Algorithmic Lower Bounds - Streaming Algorithms

Matt Kovacs-Deak

December 11, 2021

As suggested by the name streaming algorithms sequentally receive as input some (very long) data stream $\boldsymbol{x} = (x_1, ..., x_n)$, and they are required to process it with limited amount of memory, and limited processing time per datapoint x_i. Typically streaming algorithms are built to answer some question about a (global) property of the stream $\boldsymbol{x}$, and as such one can conceptualize streaming algorithms as ones that work by creating some kind of a "summary" of their input data $\boldsymbol{x}$. One example is the following: given a stream $\boldsymbol{x}$ with $x_i \in [m]$ such that there is precisely one element $k \in [m]$ that doesn't occur in $\boldsymbol{x}$, identify k. This of course would be trivial if one is allowed $O(m)$ memory, however the problem becomes interesting with the restriction that one is only allowed $O(\sqrt{m})$ memory. There is an efficient algorithm for this problem, finding it servers as an easy exercise for the reader.

Another, equivalent, albeit somewhat less natural way of formulating a streaming algorithm is the following. Starting with some initilly all zero vector $\boldsymbol{a} = (0, 0, ..., 0)$, the algorithm receives a stream $x_i = \langle j_i, c_i \rangle$. On each such update it updates the *occurancy count* of element j_i, that is it assigns $a_{j_i} \leftarrow a_{j_i} + c_{j_i}$. Given a vector $\boldsymbol{a} = (a_1, ..., a_m)$ some typical measures of interest are,

(i) Frequency moments: $F_k(\boldsymbol{a}) = \sum_j a_j^k$

(ii) Heavy hitters: $\{j : a_j > k\}$

(iii) Number of distinct elements: $\big|\{j : a_j \neq 0\}\big|$

(iv) Entropy: $E(\boldsymbol{a}) = \sum_j a_j/M \log a_j/M$ for $M = \sum_j a_j$

Notice that $F_1(\boldsymbol{a})$ is just the number of elements in the dataset.

Streaming graph algorithms

In the context of streaming algorithms concerning graphs, the vertex set of the graph is often assumed fix. With these constraint, there are two main classes of graph streaming. In *insertion only* algorithms, as the name suggests the graph "keeps growing". On the other hand, for dynamic algorithms edges can be both added and removed with each datapoint. As an example, we consider the Maximum Matching problem. Here we are given a graph $G = (V, E)$ and we need to find a matching with maximal size. We can classify streaming algorithms for the Maximum Matching problem by (1) the number of passes it makes; (2) memory it uses; (3) and the approximation factor it achieves. The following claim gives our first example of a streaming algorithm,

Claim 1. *There is a single-pass algorithm that (1) decides on (& commits to) each edge upon receipt; (2) requires $\tilde{\mathcal{O}}(n)$ amount of space; (3) and achieves an approximation factor of $1/2$.*

Proof (sketch). Initially store all the vertices using $\mathcal{O}(\log n)$ space, keeping them unmarked. Initialize $M \leftarrow \emptyset$. On receipt of edge (x, y) add it to the matching M if both x, y are unmarked. Mark x and y. ∎

There are a number of open questions regarding MAXIMUM MATCHING:

- Does the above algorithm achieve the best possible approximation factor for any single-pass algorithm with $\tilde{O}(n)$ memory? Konrad *et al.* shows that it is possible to achieve a better than 2 approximation if multiple passes are allowed.
- Is there an algorithm that achieves a constant or polylog approximation factor using $o(n)$ space only?
- What's the best we can do for bipartite graphs? Feigenbaum *et al* proved that for any $0 < \epsilon < 1/3$ there is a streaming algorithm with $\mathcal{O}(\epsilon^{-1} \log \epsilon^{-1})$ passes that achieves a $(2/3 - \epsilon)$ approximation factor on bipartite graphs.

For streaming algorithms many of the hardness results come from communication complexity.

Communication Complexity

We will discuss some results from communication complexity that will serve as building blocks for our hardness proofs. For a more complete exposition, the reader is advised to consult the book of Kushilevitz & Nisan. Assume two distant parties (Alice and Bob) each have a bitstring of length n (i.e. $x, y \in \{0,1\}^n$, respectively). How many bits would Alice and Bob need to share to determine if x=y? One trivial protocol involves sending $(n+1)$ bits. The following exercise gives two known results,

Exercise 2.

(i) Prove that any deterministic protocol requires at least n bits to determine $x \stackrel{?}{=} y$.

(ii) Show that there is a randomized protocol for determining $x \stackrel{?}{=} y$ that uses private coins, $\mathcal{O}(\log n)$ bits, and achieves a probability of error at most $1/n$.

Next, we formalize the notion of a *2-party communication protocol*[1].

Definition 3. *Let $f : \{0,1\}^{2n} \to \{0,1\}$ be some function. A t-round 2-party communication protocol Π for computing f is a seqeunce of t functions $P_1, ..., P_t : \{0,1\}^* \to \{0,1\}^*$. On inputs $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^n$ the scheme proceeds by: Alice computes $p_1 = P_1(\boldsymbol{x})$ and sends $p_1 \in \{0,1\}$ to Bob, Bob computes $p_2 = P_2(\boldsymbol{y}, p_1)$ and sends p_2 to Alice, etc. In the i^{th} round Alice (Bob) computes $p_i = P_i(\boldsymbol{z}, p_1, ..., p_{i-1})$ for $\boldsymbol{z} = \boldsymbol{x}$ ($\boldsymbol{z} = \boldsymbol{y}$) and send p_i to Bob (Alice), provided that i is odd (even). The protocol Π is valid if the last message sent p_t is equal to $f(\boldsymbol{x}, \boldsymbol{y})$ for any pair of inputs $\boldsymbol{x}, \boldsymbol{y}$.*

Given a t-round 2-party protocol Π we define the communication complexity of Π to be the maximum number of bits communicated, i.e. the maximum of the quantity $\sum_i |p_i|$, where the maximum is taken over all the possible inputs $\boldsymbol{x}, \boldsymbol{y}$. Given a function $f : \{0,1\}^{2n} \to \{0,1\}$ the communication complexity of f, denoted $C(f)$, is defined to be the minimum communication complexity of any protocol Π that is valid for f.
Next we consider some well-known problems in communication complexity,

[1]The below definition is due to Arora & Barak. See `https://www.cs.princeton.edu/courses/archive/spring08/cos598D/communicatechap.pdf`.

- INDEX: Alice receives a string $x \in \{0,1\}^n$ and Bob receives an integer $k \in [n]$. Bob would like to learn the value of x_k. We enforce the additional constraint that the protocol must be *one-way*, that is only Alice is allowed to send bits.
- INDEXSAME: Alice receives a string $x \in \{0,1\}^n$ and Bob receives an integer $k \in [n-1]$. Bob would like to learn if $x_k = x_{k+1}$. As above, we require that the procol be one-way: only Alice can share information.
- DISJOINTNESS: Alice and Bob receive n-bitstrings x and y respectively. They want to determine if the sets represented by x and y (eg. $\{j \in [n] : x_j = 1\}$) are disjoint.

The following bounds are known for these problems,

Theorem 4. *Each of the following lower bounds on communication complexity hold for both deterministic and randomized protocols:*

(i) In the one-way communication model INDEX *has communication complexity* $\Omega(n)$.

(ii) In the one-way model INDEXSAME *requires* $\Omega(n)$ *bits.*

(iii) DISJOINTNESS *requires* $\Omega(n)$ *bits in the two-way model.*

The following steps can be used to prove that $o(n)$ bits are not enough for INDEXSAME. First, show that deterministic one-way protocols require at least n bits for INDEX. Next, prove that if INDEXSAME can be done using $o(n)$ bits then so too can INDEX be done. Finally, conclude that INDEXSAME cannot be done using $o(n)$ bits.

Lower Bounds on Graph Streaming Algorithms

To prove lower bounds for the space requirements of streaming algorithms for graphs, we will produce reductions from the three communication complexity results introduced above. In these notes, our main focus will be determinstic algorithms.

The first problem we consider is the MAXIMUM CONNECTED COMPONENT problem. This problem is parameterized by some integer $k \in \mathbb{N}$ and takes as its input a simple graph $G = (V, E)$. The task is to decide if there is a connected component of G of size k? Our first result is the following,

Theorem 5. *Any single-pass streaming algorithm that solves* MAXIMUM CONNECTED COMPONENT *for some* $k \geq 3$*, requires* $\Omega(n)$ *space.*

Proof (sketch). From INDEX via a construction, see fig. 1 for a hint. ∎

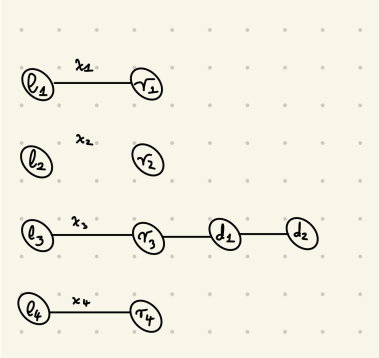

Figure 1: An example demonstrating the reduction from INDEX to MAXIMUM CONNECTED COMPONENT with $k = 4$. The construction is given for INDEX with input $\boldsymbol{x} = 1011$ and $i = 3$.

Next, we consider the problem called isTREE which is the problem of deciding whether or not a given graph $G = (V, E)$ is a tree. We have the following result,

Claim 6. *Any single-pass algorithm for* isTREE *requires* $\Omega(n)$ *space.*

Proof (sketch). Reduce from INDEXSAME similarly to the reduction given for theorem 5. ∎

A closely related problem is TREE DIAMETER. This is parameterized by an integer $k \leq 3$. Here, given a graph $G = (V, E)$ which is a tree, the task is to determine if its diameter is at least k.

Exercise 7. *Prove that any single-pass algorithm needs* $\Omega(n)$ *memory for* TREE DIAMETER.

Next, we consider the PERFECT MATCHING problem. Here given a graph $G = (V, E)$, the task is to decide if G has a perfect matching. For this we have the following result,

Claim 8. *Any single-pass streaming algorithm for* PERFECT MATCHING *requires* $\Omega(n^2)$ *space.*

Proof (sketch). Reduce from INDEX where the bitstring considered is viewed as an $n \times n$ matrix of bits, and Bob wants to learn the value of the entry in position (i, j). ∎

Our final example graph problem is SHORTEST PATH: given some simple graph $G = (V, E)$ together with two vertices $u, v \in V$, what is the length of the shortest u-v path in G? We have the following result,

Claim 9. *Any single-pass algorithm that approximates* SHORTEST PATH *with an approximation factor better than* $5/3$ *requires* $\Omega(n^2)$ *memory.*

Proof (sketch). Similar to the proof of claim 8. ∎

We close this notes with some computational problems concerning frequency numbers.

Frequency Moments

Given a stream of numbers $y_1, ..., y_m \in [n]$, let x_k be the frequency of the number $k \in [n]$. Then, as mentioned earlier (for $p \in \mathbb{R}^+ \cup \{\infty\}$) the p^{th} frequency moment of this datastream is,

$$F_k(x) = \begin{cases} \sum_i x_i^p, & \text{if } p \neq \infty \\ \max x_i, & p = \infty \end{cases} \tag{1}$$

The following results are known concerning frequency moments,

Claim 10.

(i) For any $p \in [0,2]$, there is a randomized streaming algorithm which $(1+\epsilon)$-approximates $F_p(x)$ using $O(\text{poly}(\log n, \log m))$ space.

(ii) For any $p > 2$, any randomized streaming algorithm that $(1+\epsilon)$-approximates $F_p(x)$ requires $\Omega(m^{1-2/p})$ space.

(iii) For any $p \in [0,2]$, there is exists a randomized streaming algorithm which $(1+\epsilon)$-approximates $F_p(x)$ using $O(m^{(1-2/p)})$ space.

Further reading

- Given an ordered sequence of numbers $\boldsymbol{x} = (x_1, ..., x_n)$ the Longest Increasing Subsequence problem is the problem of identifying an increasing subsequence (ie. a subsequence x_{i_j} of (x_i) with $x_{i_j} \leq x_{i_{j+1}}$ for all j) that is of maximal length. In 2012, Saks & Seshadhri gave [5] a deterministic, single-pass streaming algorithm for additively approximating the Longest Increasing Subsequence problem to within an additive δn for any given $\delta > 0$ using $O(\log^2 n/\delta)$ memory. They also considered the Longest Common Subsequence problem[2] and gave an analogous result for that one as well.

- In 2021, Chen *et al.* gave [1] a single-pass streaming algorithm for the Maximum Weight k-Matching problem. Given some weighted graph G, together with the parameter $k \in \mathbb{N}$, their probabilistic algorithm produces with high probability a maximum-weight k-matching in the input graph, using $O(k^2)$ space.

- Czumaj *et al.* recently considered [3] the Geometric Steiner Forest problem with k color classes, and the inputs on some discrete grid $[m] \times [m]$. They gave a single-pass streaming algorithm that requires $\text{poly}(k \cdot \log m)$ memory and estimates the cost of an optimal Steiner forest with an approximation ratio that can be made arbitrarily close to a certain constant, the so-called *Euclidean Steiner ratio*[3] α_2.

- Recall the **NP**-hard problem of Maximum Coverage: given some collection of subsets S_i of some finite universe Σ together with some parameter k, the task is to select k subsets such that their union has maximal cardinality. There is a straightforward greedy $(1 - e^{-1})$-approximation algorithm that runs in polynomial time. In the streaming model, McGregor & Tu gave two algorithms for the streaming formulation of Maximum Coverage, each achieving an $(1 - e^{-1} - \epsilon)$-approximation [4]:

  (i) A single-pass algorithm that requires $\tilde{O}(\epsilon^{-2}m)$ space, where m is the number of subsets considered.

  (ii) A multi-pass algorithm that requires $\tilde{O}(\epsilon^{-2}k)$ space and $\tilde{O}(\epsilon^{-1})$ passes.

- Chitnis, Cormode, Hajiaghayi, and Monemizadeh considered *parameterized* streaming algorithms for the Vertex Cover problem [2]. They proved a tight lower bound of $\Omega(k^2)$ for the space complexity of the parameterized Vertex Cover problem for any randomized streaming algorithm (both dynamic, and insert-only).

[2] Here, given two strings $\boldsymbol{x}$ and $\boldsymbol{y}$ the goal is to identify a maximal sequence that is a subsequence of both strings.
[3] The value of α_2 is between $1.1547 \leq \alpha_2 \leq 1.214$.

References

[1] J. Chen, Q. Huang, I. Kanj, and G. Xia. Optimal Streaming Algorithms for Graph Matching. *arXiv e-prints*, page arXiv:2102.06939, Feb. 2021.

[2] R. Chitnis, G. Cormode, M. Hajiaghayi, and M. Monemizadeh. Parameterized Streaming Algorithms for Vertex Cover. *arXiv e-prints*, page arXiv:1405.0093, May 2014.

[3] A. Czumaj, S. H. C. Jiang, R. Krauthgamer, and P. Veselý. Streaming Algorithms for Geometric Steiner Forest. *arXiv e-prints*, page arXiv:2011.04324, Nov. 2020.

[4] A. McGregor and H. T. Vu. Better Streaming Algorithms for the Maximum Coverage Problem. *arXiv e-prints*, page arXiv:1610.06199, Oct. 2016.

[5] M. Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. *arXiv e-prints*, page arXiv:1204.1098, Apr. 2012.