College of Computer & Information Science
Northeastern University
CS 7880: Network Algorithms

Spring 2016
4 March 2016
Scribe: Giorgos Zirdelis

- Wrap up spectral sparsifier
- Streaming
  - Model
  - Frequent items
  - Distinct items

1 Spectral sparsifier

Continuing from the previous lecture, we apply the Chernoff bounds from [Tro12] for random variable $X = \sum_{e,i} X_i$ whereas the X_i's are defined as follows:

$$X_i = \frac{Z_{e,i}}{\rho}\left(\frac{x_e x_e^T}{x_e^T x_e}\right)$$

For $\rho = \frac{c \ln n}{\varepsilon^2}$ we obtain

$$\Pr[\lambda_{\max}(X) \geq 1 + \varepsilon] \leq \exp\left(\frac{-\varepsilon^2 \rho}{3R}\right) = \frac{1}{n^{c/3}}$$

For all e, i because all eigenvalues of $\frac{x_e x_e^T}{x_e^T x_e}$ are at most one, it follows that the eigenvalues of $\frac{Z_{e,i}}{\rho}\left(\frac{x_e x_e^T}{x_e^T x_e}\right)$ are between $[0, 1/\rho]$.

2 Streaming

The motivation for streaming algorithms comes from processing of big data, e.g. large graphs with a billion nodes. We want to process the data fast, i.e. usually read once the stream and spent as little time as possible for each element of the stream. Additionally, we have limited space (memory) available while processing the stream, i.e. we cannot store the entire stream.

More formally, assume we have a stream σ of m elements, i.e. $\sigma = \langle a_1, \ldots, a_m \rangle$, where each element is drawn from a universe $\mathcal{U} = [n] = \{1, 2, \ldots, n\}$ and $m \gg n$. We are asked to compute a function $\phi(\sigma)$ of the stream.

The stream σ defines a frequency vector for each element of the universe $f = \langle f_1, \ldots, f_n \rangle$. So ϕ is actually a function of f as for the problems we are going to see, order is not important.

2.1 Frequent items

We start with the problem of finding all elements in a stream with frequency greater than $\frac{m}{k}$ for a fixed constant k, if such elements exist. That is, find all elements in $\{i : f_i > \frac{m}{k}\}$. For $k = 2$ the problem is known as finding the majority element.

We can solve this problem by going through the stream and keeping a counter for each element of the universe but then we might need $n \cdot \log m$ bits of space.

To solve this problem with less space, we start with an empty array of k rows and 2 columns. In the first column we store an element and in the second row its respective score. When a element comes, if it is in the array we increase its score by one. If it is not in the array and there is free space, we insert it with score of one. If it is not in the array and there is no free space, we decrease all scores in the array by one. If we reach a zero score we remove that element from the array. This algorithm is due to Misra and Gries [MG82].

For all $i \in \mathcal{U}$, define $\hat{f}_i$ to be the score of i-th element at the end of the stream. It holds that for all i,

$$f_i - \frac{m}{k} \leq \hat{f}_i \leq f_i$$

While the inequality $\hat{f}_i \leq f_i$ follows from the definition of $\hat{f}_i$, the argument for the other inequality is the following: for each decrement we need to make if the array is full, we "point" to (at most) k array elements. The stream σ has m elements, therefore there can be at most $\frac{m}{k}$ such decrements for each element of the array.

The space complexity for this algorithm is $O(k(\log n + \log m))$ bits. We need $\log n$ bits for representing a stream element and $\log m$ bits for its score value.

At the end of the previous algorithm all elements with $f_i > \frac{m}{k}$ will have a positive $\hat{f}_i$ score. To get an exact answer to the problem, we need to do a second pass on the stream. We count the cardinality of the elements that have $\hat{f}_i > 0$ on the first run of the algorithm, and see which ones have $f_i > \frac{m}{k}$.

2.2 Distinct items

Since we are interested a lot in efficiency, a streaming algorithm can use randomness and/or give approximate answers. Specifically, assume that $\mathcal{A}$ is a randomized algorithm with output $\mathcal{A}(\sigma)$. We say that $\mathcal{A}$ is a (ε, δ) approximation of ϕ, if for all streams it holds that

$$\left| \frac{\mathcal{A}(\sigma)}{\phi(\sigma)} - 1 \right| \leq \varepsilon \quad \text{or equivalently} \quad (1-\varepsilon)\phi(\sigma) \leq \mathcal{A}(\sigma) \leq (1+\varepsilon)\phi(\sigma)$$

with probability $1 - \delta$ for some $\varepsilon, \delta > 0$. This defines a multiplicative approximation.

Assume we have a stream σ and want to find the number of the distinct elements, or give an estimate. Notice that we are interested only in elements with $f_i > 0$. An algorithm to solve the problem approximately is due to Alon, Matias and Szegedy [AMS96], and the idea is the following: We choose a hash function $h : \mathcal{U} \to \mathcal{U}$. We hash each element of the stream and keep track of the

most consecutive trailing zeros of $h(a_i)$ (least significant bits) for all i, for which we need $\log n$ bits. If r is that number, then return 2^r.

References

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29. ACM, 1996.

[MG82] Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.

[Tro12] Joel A. Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of Computational Mathematics*, 12(4):389–434, 2012.