# Dynamic Graph Sketching

COMS 4232: Advanced Algorithms Final Project

Ashwin Padaki and Krish Singal

May 17, 2023

## Introduction

Linear sketching is a general technique for approximating high-dimensional vectors via linear dimensionality reduction. In the streaming setting, it is used to efficiently solve problems such as norm estimation and heavy hitters. Dynamic graph sketching, formally introduced in [AGM12], is the problem of computing properties of multi-graphs given stream access to its nodes and edges. The results outlined operate largely within the semi-streaming model, wherein $\tilde{O}(n)$ bits of memory are allowed. This survey has three main contributions. (1) We define the problem, detail preliminaries on $\ell_p$ sampling, and outline an algorithm to compute spanning forests in the semi-streaming model. (2) We outline some of its applications to various other graph problems, such as bipartite testing and approximate MST computation. (3) Finally, we outline a proof of the optimality of the AGM sketch.

## Contents

# 1 Preliminaries

## 1.1 Dynamic Streaming Model

This survey outlines results within the context of the dynamic graph streaming model, which is defined below.

**Definition 1 (Dynamic Graph Stream).** Consider a multi-graph $G = (V, E)$ with $|V| = n, |E| = m$. Here, $E$ should be interpreted as a vector whose coordinates indicate the multiplicity of a given edge in the graph. Accordingly, the basis vector $e_i$ is the indicator of a single edge $i \in [m]$. Graph $G$ may then be represented as a dynamic graph stream $S = \langle s_1, ..., s_t \rangle$ where $s_i = (e_i, \Delta_i) \in E \times \mathbb{Z}$ and $\sum_{i=1}^{t} e_i \Delta_i = E$.

Note that dynamic graph streams allow an edge $e \in E$ to be inserted or deleted multiple times during the course of stream $S$, a less restrictive version of the insertion-only model.

## 1.2 $\ell_p$ Sampling

When processing data streams, it is no surprise that information about the $p$-norm $\ell_p(\mathbf{x}) := ||\mathbf{x}||_p$ turns out to be a useful metric. Rather than the explicit estimation of the $p$-norm, $\ell_p$ sampling asks a related but nonetheless powerful question: can we efficiently sample from the coordinates of $\mathbf{x}$ according to the distribution induced by $\ell_p(\mathbf{x})$?

**Definition 2 ($\ell_p$ Sampler).** Consider a dynamic stream and the associated aggregate vector $\mathbf{x} \in \mathbb{R}^n$. We say that an algorithm is an $\ell_p$ sampler with error $(\epsilon, \delta)$ if it returns $\perp$ (failure) with probability at most $\delta$ and otherwise outputs coordinate $i$ of $\mathbf{x}$ with probability $(1 \pm \epsilon) \cdot \frac{|\mathbf{x}_i|^p}{\ell_p(\mathbf{x})^p}$.

In the above definition, note that there are two sources of error: $\delta$ is an upper bound on the probability of general failure, while $\epsilon$ controls the maximum multiplicative error on the true $\ell_p$ distribution in any successful case.

While $\ell_p$ sampling is an interesting area of study, for the purposes of graph streaming we care most about sketches for $\ell_0$ sampling. $\ell_0$ sampling simply requires that we return a uniformly random nonzero coordinate of the stream vector $\mathbf{x}$. This establishes connections with the related problem of linear sparse recovery.

## 1.3 Linear Sparse Recovery

Given a sparsity threshold $s \leq n$, a linear sparse recovery protocol consists of a dimension parameter $k$, a distribution of linear functions $L : \mathbb{R}^n \to \mathbb{R}^k$, and an algorithm which, on input $L(\mathbf{x})$, has the following two guarantees:

- If $\ell_0(\mathbf{x}) \leq s$ (i.e. $\mathbf{x}$ is "$s$-sparse"), then the algorithm outputs $\mathbf{x}$ with probability 1.
- If $\ell_0(\mathbf{x}) > s$ the algorithm outputs $\perp$ with probability 9/10.

We denote by SparseRecovery$_s$ the procedure with these guarantees and which provides query access to the output $\mathbf{x}$. Sparse recovery is a fairly well-studied problem and has deep connections to signal processing and coding theory. We will use the following positive result without proof, but a more in-depth discussion of linear sparse recovery can be found in Section 2.3 of [CF14].

**Lemma 1** *For $\mathbf{x} \in \mathbb{R}^n$ and $1 \leq s \leq n$, there exists a linear sparse recovery protocol where $k = O(s)$ and $L$ can be determined from $O(k \log n)$ random bits.*

The fact that $L$ is linear is crucial. Particularly in our dynamic stream context, we can maintain $s$-sparse recoverable vectors using only $O(k)$ space via the sketch $L$.

### 1.4 Algorithm for $\ell_0$ sampling

Through a clever application of this sparse recovery protocol, [JST11] constructs an efficient streaming algorithm for $\ell_0$ sampling with error $(0, \delta)$. That is, with probability at least $1-\delta$, the algorithm is guaranteed to return a uniformly random element of $J = \{i \in [n] : \mathbf{x}_i \neq 0\}$.

A naive approach would be to directly store and sample from $J$, but of course $|J| = O(n)$ in the worst case, and we would like a sublinear space algorithm. A key observation is that we can *almost* sample uniformly from $J$ with the following two-step process: pick a uniformly random subset $I \subseteq [n]$ of some given size, and then sample uniformly from $I \cap J$, which can be accomplished with $O(|I \cap J|)$ space. The major caveat is that if $|I|$ is too small, we run the risk that $I \cap J = \varnothing$, in which case our approach would fail. We might therefore intuit that there is some $I$ such that both our demands are met with high probability: $I \cap J \neq \varnothing$, and $|I \cap J|$ is sufficiently small. This intuition turns out to be correct, and while the optimal $|I|$ cannot be determined a priori, a clever trick allows us to find an appropriate setting.

We can do even better with sparse recovery: $|I \cap J|$ being sufficiently small (less than some threshold $s$) is equivalent to $\mathbf{x}_I$, the projection of $\mathbf{x}$ onto coordinates $I$, being $s$-sparse. Using $\textsc{SparseRecovery}_s$ and maintaining $L(\mathbf{x}_I)$, our space is bottlenecked not by $|I \cap J|$ but by $|L(\mathbf{x}_I)| = O(s)$. It turns out that a sufficient threshold is $s = O(\log(\frac{1}{\delta}))$, which notably does not depend on $n$ at all.

**Algorithm 1** $\ell_0$ Sampler

**procedure** $\ell_0$ SAMPLER
  Set $s := \lceil 4 \log(\frac{1}{\delta}) \rceil$
  Set $I_0 := [n]$
  **for** $k = 0, \ldots, \lfloor \log n \rfloor$ **do**
    **if** $k > 0$ **then**
      Draw $I_k \subseteq [n]$ uniformly at random with $|I_k| = 2^k$
    Set $\mathbf{x}_{I_k} \in \mathbb{R}^{2^k}$ (or $\mathbb{R}^n$ if $k = 0$) as the projection of $\mathbf{x}$ onto coordinates $I_k$
    Set $\mathbf{y}_k = \textsc{SparseRecovery}_s(\mathbf{x}_{I_k})$
    **if** $\mathbf{y_k} = \perp$ or $\mathbf{y_k} = \mathbf{0}$ **then continue**
    **else**
      Set $J_k = \{i \in I_k : (\mathbf{y}_k)_i \neq 0\}$
      Draw $j \in J_k$ uniformly at random
      **return** $j$
  **return** "FAIL"

### 1.5 Proof of Correctness

We will first show that this $\ell_0$ sampler has error $(0, \delta)$. Observe that whenever we do not return "FAIL", we return a random coordinate from some $J_k = J \cap I_k$. Since each set $I_k$ is uniformly random, a random coordinate from $J_k$ is also a random coordinate of $J$, confirming that $\epsilon = 0$. Then, it suffices to show that we return "FAIL" with probability at most $\delta$. To prove this, we will show the existence of some $k$ for which $\mathbf{x}_{I_k}$ is nonzero and $s$-sparse (or equivalently $1 \leq |J_k| \leq s$) with probability at least $1 - \delta$; if this holds, then sparse recovery yields $\mathbf{y}_k = \mathbf{x}_{I_k} \notin \{\perp, 0\}$ with probability 1, in which case we will return some $j$ and avoid failure.

First, we remark that if $|J| \leq s$ then $\mathbf{x}_{I_0} = \mathbf{x}$ is nonzero and $s$-sparse with probability $1 > 1 - \delta$. If $|J| > s$ then for any given $k \geq 1$ we have $\mathbb{E}[|J_k|] = \mathbb{E}[|I_k \cap J|] = \frac{2^k \cdot |J|}{n}$, since each coordinate in $|J|$ has probability $\frac{2^k}{n}$ of also being in $I_k$. The quantity $2^k$ ranges over all powers of 2 from 2 up to $2^{\lfloor \log n \rfloor}$. Therefore, the smallest setting of $\frac{2^k \cdot |J|}{n}$ is $\frac{2|J|}{n} \leq 2$ while the largest setting is $\frac{2^{\lfloor \log n \rfloor} \cdot |J|}{n} \geq \frac{n/2 \cdot |J|}{n} > \frac{s}{2}$. Hence, there is some $k^*$ for which $\mathbb{E}[|J_{k^*}|] = \frac{2^{k^*}|J|}{n} \in [s/3, 2s/3]$. Since $|J_{k^*}|$ can also be written as a sum of $|J| > s$ i.i.d indicator variables, a simple Chernoff bound allows us to conclude that this $|J_{k^*}| \in [1, s]$ except with probability at most $e^{-\log\left(\frac{1}{\delta}\right)} = \delta$.

There is one small technicality we must address before we can conclude that we indeed a uniformly random element in $J$. Recall that when $\mathbf{x}$ is not $s$-sparse, with probability $1/10$ the algorithm $\textsc{SparseRecovery}_s$ may return not $\perp$ but an arbitrary vector, and a random nonzero coordinate of this vector may not even be an element of $J$. To fix this, we note that by running the algorithm $\textsc{SparseRecovery}_s$ $\text{poly}(n)$ times, we can shrink this error probability from $1/10$ to, say, $\frac{\delta}{n}$. Applying a union bound over all calls to the algorithm, we can assume all of our calls to $\textsc{SparseRecovery}_s$ with non $s$-sparse inputs will return $\perp$ with probability $1 - \delta$. Hence, we will have an overall success probability of at least $(1 - \delta)^2 \geq (1 - 3\delta)$, so setting constants appropriately completes this part of the proof.

Finally, we compute the space required for our $\ell_0$ sampler. Observe that during the stream, we must persistently store the sketches used by our $\textsc{SparseRecovery}_s$ instances and the random bits describing the corresponding linear update functions.

- The $\textsc{SparseRecovery}_s$ sketches take $O(s) = O(\log\left(\frac{1}{\delta}\right))$ space, and we store $\lfloor \log n \rfloor$ such sketches.

  $\longrightarrow O(\log n \cdot \log\left(\frac{1}{\delta}\right))$ space

- The random bits describing the sketch update functions take $O(s \log n) = O(\log n \cdot \log\left(\frac{1}{\delta}\right))$ space, and we store $1 + \lfloor \log n \rfloor$ such functions.

  $\longrightarrow O(\log^2 n \cdot \log\left(\frac{1}{\delta}\right))$ space

In total, then, our $\ell_0$ sampler uses $O(\log^2 n \cdot \log\left(\frac{1}{\delta}\right))$ space. Lastly, it turns out that $O(\log^2 n)$ random bits are sufficient to sample each $I_k \leftarrow 2^{[n]}$ and also each $j \leftarrow J_k$ in a way that is "close enough" to uniformly random. We will not elaborate on this detail, but we refer the reader to Nisan's pseudorandom generator for more information.

# 2 AGM Sketch

We now study the AGM sketch, outlined in [AGM12], which employs $\ell_0$ sampling to construct a one-pass streaming algorithm to compute spanning forest.

## 2.1 Sketch Updates – Independence and Adaptivity

Consider the following application of the $\ell_0$ sampling result from section 1.4 to a graph $G = (V, E)$ with corresponding adjacency-like matrix $A_G \in \{-1, 0, 1\}^{n \times \binom{n}{2}}$ defined as

$$A_G[i, (j, k)] = \begin{cases} 1 \text{ if } i = j \text{ and } (j, k) \in E \\ -1 \text{ if } i = k \text{ and } (k, j) \in E \\ 0 \text{ otherwise} \end{cases}$$

Then define the *characteristic vector* $a^i$ as row $i$ of $A_G$ and let $\mathcal{S}$ denote the linear $\ell_0$ sampling sketch. We see that $\mathcal{S}(a^i)$ provides a random sample from $a^i$ according to the distribution induced by the $\ell_0$ norm. In particular, $\mathcal{S}(a^i)$ is a uniformly random neighbor of $i \in V$ (assuming $G$ is undirected) since every neighbor of $i$ contributes a 1 and $-1$ to characteristic vector $a^i$. The distribution induced by the $\ell_0$ norm of $a^i$ is then $\mathbb{P}(j \mid (i, j) \in \mathbb{E}) = \frac{2}{2 \cdot d(i)} = \frac{1}{d(i)}$ – the uniform distribution over all neighbors of node $i$. By virtue of $\mathcal{S}$'s linearity, $\mathcal{S}(a^u_n) = \mathcal{S}(a^u_{n-1}) + \mathcal{S}(e_n \Delta_n)$ where $a^u_t$ denotes node $u$'s characteristic vector at time $t$ and $e_t \Delta_t$ is the stream update at time $t$.

Notice that $\mathcal{S}$ is a function of $a^u$ and the sketch's internal randomness. In particular, $\mathcal{S}$ draws $O(\log^2(n) \log(\frac{1}{\delta}))$ random bits to choose the set of $I_k$'s and an eventual nonzero element from recovered $\mathbf{x}$, as described in section 1.4. Then, repeated queries to $\mathcal{S}(a^u)$ must return the same answer. Put differently, we think of drawing a deterministic linear sketch $\mathcal{S}$ uniformly at random from a total of $2^{\log^2(n) \log(\frac{1}{\delta})}$ possible such linear sketches. It is then clear that $\mathcal{S}(a^u_{t_1})$ is not independent of $\mathcal{S}(a^u_{t_2})$ for any two instances of $a^u$ during the stream. As will be seen, fresh sketches must be drawn per query to maintain independence between the randomly sampled neighbors of a given vertex.

Another caveat is the invalidity of adaptive updates to sketch $\mathcal{S}$. One may be tempted to extract all neighbors of $u$ by iteratively querying $\mathcal{S}(a^u)$ and updating the sketch with the removal of the retrieved neighbor. This, however, then implies that $O(n)$ bits of information can be retrieved from a sketch of size $O(\log^2 n \log \frac{1}{\delta})$. Information theoretically, we know that this can never occur. Mechanically, such adaptive queries break down the success guarantees given by the $\ell_0$ sampler, in turn only allowing $\log^2 n \log \frac{1}{\delta}$ bits of information to be extracted from the sketch on average.

## 2.2 Algorithm for Spanning Forest

To compute the spanning forest, we first prove the following lemma then present a Boruvska-like algorithm for the spanning forest computation.

**Lemma 2** *Let $T \subseteq V$. We denote $E_T$ to be the set of edges crossing the cut $T$. Then for $\mathbf{x} = \sum_{i \in T} a^i$, we have $|E_T| = \ell_0(\mathbf{x})$ and $\mathcal{S}(\mathbf{x}) \sim unif(E_T)$.*

**Proof**. Consider three sets of edges $A = \{(i,j) \mid i,j \in T\}$, $B = \{(i,j) \mid (i \in T \text{ and } j \notin T)\}$, and $C = \{(i,j) \mid (i \notin T \text{ and } j \in T)\}$. Notice that $B \cup C = E_T$.

For any $(i,j) \in A$ notice that $a^i[(i,j)] = 1$, $a^j[(i,j)] = -1$, and $a^k[(i,j)] = 0$ for $k \neq i,j$ and $k \in T$. Therefore, $\mathbf{x}[(i,j)] = a^i[(i,j)] + a^j[(i,j)] + \sum_{k \in T | k \neq i,j} a^k(i,j) = 1 - 1 + 0 = 0$. Similarly, for $(i,j) \in B$, $\mathbf{x}[(i,j)] = a^i[(i,j)] + \sum_{k \in T | k \neq i} a^k[(i,j)] = 1$. Lastly, for $(i,j) \in C$, $\mathbf{x}[(i,j)] = a^j[(i,j)] + \sum_{k \in T | k \neq i} a^k[(i,j)] = -1$. Therefore, $\ell_0(\mathbf{x}) = |B| + |C| = |E_T|$.

The distribution induced by the $\ell_0$ norm on $\mathbf{x}$ is then given by $\frac{1}{|E_T|}$ for $(i,j) \in E_T$ – namely, the uniform distribution over $E_T$. □

By the linearity of sketch $\mathcal{S}$, we have that $\mathcal{S}(a^{u_1}) + \mathcal{S}(a^{u_2}) = \mathcal{S}(a^{u_1} + a^{u_2})$. Lemma 2 then shows that $\mathcal{S}(a^{u_1} + a^{u_2})$ samples a neighbor from the neighborhood $N_{u_1} \cup N_{u_2}$ uniformly at random (where $N_v$ denotes the neighbors of $v$). Naturally, this leads to the following algorithm to compute spanning forests in the dynamic streaming model.

**Algorithm 2** Spanning Forest

```
1: procedure SPANNING FOREST
2:     Set t = O(log(n))
3:     Draw S_1^i, ..., S_n^i ℓ_0 sampling sketches uniformly at random for i ∈ [t]
4:     // S_k^i is the ith copy of S(a^k)
5:     Initialize V̂ = V
6:     for i ∈ [t] do
7:         for s ∈ V̂ do
8:             s' := Σ_{v∈s} S_v^i = S^i_{Σ_{v∈s} a^v}
9:             Merge s and s'
       return V̂
```

The procedure maintains $\hat{V}$ which contains one representative for each connected component. $s'$ is an $\ell_0$ sampled neighbor of connected component $s$. The `Merge` subroutine combines components $s$ and $s'$. Notice that by section 1.4, we can take $\mathcal{S}$ to be $\ell_0$ samplers with $\epsilon = 0$ and constant $\delta = \frac{1}{100}$. Because $cc(G) - |\hat{V}|$ is reduced by a constant factor with each iteration, only $t := O(log(n))$ stages of the algorithm are required. In accordance with the independence and adaptivity issues discussed in 2.1, note that we must use a fresh sketch for each node per timestep. Then, `Spanning-Forest` is an $O(nt\log^2(n)) = O(n\log^3(n))$ space algorithm.

## 2.3 Applications via Reduction

We now present two applications of the AGM sketch as proven in [AGM12].

### 2.3.1 Bipartiteness

We present a solution to bipartite testing within the semi-streaming model via a reduction to the spanning forest problem. In particular, given a graph $G = (V,E)$ we construct the graph

$D(G) = (V', E')$ where $V' = \{v_1 \mid v \in V\} \cup \{v_2 \mid v \in V\}$ and $E' = \{(u_1, v_2) \mid (u, v) \in E\} \cup \{(u_2, v_1) \mid (u, v) \in E\}$. Then,

**Lemma 3** *Let $cc(G)$ denote the number of connected components in graph $G$. Then, $cc(D(G)) = 2cc(G)$ iff $G$ is bipartite.*

**Proof**. Let $k = cc(G)$ and $G_1, ..., G_k$ be the connected components of $G$. By definition, $G_i$ and $G_j$ for $i, j \in [k]$ are disconnected, meaning that $D(G)$ contains isolated components (not necessarily connected) $D(G_1), ..., D(G_k)$. Thus, $cc(D(G)) = \sum_{i=1}^{k} cc(D(G_i))$.

We first show that for any bipartite component $G_i$, $D(G_i)$ contains exactly 2 connected components. By virtue of $G_i$ being connected, there exists a path from vertex $u \in V_i$ to any $v \in V_i$. This means that $u_1$ can reach either $v_1$ or $v_2$ for every $v \in V_i$. In particular, if $u_1$ can reach $v_j$ then $u_2$ can reach $v_{3-j}$ for $j \in \{1, 2\}$.

Let us then denote the nodes reachable by $u_1$ and $u_2$ as $S_1$ and $S_2$ respectively. From the above argument, we conclude that $S_1 \cup S_2 = V'$. We claim that $S_1$ and $S_2$ are separate connected components of $G_i$. To see that they cannot be merged, note that any path from $u_1$ to $u_2$ must have odd length (as discussed above). This, however, corresponds to an odd cycle in $G_i$ – contradiction. Therefore, $D(G_i)$ contains exactly 2 connected components. By the same line of reasoning, for $G_i$ that is not bipartite, there must exist an odd cycle. Let $u$ be a vertex in this odd cycle. It follows that $S_1$ and $S_2$ are connected since a path between $u_1$ and $u_2$ exists in $D(G_i)$. Therefore, $D(G_i)$ contains exactly one connected component.

To finish the argument, notice that for bipartite graph $G$, every $G_i$ must be bipartite. Then, $\sum_{i=1}^{k} cc(D(G_i)) = 2k$. For a graph $G$ that is not bipartite, there must exist a $G_i$ which is not bipartite. Then, $\sum_{i=1}^{k} cc(D(G_i)) < 2k$. □

We then maintain sketches of the characteristic vectors from adjacency matrix $A_{D(G)}$ and compute $D(G)$'s spanning forest using the AGM sketch in section 2.1 to obtain an $O(n \log^3 n)$ space algorithm for dynamic bipartite testing.

### 2.3.2 Approximate MST

We can also compute $(1+\epsilon)$ approximate MST via a reduction to the spanning forest problem. Let $G = (V, E)$ be a weighted graph with weights in the range $[1, W]$ where $W = poly(n)$. Denote $G_i$ to be the edge-induced subgraph given by edges with weight at least $(1+\epsilon)^i$.

**Lemma 4** *Let $\lambda_i = (1+\epsilon)^{i+1} - (1+\epsilon)^i$ and $r = \lceil \log_{1+\epsilon}(W) \rceil$. Then,*

$$w(T) \leq n - (1+\epsilon)^r + \sum_{i=0}^{r} \lambda_i \cdot cc(G_i) \leq (1+\epsilon) w(T)$$

**Proof**. Construct weighted graph $G'$ by rounding every edge weight up to the nearest power of $(1+\epsilon)$. Every edge weight is scaled by at least a factor of 1 and at most a factor of $(1+\epsilon)$, therefore

$$w(T) \leq w(T') \leq (1+\epsilon) w(T)$$

where $T'$ is an MST for $G'$. Consider the construction of $T'$ via Kruskal's algorithm. Notice that Kruskal's algorithm will greedily pick $n - cc(G_1)$ weight 1 edges (any more will result in a cycle). At this point, there will be $cc(G_1)$ connected components. Kruskal's will then pick $cc(G_1) - cc(G_2)$ weight $(1+\epsilon)$ edges. In general, on iteration $i$, Kruskal's picks $cc(G_i) - cc(G_{i+1})$ weight $(1+\epsilon)^i$ edges. Therefore, the total weight of $T'$ is given by

$$\begin{aligned} w(T') &= (n - cc(G_1)) + \sum_{i=1}^{r-1} (1+\epsilon)^i (cc(G_i) - cc(G_{i+1})) \\ &= n - (1+\epsilon)^r + \sum_{i=0}^{r} \lambda_i cc(G_i) \leq (1+\epsilon) w(T) \end{aligned}$$

□

To compute a $(1+\epsilon)$-approximation of $w(T)$, we must only compute all $cc(G_i)$. Observe that $G_i$ is a subgraph of $G_{i+1}$. Because the edge weights in $G_i$ are smaller than those of $G_{i+1}$, it follows that $T_i$ is a subgraph of $T_{i+1}$ (where $T_i$ denotes the MST of $G_i$). In particular, the spanning forest of $G_i$ is a subgraph of the spanning forest of $G_{i+1}$. We can then compute $cc(G_1), ..., cc(G_r)$ with a total of $\log(n)$ merge iterations. Notice that $r = \frac{\log(poly(n))}{1+\epsilon} = O(\frac{\log(n)}{\epsilon})$.

Just as in the AGM sketch, we must store sketches for each $G_i$ – taking $O(nr\log^2(n)) = O(\epsilon^{-1} n \log^3(n))$ space in total. The global $O(\log(n))$ merge steps require $O(n\log^2(n) \cdot \log(n)) = O(n\log^3(n))$ space. Thus, for bounded $\epsilon$, there exists a one-pass algorithm for approximate MST using $O(\epsilon^{-1} n \log^3(n))$ space.

# 3 Optimality

In Section 2, we used the AGM sketch to provide an efficient sketch for the spanning forest algorithm. This motivates the question: can we do even better? In this section, we overview a proof found in [NY19] of a matching space lower bound; that is, the AGM sketch is actually space-optimal for computing spanning forest! We begin with a short review of communication complexity, a tool we will need to prove this lower bound.

## 3.1 Communication Complexity – a quick primer

Traditional computational complexity evaluates the complexity of certain problems by the amount of computation needed to evaluate the output for a given input. Communication complexity asks a different question: for a given relation $f(x, y)$ whose inputs known by two different parties – Alice only knows $x$ and Bob only knows $y$ – how many bits of communication between the parties are necessary for one party to correctly output an element in $f(x, y)$, assuming each party can do unlimited computation on their own? A natural extension is the *n-fold* problem, $f^n(x_1, y_1, ..., x_n, y_n)$, in which Alice and Bob must compute $n$ independent instances of $f(x_i, y_i)$. Randomized communication complexity (in the public coin model) allows Alice and Bob to agree on a random communication protocol, with the goal of outputting a correct $f(x, y)$ except with some small failure probability $\delta$.

In the context of streaming problems, we often consider the one-way communication setting, wherein Alice is allowed to send only a single message $m$ to Bob, after which Bob must output a value. The reason for this is that efficient streaming algorithms for a given problem $P$ give rise to efficient one-way communication protocols for certain relations $f(x, y)$ that are determined by $P$:

1. Alice initializes a program implementing the streaming algorithm for $P$, and issues a certain sequence of updates to her sketch determined by her input $x$.

2. Alice sends a message $m$ containing the memory of the sketch to Bob.

3. Bob issues a certain sequence of updates determined by his input $y$. Bob then evaluates $P$, after which he can output a candidate solution to $f(x, y)$.

We observe that the communication cost of the above protocol (the length of $m$) is precisely the size of the sketch for solving $P$. Therefore, known lower bounds for certain communication problems translate directly to lower bounds for the corresponding streaming problem! While the condition that $P$ fully determines $f$ might seem overly restrictive at first, it holds in various settings. This is best shown through an example that is particularly relevant for our purposes, namely the communication problem of one-way *universal relation*.

**Definition 3** We define the one-way universal relation problem $\mathbf{UR}^{\subset}$ as follows. Fixing a universal set $U$, Alice receives a set $S \subset U$ and Bob receives a set $T \subset U$, with the promise that $T \subsetneq S$. Upon receiving a message $m$ from Alice, Bob must output some $i \in S \backslash T$. As before, we can extend the universal relation problem to the $n$-fold setting, in which Bob must return $i_1, \ldots, i_n$ with $i_j \in S_j \backslash T_j$.

### 3.2 Reducing Spanning Forest to $n$-fold $\mathbf{UR}^{\subset}$

We illustrate a reduction from the spanning forest problem discussed in Section 2 to the $n$-fold version of $\mathbf{UR}^{\subset}$ in the proof of the following lemma.

**Lemma 5** *Suppose there is a dynamic graph streaming algorithm $A$ on a $2n$-node graph that outputs a correct spanning forest (with failure probability $\delta$) using $C$ bits of memory. Then, there is a randomized one-way communication protocol for $n$-fold $\mathbf{UR}^{\subset}$ with $U = [n]$ using $C$ bits of communication with success probability $1 - \delta$.*

**Proof**. Consider a graph $G$ on $2n$ nodes, whose vertices are indexed $v_1, \ldots, v_n, w_1, \ldots, w_n$. Consider the following communication protocol for $n$-fold $\mathbf{UR}^{\subset}$:

1. Alice simulates $A$ on an initially empty $G$. For each $i \in [n]$, and for each $x \in S_i$, she feeds to $A$ the *insertion* update $(v_x, w_i)$ to $G$.

2. Alice sends the memory of $A$ to Bob.

3. For each $i \in [n]$, and for each $x \in T_i$, Bob feeds to $A$ the *deletion* update $(v_x, w_i)$. Bob then obtains a spanning forest from $G$. This will allow Bob to obtain an edge $(v_x, w_j)$ for each $j \in [n]$ and he sets $i_j := x$ with the guarantee that $x \in S_j \backslash T_j$.

To verify the correctness of the above protocol, observe that after Alice's insertions and Bob's deletions, the edges $(v_x, w_i)$ remaining in $G$ correspond to all $x, i \in [n]$ for which $x \in S_i \backslash T_i$. By the promise of $\mathbf{UR}^{\subset}$, each $w_j$ will have at least one incident edge, meaning a spanning forest must contain at least one incident edge of $w_j$, which allows Bob to return some $i_j \in S_j \backslash T_j$, as desired. Clearly, the above protocol takes at most $C$ bits of communication, and its failure probability is at most the failure probability of $A$, namely $\delta$. □

Using this lemma, finding a lower bound for the amount of bits necessary to solve the spanning forest problem on dynamic graph streams reduces to finding a lower bound for the amount of communication needed to solve the $n$-fold $\mathbf{UR}^{\subset}$ problem in the randomized one-way setting. As it turns out, a well-known result known as Yao's principle equates randomized communication complexity with distributional communication complexity. That is, if we show the existence of a distribution $\mathcal{D}$ over the input space for which any deterministic protocol must use $C$ bits of communication, we guarantee that any randomized protocol must use at least $C$ bits of communication. It turns out that in order to find the desired distributional lower bound, we must first introduce some information theoretic notions, which we will do in the following subsection.

### 3.3 $\theta$-Protocols and a Direct Product Lemma

Given a function $f$ and corresponding protocol $\pi$ with communication cost $C$, we can trivially upper bound the communication cost of the $n$-fold problem $f^n$ by $Cn$. Similarly, a randomized protocol with cost $C$ and success probability $p$ trivially yields an $n$-fold protocol with cost $Cn$ and success probability $p^n$. But are these optimal in general? Conversely, given an $n$-fold protocol, what is the optimal one-fold protocol? Here we define preliminaries in information theory, the notion of $\theta$-protocols, and outline a surprising result addressing these questions.

**Definition 4** Let $X$ be a random variable. The entropy of $X$ is then $H(X) = \sum_x -p(x)\log(p(x))$.

The entropy of a random variable intuitively quantifies how random it is. The reader is encouraged to verify that a random variable taking on the uniform distribution has maximal entropy, whereas that taking on a Dirac delta distribution has minimal entropy.

**Definition 5** The mutual information between random variables $X$ and $Y$ is given by $I(X;Y) = H(X) - H(X \mid Y)$. Similarly, the conditional mutual information $I(X;Y \mid Z) = H(X \mid Y) - H(X \mid Y, Z)$

The mutual information between $X$ and $Y$ intuitively quantifies the amount of information random variable $Y$ contains about $X$.

**Definition 6** Let $\mathcal{D}$ be an input distribution for communication problem $f(x, y)$. Then, a one-way $\theta$-protocol is defined to contain

1. $M_{x,y}$, a distribution over all messages Alice can send to Bob.
2. $O(m, y)$, an output function dependent only on $m$ and $y$

such that when $(X, Y) \sim \mathcal{D}$ and $M \sim M_{x,y}$ then $I(M; Y|X) \leq \theta$

Notice that the one-way $\theta$-protocol only differs from the standard one-way protocol by allowing Alice's message $M$ to depend on Bob's input $y$. In particular, $Y$ can reveal at most $\theta$ more bits of information regarding $M$ given $X$. It is then obvious that a 0-protocol is equivalent to the standard one way protocol since $I(M; Y \mid X) = 0 \iff M$ depends only on $x$.

As it turns out, $n$-fold protocols yield efficient $\theta$ protocols. This is proven by the following variant of the direct product lemma:

**Lemma 6** *Let $\mathcal{D}^n$ be a distribution over inputs $(X^{(n)}, Y^{(n)}) = (X^1, ..., X^n, Y^1, ..., Y^n)$ to an $n$-fold problem $f^n$. If there is a one-way protocol $\tau$ for $f^n$ with communication cost $C$ and success probability $p$, then there is an input distribution $\mathcal{D}'$ and $O(\frac{1}{n}\log\frac{1}{p})$-protocol $\pi$ for the one fold problem $f$ such that*

1. *$\pi$ has success probability $1 - O\left(\sqrt{\frac{1}{n}\log\left(\frac{1}{p}\right)}\right)$*
2. *$\pi$ has internal information cost $I(X; M \mid Y) \leq O(C/n)$*
3. *$\mathcal{D}'$ and $\mathcal{D}$ are "close" in distribution*

Here, the notion of "closeness" between $\mathcal{D}$ and $\mathcal{D}'$ is quantified by the KL-divergence and is outside the scope of this survey. While the proof of Lemma 6 is beyond the reach of this survey, we offer very brief intuition as to why one might expect such a statement to be true. Naturally, we can decompose any $n$-fold problem into $n$ one-fold problems. By an averaging argument, at least one of these problems must have communication cost (or in this case, the analogous internal information cost) smaller than $C/n$. We encourage the motivated reader to consult [NY19] for detailed argumentation.

### 3.4 Existing Lower Bounds for $\text{UR}^{\subset}$

The reason we construct a one-way $\theta$-protocol from an $n$-fold protocol is to utilize known lower bounds on $\theta$-protocols – thereby providing a lower bound on the communication cost of the $n$-fold problem. In particular, the universal relation problem has the following known lower bound:

**Lemma 7** *There exists a distribution $\mathcal{D}_{UR}$ such that for any distribution $\mathcal{D}'$ "close" in distribution, one-way $\eta$-protocols for $UR^{\subset}$ over $\mathcal{D}'$ with error probability $\delta$ must have internal information cost $I(X; M|Y) \geq \Omega(\log\left(\frac{1}{\delta}\right)\log^2 n)$ for $\eta \leq O(\delta^2)$.*

Notice that by Lemma 7, an $n$-fold protocol $\pi^n$ with cost $C$ and error probability $\delta$ yields a distribution $\mathcal{D}'$ and one-way $O(\frac{\delta}{n})$-protocol with internal information cost $I(X; M|Y) \leq O(C/n)$. Naturally, an application of Lemma 8 then gives $C/n \geq \Omega(\log\left(\frac{n}{\delta}\right)\log^2 n)$ implying that $C \geq \Omega(n\log^2 n \log\left(\frac{n}{\delta}\right))$. Lemma 6 then implies that any dynamic graph streaming algorithm for spanning forest with error probability $\delta$ must use $\Omega(n\log^2 n\log\left(\frac{n}{\delta}\right))$ bits of memory. When $\delta$ is a constant, the AGM sketch achieves the $\Omega(n\log^3 n)$ lower bound.

## References


[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 459–467, 2012.

[CF14] Graham Cormode and Donatella Firmani. 2014. A unifying framework for l0-sampling algorithms. Distributed Parallel Databases 32, 3 (2014), 315–335

[JST11] H. Jowhari, M. Saglam, and G. Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In PODS, pages 49–58, 2011.

[NY19] Jelani Nelson and Huacheng Yu. Optimal lower bounds for distributed and streaming spanning forest computation. In Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019, pages 1844–1860, 2019.