider a vector $v = [1, 2, 3, 4, \cdots, n]$. There are two possible methods

(Method 1) *This method was proposed during the class*

Consider another vector $v' = [0, 1, 2, 3, \cdots, n-1]$

$$y' = \langle v', x \rangle = \alpha(i^* - 1)$$
$$y = \langle v, x \rangle = \alpha i^*$$
$$\Rightarrow \alpha = y - y'$$
$$\text{So we can recover } i^* \text{ by } i^* = \frac{y}{y - y'}$$

Actually, v' doesn't have to be same as above. It should be any form of vector which has proper linear combination to recover i^*. So there could be other possible methods. For example:

(Method 2)

Consider another vector $v' = [1, 1, 1, \cdots, 1]$

$$\begin{aligned} y' &= \langle v', x \rangle = \alpha \\ y &= \langle v, x \rangle = \alpha i^* \\ \Rightarrow i^* &= \frac{y}{y'} \end{aligned} \tag{1}$$

So, we've shown how to recover the index of nonzero element when there exists only one support. However, the **Guarantee** condition above is not always true. So, an algorithm to check whether $k \neq 1$ is required, where k is a number of nonzero element in given vector x. A sketch based algorithm can be used to check the number of nonzero elements with high probability.

The number of nonzero elements can be represented in ℓ_0-norm as $\|x\|_0$, and we want to check whether $\|x\|_0 \geq 2$. This ℓ_0-norm checking step can be done using a linear sketch Ax with only $O(\log n)$ rows. We will start by checking whether $\|x\|_0 \geq 1$, then use this to check whether $\|x\|_0 \geq 2$.

Simple case $(k = 1)$ Check whether $\|x\|_0 \geq 1$, i.e. $x \neq 0$

Algorithm NZ (Based on random generation.)

1. Randomly generate a vector $v \in \{\pm 1\}^n$ with equal probability $\frac{1}{2}$ so that v has only ± 1 values.
2. Output whether $\langle v, x \rangle \neq 0$ or not.

This strategy can actually check $\|x\|_0 \geq 1$ with high probability. Suppose $x_i \neq 0$. And following equation is always true, where y_{-i} represents a vector y with coordinates without the i_{th} element.

$$\langle v, x \rangle = \langle v_{-i}, x_{-i} \rangle + v_i x_i$$

So, to have $\langle v, x, \rangle = 0$, $v_i x_i = -\langle v_{-i}, x_{-i} \rangle$ should be satisfied. And v_i determines the sign of $v_i x_i$. So the probability of having appropriate sign is a probability of choosing v_i which is equal to $\frac{1}{2}$. In this sense,

$$\begin{aligned} Pr\{\langle v, x \rangle = 0 | v_{-i}\} &\leq \frac{1}{2} \\ \Rightarrow Pr\{\langle v, x \rangle = 0\} &\leq \frac{1}{2} \end{aligned}$$

If $\|x\|_0 = 0 \Rightarrow \langle v, x \rangle = 0$, so if we do this operation for $O(\log n)$ times, we can check whether $\|x\|_0 \geq 1$ with high probability.

Advanced case $(k = 2)$ Check whether $\|x\|_0 \geq 2 \Leftrightarrow \|x\|_0 > 1$. And this is equivalent to checking $\|x\|_0 \neq 1$ given $\|x\|_0 \neq 0$.

Algorithm C (Uses Algorithm NZ. FAIL when $\|x\|_0 > 1$)

1. For $O(\log n)$ times

2. Choose set S of size $\frac{n}{2}$, where S is a subset of $[n]$. (Subset of coordinates)
3. Run **Algorithm NZ** on $(Stream \bigcap S)$ and $(Stream \bigcap \bar{S})$.
   ($\bar{S}$ denotes the complement set of S. See Figure 1)
4. If run on both S and $\bar{S}$ output nonzero, output FAIL with $\geq \frac{1}{8}$ probability ($\|x\|_0 > 1$)

So this Algorithm C can check the $\|x\|_0 > 1$ with $1 - (\frac{7}{8})^{8 \log n} \geq 1 - \frac{1}{e^{\log n}} = 1 - \frac{1}{n}$ probability, which is a high probability.

Combining all the introduced algorithms, it is possible to recover

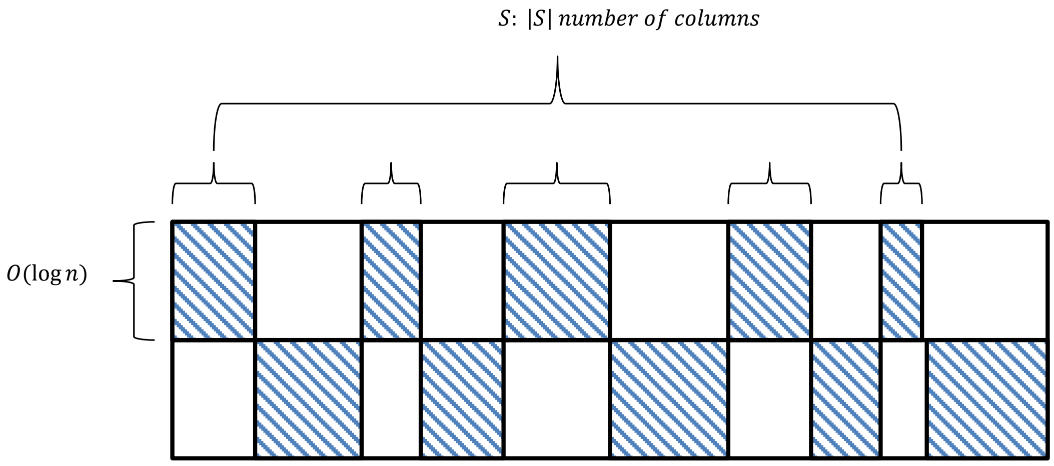

Figure 1: Structure of matrix A for checking $\|x\|_0 \geq 2$. Colored columns represents the randomly selected set of coordinates S. Others are set to 0.

2.3 Distinct Element Sampling

Now let's go back to the Warm Up problem in Section 2.1 with Case 2 where both insertions and deletions are possible. We are trying to output an element with probability $\approx \frac{1}{k}$.

Distinct Element Sampling (insertion/deletion)

```
FOR $\hat{k} = 1, 2, 4, \cdots, n$
    FOR $\log n$ different choices of $S \subseteq [n]$ of size $\frac{n}{\hat{k}}$
        $h : [n] \to [\hat{k}]$
        $S : \{i | h(i) = 0\}$
        Run Algorithm C on $(Stream \bigcap S)$
        IF result $\neq$ FAIL: return it
    END
END
```

3 Next lecture

In the next lecture, we will go back to the problem of graph sketching based on the preliminary algorithms covered today. Graph sketching algorithm in [AGM12] guarantees the use of $O(\log^c n)$ spaces. The flow of idea is quite similar to the original paper's [AGM12], so it is recommended to read the paper.

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.

[GLPS12] Anna C Gilbert, Yi Li, Ely Porat, and Martin J Strauss. Approximate sparse recovery: optimizing time and measurements. *SIAM Journal on Computing*, 41(2):436–453, 2012.

[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.