

# Lectures 4 and 5: Adversarially Robust Streaming Algorithms

DS-563 / CD-543 @ Boston University  
Instructor: Krzysztof Onak

Fall 2021

## 1 What we want to compute

**Input:** stream of at most  $m$  elements from  $[n] = \{1, \dots, n\}$ . For simplicity  $m = O(\text{poly}(n))$ .

**Want to approximate:** The number of distinct elements. Denote it  $\text{DE}(S)$  for stream  $S$ . More specifically, for some parameter  $\epsilon \in (0, 1)$ , we want to output an estimate  $F$  such that

$$(1 - \epsilon) \text{DE}(S) \leq F \leq (1 + \epsilon) \text{DE}(S).$$

**Deletions:** Sometimes deletions of items are allowed as well. In this setting, every stream item is of the form “insert  $x$ ” or “delete  $x$ .”

## 2 Adversarially robust streaming algorithms

**Model:** a game between two players, Algorithm and Adversary.

In each round:

- First, Adversary sends the next element of the stream to Algorithm.
- Second, Algorithm sends an updated estimate to Adversary.

**Outcome:** Adversary wins if *at least one* of the estimates send by Algorithm is not a  $(1 \pm \epsilon)$ -approximation to the current value.

**Resources:**

- Algorithm: as small space as possible (it’s a streaming algorithm after all!)
- Adversary: can base next updates on her own random coin tosses and previous estimates of Algorithm, does not know Algorithm’s personal coin tosses other than through the estimates that are made public

## Goal: turn non-robust algorithms into robust

**Note:** Non-robust algorithms are often referred to as *oblivious*.

### 3 Simple solution: use a different copy in each step

$\mathcal{A}$  = oblivious algorithm that provides  $(1 \pm \epsilon)$ -approximation w.p.  $1 - \delta/m$

**Solution:** Keep and update  $m$  independent copies of  $\mathcal{A}$ . In round  $i$ , return the estimate from the  $i$ -th copy.

**Problem:** multiplicative overhead over  $\mathcal{A}$  is  $\Theta(m)$ , so we could just store the entire input. . .

### 4 Technique 1: “Sketch/algorithm switching” (Insertion Only)

$\mathcal{A}$  = oblivious algorithm that provides  $(1 \pm \epsilon/20)$ -approximation w.p.  $1 - \delta/m^2$

---

**Algorithm 1:** Insertion-only  $\text{DE}(S)$  approximation

---

```
1 estimate  $\leftarrow$  0
2 index  $\leftarrow$  1
3  $t = O(\epsilon^{-1} \log m)$  independent copies  $\mathcal{A}_1, \dots, \mathcal{A}_t$  of  $\mathcal{A}$ 
4 foreach stream item  $x$  do
5     pass  $x$  to each  $\mathcal{A}_i$  and process it independently
6     if estimate from  $\mathcal{A}_{\text{index}} \geq (1 + \epsilon/2)\text{estimate}$  then
7         estimate  $\leftarrow$  estimate from  $\mathcal{A}_{\text{index}}$ 
8         index  $\leftarrow$  index + 1
9     output estimate
```

---

**Observation (space usage):** If all estimates of  $\mathcal{A}_i$ 's that we look at are good, then  $\sim \log_{1+\epsilon/2} m = O(\epsilon^{-1} \log m)$  copies of  $\mathcal{A}$  suffice because  $\text{DE}(S) \in \{0, \dots, m\}$ .

**Why:** The estimate we provide cannot increase too many times from 0 and then roughly 1 to roughly  $m$ , which bounds the maximum number of distinct elements in the stream.

**Another observation:** If all estimates of  $\mathcal{A}_i$ 's that we look at are good, then the algorithm provides a good approximation to  $\text{DE}(\dots)$  for all prefixes of the stream.

**Why:** If we are switching to a new  $\mathcal{A}_i$ , it clearly provides a good approximation. Otherwise, we know we cannot be much farther away from the correct answer than a factor of roughly  $(1 + \epsilon/2)(1 + \epsilon/20)$ , which is bounded by  $(1 + \epsilon)$  as long as  $\epsilon \in (0, 1)$ .

**Obstacle:** So it remains to prove that these estimates are “good” for the oblivious algorithms,  $\mathcal{A}_i$ 's, that we use, when we use them. In the simple solution (Section 3), we used the fact that nothing has been revealed to Adversary about a given  $\mathcal{A}_i$  until we used it for providing an estimate and then we would immediately throw this  $\mathcal{A}_i$  away and never use it again. Here, however, we make multiple queries to  $\mathcal{A}_{\text{index}}$  to track when we cross a given threshold. This does provide some additional information to Adversary, who knows that our estimate has not crossed the threshold and therefore, knows that some settings of internal coin tosses in  $\mathcal{A}_{\text{index}}$  are not possible.

**Getting around the obstacle:**

- Assume Adversary is deterministic. If she's not, by averaging, there must be a setting of her random coin tosses for which she manages to break our algorithm with at least the same probability.

- Prove by induction: Algorithms  $\mathcal{A}_i$ , for  $i \in \{1, \dots, k\}$ , when we query them, give all good estimates with probability at least  $1 - k\delta/m$ .
- The inductive step reasoning: Consider the moment when we increase *index* to  $k + 1$  and start using  $\mathcal{A}_{k+1}$  to track when we cross the new threshold. Note that while we use  $\mathcal{A}_{k+1}$ , we keep giving to Adversary a fixed estimate, which is stored in variable *estimate*. Since Adversary is deterministic, we can simulate the stream that Adversary would produce if it received the value stored in *estimate* as our estimate till the end of the stream.  $\mathcal{A}_{k+1}$  has to provide a good estimate for some prefix of this stream, until it provides an estimate that is at least  $(1 + \epsilon/2)\text{estimate}$ . Therefore, it suffices that  $\mathcal{A}_{k+1}$  provides good estimates throughout this fixed stream of updates. Since this stream is fixed, it is not a problem that  $\mathcal{A}_{k+1}$  is oblivious, and via the union bound, it achieves this goal with probability at least  $1 - m \cdot \delta/m^2 = 1 - \delta/m$ . By another application of the union bound and the inductive assumption,  $\mathcal{A}_i$ 's for  $i \in \{1, \dots, k+1\}$  provide good approximations when queried throughout our algorithm with probability at least  $1 - k\delta/m - \delta/m = 1 - (k+1)\delta/m$ .
- All internal estimates by  $\mathcal{A}_i$ 's and estimates sent back to Adversary are therefore within the allowed range with probability at least  $1 - m \cdot \delta/m = 1 - \delta$ .

**Space usage:** Since an oblivious algorithm for approximating DE with properties as listed above needs only  $O(\text{poly}(\frac{\log n}{\epsilon}))$  space, the total space is  $O(\text{poly}(\frac{\log n}{\epsilon})) \cdot O(\frac{\log m}{\epsilon}) = O(\text{poly}(\frac{\log n}{\epsilon}))$ .

## 5 Technique 2: Sparse–Dense Trade-offs (Insertion/Deletion Streams)

**When are deletions problematic:** When the number of distinct elements can change significantly very often. Sample stream: “insert 5”, “delete 5”, “insert 5”, “delete 5”, ...

Technique 1 builds on the fact that the actual value cannot change significantly too often. We won't define it here formally, but this value is known as the *flip number*. Unfortunately, in the example above, the flip number is  $\Omega(m)$ , where  $m$  is the length of the stream.

**Observation:** Significant changes can only happen very often when the current number of distinct elements is small. If the number of distinct elements is  $(1 \pm \epsilon/3)T$ , then over the next  $\epsilon T/3$  updates,  $T$  will still be a  $(1 \pm \epsilon)$ -multiplicative approximation to the number of distinct elements.

**(Definitions) Sparsity of a vector  $v$ :** We say that a vector  $v$  is *k-sparse* if it has at most  $k$  non-zero coordinates. Analogously, we say that a vector  $v$  is *k-dense* if it has at least  $k$  non-zero coordinates.

**(Auxiliary Tool) Sparse recovery:**

For any  $k$ , there is a (linear sketching) streaming algorithm that uses  $k \text{ polylog}(n)$  space and can recover all  $k$ -sparse frequency vectors over a stream of  $m$  arbitrary deletions and insertions.

The algorithm provides the recovery guarantee for all vectors with probability at least  $1 - 1/n^3$ , where the probability is taken over the initial selection of internal coin tosses.

Note that this works for recovering  $k$ -sparse vectors even if in the meantime, the vector was arbitrarily dense.

**Solution:**

- Create two regimes: sparse and dense. Handle them differently.
- Sparse regime (for at most  $2\sqrt{m}$ -sparse vectors): Store the vector explicitly. A sparse representation uses  $O(\sqrt{m})$  words. In this case, we know the number of distinct elements exactly.
- Dense regime (at least  $\sqrt{m}$ -dense vectors): Run in parallel (from the very beginning of the algorithm, ignoring the sparse/dense regime distinction)  $3\sqrt{m}/\epsilon$  parallel copies of an oblivious algorithm that gives a  $(1 \pm \epsilon/3)$ -approximation with probability  $1 - \delta/m$ . In the dense regime, use a new unused copy every  $\frac{\epsilon}{3}\sqrt{m}$  updates to give an estimate, and immediately dispose this algorithm. Stick to this estimate for  $\frac{\epsilon}{3}\sqrt{m}$  updates.
- Switching between regimes:
  - From sparse to dense: When the vector becomes  $2\sqrt{m}$ -dense, which we track exactly, forget it and just switch to the dense regime.
  - From dense to sparse: If a new approximation of DE becomes lower than  $\frac{3}{2}\sqrt{m}$ , switch to sparse regime. We need to recover the frequency vector exactly in this case, which can be achieved using the sparse recovery tool.

**Space usage:**  $O(\sqrt{m}) \text{ poly}(\epsilon^{-1} \log n)$  words

- Exact vector in the sparse regime:  $O(\sqrt{m})$  words
- Sparse recovery (one instance needed):  $O(\sqrt{m} \text{ polylog}(n))$  words
- $O(\epsilon^{-1}\sqrt{m})$  copies of an oblivious DE streaming algorithm:  $O(\sqrt{m} \text{ poly}(\epsilon^{-1} \log n))$  words

**Best currently known:**  $O(m^{1/3} \cdot \text{poly}(\epsilon^{-1} \log n))$  words. (It's a great open question whether  $m^{\Omega(1)}$  is needed.)