**CS 15-851: Algorithms for Big Data** Spring 2024

Lecture 11 - 04/04

*Prof. David Woodruff* *Scribe: Harrison Wiseblatt*

# 1 Adversarially Robust Streaming Algorithms

**Notation**: in the classical streaming algorithms model updates to a large vector $x \in \mathbb{R}^n$, at time $t = 1, 2, \cdots$, get update $(i_t, \Delta_t)$ causing change $x_{i_t} = x_{i_t} + \Delta_t$.

**Definition**: If we restrict $\Delta_t \geq 0$, the stream is **insertion-only**.

## 1.1 Constant probability of success at end of stream $\implies$ can get constant probability of success at all timesteps

If you have an algorithm that has constant probability of success at the end of a stream, you can get one that's correct throughout the stream, you can repeat it $\Theta(\log n)$ times in parallel at all time steps to get an algorithm that's correct with constant probability at all time steps in the stream (by ex., taking the median of the computed values).

## 1.2 $F_2$ estimation

Goal: output $\tilde{F_2}$ such that $(1-\varepsilon) \cdot F_2 \leq \tilde{F_2} \leq (1+\varepsilon) \cdot F_2$ with $F_2 = \sum_i x_i^2$.
We can use a random sketching matrix $S \in \{-\varepsilon, \varepsilon\}^{\frac{1}{\varepsilon^2} \times n}$ with 4-wise independent entries. With this, we have $E[|S \cdot x|_2^2] = |x|_2^2$ and $\text{Var}[|S \cdot x|_2^2] = O(\varepsilon^2 |x|_2^4)$ to get a $\varepsilon$ approximation with constant probability.

## 1.3 $F_0$ Estimation in insertion streams

$F_0$ estimation: Goal: output $\tilde{F_2}$ such that $(1-\varepsilon) \cdot F_0 \leq \tilde{F_0} \leq (1+\varepsilon) \cdot F_0$ with $F_0 = |\{i : x_i \neq 0\}|$. If we know that it's just an insertion stream (ie., all entries we see are unique), we can choose a hash function $h : [n] \to [M]$ with $M = O(n^2)$, with constant probability we get no collisions, we can maintain the smallest $t = \frac{100}{\varepsilon^2}$ hash values in the stream.

Intuition: if you see $F_0$ items, you expect that minimum hash value will be around roughly $\frac{1}{F_0}$, since there are no deletions you don't have to worry about updates. Then, your estimate by only keeping track of the min (call it $X$) is $\tilde{F_0} = \frac{1}{X}$.

The intuitive approach doesn't work to get a $\varepsilon$ approximation because of variance issues, but if you take $\frac{100}{\varepsilon^2}$ smallest items, if you instead take the $t$-th smallest hash value, you expect the smallest hash value is around $\frac{M}{F_0}$, so $v$ should be around $\frac{tM}{F_0}$.
One can use Chebyshev's inequality to show that this will work with constant probability, and uses $O(\frac{\log(n)}{\varepsilon^2})$ memory.

# 2 Tracking algorithms

These are the class of streaming algorithms where we care about the result across all time steps. Let's see what we can do without needing to do the process of repeating a "correct at the end" algorithm $\Theta(\log n)$ many times.

Let $x^{(t)}$ = stream vector after updates $1, 2, \cdots, t$.
Algorithm must output $R^{(t)} = (1 \pm \varepsilon) f(x^{(t)})$ at **all** timesteps $t$, not just at the end.

This can lead way to adversaries being able to mess with your input if they can track the outputs $R^{(t)}$ over time and change their inputs in response to them.

## 2.1 Motivation for adverserial streaming model

In classic streams, data is fixed before the algorithm starts (ie., future data does not depend on $R^{(t)}$). But, typically, your future data depends on your past decisions (ie., you may change your inputs to the streaming algorithm when you look at its output), so the future data depends on the current randomness. There are **no known guarantees** for problems in this case. This is because if your query can depend on the state of the algorithm, in an algorithm with sketching you could (for example) adverserially choose your input to be in the kernel of your sketching matrix. It's bad if your input can depend on your randomness.

## 2.2 Adversarial streaming model

An adversary can see $R^{(t)}$, then gets to choose $(x_{t_{i+1}}, \Delta_{t_{i+1}})$. The goal of the adversary is to make the algorithm fail to approximate a $\varepsilon$ approximation, and (unless stated otherwise), the adversary has unbounded computational power (with knowledge of the entire history of interactions). Note: this is the **black box model**, where we assume the adversary doesn't know the internal state of the algorithm and can only observe its outputs.

**Theorem (Alon, Matias, Szegedy '96)** the $F_2$ sketching matrix given in Section 1.2 above is not adversarially robust, even in insertion-only streams. Basically most non-deterministic sketches from this class will fall under that trap.

# 3 Generic transformations

We can transform any streaming algorithm $A$ into an adversarially robust algorithm $A'$: **sketch switching** and **computation paths**.
**Definition** for a function $f : \mathbb{R}^n \to \mathbb{R}$ define the $\varepsilon$-flip number $\lambda_\varepsilon(f)$ to be the maximum number of times $f(x^{(t)})$ can change by a factor of $(1+\varepsilon)$ after poly$(n)$ updates.

Ex., if you're estimating $F_2$ with $\Delta_t = 1 \forall t$, to estimate $f(X) = |x|_2^2$, we have

$$\lambda_\varepsilon \leq \log_{1+\varepsilon}(\text{poly}(n)) = O(\frac{\log n}{\varepsilon}).$$

## 3.1 Sketch switching

Keep $\lambda_\varepsilon(f)$ many independent sketches, only use the output of one sketch $S_i$ at a time, answer $R^(t)$ by the state of $S_i$ until it's no longer a $(1+\varepsilon)$ approximation, then information about $S_i$ was leaked so throw it away and start using the next one. This algorithm has the property that as soon as we reveal any information of the algorithm, we make it irrelevant, since the adversary only sees one output per sketch so can't learn anything about its internals.

Formally:
1) Create $O(\lambda_{\frac{\varepsilon}{12}}(f))$ independent sketches $S_1, \cdots, S_k$, each providing a $(1 \pm \frac{\varepsilon}{10}$-approximation. Set $i = 1$ and $R^{(0)} = 0$.
2) At time $t$, if Estimate $S_i) \notin (1 \pm \frac{\varepsilon}{3})R^{(t-1)}$, set $R^{(t)} =$ Estimate $(S_i)$ and throw out $S_i$, set $i = i+1$. Otherwise, output $R^{(t)} = R^{(t-1)}$, so we reveal no information about our algorithm (ie., we only output the result of $S_i$ the first time we use it and never update it until it's an insufficient approximation).

Intuition about correctness: we only need $S_{i+1}$ to be a good approximation for the stream the adversary generates to try to fool $S_i$, but either $S_i$ will either be correct forever, or it will end, but with constant probability $S_{i+1}$ will be correct for whatever stream the adversary generates (since it's independent of $S_{i+1}$).

**Formal proof**:

*Proof.* we can WLOG assume the adversary is deterministic. This is because if a randomized adversary can break your thing with probability $\geq \frac{2}{3}$, that $\frac{2}{3}$ probability is over your randomness and theirs, so by averaging there's some fixed choice of their random coins that still will break your thing with probability at least $\frac{2}{3}$.

This fixes the part of the stream the adversary gives $S^i$ after $S^{i-1}$ returned an answer out (that stream doesn't depend on $S^i$).
The stream doesn't depend on $S^i$ (but it could depend on the previous sketches), so $S^i$ is correct at all positions in the new stream.

$S^i$ outputs old value Out until Out' $\neq (1 \pm \frac{\varepsilon}{3}) \cdot$ Out (so just fix what the adversary does if it saw out forever).
It's a concern that you learn something about $S^i$ until it outputs Out', but $S^i$ is correct on whatever fixed stream you choose until $S^i$ outputs Out', so that doesn't matter.

Finally, if $S^i$ is a $(1 \pm \frac{\varepsilon}{10})$ approximation, then Out' $\neq (1 \pm \frac{\varepsilon}{3})$ Out, then $f$ has changed by a $1 \pm \frac{\varepsilon}{12}$ factor, so the number of sketches we need is bounded by $\lambda_{\frac{\varepsilon}{12}}(f)$, and if $f$ changes by a $(1 \pm \varepsilon)$ factor, then Out' $\notin (1 \pm \frac{\varepsilon}{3})$ Out, so we are always correct.

■

So, on $F_2$ estimation, this will give we have $O(\frac{\log^2 n}{\varepsilon^3})$ total memory used (because we have $O(\frac{\log n}{\varepsilon})$ many sketches, each of which uses $O(\frac{\log n}{\varepsilon^2})$ memory.