CS 395T: Sublinear Algorithms Fall 2016

Lecture 8 — Sep. 20, 2016

Prof. Eric Price *Scribe: Ger Yang & Mei Wang*

1 Overview

In this lecture, we mainly focus on sketches for a fundamental sampling problem, known as l_0 -sampling. Afterwards, we begin to look at "exact" sparse recovery problem. Based on these, we will start graph sketching in the next lecture. This is a useful tool where we have subgraphs with streams of edges insertions and deletions, and we can compute some properties of this graph.

2 l_0 Sampling

As in turnstile stream model, given a vector $x \in \mathbb{R}^n$, an l_0 sampling of x is a uniform distribution over random samples from the non-zero indices of x, which is denoted as $supp(x)$. Now given $y = Ax$ as linear sketching model, our goal is to choose a sparse and randomized matrix $A \in \mathbb{R}^{m\times n}$, and output each $i \in supp(x)$ with probability

$$\frac{1 \pm \epsilon}{|supp(x)|} = \frac{1 \pm \epsilon}{||x||_0},$$

where ϵ is a constant in class.

If there is no other constraints, the algorithm is easy – just set A as an identity matrix and compute it. And we want to make the ϵ to be small in order to use sublinear space, and make A to be sparse to update and compute the Ax quickly so as to sketch in sublinear time.

We can solve this problem with

$$m = O(\frac{1}{\epsilon^2} \log^2(\frac{n}{\delta})).$$

So the question is, how can we do this?

2.1 Reduction to Insertion only turnstile model

The first step is to simplify the problem. We can get a simpler description of this problem as an insertion only turnstile model. As solved in previous class, we can use the LogLog algorithm in turnstile model. We choose a hash function: $h : [n] \to (0, 1)$. When it is fully independent, pick $i \in supp(x)$ minimizing $h(i)$. If it is not a truly random hash, select from a k-wise independent family of hash functions which suffices the "min-wise independence", that is $(\frac{1}{\epsilon^2})$ or $\log^c \frac{1}{\epsilon}$.

2.2 1-sparse recovery

We begin with a simpler case when $||x||_0 = 1$ instead of k.

Recall the Problem 2(b) in homework set 1: Give an algorithm that detects if x has a single non-zero entry, and if so finds that location. That is, give an algorithm to compute i if $x = e_i$, or compute $\perp$ otherwise with probability $1 - \delta$. Then the algorithm could be:

1. Check if $||x||_0 = 1$.
2. If so, output $i \in supp(x)$.

The method to check whether $||x||_0 = 1$ is: let $a = \sum x_i$, $b = \sum ix_i$, then $\frac{b}{a} \in supp(x)$ if $||x||_0 = 1$.

2.3 k-sparse recovery

If we know $||x||_0 = k$, then we can reduce it and use 1-sparse recovery as a black box.

1. Hash the coordinates of x into $B = O(k)$ buckets.
2. Run hw 2(b) on the first bucket, it can take $O(\log \frac{1}{\delta})$ words, which is $O(\log n \log \frac{1}{\delta})$ bits.
3. Then if the bucket is empty or at least 2, output $\perp$ with $1 - \delta$ probability; else output who leaves there with $1 - \delta$ probability.

Let

$$\begin{aligned} P &= \mathbb{P}[\text{Element } i \text{ is alone in the first bucket}] \\ &= \mathbb{P}[h(i) = 0 \cap \text{none of } h(j) = 0,\ j \in supp(x) \setminus i\ ] \\ &= \mathbb{P}[h(i) = 0] - \mathbb{P}[h(i) = 0 \cap \text{any } h(j) = 0,\ j \in supp(x) \setminus i] \end{aligned}$$

Now we want to get the bounds of P, so that each thing has a roughly equal chance of being found. Then the upper bound is:

$$P \leq \mathbb{P}[h(i) = 0] = \frac{1}{B}$$

And the lower bound is:

$$\begin{aligned} P &\geq \mathbb{P}[h(i) = 0] - k\,\mathbb{P}[h(i) = 0 \cap h(j) = 0, i \neq j] \\ &= \frac{1}{B} - k\frac{1}{B^2} \\ &= \frac{1}{B}(1 - \frac{k}{B}) \end{aligned}$$

Set $B \geq \frac{k}{\epsilon}$, then $P \in [\ \frac{1-\epsilon}{B}, \frac{1}{B}\ ]$. Then the chance of anything that is alone in the first bucket equals to $kP \in [\epsilon(1-\epsilon), \epsilon]$. Repeat for $O(\frac{1}{\epsilon} \log \frac{1}{\delta})$ times, then we can get $O(\frac{1}{\epsilon} \log \frac{1}{\delta} \log(\frac{1}{\epsilon\delta}))$.

2.4 General algorithm

What if we don't know k? Notice that we only use k to set B.

1. We need $B \geq \frac{k}{\epsilon}$ to be ϵ -accurate.
2. We need $B \leq \frac{k}{\epsilon}$ to output anything not equals to $\perp$ with no less than ϵ probability for each hash.

So if $B \leq c\frac{k}{\epsilon}$, the chance of any element can be found is not less than $\frac{k(1-\epsilon)}{B}$. Then run the algorithm for $k = 1, 2, 4, 8, \ldots$, then output element found by the largest k that finds anything. The space should be $O(\frac{\log n}{\epsilon} \log \frac{\log n}{\delta} \log(\frac{\log n}{\epsilon\delta}))$.

3 Exact Sparse Recovery

In the *sparse recovery* problem, we assume that the data stream $x \in \mathbb{R}^n$ is a k-sparse vector, i.e., $\|x\|_0 \leq k$. We are free to choose $A \in \mathbb{R}^{m\times n}$. Effectively, what we are observing is the vector $y = Ax$, which is on a lower dimension than x. Our goal for the sparse recovery problem is to recover x from y. In the *exact sparse recovery* problem, we would like to recover x exactly with high probability.

Before we think about how to solve this problem, we throw out the following questions:

1. How big should m be?
2. How fast can we do it?
3. Do we even need randomization?

For the first problem, it turn out $m = 2k$ words is sufficient, if the running time is not important. We show this claim in the following lemma:

Lemma 1. *Let $A \in \mathbb{R}^{m\times n}$ be a i.i.d. random Gaussian matrix. Suppose we are given $y = Ax$, where $\|x\|_0 \leq k$. If $m = 2k$, then we can recover x from y uniquely with probability* 1.

Proof. We prove this by contradiction. Assume with some positive probability, there is some other solution $x' \neq x$ such that $y = Ax'$ and $\|x'\|_0 \leq k$. We can see that the vector $x - x' \neq 0$ is in the null space of A:

$$A(x - x') = 0$$

Let $S = supp(x - x')$ be the support of $x - x'$. Owing to the k-sparsity of x and x', we have $|S| \leq 2k$. Denote A_S the matrix obtained by choosing each column from A if the index is in S. Also, for any vector $x \in \mathbb{R}^n$, x_S is defined analogously. Now we have

$$A_S(x - x')_S = 0$$

Since $x - x'$ is a nonzero vector, we can conclude that A_S has a linear dependence among its columns. If $m < 2k$, then the linear dependency is reasonable since A_S is not a full-rank matrix. If $m \geq 2k$, then the event that A_S has linearly dependent columns has probability measure zero. This contradicts to our assumption. As a result, $m = 2k$ is sufficient to recover x from y uniquely with probability 1. □

3.1 Fourier Matrix Approach

The first way to deal with the exact sparse recovery problem is the Fourier matrix approach. For example, if we choose A to be the Vandermonde matrix, it is well-known that with $2k$ samples, x can be recovered in polynomial time. The other ways to do the Fourier transform is to use the Berlekamp-Massey algorithm or the syndrome decoding for Reed-Solomon coding.

3.2 An $O(k \log k)$-Algorithm

We adapt the similar approach as we have done in l_0-sampling for the exact sparse recovery problem. More specifically, we hash the vector x into B buckets using a pairwise independent hash function. Then, we can see that there are two possible outcomes for each $i \in supp(x)$:

1. It is a good coordinate: there is no collision within the bucket it belongs to.
2. It is a bad coordinate: collision happens.

For each good coordinate $i \in supp(x)$, it turns out that we are able to get the correct estimate $\tilde{x}_i = x_i$ using the method we solve Problem 2b in the first problem set. Precisely, we can estimate $\tilde{x}_i$ through the following algorithm:

1. Let $S_j = \{i : h(i) = j\}$.
2. Compute $z'_j = \sum_{i \in S_j} i x_i$.
3. Compute $z_j = \sum_{i \in S_j} x_i$.
4. Let $\tilde{i} = z'_j / z_j$.
5. Set $\tilde{x}_{\tilde{i}} = z_j$.

If there is no collision in the j-th bucket, then $\tilde{i} = i$ and $\tilde{x}_{\tilde{i}} = x_i$. Next, we show that the probability that the i-th coordinate is good depends on the number of buckets B. The analysis basically follows from what we have done in the section of l_0-sampling. We let

$$\begin{aligned} p_{ij} &= \mathbb{P}[\text{Element } i \text{ is alone in bucket } j] \\ &= \mathbb{P}[h(i) = j \cap \text{There is no } k \in supp(x) \setminus i \text{ such that } h(k) = j] \\ &= \mathbb{P}[h(i) = j] - \mathbb{P}[h(i) = j \cap \text{There is some } k \in supp(x) \setminus i \text{ such that } h(k) = j] \end{aligned}$$

Since $p_{ij} \geq \frac{1}{B}\left(1 - \frac{k}{B}\right)$, we can see that

$$\begin{aligned} p_i &= \mathbb{P}[\text{Element } i \text{ is alone in his bucket}] \\ &= \sum_{j=1}^{B} p_{ij} \geq 1 - \frac{k}{B} \end{aligned}$$

which means the probability of i being a good coordinate is at least $1 - \frac{k}{B}$.

For bad coordinates, we show that the expected number of errors can be made small. We can see that for each bad coordinate, there is at most 1 more coordinate getting an error estimate. As a result, let t be the number of coordinates in the support of x that are not alone in the bin, and we can say

$$\|\tilde{x} - x\|_0 \leq 2t$$

Taking the expectation on both sides gives us

$$\mathbb{E}[\|\tilde{x} - x\|_0] \leq \mathbb{E}[2t] = 2\sum_{i=1}^{k}(1 - p_i) \leq \frac{2k^2}{B}$$

By setting $B = 4k/\delta$ and applying Markov's inequality we get

$$\|\tilde{x} - x\|_0 \leq \frac{k}{2} \quad \text{w.p. } 1 - \delta$$

This gives us the following lemma:

Lemma 2. *In $B = O(k/\delta)$ words, we can find $\tilde{x}$ such that $\|\tilde{x} - x\|_0 \leq k/2$ with probability at least $1 - \delta$.*

The next thing to do is to improve the accuracy by repeating the above procedure $\log k$ times. A naive analysis shows that this would require us $O(k \log^2 k)$ space to achieve a $3/4$ successful probability. However, we can use a trick to get a bound of using only $O(k)$ words instead of $O(k \log k)$ by working on the residues. Observe that in Lemma 2 we showed that, in $B = O(k/\delta)$ words and with high probability, the vector $\tilde{x} - x$ is $k/2$-sparse. If we further feed the vector $\tilde{x} - x$ into the algorithm described above and get $\tilde{x}^{(2)}$, we can find that $\|\tilde{x}^{(2)} - \tilde{x} + x\|_0 \leq k/4$ with high probability, which reduces the error from $k/2$ to $k/4$. This means that $\tilde{x} - \tilde{x}^{(2)}$ is a better estimate. This procedure can be formulated as the following algorithm

1. For $i = 1 \ldots \log k$, do
   (a) Generate $A^{(i)} \in \mathbb{R}^{\frac{8k}{2^i \delta_i} \times n}$
   (b) Find $A^{(i)}x$ and $A^{(i)}x^{(i-1)}$
   (c) Compute $\tilde{x}^{(i)} = A^{(i)}(x - \tilde{x}^{(i-1)})$.
2. Return $\tilde{x} = \sum_{i=1}^{\log k}(-1)^{i+1}\tilde{x}^{(i)}$

Theorem 3. *The above algorithm finds the exact sparse recovery in $O(k \log k)$ space with probability $3/4$.*

Proof. By Lemma 2, we can see that each iteration achieves $k/2^i$ error with probability at least $1-\delta_i$. Therefore, the algorithm returns the incorrect solution with probability at most $\delta = \sum_{i=1}^{\log k} \delta_i$. By setting $\delta_i = \left(\frac{3}{4}\right)^i \delta_1$ for each $i \in \{1, 2, \ldots, \log k\}$, we can see that $\delta = \frac{\delta_1}{1-3/4} = 4\delta_1$. Besides, the number of words we are keeping in the algorithm is

$$\sum_{i=1}^{\log k} O\left(\frac{k}{2^i \delta_i}\right) = O(k)\sum_{i=1}^{\log k}\left(\frac{2}{3}\right)^i = O(k)$$

By setting $\delta = 3/4$ we can get the desired result. $\square$