1 Connectivity in Dynamic Graphs

Dynamic graph model: The input stream contains insertions and deletions of edges to $G$. Recall that we assume $V = [n]$.

The tool of choice is linear sketching, where decrements are supported by definition.

Motivations:

a) updates to the graph like removing hyperlinks or un-friending

b) the graph is distributed (each site contains a subset of the edges), and their linear sketches can be easily combined

Theorem [Ahn, Guha and McGregor, 2012]: There is a streaming algorithm with storage $\tilde{O}(n)$ that determines whp whether the graph is connected (In fact, it computes a spanning forest and can determine which pairs of vertices are connected.)

Idea: To grow (increase) connected components, we need to find an outgoing edge from each current component. Using $\ell_0$-sampling and especially its linear-sketch form, we can pick an outgoing edge from an arbitrary set.

Notation: Let $N = \binom{n}{2}$, and for each vertex $v$ define a vector $x^v \in \mathbb{R}^N$ that is 0 except at coordinates

$$x^v_{\{v,j\}} = \begin{cases} +1 & \text{if } (v,j) \in E \text{ and } v < j \\ -1 & \text{if } (v,j) \in E \text{ and } v > j \end{cases}$$

Algorithm AGM:

Update (on a stream/dynamic graph $G$):

For each vertex $v$, create a virtual stream for $x^v \in \mathbb{R}^N$ and maintain an $\ell_0$-sampler for this $x^v$ (using the same coins, as these are linear sketches that can be added).

*These notes summarize the material covered in class, usually skipping proofs, details, examples and so forth, and possibly adding some remarks, or pointers. The exercises are for self-practice and need not be handed in. In the interest of brevity, most references and credits were omitted.*
Repeat the above log \( n \) times independently (i.e., log \( n \) “levels” of samplers for each \( v \in V \)).

Output (to determine connectivity):

Initialize a partition \( \Pi = \{\{1\}, \ldots, \{n\}\} \) where each vertex is in a separate connected component.

Now repeat for \( l = 1, \ldots, \log n \):

1. For each connected component \( Q \in \Pi \), sum the samplers (more precisely, their sketches) for all \( v \in Q \) from level \( l \), to obtain a sampler for \( \sum_{v \in Q} x^v \). Then activate the sampler to pick a coordinate from \([N]\) (which we will see is a random outgoing from \( Q \)).

2. Use the \(|Q|\) sampled edges to merge connected components and update \( \Pi \)

Output “connected” if all the vertices are merged into one connected component.

**Analysis:** To simplify the analysis, we assume henceforth that \( G \) is connected (see below), and that the samplers are perfect (i.e. ignore their polynomially-small error probability).

**Exer:** Extend the analysis to the case that \( G \) is not connected, to determine whether \( s, t \in V \) given at query time, are connected.

**Claim 1:** If the number of connected components at the beginning of an iteration is \( k > 1 \) (and the samplers succeed in producing outgoing edges), then their number at the end of the iteration is at most \( k/2 \).

Exer: prove this claim.

**Claim 2:** Fix a set \( Q \subset V \). Then \( \sum_{v \in Q} x^v \) is nonzero only in coordinates \( \{i, j\} \) corresponding to an edge outgoing from \( Q \), i.e., \(|Q \cap \{i, j\}| = 1\).

**Proof:** Was seen in class.

**Storage:** The main storage is for \( \ell_0 \)-samplers for every vertex. Each one requires \( O(\log^3 n) \) bits, and we need fresh randomness in each of the \( O(\log n) \) iterations (levels), to avoid potential dependencies. Thus the total storage is \( O(n \log^4 n) \) bits.

## 2 Triangle Counting

**Goal:** Report the number of triangles, denoted by \( T \), in a graph \( G \) given as a stream of \( m \) edges on vertex set \( V = [n] \).

**Motivation:** The relative frequency of how often 2 friends of a person know each other is defined as

\[
F = \frac{3T}{\sum_{v \in V} \binom{\deg(v)}{2}}.
\]

We can compute \( \sum_{v \in V} \binom{\deg(v)}{2} \) exactly in \( O(n) \) space, by maintaining the degree of every vertex, and we can also approximate it using polylog\( (n) \) space using algorithms that estimate \( \ell_2 \)-norm.

Distinguishing \( T = 0 \) from \( T = 1 \) is known to require \( \Omega(m) \) space [Braverman, Ostrovsky, and Vilenchik, 2013].
We will henceforth assume a known lower bound $0 < t \leq T$.

**First Approach [Bar-Yossef, Kumar and Sivakumar, 2002]:**

Idea: use frequency moments.

Define vector $x \in \mathbb{R}^{\binom{n}{3}}$, where every coordinate $x_S$ counts the number of edges internal to a subset $S \subset V$ of 3 vertices.

Then $T = \# \{ S \subset V, |S| = 3 : x_S = 3 \}$.

**Lemma:** Let $F_p = \|x\|_p^p$ be the frequency moments for $p = 0, 1, 2$. Then

$$T = F_0 - 1.5F_1 + 0.5F_2.$$ 

Proof: As seen in class it suffices to verify that each coordinate $x_S$ contributes the same amount to both sides.

**Why such formula exists?:** We are looking for a polynomial $f(x_S) : \mathbb{R} \to \mathbb{R}$ with specific values $f(3) = 1$ and $f(2) = f(1) = f(0) = 0$. We can do polynomial interpolation. It would generally require degree 3, but $F_0 = 1_{\{x_S > 0\}}$ gives an extra degree of freedom.

**Algorithm 1:**

Update: Maintain the frequency moments $p = 0, 1, 2$ of vector $x \in \mathbb{R}^{\binom{n}{3}}$. Initially $x = 0$, and when an edge $(u, v)$ arrives, increment $x_S$ for every $S$ of the form $\{u, v, w\}$.

Output: Compute moment estimates $\hat{F}_p$ and report $\hat{T} = \hat{F}_0 - 1.5\hat{F}_1 + 0.5\hat{F}_2$.

**Correctness:** As was seen in class, suppose we compute frequency estimates $\hat{F}_p \in (1 \pm \gamma)F_p$. Then if we set suitable $\gamma = O\left(\frac{\epsilon t}{mn}\right)$, we would get additive error $\epsilon t \leq \epsilon T$.

**Storage:** The storage requirement is $O(\gamma^{-2} \log n) = O(\epsilon^{-2}(\frac{mn}{t}) \log n)$ words.