

Lecture 1

September 1, 2020

*Instructor: Sepehr Assadi**Scribe: Sepehr Assadi*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

1 Graph Streaming

Motivation

Massive graphs appear in most application domains nowadays: web-pages and hyperlinks, neurons and synapses, papers and citations, or social networks and friendship links are just a few examples. Many of the computing tasks in processing these graphs, at their core, correspond to classical problems such as reachability, shortest path, or maximum matching—problems that have been studied extensively for almost a century now. However, the traditional algorithmic approaches to these problems do not accurately capture the challenges of processing massive graphs. For instance, we can no longer assume a random access to the entire input on these graphs, as their sheer size prevents us from loading them into the main memory.

To handle these challenges, there is a rapidly growing interest in developing algorithms that explicitly account for the restrictions of processing massive graphs. Graph streaming algorithms have been particularly successful in this regard; these are algorithms that process input graphs by making **one or few sequential passes** over their edges while using a **small memory**. Not only streaming algorithms can be directly used to address various challenges in processing massive graphs such as I/O-efficiency or monitoring evolving graphs in realtime, they are also closely connected to other models of computation for massive graphs and often lead to efficient algorithms in those models as well. Another remarkable advantage of graph streaming algorithms is that we have an extremely powerful tool at our disposal for proving lower bounds on their capabilities: communication complexity. These lower bounds can provide invaluable insights into the problems at hand and guide us in designing better algorithms or adjusting our expectations by making more realistic assumptions.

(Semi-)Formal Definition of the Model

Suppose $G = (V, E)$ is a graph with n vertices (in general, G can be undirected or directed, unweighted or weighted, and so on). We assume that $V := \{1, \dots, n\}$ and so the only unknown about G is the set of its edges E . A *graph stream* σ is then an arbitrarily-ordered sequence of edges of G , i.e., $\sigma := \langle e_1, \dots, e_m \rangle$ for $e_i \in E$. A streaming algorithm can make one or a few passes over this stream (in the same order), and at the end of the last pass, should output the solution to some problem on G .

Of course, naively, the streaming algorithm can simply store the entire edges of the graph in the stream and solve the problem at the end using any offline algorithm; this however may require up to $\Theta(n^2)$ space which is prohibitively large. As such, we are interested in algorithms that use much smaller space, typically $\tilde{O}(n)$ space¹—these algorithms are called *semi-streaming* algorithms and were introduced by Feigenbaum et.al. [10] who initiated the systematic study of graph streams². We shall note however that both regimes when memory is $\gg n$, but $o(n^2)$, and when it is $\ll n$, typically $\text{polylog}(n)$, are also gaining increasing attention; we will discuss such algorithms later in this course.

¹Throughout the course, we use the notation $\tilde{O}(f) := O(f \cdot \text{polylog}(f))$ to hide poly-log factors.

²It should be noted that there are some work prior to [10] that considered graphs in the streaming model, for instance [4, 13].

In the graph streaming model, the two main resources are the space complexity of the algorithm and the number of passes—for the latter, one almost always considers single-pass algorithms much more favorable than multi-pass algorithms, but even with more number of passes, optimizing the number of passes is a priority often down to the last constant³.

Different variants. The above only describes the most vanilla version of the graph streaming model. There are various extensions that one may want to consider:

- **Order of arrival of edges:** The definition of graph streams given above leads to a “doubly worst case” analysis of streaming algorithms: adversarial graphs and adversarial stream orderings. However, one can also consider models such as *random-arrival* where the stream is a random permutation of edges (see, e.g. [14]) or *adjacency-arrival* where edges of a single vertex appear next to each other (see, e.g. [17]); note that in the latter model, each edge appears twice.
- **Dynamic graphs:** Another popular extension of graph streams is to consider the case when edges can be *deleted* from the stream as well. For instance, this can be in form of a *dynamic graph stream* in which each entry of the stream either inserts a new edge or delete a previously inserted edge [1] or in a *sliding window* model where the goal is to compute the solution only over the last W edges seen in the stream [9].
- **Verifiable graph stream computation:** Another line of work considers *verification* problems in the context of graph streaming problems. These algorithms capture the scenario when a “computationally weak” verifier outsources the storage and processing of the input graph to a powerful but untrusted prover; here, the verifier needs only to verify the solution returned by an all knowing prover in a streaming fashion as opposed to solving the problem entirely from the scratch. Several variants of verifiable graph streams have been studied such as *annotated data streams* [7], *Artur-Merlin streaming algorithms* [12], and *streaming interactive proofs* [8].

We will discuss some of these variants throughout this course.

2 Warm-Up: Some Simple Graph Streaming Algorithms

We start by describing several simple graph (semi-)streaming algorithms. These results are presented first in the work of Feigenbaum et.al. [10] or are otherwise folklore (see [16]).

Undirected Connectivity

Problem 1. Given an undirected graph $G = (V, E)$, decide whether or not G is connected.

The algorithm for this problem is to simply maintain a forest and add each arriving edge if it does not create a cycle, i.e., introduces a “new connectivity information”. Formally,

Algorithm 1. A single-pass semi-streaming algorithm for connectivity.

- (i) Let $F \leftarrow \emptyset$;
- (ii) For any edge e in the stream, add e to F if $F \cup \{e\}$ does not have a cycle.
- (iii) Return G is connected iff F is connected.

³In certain cases, one also tries to minimize the runtime of the algorithm, in particular, the update time per each element of the stream. However, in this course, we primarily focus on space- and pass-complexity of the algorithms and may even use algorithms with exponential (or larger) runtime.

Algorithm 1 uses $O(n)$ space⁴ as F at every point is a forest and hence contains at most $n - 1$ edges. It thus remains to prove its correctness (see **Figure 1** for an illustration).

Lemma 1. *The forest F in **Algorithm 1** is connected iff the input graph G is connected.*

Proof. Since F is a subgraph of G , F being connected immediately implies that G is also connected. For the other direction, suppose towards a contradiction that G is connected but F is not. Since F is not connected, there exists a cut $(S, V \setminus S)$ of vertices where F has no edges crossing this cut. At the same time, since G is connected, there should be an edge e in the cut $(S, V \setminus S)$. Now consider the time e arrived in the stream. At that point, $F \cup \{e\}$ cannot form a cycle as $e = (u, v)$ is the unique cut-edge of $(S, V \setminus S)$ and thus there is no other path from $u \in S$ to $v \in V \setminus S$ in $F \cup \{e\}$. As such, e should have been added to F also by **Algorithm 1**, a contradiction that $(S, V \setminus S)$ has no cut edge in the final F . \square

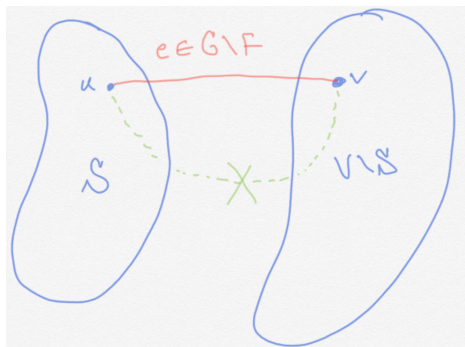


Figure 1: A simple illustration of the proof of **Lemma 1**. If $e = (u, v)$ is the unique cut edge of $(S, V \setminus S)$ in $F \cup \{e\}$, then e cannot be a part of any cycle.

Optimality? Any single-pass semi-streaming algorithm for connectivity requires $\Omega(n \log n)$ bits of space ($\Omega(n)$ space in the language of this course) by a result of Sun and Woodruff [21] (we will prove a slightly weaker version of this by the end of this note). As such, **Algorithm 1** is asymptotically optimal.

Edge Connectivity

Problem 2. Given an undirected graph $G = (V, E)$ and an integer $k \geq 1$, decide whether or not G is k -edge-connected, i.e., at least k edges needs to be removed from G to make it disconnected.

The algorithm is similar to the previous part by trying to maintain a “certificate” for k -connectivity, namely, a *sparser* subgraph of G which preserves the k -connectivity property of G . We shall note that the algorithm we present is not necessarily semi-streaming (when $k \gg \text{polylog}(n)$); we will revisit this problem of designing semi-streaming algorithms for edge connectivity later in this course.

Algorithm 2. A single-pass streaming algorithm for k -edge-connectivity.

- (i) Let $F_1, F_2, \dots, F_k \leftarrow \emptyset$;
- (ii) For any edge e in the stream, if there is any F_i such that $F_i \cup \{e\}$ has no cycle, add e to this F_i .
- (iii) Return G is k -connected iff $F := F_1 \cup \dots \cup F_k$ is k -connected.

⁴For simplicity and to follow the graph streaming convention, we typically measure the space in terms of machine words of size $\Theta(\log n)$ as opposed to bits which is more common in the traditional streaming model and communication complexity.

Algorithm 2 uses $O(nk)$ space as it maintains k spanning forests at all times. We now prove that it outputs a correct answer (see **Figure 2** for an illustration).

Lemma 2. *The subgraph $F = F_1 \cup \dots \cup F_k$ in **Algorithm 2** is connected iff the input graph G is k -connected.*

Proof. Since F is a subgraph of G , F being k -connected immediately implies that G is also k -connected. We now prove the other direction.

Suppose towards a contradiction that G is k -connected but F is not. Since F is not k -connected, there exists a cut $(S, V \setminus S)$ of vertices where F has less than k edges crossing this cut. This in particular means that at least one of the forests F_i in **Algorithm 2** has no edge in this cut as the edges across F_i 's are disjoint. But then adding e to F_i does not create a cycle and thus e should have been part of F_i , a contradiction. \square

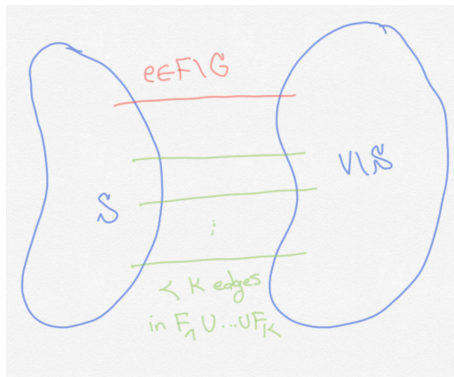


Figure 2: A simple illustration of the proof of **Lemma 2**. An edge $e \in G \setminus F$ cannot form a cycle with every one of F_1, \dots, F_k if less than k edges in F belong to the cut $(S, V \setminus S)$.

Optimality? Sun and Woodruff [21] also proved that any deterministic single-pass algorithm for k -connectivity requires $\Omega(nk \log n)$ bits; the bound degrades to $\Omega(nk)$ for randomized algorithms [21]. As such, **Algorithm 1** is also optimal up to constant factor and $\Theta(\log n)$ -factor for deterministic and randomized algorithms, respectively.

Shortest Path

Problem 3. Given an undirected graph $G = (V, E)$ and $s, t \in V$, find a shortest path from s to t in G .

Let us first point out that the knowledge of s and t in this problem is in some sense irrelevant. Consider any stream that inserts an arbitrary graph between vertices in $V \setminus \{s, t\}$ and at the very end, inserts two edges (s, s') and (t', t) for two arbitrary $s', t' \in V \setminus \{s, t\}$. At this point, the algorithm is forced to return a shortest path from s' to t' as the shortest s - t path is certainly of the form $s \rightarrow s' \rightsquigarrow_P t' \rightarrow t$ where P is a shortest s' - t' path.

The above observation means that any streaming algorithm for s - t shortest path on n vertex graphs, can be turned into a *data structure* from which we can recover a shortest path between any two pairs of vertices in a graph on $n - 2$ vertices: simply run the streaming algorithm on the $(n - 2)$ -vertex graph and store the content of the memory of the streaming algorithm as a data structure; then, given a query s', t' to the data structure, continue running the algorithm on the two last edges of the stream which are (s, s') and (t', t) . This immediately means that the memory content of the streaming algorithm has to be of size $\Omega(n^2)$ (take the $(n - 2)$ -vertex graph for the data structure to be a clique).

So far, we established that we cannot obtain an exact solution to the shortest path problem and have to settle for approximation. Moreover, we learned that we effectively need to solve an all-pairs shortest problem and not a single-source one. This motivates us to find a *graph spanner* defined as follows:

Definition 3. For a graph $G = (V, E)$ and parameter $\alpha \geq 1$, we say that a subgraph $H = (V, E_H)$ for $E_H \subseteq E$ is an α -spanner iff for all pairs of vertices $u, v \in V$:

$$\text{dist}_G(u, v) \leq \text{dist}_H(u, v) \leq \alpha \cdot \text{dist}_G(u, v),^5$$

where $\text{dist}_*(u, v)$ denote the length of the shortest path between u and v in the corresponding graph $*$.

We are going to design a streaming algorithm for finding an α -spanner to approximate shortest path problem (think of spanners as “certificate” for shortest path similar to k -spanning forests for k -connectivity). The algorithm is simply go over the edges of the stream and include an edge if it does not create a “short” cycle and hence cannot “help” in reducing the distance of its endpoints by much (think of this as a generalization of Algorithm 1). Again, this algorithm is not always semi-streaming (depending on the choice of α); we will revisit semi-streaming algorithms for shortest path later in this course.

Algorithm 3. A single-pass streaming $(2k - 1)$ -approximation algorithm for s - t shortest path.

- (i) Let $F \leftarrow \emptyset$;
- (ii) For any edge e in the stream, add e to F if $F \cup \{e\}$ does not have a cycle of length $\leq 2k$.
- (iii) Return the s - t shortest path in F .

Let us first prove that F is $(2k - 1)$ -spanner of G . This will immediately imply the correctness of the algorithm by the definition of spanner as $\text{dist}_F(s, t) \leq (2k - 1) \cdot \text{dist}_G(s, t)$.

Lemma 4. *The subgraph F in Algorithm 3 is a $(2k - 1)$ -spanner of the input graph G .*

Proof. We say that an edge $(u, v) \in G$ is *stretched* by α iff $\text{dist}_F(u, v) = \alpha$, i.e., in place of the edge (u, v) in F , we now have a path of length α . Any edge $(u, v) \in G \setminus F$ is stretched by at most $(2k - 1)$ because the reason we did not add (u, v) to F was existence of a cycle of length $\leq 2k$ in F (that inevitably includes both u and v); but this means that there is a path of length $\leq 2k - 1$ between u and v in F .

Consider a pair of vertices $x, y \in V$ and the shortest path $x = w_0 \rightarrow w_1 \rightarrow w_2 \dots w_\ell \rightarrow y = w_{\ell+1}$ in G with length ℓ . Every edge in this path is stretched by at most $(2k - 1)$ in F and thus there is a *walk* of length $(2k - 1) \cdot \ell$ in F between x, y . This immediately implies also that $\text{dist}_F(x, y) \leq (2k - 1) \cdot \text{dist}_G(x, y)$ as well as we can shortcut the edges of the walk into a path. Hence, F is a $(2k - 1)$ -spanner, concluding the proof. \square

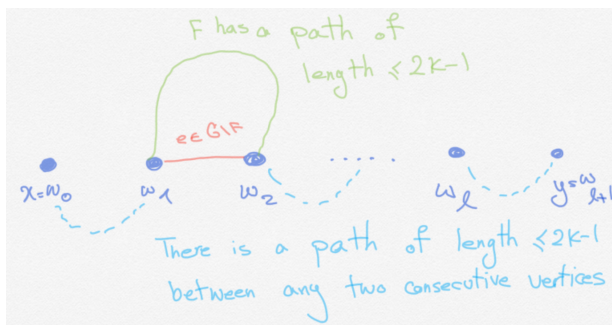


Figure 3: A simple illustration of the proof of Lemma 4. As we only remove skip edge of G in G that are part of a $\leq 2k$ cycle, every edge of a shortest path between two vertices x, y in G is stretched by at most $(2k - 1)$ in F ; this means that the length of x - y shortest path in F is at most $(2k - 1)$ times larger than G .

⁵Notice that the interesting inequality is the right one; since $H \subseteq G$, the left inequality always hold.

It only remains to analyze the space complexity of [Algorithm 3](#). For this, we can use a classical result in graph theory referred to as the Moore Bound: any graph with no cycle of length $\leq 2k$ can only have $O(n^{1+1/k})$ edges [6] (see also [2, 3] for sharper bounds for irregular graphs). This can be proven as follows (this proof is mostly an exercise in graph theory and has not much to do with streaming algorithms; thus, the reader may decide to skip it entirely).

Proposition 5 (Moore Bound). *Any graph G with $m \geq 2 \cdot n^{1+1/k}$ edges contain a cycle of length $\leq 2k$.*

Proof. We first claim that G has an induced subgraph H with minimum degree $\delta(H) > m/n$ (at least half the average degree). This can be proven as follows: if $\delta(G) > m/n$ we are done already; if not, pick any vertex $v \in G$ with $\deg(v) \leq m/n$ and remove it from the graph. As removing a vertex of degree $\leq m/n$ reduces number of edges by $\leq 2m/n$, this will reduce the average degree of the remaining graph to:

$$\geq \frac{\sum_{u \in V} \deg(u) - 2m/n}{n-1} = \frac{2m \cdot (1 - 1/n)}{n \cdot (1 - 1/n)} = 2m/n.$$

As such, this process never decreases the average degree and hence needs to terminate in a non-empty subgraph H with $\delta(H) > m/n$.

We now have a subgraph H with minimum degree $d > 2n^{1/k}$ over at most n vertices. Suppose towards a contradiction that H has no cycle of length $\leq 2k$. Now, consider growing a BFS tree of depth k from any vertex v of H . The vertices of this tree cannot be visited in more than one path from v as otherwise any vertex x reachable from v via more than 2 paths of length $\leq k$, creates a cycle of length $\leq 2k$ already. But this means that we should visit

$$1 + d + d \cdot (d-1) + d \cdot (d-1)^2 + \dots + d \cdot (d-1)^{k-1} \geq (d-1)^k$$

distinct vertices in this BFS tree. But since $(d-1) > n^{1/k}$, we need to visit more than n vertices, a contradiction. This means H has $\leq 2k$ -length cycle, concluding the proof. \square

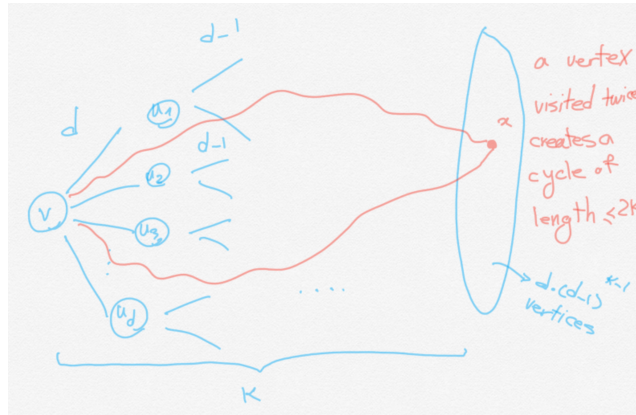


Figure 4: A simple illustration of the proof of [Proposition 5](#). The vertices in the first k layer of the BFS tree from v are distinct, i.e., only have one path from v —otherwise, we will find a cycle of length $\leq 2k$ already.

Remark. The **Erdős-Girth** conjecture states that for every k , there are graphs with $\Omega(n^{1+1/k})$ edges with no cycle of length $\leq 2k$, meaning that Moore bound is tight. This conjecture is widely believed and is known to be true for several small choices of k . However, the general conjecture is still open and the best known construction imply a lower bound of $\Omega(n^{1+1/2k-1})$ edges (a random graph of average degree $\Theta(n^{1/2k-1})$ satisfy this).

Back to [Algorithm 3](#): considering that the subgraph F maintained by [Algorithm 3](#) has no $\leq 2k$ -cycle, we obtain that its space is $O(n^{1+1/k})$ by [Proposition 5](#). This in particular means that for $k = \Theta(\log n)$, we obtain a semi-streaming $O(\log n)$ -approximation to s - t shortest path (and all-pairs shortest path).

Optimality? Feigenbaum et.al. [11] proved that no semi-streaming algorithm can output an $o(\log n)$ -approximation to s - t shortest path in a single pass, implying the optimality of the above semi-streaming algorithm. See [11] for more on the space-approximation tradeoffs for this problem.

3 Communication Complexity (for Graph Streams)

Motivation

We now start the second part of this note, communication complexity, which is the second main component of this course. The problem we are interested in is the following: Suppose we have a graph $G = (V, E)$ and we partition its edges between Alice and Bob, as E_A and E_B , respectively. How many bits of communication is needed between Alice and Bob to solve some problem on G , say, find a spanning forest of G ?⁶

Why do we care about this in this course? Well, as we shall see soon, any graph streaming algorithm will also imply a communication protocol with communication proportional to the space of the streaming algorithm times its number of passes. Put differently, communication complexity *lower bounds* will also imply lower bounds for streaming algorithms; in fact, this is the de facto method for proving streaming lower bounds.

Before continuing however, a disclaimer is in order: Communication complexity is a highly general branch of theoretical computer science (TCS) with various aspects way beyond graph streaming; however, in this course, we are primarily focused on this model in the context of graph streaming and as such by necessity have to ignore numerous other fascinating aspects of this model. We refer the interested reader to the following excellent textbooks by Kushilevitz and Nisan [15], and by Rao and Yehudayoff [19] on basics of communication complexity, and by Roughgarden [20] for implications of communication complexity in proving lower bounds in algorithmic models.

(Semi-)Formal Definition of the Model

Let us now define communication complexity more generally for arbitrary functions/relations (and not only on graphs). Suppose we have a Boolean function $f : \{0, 1\}^N \times \{0, 1\}^N \rightarrow \{0, 1\}$ (one can consider different domains for the function, different ranges, relations instead of Boolean functions, or even functions with more than two arguments corresponding to *multi-party* communication models—for brevity, we stick to the most basic setting in this lecture but will consider other settings as well throughout the course). Function f defines a *communication problem* as follows: there are two players, say Alice and Bob, who get inputs $x \in \{0, 1\}^N$ and $y \in \{0, 1\}^N$, respectively, and their goal is to compute $f(x, y)$.

Considering neither Alice nor Bob has the entire input on their own, the players need to *communicate* with each other to determine the value of $f(x, y)$. The communication happens according to a *protocol* and is done as follows: Alice sends a single message m_1 to Bob based solely on her input x ; after receiving this message, Bob responds back with a message m_2 of his own which is a function of his input y and the message m_1 of Alice; the players continue this throughout the protocol; eventually, the last player receiving a message outputs the solution to $f(x, y)$.

The main measure of efficiency in this model is the communication cost of the protocols, defined as follows.

Definition 6 (Communication cost). The communication cost of a protocol π , denoted by $\|\pi\|$, is the *worst-case* number of bits communicated between Alice and Bob in π over any choice of inputs x, y .

Definition 7 (Deterministic communication complexity). The *deterministic* communication complexity of function f is defined as $D(f) = \min_{\pi} \|\pi\|$, where π ranges over all protocols that can solve f .

⁶The answer to this particular question is $O(n \log n)$ bits—Alice sends a spanning forest of her input to Bob. Does this remind you of an algorithm we saw in this lecture?

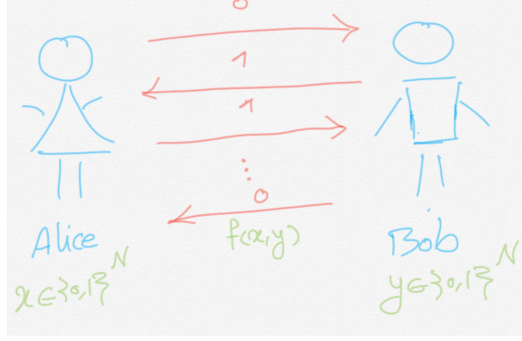


Figure 5: Alice and Bob computing $f(x, y)$.

Note that $D(f) = O(N)$ for any function $f : \{0, 1\}^N \times \{0, 1\}^N \rightarrow \{0, 1\}$ as Alice can send all her N -bit input to Bob and Bob can compute the solution on its own.

Randomized Communication Complexity. We can also consider randomized communication complexity setting wherein Alice and Bob have access to random bits. There are two ways of introducing random bits to the communication model: the *private coin/randomness* model where Alice and Bob have access to separate sources of randomness on their own, and the *public coin/randomness* model, where players have access to a shared source of randomness. We require that a randomized protocol for a problem f to output the correct answer to $f(x, y)$ for any given x, y to Alice and Bob, with probability at least $2/3$ (or some other constant strictly more than half⁷).

At first glance, the notion of public coin may sound rather strange. However, there are multiple reasons that motivate considering public coins in addition to (or instead of) private coins. One particularly important reason is that public coin protocols are somewhat “mathematically nicer” to work with as they can be considered as distributions over deterministic protocols. But an important question remains still: can it be that by allowing public coins, we give “too much power” to the protocols? The answer, perhaps surprisingly at first, is *No!* (this proof is again mostly an exercise in probabilistic analysis and has not much to do with streaming algorithms and thus can be skipped by the reader).

Proposition 8 (Newman’s Theorem [18]). *Any public coin protocol π for a communication problem f with domain $\{0, 1\}^N \times \{0, 1\}^N$ with probability of success at least $1 - \epsilon$ can be simulated by a private coin protocol θ such that $\|\theta\| \leq \|\pi\| + O(\log N + \log(1/\delta))$ and θ outputs the correct answer to f w.p. at least $1 - \epsilon - \delta$.*

Proof. Recall the following additive chernoff bound (or Hoeffding inequality). Given t independent random variables $Z_1, \dots, Z_t \in [0, 1]$ and $\bar{Z} = \sum_i^t Z_i/t$, for any $\alpha > 0$, we have

$$\Pr(\bar{Z} - \mathbb{E}[\bar{Z}] > \alpha) \leq \exp(-2\alpha^2 t). \quad (1)$$

Let $\pi(x, y, r)$ denote the *deterministic* output of the protocol π on inputs x and y , and public randomness r . Define the indicator random variable $Z(x, y, r) \in \{0, 1\}$ which is 1 iff $\pi(x, y, r) \neq f(x, y)$, i.e., the protocol outputs errs in the answer. By the guarantee of the protocol π , we have that for any x, y :

$$\mathbb{E}_r[Z(x, y, r)] = \Pr(\pi \text{ errs on } (x, y)) \leq \epsilon.$$

Now, suppose we sample public coins r_1, \dots, r_t for $t = \lceil 2N/\delta^2 \rceil$. By the additive Chernoff bound, we have:

$$\begin{aligned} \Pr_{r_1, \dots, r_t} \left(\frac{\sum_{i=1}^t Z(x, y, r_i)}{t} \geq \epsilon + \delta \right) &\leq \Pr_{r_1, \dots, r_t} \left(\frac{\sum_{i=1}^t Z(x, y, r_i)}{t} - \mathbb{E} \left[\frac{\sum_{i=1}^t Z(x, y, r_i)}{t} \right] \geq \delta \right) \\ &\leq \exp(-2\delta^2 t) \leq \exp(-4N). \end{aligned}$$

⁷This is because we can always boost the probability of success of the algorithm by running it multiple times in parallel and using majority/median trick; this only increases the communication by a constant factor.

As such, by union bound over all choices of $(x, y) \in \{0, 1\}^N \times \{0, 1\}^N$, we have,

$$\Pr_{r_1, \dots, r_t} \left(\text{exists } (x, y) \text{ s.t. } \frac{\sum_{i=1}^t Z(x, y, r_i)}{t} \geq \varepsilon + \delta \right) \leq 2^{2N} \cdot \exp(-4N) < 1.$$

Thus, one can find a collection of t choices of public coins r_1, \dots, r_t such that $\frac{\sum_{i=1}^t Z(x, y, r_i)}{t} < \varepsilon + \delta$ for all possible inputs x, y . In the following we fix such choice of r_1, \dots, r_t .

The protocol θ works as follows. Alice first *privately* samples $r' \in \{r_1, \dots, r_t\}$ uniformly at random and sends it to Bob using $O(\log t) = O(\log N + \log(1/\delta))$ bits. The players then together run the *deterministic* protocol $\pi(x, y, r)$ from now on and output the same answer.

The communication cost of this protocol is clearly at most $O(\log N + \log(1/\delta))$ more than the communication cost of π . Moreover, for any input (x, y) :

$$\Pr_{r'}(\theta(x, y, r') \text{ errs}) = \Pr_{r' \in \{r_1, \dots, r_t\}}(\pi(x, y, r') \text{ errs}) = \frac{1}{t} \cdot \sum_{i=1}^t \Pr(\pi(x, y, r_i) \text{ errs}) = \frac{1}{t} \sum_{i=1}^t Z(x, y, r_i) \leq \varepsilon + \delta,$$

by the definition of r_1, \dots, r_t . This concludes the proof. \square

Remark. Newman's theorem can be seen as a very basic *pseudo-random number generator* (PSG): We were able to reduce the entire random bits needed by π to only $O(\log N + \log(1/\delta))$ bits (and thus communicate it between the players) at the cost of only paying an additive factor of δ in the algorithm. Note that aside from transforming public coins to private coins, this theorem also implies that any constant-error protocol can be made to work with only $O(\log N)$ bits of randomness.

Equipped with Newman's theorem, we can henceforth only focus on public coin protocols and present our definitions only for such protocols (and if needed, one can infer the results for private coin protocols by applying Newman's theorem and "pay" a minimal penalty in the bounds). In the following, by randomized protocols, we always mean public coin protocols (note that a public coin protocol does *not* need access to private coins in this setting)⁸.

Definition 9 (Communication cost). The communication cost of a randomized protocol π is the *worst-case* number of bits communicated between Alice and Bob in π over any choice of inputs x, y and the randomness of the protocol.

Remark: Throughout this course, we always assume that the length of messages communicated in every protocol is always equal to its communication cost by padding each message to a fixed length; this can only increase the cost of protocols by a constant factor.

We can view a public coin protocol as a distribution over deterministic protocols obtained by first using the public coins to sample the deterministic protocol and then running the deterministic protocol on the input. As such, we can alternatively define the communication cost of π as the maximum communication cost of any deterministic protocol in the support of this distribution.

Definition 10 (Randomized communication complexity). The *randomized* communication complexity of function f is defined as $R(f) = \min_{\pi} \|\pi\|$, where π ranges over all randomized protocols that can solve f with probability of success at least $2/3$.

We will also define a distributional version of communication complexity.

⁸At some point in the course, we will study *information* complexity wherein interestingly, public coins are obsolete but private coins are (seemingly) necessary.

Definition 11. Let μ be a distribution on inputs (x, y) . We define the distributional communication complexity of f over distribution μ as $D_\mu(f) = \min_\pi \|\pi\|$ where π ranges over all deterministic protocols that output the correct answer on inputs sampled from μ with probability at least $2/3$.

The celebrated Yao’s minimax principle [22] relates distributional and randomized communication complexity (we also refer the interested reader to [5] for a very recent and exciting development on this result).

Proposition 12 (Yao’s minimax principle [22]). *For any problem $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$,*

- (i) $D_\mu(f) \leq R(f)$ for all input distributions μ ; and,
- (ii) $D_\mu(f) = R(f)$ for some input distribution μ .

Yao’s minimax principle gives us a way of proving lower bounds for randomized protocols by instead considering deterministic ones on random inputs (we typically only use the first part which follows from a simple averaging argument—the second part implies that this approach can always give us the tightest possible bound *if* we are able to find the “right” distribution μ).

We prove this proposition in the following. The first part, called the “easy direction” has a one line proof and will be necessary for the rest of this course (so that we can see how easily it can be extended to other settings). The second part, called the “hard direction” has a more interesting proof which we include here for completeness; however, the reader can safely skip this part.

Proof of Proposition 12. We prove each part as follows.

Part (i) – the easy direction: Let π be a randomized protocol for f with $\|\pi\| = R(f)$. Note that π can be seen as first sampling random bits r and then running the deterministic protocol π^r over the input. By the guarantee of the protocol:

$$2/3 \leq \Pr_{(x,y) \sim \mu, r} (\pi^r(x, y) \text{ is correct}) = \mathbb{E}_r \Pr_{(x,y) \sim \mu} (\pi^r(x, y) \text{ is correct}).$$

(because r is chosen independent of x, y)

As maximum is at least as large as expectation, there is some r^* such that

$$\Pr_{(x,y) \sim \mu} (\pi^{r^*}(x, y) \text{ is correct}) \geq 2/3.$$

But now π^{r^*} is a deterministic protocol with $\|\pi^{r^*}\| \leq \|\pi\|$ (by worst-case definition of communication cost) and probability of success more than $2/3$ on μ . As such,

$$D_\mu(f) \leq \|\pi^{r^*}\| \leq \|\pi\| = R(f),$$

as desired.

Part (ii) – the hard direction: Let $C := \max_\mu D_\mu(f)$. We are going to prove that $R(f) = C$ also which implies the second part.

Consider a game⁹ between two players called the *Input player* and the *Algorithm player*. The set of strategies of the Input player are all inputs on $\Omega := \{0, 1\}^N \times \{0, 1\}^N$ and the set of strategies of the Algorithm player are all deterministic protocols with communication cost at most C , denoted by Π ; for fixed N, C , both sets are finite.

For any $(x, y) \in \Omega$ as a strategy of the Input player and any deterministic protocol $\pi \in \Pi$ as the strategy of the Algorithm player, we define:

⁹This is a game-theoretic notation of a game.

- $v((x, y), \pi)$: the outcome of the game which is 1 if $\pi(x, y) = f(x, y)$ (protocol is correct) and is 0 if $\pi(x, y) \neq f(x, y)$ (protocol errs).

On a choice of (pure) strategies $(x, y), \pi$ by the players, we define the payoff of the Algorithm player as $v((x, y), \pi)$ and for the Input player as $-v((x, y), \pi)$. As such, the Algorithm player would like to maximize $v((x, y), \pi)$ (by choosing π), while the Input player tries to minimize it (by choosing x, y). Notice that this is a *zero-sum* game.

Let Δ_Ω denote the set of all distributions on strategies (inputs) of the Input player and Δ_Π denote the set of all distributions on strategies (deterministic protocols) of the Algorithm player. This way, Δ_Ω and Δ_Π denote the set of all mixed strategies for the Input player and Algorithm player, respectively. Considering this is a zero-sum game, Von Neumann’s Minimax Theorem asserts that,

$$\min_{\mu \in \Delta_\Omega} \max_{\pi \in \Delta_\Pi} \mathbb{E}_{(x,y) \sim \mu} [v((x, y), \pi)] = \max_{\pi^r \in \Delta_\Pi} \min_{(x,y) \in \Omega} \mathbb{E}_{\pi \in \pi^r} [v((x, y), \pi)]. \quad (2)$$

The LHS in Eq (2) corresponds to picking any possible distribution on inputs and then running the “best” deterministic protocol on this distribution and measuring the probability of success of the protocol. As $D_\mu(f) \leq C$ for all μ , and by definition, the LHS is at least $2/3$.

The RHS in Eq (2) corresponds to picking any distribution over deterministic protocols, i.e., a (public-coin) randomized protocol, and then running this protocol on “worst” input and measure the probability of the success of the protocol. By the lower bound on LHS and Eq (2), this is at least $2/3$. This means that there exists a randomized protocol π^{r^*} with communication cost C (the arg max of RHS in Eq (2)) that achieves a probability of success at least $2/3$ on any input chosen for Alice and Bob. Hence, $R(f) \leq \|\pi^{r^*}\| \leq C$ as desired, concluding the proof. \square

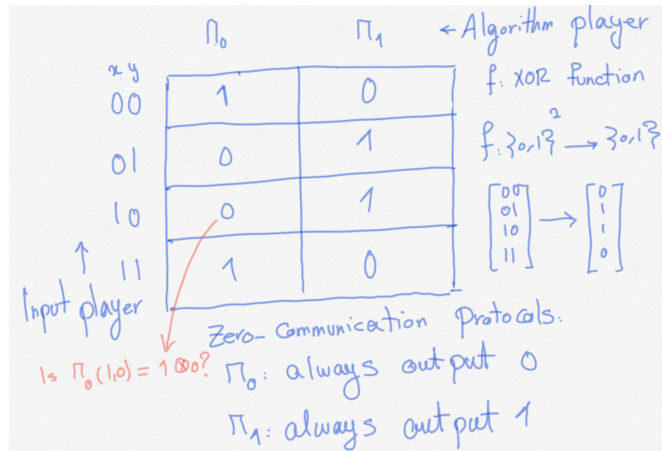


Figure 6: A simple illustration of the game in Yao’s minimax principle proof. Here, we have a toy setting: each player gets a single bit and the function to compute is XOR. We also focus only on zero-error protocols: the one that outputs 0 all the time and the one that outputs 1. In this game, any pure strategy of each player has a counter move that allows the second player to win. However, the best mixed strategy for both Input player and Algorithm player is the uniform distribution, leading to value of $1/2$ for this game, which is the best probability of success for zero-communication protocols.

Application to Graph Streaming Lower Bounds

Let us now formalize the connection between graph streaming and communication complexity lower bounds.

Proposition 13. *Suppose Alg is a $p(n)$ -pass $s(n)$ -space streaming algorithm for some problem $P(G)$ on n -vertex graphs G . Then, there is a communication protocol for the communication problem associated with P that uses $2 \cdot p(n) - 1$ messages between Alice and Bob, and $O(p(n) \cdot s(n))$ communication,*

Proof. Let E_A and E_B be the partitioning of edges of Alice and Bob and define the stream $\sigma = \langle E_A \circ E_B \rangle$ where the ordering of edges inside each E_A and E_B can be anything. Alice will run the streaming algorithm Alg on the first part of the stream and once finished will send the content of memory to Bob. Bob then continues to run the algorithm on the second part of the stream and send the memory at the end to Alice. This requires 2 messages each of size at most $s(n)$ and allows the players to simulate one pass of the streaming algorithm entirely. The players continue doing this except that in the last pass, instead of Bob sending a message to Alice, he simply outputs the solution. This means that in total $2p(n) - 1$ messages get communicated leading to a protocol with $O(p(n) \cdot s(n))$ communication. \square

The takeaway message of [Proposition 13](#) is clear: communication complexity lower bounds imply graph streaming lower bounds. We will use this repeatedly throughout the course. However, an important remark is in order.

Remark. Even though communication complexity lower bounds imply graph streaming lower bounds, often times we cannot hope to use them directly to prove “interesting” lower bounds for graph streams. The communication model is much more stronger than streaming and there are numerous cases when we may have an efficient communication protocol for a problem but no graph streaming algorithms.

Fortunately, the connection between the two models is deeper than this. For instance, [Proposition 13](#) shows that if our goal was to prove a streaming lower bound for *single-pass* algorithms, then we can focus our attention to communication protocols that only involve Alice sending a single message to Bob and Bob outputting the solution; such protocols are much weaker than general communication protocols, allowing us to prove stronger lower bounds (we will see an example of this by the end of this note)—this connection holds for p -pass streaming algorithm and $(2p - 1)$ -round communication protocols.

Throughout the course, we shall see several natural restrictions that we can put on communication protocols that allows us to prove stronger lower bounds and extend them to lower bounds for graph streaming algorithms as well.

4 One Way Communication Complexity: The Index Problem

We are now going to look at one of the most important communication problems in the context of streaming algorithms, the *Index* problem.

Problem 4. In the index communication problem Ind , Alice gets a string $x \in \{0, 1\}^N$ and Bob gets an index $i \in [N]$; the goal for the players is to output x_i , i.e., $Ind(x, i) = x_i$.

Before studying the communication complexity of this problem, let us see how it can be used to prove streaming lower bounds for the connectivity problem.

Theorem 14. *Any p -pass streaming algorithm for deciding whether an n -vertex graph is connected or not requires $\Omega(R(Ind_N)/p)$ bits of space, where Ind is the index problem on $\{0, 1\}^N$ for $N = n - 2$.*

Proof. We are going to prove that $R(connectivity_n) \geq R(Ind_N)$ where $connectivity_n$ is the communication problem on an n -vertex graph whose edges are partitioned between Alice and Bob. The theorem then follows from [Proposition 13](#). The proof is via the following reduction:

- (i) The players set the vertices of the graph to be s, t, v_1, \dots, v_N (so $N + 2 = n$ vertices).
- (ii) Given $x \in \{0, 1\}^N$, Alice adds an edge in E_A between s and v_i for every $i \in [N]$ such that $x_i = 1$.
- (iii) Given $i \in [N]$, Bob adds an edge in E_B between t and s and t and all v_j with $j \neq i$.

We claim that the graph G in this reduction is connected iff $Ind(x, i) = x_i = 1$. This is simply because all vertices other than v_i are connected to t by an edge of Bob and if $x_i = 1$, v_i has an edge to s , hence G is connected, while when $x_i = 0$, v_i is a singleton vertex, making G disconnected.

As this reduction can be done without any communication between the players, Alice and Bob can create G and run a protocol for $connectivity_n$ on G to solve Ind_N as well. Thus, any protocol for the former problem would also solve the latter with the same communication, implying that $R(connectivity_n) \geq R(Ind_N)$. \square

Remark. The type of reduction from Index in [Theorem 14](#) is quite common in the (graph) streaming model and allows one to prove numerous lower bounds for the problems at hand. We shall see several such reductions throughout the course. However, it should also be noted that not every graph streaming lower bound can be proven this easily (including some highly sophisticated reductions from Index).

The above theorem suggests that we can prove a lower bound for space complexity of the connectivity problem by lower bounding $R(Ind)$ instead. Alas, it is easy to see that even $D(Ind) = O(\log n)$ as Bob can simply send his index to Alice and Alice solves the problem. This will not allow us to prove any meaningful lower bound for the streaming connectivity problem. This is precisely what was remarked after [Proposition 13](#): to prove lower bounds for single-pass algorithms, we can focus solely on protocols that involve Alice sending a single message to Bob. This brings us to the final topic of this lecture.

One-way Communication Complexity. In the *one-way* communication model, we only allow Alice to send a single message to Bob and Bob then needs to output the answer. We again define the communication cost of the protocol as the worst-case bit-length of the message of Alice.

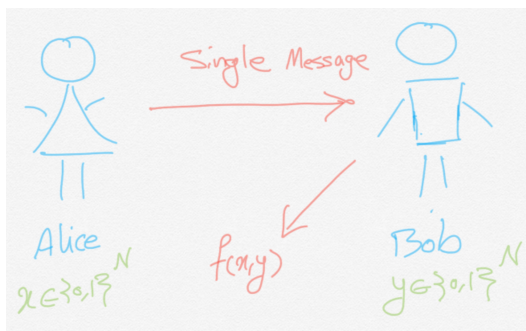


Figure 7: Illustration of a one way protocol.

We can then define:

- One-way deterministic communication complexity – $\vec{D}(f) := \min_{\pi} \|\pi\|$ where π ranges over all deterministic *one-way* protocols for f ;
- One-way randomized communication complexity – $\vec{R}(f) := \min_{\pi} \|\pi\|$ where π ranges over all randomized *one-way* protocols for f with probability of success at least $2/3$;
- One-way distributional communication complexity – $\vec{D}_{\mu}(f) := \min_{\pi} \|\pi\|$ where π ranges over all deterministic *one-way* protocols for f with probability of success at least $2/3$ over the distribution μ ;

Let us examine the one-way communication complexity of Ind . It is easy to prove that $\vec{D}(Ind) \geq n$ as follows: By pigeonhole principle, if the message of Alice has size less than n , then at least two different strings $x \neq y \in \{0,1\}^N$, are mapped to the same message M . Now let i be an index where $x_i \neq y_i$. The output of Bob is a deterministic function of M and i and is thus wrong for one of x_i and y_i , a contradiction.

This argument however does not work for randomized algorithms when the answer is allowed to be wrong with probability $1/3$ (in fact, it is easy to see that $\vec{R}(Ind) \leq 2N/3$). In the following, we are going to use a more general argument to prove that $\vec{R}(Ind) = \Omega(N)$. This will allow us to apply [Theorem 14](#) and prove the desired lower bound for single-pass streaming algorithms of connectivity without proving any lower bound on $R(\text{connectivity})$ directly¹⁰.

Theorem 15. *The one-way randomized communication complexity of Index is $\vec{R}(Ind) = \Omega(N)$.*

Proof of Theorem 15. We prove the lower bound for protocols of Ind that output the correct answer with probability of success at least $1 - \delta$ for any $\delta < 0.1$ as opposed to $2/3$; this is without loss of generality as the communication cost of these protocols are within a constant factor of “standard” protocols by simply running the $1/3$ -error protocol in parallel $O(1)$ time independently and taking the majority answer.

We will use Yao’s minimax principle in part (i) of [Proposition 12](#) by showing that there is a distribution μ where $\vec{D}_\mu^\delta(Ind) = \Omega(N)$ where $\vec{D}_\mu^\delta(Ind)$ here is the distributional complexity of protocols over μ that output the correct answer with probability $\geq 1 - \delta$. The distribution μ is as follows:

- **Distribution μ :** Sample $x \in \{0, 1\}^N$ and $i \in \{0, 1\}^N$ independently and uniformly at random.

Now consider any deterministic one-way protocol π for Ind on the distribution μ with probability of success at least $1 - \delta$. By definition,

$$\Pr_{x \in \{0, 1\}^N} \Pr_{i \in [N]} (\pi \text{ errs on input } (x, i)) \leq \delta; \quad (3)$$

Let \mathcal{X} be the subset of $\{0, 1\}^N$ defined as follows:

$$\mathcal{X} := \left\{ x \in \{0, 1\}^N \mid \Pr_{i \in [N]} (\pi \text{ errs on input } (x, i)) < 2\delta \right\}.$$

We claim that $|\mathcal{X}| \geq 2^{N-1}$; otherwise (recall that choices of i and x in μ are independent and uniform):

$$\begin{aligned} \Pr_{x \in \{0, 1\}^N} \Pr_{i \in [N]} (\pi \text{ errs on input } (x, i)) &= \Pr_{x \in \mathcal{X}} \Pr_{i \in [N]} (\pi \text{ errs on input } (x, i)) + \Pr_{x \in \overline{\mathcal{X}}} \Pr_{i \in [N]} (\pi \text{ errs on input } (x, i)) \\ &\geq \Pr_{x \in \overline{\mathcal{X}}} \Pr_{i \in [N]} (\pi \text{ errs on input } (x, i)) \\ &\geq \frac{|\overline{\mathcal{X}}|}{2^N} \cdot 2\delta \\ &\quad (\text{by the definition of } x \notin \mathcal{X}, \Pr_{i \in [N]} (\pi \text{ errs on input } (x, i)) \geq 2\delta) \\ &> \frac{2^{N-1}}{2^N} \cdot 2\delta = \delta, \\ &\quad (\text{by the contradicting assumption on size of } \mathcal{X}, \text{ we have } |\overline{\mathcal{X}}| > 2^{N-1}) \end{aligned}$$

which contradicts [Eq \(3\)](#). Additionally, for any $x \in \mathcal{X}$, we define:

$$E(x) := \{i \in N \text{ such that } \pi(x, i) \text{ errs}\}.$$

By definition, for any $x \in \mathcal{X}$, $|E(x)| < 2\delta N$. Because of this, across all $x \in \mathcal{X}$, the number of choices for $E(x)$ is at most:

$$\begin{aligned} \# \text{ of } (< 2\delta N)\text{-subsets of } [N] &= \sum_{k=0}^{2\delta N-1} \binom{N}{k} \leq \left(\frac{1}{2\delta}\right)^{2\delta N} = 2^{2\delta \cdot \log(1/2\delta)N}. \\ &\quad (\text{as } \binom{a}{b} \leq (e \cdot a/b)^b \text{ and that } \binom{a}{b} \leq \binom{a}{b+1} \text{ for } b+1 \leq a/2) \end{aligned}$$

¹⁰Strictly speaking, connectivity_n is not the best example of proving one-way lower bounds in place two-way ones since one can even prove that $R(\text{connectivity}_n) = \Omega(n)$ (as we shall see later in the course); if you want a “real” example, consider directed s - t reachability problem: $R(s\text{-}t\text{-reachability}) = \tilde{O}(n)$ but $\vec{R}(s\text{-}t\text{-reachability}) = \Omega(n^2)$; you are strongly encouraged to prove both statements on your own (the first one uses BFS and the second is a reduction from Index).

Combined with the lower bound of 2^{N-1} on the size of \mathcal{X} and pigeon hole principle, this implies that there is a set \mathcal{X}^* such that for all $x, y \in \mathcal{X}^*$, $E(x) = E(y)$ and

$$|\mathcal{X}^*| \geq \frac{2^{N-1}}{2^{2\delta \cdot \log(1/2\delta)N}} = 2^{(1-2\delta \cdot \log(1/2\delta))N-1}.$$

We are now done by a similar argument as in the deterministic case. We claim that any $x \in \mathcal{X}^*$ should be mapped to a distinct message $M(x)$; this is simple because we can recover x from $M(x)$ by simply computing the string $y = \pi(x, 1), \pi(x, 2), \dots, \pi(x, n)$ which is a function of $M(x)$ only and then flip every bit in $E(x)$ which is a function of \mathcal{X}^* and not x . This implies that:

$$\|\pi\| \geq \log |\mathcal{X}^*| \geq (1 - 2\delta \cdot \log(1/2\delta))N - 1 = \Omega(N),$$

for any $\delta \leq 0.1$. □

Plugging the bound in [Theorem 15](#) in [Theorem 14](#) (for $p = 1$ and after replacing $R(Ind)$ by $\vec{R}(Ind)$ in the theorem statement), we obtain a lower bound of $\Omega(n)$ bits on the space needed by single-pass streaming algorithms for connectivity (this is within an $O(\log n)$ factor of the optimal bounds proven in [\[21\]](#)).

References

- [1] K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467, 2012. [2](#)
- [2] N. Alon, S. Hoory, and N. Linial. The Moore bound for irregular graphs. *Graphs Comb.*, 18(1):53–57, 2002. [6](#)
- [3] A. Babu and J. Radhakrishnan. An entropy based proof of the Moore bound for irregular graphs. *CoRR*, abs/1011.1058, 2010. [6](#)
- [4] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 623–632, 2002. [1](#)
- [5] S. Ben-David and E. Blais. A new minimax theorem for randomized algorithms. *CoRR*, abs/2002.10802. To appear in FOCS 2020, 2020. [10](#)
- [6] B. Bollobás. *Extremal graph theory*. Courier Corporation, 2004. [6](#)
- [7] A. Chakrabarti, G. Cormode, and A. McGregor. Annotations in data streams. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 222–234, 2009. [2](#)
- [8] G. Cormode, J. Thaler, and K. Yi. Verifying computations with streaming interactive proofs. *Proc. VLDB Endow.*, 5(1):25–36, 2011. [2](#)
- [9] M. S. Crouch, A. McGregor, and D. Stubbs. Dynamic graphs in the sliding-window model. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 337–348, 2013. [2](#)
- [10] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, pages 531–543, 2004. [1](#), [2](#)
- [11] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the data-stream model. *SIAM J. Comput.*, 38(5):1709–1727, 2008. [7](#)

- [12] T. Gur and R. Raz. Arthur-merlin streaming complexity. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 528–539, 2013. [2](#)
- [13] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In *External Memory Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, May 20-22, 1998*, pages 107–118, 1998. [1](#)
- [14] C. Konrad, F. Magniez, and C. Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 231–242, 2012. [2](#)
- [15] E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, 1997. [7](#)
- [16] A. McGregor. Graph stream algorithms: a survey. *SIGMOD Rec.*, 43(1):9–20, 2014. [2](#)
- [17] A. McGregor, S. Vorotnikova, and H. T. Vu. Better algorithms for counting triangles in data streams. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 401–411, 2016. [2](#)
- [18] I. Newman. Private vs. common random bits in communication complexity. *Inf. Process. Lett.*, 39(2):67–71, 1991. [8](#)
- [19] A. Rao and A. Yehudayoff. *Communication Complexity: and Applications*. Cambridge University Press, 2020. [7](#)
- [20] T. Roughgarden. Communication complexity (for algorithm designers). *Found. Trends Theor. Comput. Sci.*, 11(3-4):217–404, 2016. [7](#)
- [21] X. Sun and D. P. Woodruff. Tight bounds for graph problems in insertion streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, pages 435–448, 2015. [3](#), [4](#), [15](#)
- [22] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 222–227, 1977. [10](#)