Streaming Algorithms and their lower bounds

→ Streaming Algorithms are often used to process data streams of Big Data.
→ Big Data nowadays are everywhere: gene regulatory networks, brain networks, health/disease networks, and online social networks like Facebook, Google+, Twitter
→ most of these big data (networks) are not static and evolves over the time.
→ we often need algorithms which manipulate the data immediately as it streams (by using relatively little memory, much less than the input size and also limited processing time per item)
→ the stream of the input can be a sequence of items such as numbers or edges of a graph
→ we often produce approximate answers based on summary or "sketch" of the data stream in memory.
→ A common technique for creating a sketch is sampling at Random
→ The field of streaming algorithms was popularized esp. by a paper of Alon, Matias and Szegedy which won Gödel Prize
→ when the streaming are items, we often have a vector $a=(a_1,\dots,a_n)$ initialized to the zero vector $\vec{0}$ and the input is a stream of updates in the form $\langle i,c\rangle$ in which a_i is incremented by some possibly negative integer c.
→ In the semi streaming model for graphs, the input is the stream of edges (and if an edge appears with a negative sign, it means deletion.)
→ there is a more general <u>sliding window</u> model as well, in which the function of interest is computing over a fixed-sized window of the stream (as data streams new items added to the window and items from the end of window are deleted.
→ In some cases we might have ability of several passes over the stream.
→ streaming algorithms have many similarities with online algorithms since they both require decisions before seeing all data but they are ~~identical~~ NOT since streaming algorithms can defer actions though they have limited memory.

Some important streaming problems for items:

1- the kth frequency moment, i.e., $F_k(\vec{a}) = \sum_{i=1}^{n} a_i^k$.

2- Heavy hitters: Find all elements i whose frequency $a_i > T$

3- Counting distinct elements

4- Entropy $E(\vec{a}) = \sum_{i=1}^{n} \frac{a_i}{m} \log \frac{a_i}{m}$ where $m = \sum_{i=1}^{n} a_i$

→ Note that in all streaming algorithms, we want to minimize space (and then update time) even through multiple passes.

→ The accuracy of the algorithm is often stated as an (ε, δ) approximation meaning that the algorithm achieves an error of less than ε with probability $1-\delta$.

Lower bounds

The most common technique for computing lower bounds are communication complexity. Indeed we have lots of interesting lower bounds for massive graph problems and the semi-streaming model in which edges E are coming in the stream (often in adversarial order, but some time in a random order) and the storage space of an algorithm is bounded by $\tilde{O}(n) = O(n \text{polylog} n)$, where $n = |V|$.

Let's see a simple graph streaming algorithm.

Streaming algorithm for matching:

[Maximal matching Alg.]

Whenever a new edge is coming if both end points are free, add it to matching

The space is $O(n)$ [semi-streaming] and approximation factor is 2. [each seleted edge ruins at most two edges of the opt]

For open problem (1): All [Konrad, Magniez, Mathieu show - 2 passes are enough - Possible for random ordering]

Two important open problems just here:

1) Can we get a constant factor better than 2 with $o(m)$ space, where $m = |E|$?

2) Can we get a constant or even polylog approximation factor of the SIZE of matching in $o(n)$ space?

Note that above problems are interesting for one passe even for bipartite graphs otherwise e.g. for any $0 < \varepsilon < \frac{1}{3}$ and a bipartite graph, there is a $\frac{2}{3} - \varepsilon$ approx. factor in $O(\frac{\log \frac{1}{\varepsilon}}{\varepsilon})$ passes [Feigenbaum, Kannan, McGregor, Suri, Zhang '05]

Now let's see some hardness proofs, but first we need a bit of communication complexity.

Let $f: X \times Y \to Z$ be a function. The (2-party) communication model consist of two players, Alice & Bob. Alice is given an input $x \in X$ and Bob is given an input $y \in Y$ and they want to compute $f(x,y)$. Alice does not know y while Bob does not know x; thus they need communicate (exchange bits) according to an agreed upon protocol. The communication complexity of f is the minimum over all communication protocols of the maximum over all $x \in X$ and all $y \in Y$ of the number of bits that need to be exchanged to compute $f(x,y)$.

The protocol can be deterministic or randomized. If one player sends information it is called one-way protocol (we need to specify which player is the sender and which player is the receiver). In this case only the receiver needs to be able to compute f.

Two well-known problem from the area of communication complexity are INDEX & Disjointness

1) INDEX: Let Alice have a string $x \in \{0,1\}^n$ and Bob have a natural number $i \in [n]$. Bob wants to compute $\text{Index}(x,i) = x_i$ by receiving a single message from Alice (one-way complexity model)

It is well-known that $\Omega(n)$ bits is needed to be exchanged in this case. (see FKMSZ'05 and FKMSZ'08 for problems in graph streaming via this problem we mention some here as well).

(2) DISJOINTNESS: Let Alice have a string $x \in \{0,1\}^n$ and Bob have a string $y \in \{0,1\}^n$. Bob wants to compute whether there is an index i such that $x_i = y_i = 1$, i.e., Disjointness(x,y) = "is there an $1 \le i \le n$ such that $x_i = y_i = 1$?" by exchanging as many messages as needed (multi-way complexity) model. We might have also $\sum x_i = \sum y_i = \lfloor \frac{n}{4} \rfloor$

Again it is well-known that we need $\Omega(n)$ bits to be exchanged between Alice & Bob even over multi passes (the main difference to INDEX). Thus this problem is useful for proving hardness for graph streaming even in the multi-pass model. Note that above lower bounds work even when we have randomized protocols that answers $f(x,y)$ correctly with prob. $\frac{2}{3}$, where prob. is taken over all public coin toss.

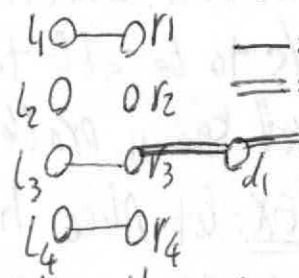
let's see some example now:

Max-Conn-Comp(k): let $G(V,E)$ be a forest. Is there a connected component of size at least $k \geq 3$ in G, for any given k.

Thm: (☆) Any single-pass streaming graph algorithm solving Max-Conn-Comp(k) on forests needs $O(n)$ space.

Pf: We give a reduction from INDEX to Max-Conn-Comp(k). Assume there is an algorithm A with space $o(n)$ for Max-Conn-Comp(k). Let's construct a graph $G = (V = V_l, V_r, V_d, E = E_{Alice} \cup E_{Bob})$ of Max-Conn-Comp(k)

let $V_l = \{l_1, \dots l_n\}$, $V_r = \{r_1, \dots r_n\}$, $V_d = \{d_1, \dots d_{k-2}\}$ and $E_{Alice} = \{\{l_j, r_j\} | x_j = 1, j \in [1,n]\}$ and $E_{Bob} = \{\{r_i, d_1\}\} \cup \{\{d_j, d_{j+1}\} | j \in [1, k-3]\}$. See Fig 1 for $IN(1011, 3)$ and $k=4$



First note that by construction G is a forest and

Max-Conn-Comp(k) = 1 $\Leftrightarrow$ Index(x, i) = 1

(if Index(x,i) = 0, then the max component has size k-1).

Now let's assume we stream edges of E_{Alice} and run algorithm A on it and as a result we have a space S of $o(n)$ as the working memory. Alice sends to Bob. Now Bob is streaming his edges by continue running A on the stream intialized by space S that he got from Alice already. Now at the end we have the execution of A on G and thus Max-Conn-Comp(k) = 1 $\Leftrightarrow$ Index(x, i) = 1. But here we sent $o(n)$ bits from Alice to Bob which is a contradiction that INDEX needs $O(n)$ bit exchange. Note that here we assumed both Alice & Bob have a common knowledge about Algorithm A which is fine for INDEX. The hardness of INDEX comes from the fact that Alice does not know i. (Public)

let's see another example:

Is-Tree. let $G = (V, E)$ be a graph. Is G a tree?

First let's introduce another problem in communication complexity.

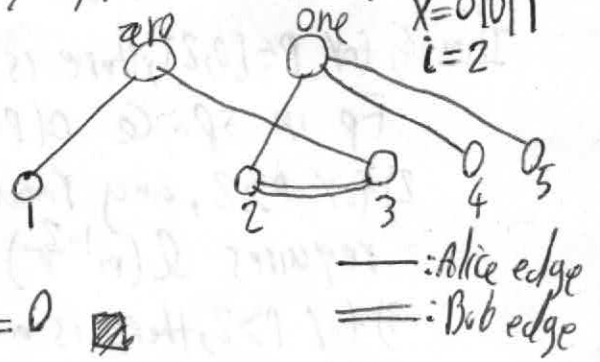
INDEX-SAME: let Alice have a string $x \in \{0,1\}^n$ and Bob have a natural number $i \in [n-1]$. Bob wants to compute Index-Same(x,i) = "is $x_i = x_{i+1}$?" by receiving a single message from Alice.

Thm: Any single-pass protocol for Index-Same needs $\Omega(n)$ memory.

Pf: we give a reduction from Index(x,i) to Index-Same(x',i').
Let $x' = x'_1 \cdots x'_n$ where $x'_i = 01$ if $x_i = 0$ and $x'_i = 11$ otherwise and let $i' = 2(i-1)+1$.
It is easy to see that Index$(x,i)=1 \iff$ Index-Same$(x',i')=1$. □

Thm: Any single-pass streaming algorithm solving Is-tree needs $\Omega(n)$ space.

Pf: the strategy of the proof is very similar to proof of Thm(★) except we use a reduction from Index-Same to Is-tree. Let G be a graph with $V = \{zero, one, 1, \ldots, n\}$ and $E = E_{Alice} \cup E_{Bob}$, where $E_{Bob} = \{i, i+1\}$ and $E_{Alice} = \{\{i, zero\} \mid x_i = 0\} \cup \{\{i, one\} \mid x_i = 1\}$.



Note that we have $n+2$ vertices and $n+1$ edges in G. Thus either G is a tree or have a cycle. Indeed it is easy to see
Is-tree(G) = YES $\iff$ INDEX-SAME$(x,i) = 0$ ▨

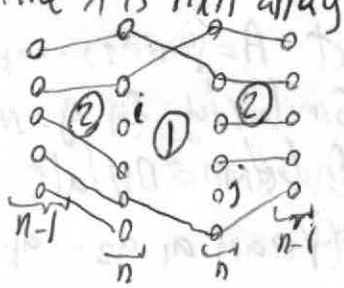
TREE-DIAM(K): let $G(V,E)$ be a tree. Is the diameter of G is at least $K \geq 3$?

Exercise: prove any single-pass streaming Alg. for TREE-DIAM(K) needs $\Omega(n)$ memory.

PerfectMatching(G)? is there a perfect matching in G.

Thm: Any single-pass streaming Alg for Perfect_Matching needs $\Omega(m) = \Omega(n^2)$ memory.

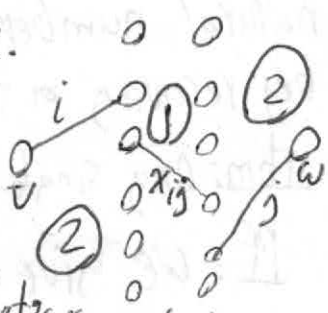
Pf: Again via a reduction from Index. This time assume x is $n \times n$ array and Bob wants to know x_{ij}, i.e., index(i,j). Thus we need space $\Omega(n^2)$. Here $e_{ij} \in E$ iff $x_{ij} = 1$ and there is no edge to i and j in edge set ② iff Bob index is (i,j). We stream edgeset ① (for Alice) and ② for Bob and thus we can solve INDEX ★



shortest-path(v,w): what is the length of shortest path from v to w?

Thm: Approximation better than $\frac{5}{3}$ requires $\Omega(n^2)$ space.

Pf: The proof is essentially the same as previous proof with the following graph. stream = ① ②. ∎



Now let's see an example for Disjointness & Frequency problem. Let's remember:

Frequency Moments: Given a data stream $a_1, \dots a_n$ where each $a_i \in [m]$, the frequency of $i \in [m]$ in the stream is $x_i = |\{j : a_j = i\}|$. The vector $x = (x_1, x_2, \dots x_m)$ is called the frequency vector. The pth frequency moment of the input is defined as follows:

$$F_p = \begin{cases} |\{i : x_i \neq 0\}| & \text{if } p = 0 \\ \max_i x_i & \text{if } p = \infty \\ \|x\|_p^p = \sum x_i^p & \text{otherwise.} \end{cases}$$

Note that F_0 is the number of distinct elements in the stream while F_1 is the number of elements (with repetition)

Indyk and Woodruff [IW05] prove the following:

Thm: 1) for $p \in [0,2]$, there is a randomized streaming algorithm that $(1+\varepsilon)$-approximates F_p in space $O(\text{poly}(\log m, \log n))$.

2) for $p > 2$, any randomized streaming algorithm that $(1+\varepsilon)$ approximates F_p requires $\Omega(m^{1-\frac{2}{p}})$ space

3) for $p > 2$, there is a randomized streaming algorithm that $(1+\varepsilon)$-approximates F_p in space $O(m^{1-\frac{2}{p}})$.

All lower bounds above are done via similar communication complexity mentioned in this lecture, let's see a simple one for F_∞

Thm Any randomized streaming algorithm that computes F_∞ requires $\Omega(m)$ space.

Pf: It is by a reduction from Disjointness. since we know that any randomized protocol for computing disjointness requires $\Omega(n)$ space, the theorem follows.

Let $A = \{a_1, a_2, \dots a_k\}$ be the set of indices i that $x_i = 1$. Define $B = \{b_1, b_2, \dots b_k\}$ Similarly for y. Note that since length of x and y are both n, $A, B \subseteq [n]$. Alice & Bob then simulate the streaming algorithm R with space C on the data stream $a_1\ a_2 \dots a_k\ b_1 b_2 \dots b_k$. Note that Alice has only the first half of

half of the stream, while Bob has only the second half. Alice runs the streaming algorithm on her part $(a_1, a_2, \ldots a_k)$ and passes the snapshot of the memory used at this point to Bob which is of size at most C. Bob will continue running the algorithm till the end of the input. Note that $m = n$ and F_∞ for this stream is either 1 or 2 depending on whether the sets are disjoint or not. Thus Alice and Bob can determine if the sets are disjoint using the output of the algorithm. ▨

→ Note that the proof shows that for every $\epsilon > 0$, even $2-\epsilon$ approximation F_∞ is not possible in $o(m)$ space.

→ Also the above reduction works even in case of a streaming algorithm that makes multiple (say constant p) passes on the input. In this case we get a protocol where at most $O(pC)$ bits are exchanged. Hence the above proof rules out even multipass streaming algorithms that use small space for this problem since Disjointness hardness works for multi-rounds as well.

Another example of use of Disjointness for Max-Conn-Comp(k).

Exercise: Any multi-pass streaming graph algorithm solving Max-Conn-Comp(k) on forest graphs needs $O(n)$ memory space.

Hint: use Disjointness instead of Index for reduction (Now x_i to $d_{i,1}$ is one iff $y_i = 1$).

Some more hardness results:

Thm: [Guruswami, Onak '13]
solving the following graph problems in p passes requires $\Omega\left(\frac{n^{1+\Omega(1/p)}}{p^{O(1)}}\right)$ bits of space

undirected graphs: Are v and w at distance at most $2(p+1)$?
Is there a perfect matching? (implies the result for size as well)

Directed graphs: Is there a directed path from v to w?

Thm [FKMSZ'05]: ([FKMSZ'04] already showed lower bound $\Omega(n^2)$ for one pass)
t-approximation of distance between two nodes in one pass requires $\Omega(n^{1+\frac{1}{t}})$ space

Thm [Goel, Kapralov, Khanna '12, Kapralov '12]
There is a $n^{1+\Omega(\frac{1}{\log\log n})}$ lower bound for $(1-e^{-1}+\delta)$-approximation of matching in one pass.

To give ideas about the proof of Thm A, first we need to define another important communication complexity problem:

Pointer chasing: Input: k functions $f_i : [n] \to [n]$
Goal: Compute $f_k(f_{k-1}(\cdots f_2(f_1(1))\cdots))$

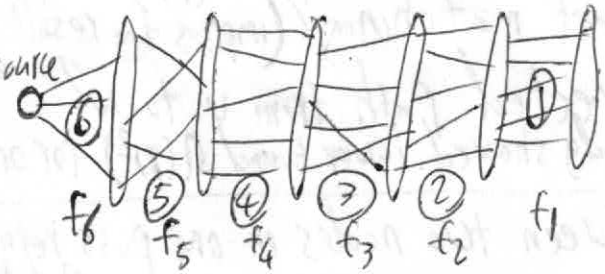
Two player version:
what players have

Alice	Bob
$f_A = f_2, f_4, f_6, \cdots$	$f_B = f_1, f_3, f_5, \cdots$

If Bob speaks first, [Nisan & Wigderson '93] proved that the communication complexity of any k-round (2-player) communication protocol to solve this problem is $\Omega(n)$, when $k = \Theta(1)$.

In k-player version: player i has function f_i and each round players speak in order player 1 through player k.
Guha & Mc Gregor (2007) proved computing the function in any k-round (player) communication protocol needs $\Omega(n)$ bit exchange, when $k = \Theta(1)$.

Now roughly the ideas are as follows: we can prove for any algorithm that computes the first k layers of a BFS tree from a prescribed node with probability at least $\frac{2}{3}$ requires either greater than $\frac{k}{2}$ passes or $\Omega(n^{1+\frac{1}{k}})$ space, i.e., with less than $\frac{k}{2}$ passes we need at least $\Omega(n^{1+\frac{1}{k}})$ space.



each f_i plays the same role that we saw for matching case and the stream would be ①, ②, ③, ④, ⑤, ⑥ in order.
These are the ideas but you can see [FKMSZ'05] for more details