Sublinear Time and Space Algorithms 2016B – Lecture 5
ℓ_0-sampling and connectivity in dynamic graphs*

Robert Krauthgamer

1 ℓ_0-sampling

Problem Definition (ℓ_p-sampling): Let $x \in \mathbb{R}^n$ be the frequency vector of the input stream. The goal is to draw a random index from $[n]$ where each i has probability $\frac{|x_i|^p}{\|x\|_p^p}$.

We will see today the case $p = 0$, where the goal is to draw a uniformly random i from the set $\text{supp}(x) = \{i \in [n] : x_i \neq 0\}$.

Algorithms may have some errors either in the probabilities being approximately correct (e.g., $\pm\delta$) and/or that with some probability the algorithm gives a wrong answer (returns FAIL or a sample not according to the desired distribution).

Framework for ℓ_0-sampling [following Cormode and Firmani, 2014]:

(A) Subsample the coordinates of x with geometrically decreasing rates

(B) Detect if the resulting vector y is 1-sparse

(C) If y is 1-sparse, recover its nonzero coordinate.

(A) Subsampling:

The algorithm chooses a random hash function $h : [n] \to [\log n]$, such that for each $i \in [n]$,

$$\Pr[h(i) = l] = 2^{-l}, \qquad \forall l \in [\log n].$$

(The probabilities do not add to 1, and in the remaining probability we can set $h(i)$ to nil, i.e., no level.)

For each $l \in [\log n]$, create a virtual stream for $h^{-1}(l)$, formally define $y^{(j)} \in \mathbb{R}^n$ which is obtained from x by zeroing out coordinates outside $h^{-1}(l)$.

Observe that y is obtained from x by a linear map.

Lemma: If $x \neq 0$, then there exists $l \in [\log n]$ for which $\Pr[|\text{supp}(y)| = 1] = \Omega(1)$.

*These notes summarize the material covered in class, usually skipping proofs, details, examples and so forth, and possibly adding some remarks, or pointers. The exercises are for self-practice and need not be handed in. In the interest of brevity, most references and credits were omitted.

Proof: Was seen in class.

Exer: Show that whenever $\text{supp}(y)$ contains only one coordinate, that coordinate is indeed drawn uniformly from $\text{supp}(x)$.

Exer: Show how to achieve a similar guarantee using a hash function h that is only pairwise independent. (However, now the "surviving" coordinate might be non-uniform.)

The success probability can be increased to $1-\delta$ by $O(\log\frac{1}{\delta})$ repetitions for each level l. The result is a linear sketch of size (dimension) $O(\log n \log\frac{1}{\delta})$ words.

(C) Sparse recovery (of a 1-sparse vector): Suppose $y \in \mathbb{R}^n$ (which is $y^{(l)}$ from above) is 1-sparse. How can we find which coordinate i is nonzero?

Compute $A = \sum_i y_i$ and $B = \sum_i i \cdot y_i$ and report their ratio B/A.

For 1-sparse vector the output is always correct, as this step is deterministic.

Notice that A, B form a linear sketch whose size (dimension) is 2 words. Moreover, they can be maintained over the original stream x (no need to maintain the virtual stream y explicitly).

(B) Detection (if a vector is 1-sparse):

Lemma: There is a linear sketch to detect whether y is 1-sparse, using $O(\log n)$ words and achieving one-sided error probability $1/n^3$ (i.e., if $|\text{supp}(y)| = 1$ it always accepts, otherwise it accepts with probability at most $1/n^3$).

Proof: Was seen in class, using linearity of the AMS sketch.

Exer: Show how to improve the storage to $O(1)$ words by a more direct approach.

Hint: Use a linear map (of y) with random coefficients from $[-n^3, n^3]$. Or coefficients R^i for y_i where R is picked at random from a finite field of size $O(n^3)$.

Overall Algorithm:

The algorithm goes over the levels l in a fixed order, and reports the first coordinate that is recovered and passes the detection test (otherwise FAIL).

Storage: The total storage is $O(\log^2 n \log\frac{1}{\delta})$ words, not including randomness.

However, using limited randomness in the subsampling (necessary to reduce randomness) might introduce some bias to the uniform probabilities.

Variations of this approach: Detection and recovery of vectors with sparsity $s = 1/\varepsilon$ instead of $s = 1$, using k-wise independent hashing in the subsampling, or using Nisan's pseudorandom generator to reduce storage.

Theorem [Jowhari, Saglam, Tardos, 2011]: There is a streaming algorithm with storage $O(\log^2 n \log\frac{1}{\delta})$ bits, that with probability at most δ reports FAIL, with probability at most $1/n^2$ reports an arbitrary answer, and in all other cases produces a uniform sample from $\text{supp}(x)$.

2 Streaming of Graphs

Basic model: Consider an input stream that represents a graph $G = (V, E)$ as a sequence of edges on the vertex set $V = [n]$. Denote $m = |E|$.

It can be viewed as a sequence of edge insertions to a graph.

Remark: We will consider later a more general model that allows edges deletions (called dynamic graphs).

Semi-streaming: The usual aim is space requirement $\tilde{O}(n)$, which can generally be much smaller than $O(m)$, by trivially storing the current graph explicitly (though it does not account for extra workspace an algorithm may need).

For many problems, $\Omega(n)$ storage is required (even to get approximate answers).

Connectivity: Determine whether the graph G is connected (or even which pairs $u, v \in V$ are connected).

Can be solved in the insertions-only model with storage requirement $O(n)$ words.

Just store a spanning tree...

Distances: Maintain all the distances in the graph (between every pair $u, v \in V$).

Theorem: Can be solved within approximation $2k - 1$ (for integer $k \geq 1$) in the insertions-only model with storage requirement $O(n)$ words.

Just apply a greedy spanner construction by [Althofer, Das, Dobkin, Joseph and Soares, 1993].

3 Dynamic Graphs

Dynamic graph model: The input stream contains insertions and deletions of edges to G.

The tool of choice is linear sketching, where decrements are supported by definition.

Motivations:

a) updates to the graph like removing hyperlinks or un-friending

b) the graph is distributed (each site contains a subset of the edges), and their linear sketches can be easily combined

Theorem [Ahn, Guha and McGregor, 2012]: There is a streaming algorithm with storage $\tilde{O}(n)$ storage that can determine whp whether the graph is connected (or whether a pair of vertices are connected).

Idea: To grow (increase) connected components, we need to find an outgoing edge from each current set. Using ℓ_0-sampling and especially its linear-sketch form, we can pick an outgoing edge from an arbitrary set.

Notation: Let $N = \binom{n}{2}$ and for each vertex v, define the vector $x^v \in \mathbb{R}^N$ which is 0 except that

$$x_v(\{v,j\}) = \begin{cases} +1 & \text{if } (v,j) \in E \text{ and } v < j \\ -1 & \text{if } (v,j) \in E \text{ and } v > j \end{cases}$$

Algorithm AGM:

Update (on a stream/dynamic graph G): Maintain an ℓ_0-sampler for x_v for each vertex v (using the same coins, so that they can be added), but repeat this sampler $\log n$ independent copies.

Output (to determine connectivity): start with each vertex forming its own connected component (formally, a partition Π of V into n singletons). Now repeat the following $\log n$ times:

1. For each connected component $Q \in \Pi$, pick a random outgoing edge by summing-up (fresh copies) of one sampler for each $v \in Q$

2. Use the edges sampled in step 1 to merge connected components (parts in current Π)

Output "connected" if all the vertices are merged into one connected component.

Analysis: To simplify the analysis, we assume henceforth that G is connected (see below), and that the samplers are perfect (i.e. ignore their polynomially-small error probability).

Exer: Extend the analysis to the case that G is not connected, to determine whether s, t are connected.

Claim 1: In each iteration, if the number of connected components is $k > 1$ then at the end of the iteration it is at most $k/2$.

Exer: prove this claim

Claim 2: Fix a set $Q \subset V$. Then $z_Q = \sum_{v \in Q} x_v$ is nonzero only in coordinates corresponding (i,j) corresponding to an edge outgoing from Q, i.e., $|Q \cap \{i,j\}| = 1$.

Proof: Was seen in class.

Storage: The main storage is for ℓ_0-samplers for every vertex. Each one requires $O(\log^3 n)$ bits, and we need fresh randomness in each of the $O(\log n)$ iterations, to avoid potential dependencies. Thus the total storage is $O(n \log^4 n)$ bits.