# CMSC 858F: Algorithmic Lower Bounds: Fun with Hardness Proofs
# Fall 2014
# Introduction to Streaming Algorithms

**Instructor:** Mohammad T. Hajiaghayi
**Scribe:** Huijing Gong

November 11, 2014

## 1 Overview

In the previous lectures, we looked at the online algorithms and studies some of the related problems. Now we look into streaming algorithms, which have many similarities with online algorithm. They both require decisions before seeing all data but streaming algorithms can defer actions with limited memory. In these two lectures, we introduce streaming algorithms, related communication complexity problems and study examples of solving for lower bounds in this area.

## 2 Introduction to Steaming Algorithm

Streaming algorithms are often used to process data streams of big data, which is widely used in brain networks, and online social networks like Facebook, Twitter. In the most cases, these big data(networks) are dynamic which evolves over the time[1]. Streaming algorithms is introduced with the purpose of manipulating the data immediately as it streams. The study of this field was initiated by a seminar paper of Alon, Matias and Szegedy[2], which won the Gödel prize.

*Streaming Algorithms* are algorithms for processing data streams in which the input is presented as a sequence of items and can be examined in only a few passes(typically just one) by using relatively little memory(much less than the input size), and also limited processing time per item. We often produce approximate answers based on a summary or "sketch" of the data stream in memory[7]. A common technique for creating a "sketch" is sampling at random.

The data stream of the input can be a sequence of items such as integers or edges of a graph. In many cases when the streaming data are items, a vector

$\vec{a} = (a_1, ..., a_n)$ is initialized to the zero vector $\vec{o}$ and the input is a stream of updates in the form of $< i, c >$, in which $a_i$ is incremented by an integer $c$, i.e. $a_i = a_i + c$. Note that $c$ can be negative here. Below are four important streaming problems for items based on the vector $\vec{a}$.

1. Evaluate the $k^{th}$ frequency moment: $F_k(\vec{a}) = \sum_{i=1}^{n} a_i^k$.
2. Find heavy hitters, which is to find all the elements i with frequency $a_i > T$.
3. Count the number of distinct elements.
4. Calculate 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 here in all streaming algorithms, we want to minimize space, and then update time, even through multiple passes. The accuracy of the algorithm is often defined as an $(\epsilon, \delta)$-*approximation*. It means the algorithm achieves an error of less than $\epsilon$ with probability $1 - \delta$.

## 2.1 Lower Bounds in Graph Streaming

There are two main versions of graph streaming: inserting only version and dynamic version, while dynamic version allows both insertion and deletion. In both versions, $< i, j >$ can represent adding an edge $< v_i, v_j >$ to the graph. In dynamic version, if $< i, j >$ appears for the second time, it means deletion, which also can be represented by $- < i, j >$.

Here we introduce three popular models of graph streaming algorithms:

1. *Semi-streaming* model sets the input to be the stream of edges. If an edges appears with a negative sign, it represents deletion. Briefly speaking, semi-streaming requires $\widetilde{O}(n)$ for $n$ edges.
2. *Parameterized stream* model is a combination p-parameterized algorithm and stream algorithm. $\widetilde{O}(k) = k \cdot \text{polylog } n$, where $k$ is the size of parameter.
3. *Sliding window* model is a more generalized one, in which the function of interest is computing over a fixed size window of the stream according to the time and the update. Note that when new items are added to the window and items from the end of window are deleted.

In fact, there are lots of interesting lower bounds for massive graph problems, especially in the semi-streaming model where edges $E$ are streaming in as inputs (often in adversarial order, but sometimes in a random order) with the bounded storage space, which is $\widetilde{O}(n) = O(n \cdot \text{polylog } n)$, where $n = |V|$.

We first look at a simple graph streaming algorithm.

### 2.1.1 Streaming Algorithm for Matching

Given a graph $G = (V, E)$, a matching $M$ in G is a set of pairwise non-adjacent edges. A*maximal match* is a matching $M$ of $G$ that no edges not in $M$ can be added to $M$.

*Maximal matching algorithm*: In the semi-streaming model, a sequence of edges are streaming in one by one. Whenever a new edge is coming if both end vertices are unmarked, add it to matching set $M$ and mark both end vertices. Since each selected edge ruins at most two edges in the optimal solution, the approximation factor of this algorithm is 2, i.e. $Alg(\text{worst case}) = 1/2 \cdot Opt$. The storage space is $\widetilde{O}(n)$.

Here we introduce two important open problems extended from maximal matching problem as follows:

1. Can we get a constant factor better than 2 in one pass with $\widetilde{O}(n)$ space, where $m = |E|$? Konrad, Magniez, and Mathieu's work [6] shows that it is possible to get better than 2-approximation if the algorithm is allowed to have two passes or if the input is in random ordering.

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

Note that one pass makes the above problems hard and interesting even for bipartite graphs. Otherwise for example for any $0 < \epsilon < 1/3$ and a bipartite graph, there is a $2/3 - \epsilon$ approximation factor in $O(\frac{\log 1/\epsilon}{\epsilon})$ passes[3].

Before going into some hardness proofs, let's get insight into communication complexity, which is used as the most common technique for computing lower bounds in streaming algorithms.

### 2.1.2 Communication Complexity

Let $f : X \times Y \to Z$ be a function. The 2-party communication model consists of two players, Alice and Bob. Alice is given an input $x \in X$ and Bob is given an input $y \in Y$. Their goal is to compute $f(x, y)$. Both don't know other player's input, which requires them to communicate and exchange bits according to an agreed upon protocol. The *communication complexity* of $f$ is the minimum number of bits exchanged between Alice and Bob in the worst case of all possible communication protocols.

The protocol can be randomized or deterministic. *One-way protocol* means only one player sends information and the other one receives information, where both roles, sender and receiver, needed to be specified and the receiver needs to be able to compute $f$.

*INDEX* and *DISJOINTNESS* are two well-known problem in the area of communication complexity.

1. INDEX: In one-way complexity model, given that Alice has a string $x \in \{0, 1\}^n$ and Bob has a natural number $i \in [n]$. Bob wants to compute

Index$(x, i) = x_i$ by receiving a single message from Alice. Alice needs $\Omega(n)$ bits sent to Bob. (See FKMSZ'05[3] and FKMSZ'08[4] for related problems.)

2. DISJOINTNESS: In the *multi-way complexity model* (two players can exchange as many message as needed), given that Alice has a string $x \in \{0,1\}^n$ and Bob has a string $y \in\in \{0,1\}^n$. Bob wants to compute whether there is an index $i$, such that $x_i = y_i = 1$, i.e. DISJOINTNESS$(x, y)$ = "true if and only if there exists an $1 \leq i \leq n$, such that $x_i = y_i = 1$?" We might also have $\sum x_i = \sum y_i = \lfloor n/4 \rfloor$ [1]. $\Omega(n)$ bits are required to be exchanged between Alice and Bob even with multi-passes, which is the main difference to INDEX problem. Therefore this problem can be used to prove hardness for graph streaming problems even in multi-pass model.

Note that above lower bounds, $\Omega(n)$ for both INDEX and DISJOINTNESS problem, work even for randomized protocols which answers $f(x, y)$ correctly with probability 2/3, where probability is taken over all by public coin toss.

# 3 Examples of Steaming Algorithms

Now we look at some examples of steaming algorithm problems.

## 3.1 Max-Conn-Comp

**Definition 1** Max-Conn-Comp(k)*: Given a forest* $G(V, E)$ *is there a connected component of size at least* $k \geq 3$ *in G, for any given k. (Forest is a graph with no cycle.)*

**Theorem 1** *Any single-pass streaming graph algorithm that solves Max-Conn-Comp(k) problem on forest needs* $\Omega(n)$ *space.*

**Proof:** Proved by reduction from INDEX to Max-Conn-Comp(k).

Assume there is an algorithm $\mathcal{A}$ with space $o(n)$ for Max-Conn-Comp(k). Given an instance of INDEX, INDEX(x, i), where $|X| = n$:

1. Construct a graph $G = (V = V_L, V_r, V_d, E = E_{Alice} \cup E_{Bob})$ of Max-Conn-Comp(k) as follows: Let $V_l = \{l_1, l_2, ..., l_n\}, V_r = \{r_1, r_2, ..., r_n\}, V_d = \{d_1, d_2, ..., d_{k-2}\}$, $E_{Alice} = \{\{l_j, r_j\} | x_j = 1 \text{ for } j \in [1, n]\}$, and $E_{Bob} = \{r_i, d_1\} \cup \{\{d_j, d_{j+1}\} | j \in [1, k-3]\}$. (See Figure 1 for an example of constructing an instance $G$ from INDEX(1011, 3) and k = 4.)

2. Stream edges of $E_{Alice}$ and run algorithm $\mathcal{A}$ on it and as a result we have a space $S = o(n)$ as the working memory that Alice sends to Bob.

3. Then it is Bob's turn to stream his edges by continuing running algorithm $\mathcal{A}$ on the stream initialized by the space S which Bob received from Alice in the previous step.

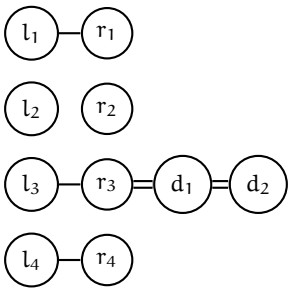
Analysis:

Eventually we have execution of algorithm $\mathcal{A}$ on $G$. By construction, $G$ is a forest. Since if INDEX(x, i) = 0, then the max component has size $k-1$, which makes Max-Conn-Comp(k) = 0. If INDEX(x, i) = 1, then clearly Max-Conn-Comp(k) = 1. Thus we have INDEX(x, i) = 0 $\iff$ Max-Conn-Comp(k) = 1.

According to the assumption, Alice sends $o(n)$ bits to Bob. It contradicts with the fact that solving INDEX problem needs $\Omega(n)$ bits to exchange. Therefore we prove that Max-Conn-Comp(k) problem also needs $\Omega(n)$ space for single-pass streaming graph algorithm to solve.

Note that we assumed both Alice and Bob share common knowledge about public algorithm $\mathcal{A}$ for INDEX problem. The hardness of INDEX is due to the fact that Alice doesn't know $i$.

(The reduction analysis of the many following problems are essentially very similar as this reduction from INDEX to Max-Conn-Comp, thus some repeated parts of proof are skipped for the rest of many problems.)




(−: Alice edge. =: Bob edge)

Figure 1: Instance of Max-Conn-Comp(k) Based on INDEX(1011, 3)

■

## 3.2 Is-Tree

**Definition 2** Is-Tree(G)*: Given* $G = (V, E)$ *as a graph, compute if G is a tree.*

Before solving this problem, we first introduce another problem, INDEX-SAME(x, i), in communication complexity as a useful tool.

**Definition 3** INDEX-SAME(x, i)*: Given that Alice has a string* $x \in \{0,1\}^n$ *and Bob has a natural number* $i \in [n-1]$*, Bob wants to compute if* $x_i = x_{i+1}$ *by receiving a single message from Alice.*

**Theorem 2** *Any single-pass streaming algorithm for INDEX-SAME need* $\Omega(n)$ *space of memory.*

**Proof:** Proved by reduction from INDEX(x, i) to INDEX-SAME(x', i').

Given an instance of INDEX problem (x, i), construct an instance of INDEX-SAME(x', i') as follows: Let $x' = x'_1, x'_2, ..., x'_n$ when $x'_i = 01$ if $x_i = 0$ , and $x'_i = 11$ otherwise. Let $i' = 2(i-1)+1$.

Easy to check that INDEX(x, i) $\iff$ INDEX-SAME(x', i').

■

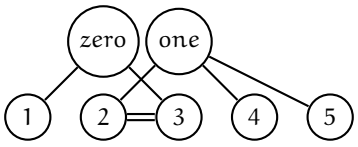
**Theorem 3** *Any single-pass streaming algorithm solving Is-Tree(G) need* $\Omega(n)$ *memory space.*

**Proof:** Essentially we use a similar proof strategy to the proof of Theorem 1. Here we show the reduction from INDEX-SAME to Is-Tree.

Given any instance of INDEX-SAME (x, i), we construct a instance graph $G = (V, E)$ of Is-Tree as follows:

Let $V = \{zero, one, 1, 2, ..., n\}$ and $E = E_{Alice} \cup E_{Bob}$, where $E_{Bob} = \{i, i+1\}$ and $E_{Alice} = \{\{i, zero\}|x_i = 0\} \cup \{\{i, one\}|x_i = 1\}$. (See Figure 2 for an example of constructing G from INDEX-SAME(01011, 2).)

By construction, there are $n+2$ vertices and $n+1$ edges in G. Therefore either G is a tree or G has a cycle. If G has a cycle, then the two nodes connected by Alice edge must connect to the same node which is either node zero or one. Either case makes INDEX-SAME(x, i) = 1. If G is a tree, then the two nodes connected by Alice edge connect to two different nodes, one is node zero and the other is node one, which indicates that INDEX-SAME(x, i) = 0. Therefore we have Is-Tree(G)=YES $\iff$ INDEX-SAME(x, i) = 0.




(−: Alice edge. =: Bob edge)

Figure 2: Instance of IsTree(G) Based on INDEX-SAME(01011, 2)

■

**Definition 4** TREE-DIAM(k)*: Given* $G(V, E)$ *be a tree, compute if there is a diameter of* G *is at least* $k \geq 3$.

**Theorem 4** *Any single-pass streaming algorithm for TREE-DIAM(k) needs* $\Omega(n)$ *memory.*

**Proof:** Exercise. ■

## 3.3 Perfect-Matching

**Definition 5** Perfect-Matching(G)*: Given a graph* $G(V, E)$*, compute if there is a perfect matching in* $G$*.*

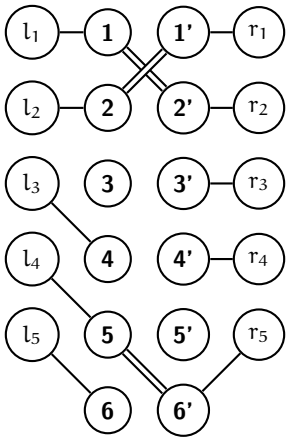
**Theorem 5** *Any single-pass streaming algorithm for Perfect-Matching need* $\Omega(m) = \Omega(n^2)$ *memory.*

**Proof:** Proved by reduction from INDEX to Perfect-Matching. Here we assume string x in INDEX problem is a $n \times n$ array and Bob wants to compute $x_{ij}$. Then an instance of INDEX can be written as (x, i, j). Then we need space $\Omega(n^2)$ to solve this INDEX(x, i, j) problem.

Assume we have an algorithm $\mathcal{A}$ solving Perfect-Matching(G) only needs $o(n^2)$. Given an instance INDEX(x, i, j), we construct a graph $G(V, E)$ as follows: Let $V = \{l_1, l_2, ..., l_{n-1}, r_1, r_2, ..., r_{n-1}, 1, 2, ..., n, 1', 2', ..., n'\}$ and $E = E_{Alice} \cup E_{Bob}$. $E_{Alice} = \{\{i, j'\} | x_{i,j} = 1\}$. $E_{Bob} = \{\{l_k, k\} | 1 \leq k < i\} \cup \{\{l_k, k+1\} | i \leq k \leq n-1\} \cup \{\{r_k, k\} | 1 \leq k < j\} \cup \{\{r_k, k+1\} | j \leq k \leq n-1\}$. See Figure 3 for constructing $G$ from INDEX(x, 3, 5) where

$$x = \begin{matrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

We stream Alice edges and then stream Bob edges. Thus we can solve INDEX(x, i, j) according to algorithm $\mathcal{A}$. It is easy to check that INDEX(x, i, j) = 1 $\iff$ Perfect-Matching(G) = YES. However it contradicts with the fact that algorithm to solve INDEX(x, i, j) needs space $\Omega(n^2)$, where $|X| = n$. Therefore we can prove this theorem as desired.




(−: Alice edge. =: Bob edge)

Figure 3: Instance of Perfect-Matching(G) Based on INDEX(x, 3, 5)

■

### 3.4 Shortest-Path

**Definition 6** Shortest-Path(v,w)*: Given a graph* $G(V, E)$*, compute the length of the shortest path from* $v$ *to* $w$*.*

**Theorem 6** *Any streaming algorithm that approximates Shortest–Path(v,w) with factor better than* $\frac{5}{3}$ *needs* $\Omega(n^2)$ *space.*

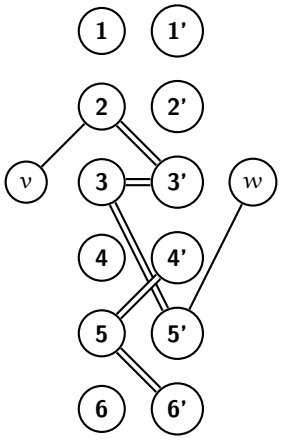
**Proof:** The proof here is essentially the same with the proof of Perfect-Matching problem. We use reduction from INDEX problem where the input string $x$ is also set as $n \times n$ array.

Assume we have an algorithm $\mathcal{A}$ solving Shortest-Path(v, w) only needs $o(n^2)$. Given an instance INDEX(x, p, q), we construct a graph $G(V, E)$ as follows: Let $V = \{v, w, 1, 2, ..., n, 1', 2', ..., n'\}$ and $E = E_{Alice} \cup E_{Bob}$. $E_{Alice} = \{\{i, j'\} | x_{i,j} = 1\}$. $E_{Bob} = \{v, p\} \cup \{q, w\}$. See Figure 4 for constructing instance of Shortest-Path(v, w) from INDEX(x, 3, 4) where

$$x = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

We stream Alice edges and then stream Bob edges. Thus we can solve INDEX(x, p, q) according to algorithm $\mathcal{A}$. Clearly INDEX(x, p, q) = 1 $\iff$

Shortest-Path(v, w) = 3 and INDEX(x, p, q) = 0 $\iff$ Shortest-Path(v, w) $\geq 5$ ($\infty$ if there is no path from $v$ to $w$). However it has contradiction with the fact that algorithm solving INDEX(x, p, q) needs space $\Omega(n^2)$, where $|X| = n$.




(−: Alice edge. =: Bob edge)

Figure 4: Instance of Shortest-Path(v, w) Based on INDEX(x, 3, 4)

■

## 3.5 Frequency Moments

**Definition 7** *Given a data stream* $y_1, y_2, ..., y_n$ *where* $y_i \in [m]$ *for all* $y_i$*, the frequency of* $k \in [m]$ *in the stream is* $x_k = |\{j | y_j = k\}|$*. The* frequency vector *is the vector* $x = (x_1, x_2, ..., x_m)$*. The* $p^{th}$ frequency moment *of the input stream is defined as follows:*

$$f_p = \begin{cases} |\{i | x_i \neq 0\}| & \text{if } p = 0 \\ \max_i x_i & \text{if } p = \infty \\ \sum_{i=1}^{m} x_i^p & \text{otherwise.} \end{cases}$$

Note that $F_0$ is the number of distinct elements of the input. $F_1$ is the number of elements (with repetition).

**Theorem 7** *Indyk and Woodruff prove the following statement [5]:*

1. *For* $p \in [0, 2]$*, there exists a randomized streaming algorithm which* $1 + \epsilon$*-approximates* $F_p$ *in space* $O(poly(\log m, \log n))$*.*
2. *For* $p > 2$*, any randomized streaming algorithm that* $1 + \epsilon$*-approximates* $F_p$ *requires* $\Omega(m^{1-\frac{2}{p}})$ *memory space.*

3. *For* $p \in [0, 2]$*, there exists a randomized streaming algorithm which* $1 + \epsilon$*-approximates* $F_p$ *in space* $\Omega(m^{1-\frac{2}{p}})$.

# References

[1] Hajiaghayi, Mohammad T. *Algorithmic Lower Bounds: Streaming Algorithms Lectures– Hand-written notes*. Fall 2014. University of Maryland.

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

[3] Feigenbaum, Joan and Kannan, Sampath and McGregor, Andrew and Suri, Siddharth and Zhang, Jian. *On graph problems in a semi-streaming model*. Theoretical Computer Science. Volume 348, Page 207–216, 2005.

[4] Feigenbaum, Joan and Kannan, Sampath and McGregor, Andrew and Suri, Siddharth and Zhang, Jian *Graph distances in the data-stream model*. SIAM Journal on Computing. Volume 38, Page 1709–1727, 2008.

[5] Indyk, Piotr and Woodruff, David *Optimal approximations of the frequency moments of data streams*. Proceedings of the thirty-seventh annual ACM symposium on Theory of computing. Page 202–208, 2005.

[6] C. Konrad and F. Magniez and C. Mathieu. *Maximum matching in semi-streaming with few passes*. Proceedings of 15th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems. Page 231–242, 2012.

[7] *Streaming algorithms* Wikipedia: The Free Encyclopedia. 7 September 2014. http://en.wikipedia.org/wiki/Streaming_algorithm/