25.12.2024 ***Adaptive data analysis***

Lecture 8: Additional tools and applications for DP-stability

Source: Lecture notes by Aaron Roth and Adam Smith

Lecturer: Uri Stemmer

DP-Stability Amplification

Now we will see a method that allows us to enhance the stability guarantee of a given algorithm. Suppose we have an algorithm $\mathcal{A}$ that operates on databases over a domain X and guarantees (1,δ)-DP-stability. We define the following algorithm:

Algorithm $\mathcal{B}$

Input: Database $S \in X^n$

(1) Construct a database $T \subseteq S$ by including each element $x_i \in S$ independently with probability ε

(2) *Return* $\mathcal{A}(T)$

Theorem: If algorithm $\mathcal{A}$ satisfies (1,δ)-DP-stability, then algorithm $\mathcal{B}$ satisfies (ε,εδ)-DP stability.

Proof:

Fix an event F and a pair of neighboring databases S and $S' = S \cup \{x'\}$. Consider the execution of $\mathcal{B}$ *on* S' and note that if $x' \notin T$ then the output is distributed exactly as the output of $\mathcal{B}$ *on* S. On the other hand*, if* $x' \in T$ the output is distributed "similarly" in the sense of stability, with parameters $(1, \delta)$.

That is,

$$\Pr[\,\mathcal{B}(S') \in F \mid x' \notin T\,] = \Pr[\,\mathcal{B}(S) \in F\,]$$

And,

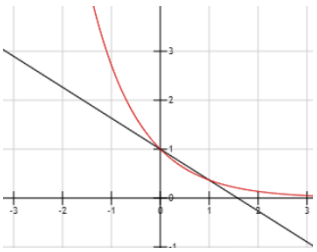
$$e^{-1} \cdot \Pr[\,\mathcal{B}(S) \in F\,] - \delta/e \;\leq\; \Pr[\,\mathcal{B}(S') \in F \mid x' \in T\,] \;\leq\; e \cdot \Pr[\,\mathcal{B}(S) \in F\,] + \delta$$

Therefore,

$$\begin{aligned}\Pr[\,\mathcal{B}(S') \in F\,] &= (1-\varepsilon) \cdot \Pr[\,\mathcal{B}(S') \in F \mid x' \notin T\,] + \varepsilon \cdot \Pr[\,\mathcal{B}(S') \in F \mid x' \in T\,] \\ &\leq (1-\varepsilon) \cdot \Pr[\,\mathcal{B}(S) \in F\,] + \varepsilon \cdot e \cdot \Pr[\,\mathcal{B}(S) \in F\,] + \varepsilon \cdot \delta \\ &= \big(1 + \varepsilon(e-1)\big) \cdot \Pr[\,\mathcal{B}(S) \in F\,] + \varepsilon \cdot \delta \\ &\leq e^{2\varepsilon} \cdot \Pr[\,\mathcal{B}(S) \in F\,] + \varepsilon \cdot \delta\end{aligned}$$

The reverse direction holds similarly:

$$\begin{aligned}\Pr[\,\mathcal{B}(S') \in F\,] &= (1-\varepsilon) \cdot \Pr[\,\mathcal{B}(S') \in F \mid x' \notin T\,] + \varepsilon \cdot \Pr[\,\mathcal{B}(S') \in F \mid x' \in T\,] \\ &\geq (1-\varepsilon) \cdot \Pr[\,\mathcal{B}(S) \in F\,] + \varepsilon \cdot e^{-1} \cdot \Pr[\,\mathcal{B}(S) \in F\,] - \varepsilon \cdot \delta/e \\ &= \big(1 - \varepsilon(1 - e^{-1})\big) \cdot \Pr[\,\mathcal{B}(S) \in F\,] - \varepsilon \cdot \delta/e \\ &\geq e^{-\varepsilon} \cdot \Pr[\,\mathcal{B}(S) \in F\,] - \varepsilon \cdot \delta/e\end{aligned}$$



*(*where the last inequality holds for all $0 \leq \varepsilon \leq 1$*)*

q.e.d.

A DP-stable variant of AboveThreshold

Recall algorithm AboveThreshold which we studied in the context of transcript compression. It turns out that this algorithm also has a DP-stable version:

***Algorithm* DPAboveThreshold**$(\boldsymbol{S \in X^*, t, \varepsilon, \{f_i\}})$
Let $\hat{t} \leftarrow t + \text{Lap}\left(\frac{2}{\varepsilon}\right)$ *For* $i = 1,2,3, \ldots$ *(until the algorithm halts)* *Obtain the next statistical query* $f_i: X \to [0,1]$ *Denote* $\hat{f}_i = f_i(S) + v_i = \sum_{x \in S} f_i(S) + v_i$ *where* $v_i \leftarrow \text{Lap}\left(\frac{4}{\varepsilon}\right)$ *If* $\hat{f}_i \geq \hat{t}$ *then output* $a_i = \top$ *and halt* *Else output* $a_i = \perp$ *and continue*

Theorem: Algorithm DPAboveThreshold satisfies (ε,0)-DP-stability (no matter how many queries it received before stopping).

Proof:
Fix two neighboring databases S, S' and fix a sequence of queries $f_1, f_2, \ldots$
Denote $A(S) = \text{DPAboveThreshold}(S, t, \varepsilon, \{f_i\})$, and note that the algorithm's output always takes the form:

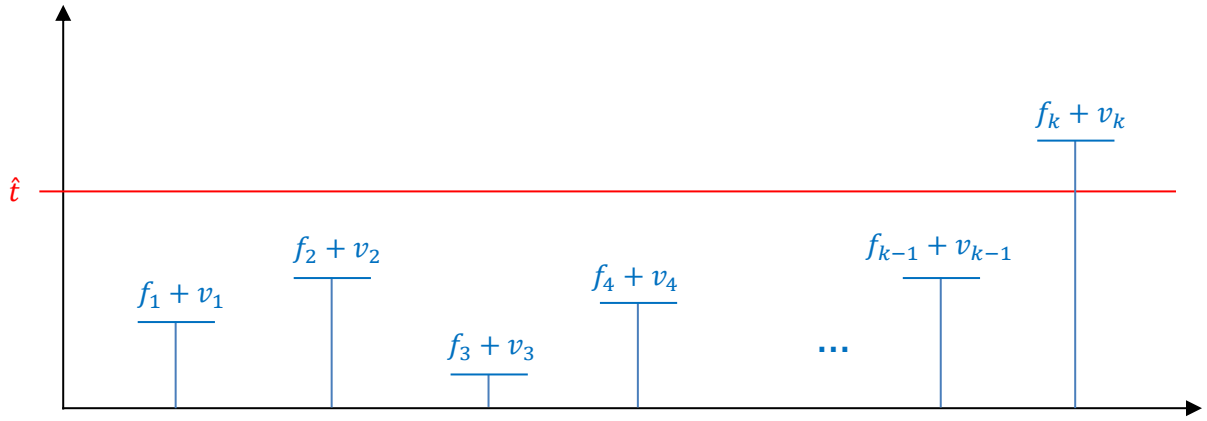
$$a_1 = \perp,\ a_2 = \perp,\ a_3 = \perp,\ \ldots,\ a_{k-1} = \perp,\ a_k = \top$$

(The only question is what is the length of the sequence, i.e., what is k).

Fix a possible output $\vec{a} = (a_1 = \perp, \ldots, a_{k-1} = \perp, a_k = \top)$.
We need to show that

$$\Pr[A(S) = \vec{a}] \leq e^{\varepsilon} \cdot \Pr[A(S') = \vec{a}]$$

Intuition:



Note: It suffices to show that for any fixture of $v_1, v_2, \dots, v_{k-1}$ it holds that

$$\Pr_{\hat{t},v_k}[A(S) = \vec{a}] \leq e^{\varepsilon} \cdot \Pr_{\hat{t},v_k}[A(S') = \vec{a}]$$

Because then

$$\Pr[A(S) = \vec{a}] = \sum_{v_1,\dots,v_{k-1}} \Pr[A(S) = \vec{a} | v_1, \dots, v_k] \cdot \Pr[v_1, \dots, v_k] \leq \cdots$$

So let us fix $v_1, v_2, \dots, v_{k-1}$ and compute

$$\Pr_{\hat{t},v_k}[A(S) = \vec{a}] = \Pr_{\hat{t},v_k}\left[\hat{t} > \max_{i<k}\left\{f_i(S) + v_i\right\} \text{ AND } f_k(S) + v_k \geq \hat{t}\right] = ((1))$$

Recall that $v_k \sim \text{Lap}\left(\frac{4}{\varepsilon}\right)$ and so by the properties of the Laplace distribution we have that

$$((1)) \leq e^{\varepsilon/2} \cdot \Pr_{\hat{t},v_k}\left[\hat{t} > \max_{i<k}\left\{f_i(S) + v_i\right\} \text{ AND } f_k(S) + v_k - 2 \geq \hat{t}\right]$$

$$= e^{\varepsilon/2} \cdot \Pr_{\hat{t},v_k}\left[\hat{t} \in \left(\max_{i<k}\left\{f_i(S) + v_i\right\}, f_k(S) + v_k - 2\right)\right] = ((2))$$

Recall that $\hat{t} \leftarrow t + \text{Lap}\left(\frac{2}{\varepsilon}\right)$ and so

$$((2)) \leq e^{\varepsilon/2} \cdot e^{\varepsilon/2} \cdot \Pr_{\hat{t},v_k}\left[\hat{t} \in \left(\max_{i<k}\left\{f_i(S) + 1 + v_i\right\}, f_k(S) + v_k - 1\right)\right]$$

$$\leq e^{\varepsilon} \cdot \Pr_{\hat{t},v_k}\left[\hat{t} \in \left(\max_{i<k}\left\{f_i(S') + v_i\right\}, f_k(S') + v_k\right)\right] = e^{\varepsilon} \cdot \Pr_{\hat{t},v_k}[A(S') = \vec{a}]$$

Where the last inequality follows from the fact that $|f_i(X) - f_i(X')| \leq 1$ and thus the final segment contains the previous segment, leading to a higher probability of entering it.

Accuracy Analysis of DPAboveThreshold:

Assume at most k queries are made: $f_1, f_2, \dots, f_k$.

During execution, we sample at most $k + 1$ Laplace noises with parameters $\frac{4}{\varepsilon}$ or $\frac{2}{\varepsilon}$. As we learned, for each of these noises, the following holds:

$$\Pr\left[\left|\text{Lap}\left(\frac{4}{\varepsilon}\right)\right| > \frac{4}{\varepsilon} \ln\left(\frac{k+1}{\beta}\right)\right] \leq \frac{\beta}{k+1}$$

Thus, with probability at least $1-\beta$ <u>all</u> noises are at most $\frac{4}{\varepsilon}\ln\left(\frac{k+1}{\beta}\right)$ in absolute value.

In this case:

- If the algorithm outputs $\perp$ in step i then $f_i(S) < t + \frac{8}{\varepsilon}\ln\left(\frac{k+1}{\beta}\right)$.
- If the algorithm outputs $\top$ in step i then $f_i(S) \geq t - \frac{8}{\varepsilon}\ln\left(\frac{k+1}{\beta}\right)$.

An extension of DPAboveThreshold

Algorithm ThresholdMonitor

Input: *Database* $S \in X^n$, *threshold* t, *parameter* k, *and a stream of queries* $f_i: X \to [0,1]$

1. *In each round* i *when receiving a query* f_i *do the following:*
   a) *Let* $\widehat{f_i} \leftarrow f_i(S) + w_i = \sum_{x \in S} f_i(S) + w_i$ *where* $w_i \sim \mathbf{Lap}(b)$ *for* $b = O\left(\frac{\sqrt{k}}{\varepsilon}\log\frac{k}{\delta}\right)$
   b) *If* $\widehat{f_i} < t$ *then output* $a_i = \perp$
   c) *Otherwise, output* $a_i = \top$ *and delete from* S *every* x *such that*

$$\sum_{\ell \in [i]:\, a_\ell = \top} f_\ell(x) \geq k$$

<u>Theorem:</u> *Algorithm* ***ThresholdMonitor*** *is* (ε, δ)*-DP-stable.*

We will not see the proof of this theorem, but we will try to cover some of its underlying intuition. For simplicity we will assume that k=1, that the queries are binary, and allow ourselves to add noises with a somewhat larger magnitude.

Intuition: Step 1 – Noiseless threshold

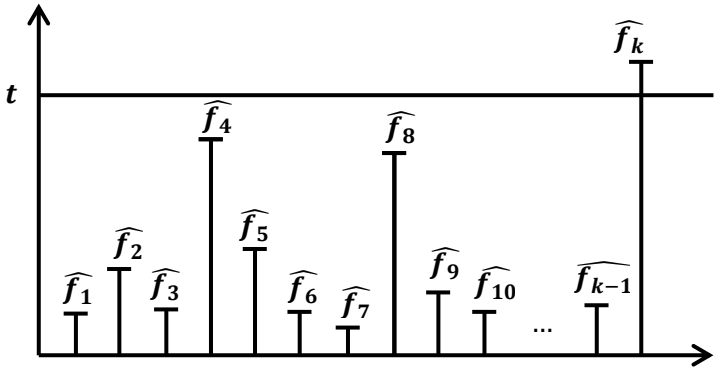
Algorithm AboveNoiselessThreshold

Input: *Database* $S \in X^n$, *threshold* t, *and a stream of queries* $f_i: X \to \{0,1\}$

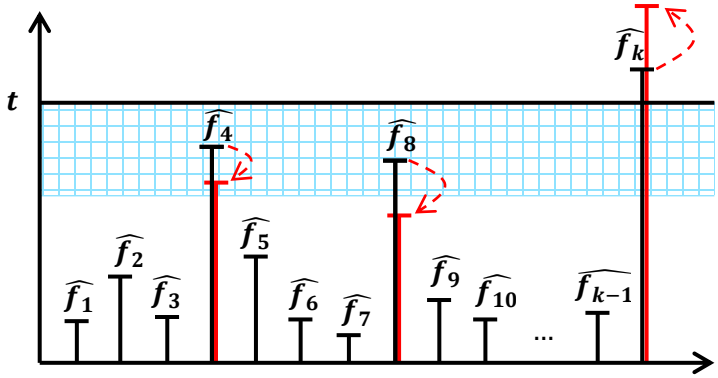
2. *In each round* i *when receiving a query* f_i *do the following:*
   a) *Let* $\widehat{f_i} \leftarrow f_i(S) + w_i$ *where* $w_i \sim \mathbf{Lap}(b)$ *for* $b = O\left(\frac{1}{\varepsilon}\log\frac{1}{\delta}\right)$
   b) *If* $\widehat{f_i} < t$ *then output* $a_i = \perp$
   c) *Otherwise, output* $a_i = \top$ *and* ***halt***

<u>Thorem:</u> Algorithm **AboveNoiselessThreshold** is DP-stable

Wrong Proof: Fix neighboring databases S, S', fix $f_1, f_2, \ldots$, and fix an output vector $\vec{a} = (a_1, \ldots, a_{k-1}, a_k) = (\perp, \ldots, \perp, \top)$. As in the analysis of **DPAboveThreshold**, we get $\vec{a}$ during an execution on S if



- **Consider a "small" strip below the threshold**
- Define Event E: at most $\log\frac{1}{\delta}$ of the $\widehat{f_i}$ fall inside the strip
- $\Pr[E] \geq 1 - \delta$. Intuitively, similar to a geometric RV
- Adjust noises only for times i in strip (and for $i = k$)
- Intuitively, times below the strip result in $\perp$ w.o.p. anyways



There are two issues here:

1. Analysis or $\Pr[E]$ assumes that the strip is "small" and therefore the probability of being in the strip is not much larger than the probability of being above it. But if f_i is just below the strip, then $\Pr[\text{below strip}] \geq 1/2$ and also $\Pr[a_i = \top] \gtrsim 1/2$. So it is not really true that "times below the strip result in $\perp$ w.o.p. anyways".

2. In order to "adjust noises in strip" we need to know which time steps are in the strip. Conditioning on that changes the distribution of the noises, which complicates the analysis.

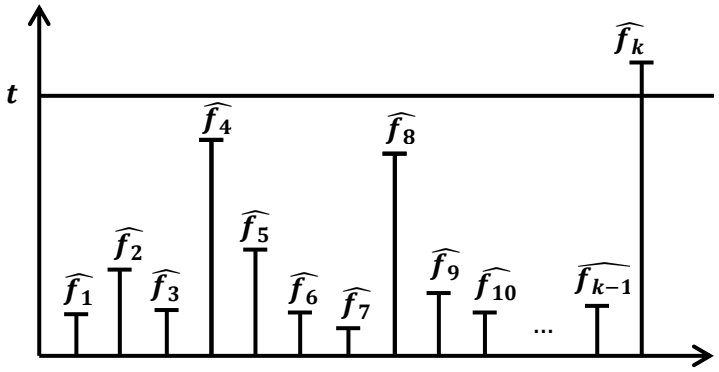
Resolution: Add two independent noises to each query

- Allow us to define E based on one of the noises, and argue about privacy using the second. This overcomes problem 2
- It also overcomes problem 1 because these noises will be on different scales

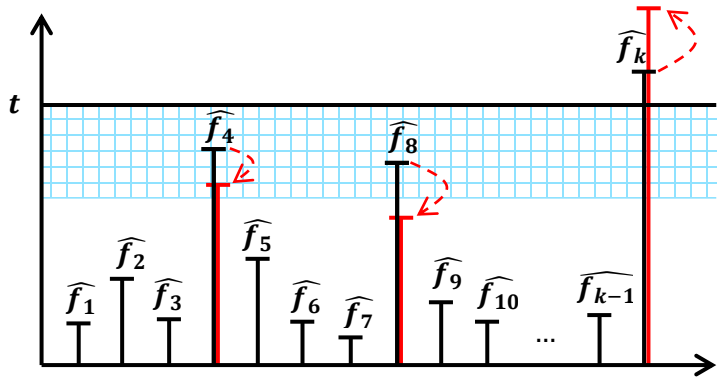
Algorithm AboveNoiselessThreshold (take 2)
Input: *Database* $S \in X^n$*, threshold* t*, and a stream of queries* $f_i: X \to \{0,1\}$ 3. *In each round* i *when receiving a query* f_i *do the following:* a) *Let* $\hat{f}_i \leftarrow f_i(S) + w_i + v_i$ *where* $w_i \sim \mathbf{Lap}\left(\frac{10}{\varepsilon}\log^2\frac{1}{\delta}\right)$ *and* $v_i \sim \mathbf{Lap}\left(\frac{1}{\varepsilon}\log\frac{1}{\delta}\right)$ b) *If* $\hat{f}_i < t$ *then output* $a_i = \perp$ c) *Otherwise, output* $a_i = \top$ *and* ***halt***

Thorem: Algorithm **AboveNoiselessThreshold** is DP-stable

Proof idea: Fix neighboring S, S', *fix* $f_1, f_2, \ldots$, and fix an output vector $\vec{a} = (a_1, \ldots, a_{k-1}, a_k) = (\perp, \ldots, \perp, \top)$. As before, we get $\vec{a}$ during an execution on S if



- **Consider a strip of size $\frac{1}{\varepsilon}\log^2\frac{1}{\delta}$ below the threshold**
- Denote $\bar{\bar{f}}_i = f_i(S) + w_i$
- Define Event E: at most $\log\frac{1}{\delta}$ of the $\bar{\bar{f}}_i$ fall inside (or above) the strip
- $\Pr[E] \geq 1 - \delta$. The probability of in strip isn't much larger than the probability of above strip, where we get $\top$ w.h.p. over v_i
- Adjust noises v_i only for times i in which $\bar{\bar{f}}_i$ fall inside or above the strip
- Times below the strip result in $\perp$ w.o.p. over sampling v_i



Intuition: Step 2 – Do not halt

Algorithm ThresholdMonitor (simplified)
Input: *Database* $\boldsymbol{S} \in \boldsymbol{X^n}$*, threshold* $\boldsymbol{t}$*, and a stream of queries* $\boldsymbol{f_i}: \boldsymbol{X} \to \{\boldsymbol{0}, \boldsymbol{1}\}$ *4. In each round* $\boldsymbol{i}$ *when receiving a query* $\boldsymbol{f_i}$ *do the following:* *a) Let* $\widehat{\boldsymbol{f_i}} \leftarrow \boldsymbol{f_i}(\boldsymbol{S}) + \boldsymbol{w_i} + \boldsymbol{v_i}$ *where* $\boldsymbol{w_i} \sim \mathbf{Lap}\left(\frac{\boldsymbol{10}}{\boldsymbol{\varepsilon}}\log^2\frac{\boldsymbol{1}}{\boldsymbol{\delta}}\right)$ *and* $\boldsymbol{v_i} \sim \mathbf{Lap}\left(\frac{\boldsymbol{1}}{\boldsymbol{\varepsilon}}\log\frac{\boldsymbol{1}}{\boldsymbol{\delta}}\right)$ *b) If* $\widehat{\boldsymbol{f_i}} < \boldsymbol{t}$ *then output* $\boldsymbol{a_i} = \perp$ *c) Otherwise, output* $\boldsymbol{a_i} = \top$ *and delete from* $\boldsymbol{S}$ *every* $\boldsymbol{x}$ *such that* $\boldsymbol{f_i}(\boldsymbol{x}) = \boldsymbol{1}$

Thm: Algorithm **ThresholdMonitor** is DP-stable

Proof intuition: Fix $\boldsymbol{S}$ and $\boldsymbol{S'} = \boldsymbol{S} \cup \{\boldsymbol{x'}\}$

- Once we reach an iteration $\boldsymbol{i^*}$ in which $\boldsymbol{a_{i^*}} = \top$ and $\boldsymbol{f_{i^*}}(\boldsymbol{x'}) = \boldsymbol{1}$ then $\boldsymbol{x'}$ is deleted from the data
- After $\boldsymbol{i^*}$ the two executions are identical
- Time steps $\boldsymbol{i} \leq \boldsymbol{i^*}$ in which $\boldsymbol{f_i}(\boldsymbol{x'}) = \boldsymbol{0}$ are also not very interesting, since $\boldsymbol{f_i}(\boldsymbol{S}) = \boldsymbol{f_i}(\boldsymbol{S'})$
- So, we only need to argue about time steps $\boldsymbol{i} \leq \boldsymbol{i^*}$ such that $\boldsymbol{f_i}(\boldsymbol{x'}) = \boldsymbol{1}$
- In these time steps, the output looks like $(\perp, \perp, \ldots, \perp, \top)$ and the previous analysis holds

Runtime of Algorithms in the Adaptive Model

We consider a situation similar to streaming, but the focus is on runtime rather than memory

Assume we have an algorithm $\mathcal{A}$ (in the non-adaptive model) with the following properties:

- Let X be some domain (the inputs to our algorithm will be elements from the domain X), and let $f: X \to \mathbb{R}$ be some function we want to compute/approximate on our inputs.
  - For example, perhaps X is the set of all graphs, and given a graph $G \in X$, the function $f(G)$ returns the size of the global min-cut in the graph G.

- At the start of its execution, the algorithm $\mathcal{A}$ receives the initial input $x_0 \in X$. We denote the runtime of $\mathcal{A}$ on the initial input x_0 as t_{init} (this is the initialization runtime).
  - For example, a graph.

- At time i, we receive an update to our input $u_i: X \to X$. We can think of u_i as a function mapping inputs to updated inputs, i.e., $x_i \leftarrow u_i(x_{i-1})$. In other words, at each step i, our current input is updated. We denote the runtime of the algorithm for each update as t_{update}.
  - For example, at each time step, one edge is added/removed from the graph.

- Occasionally, during the execution, we are asked to evaluate $f(x_i)$, where x_i is the current input. These are the times when the algorithm is "queried". We denote the runtime of the algorithm for each query as t_{query}.
  - For example, when queried, we must return an approximation of the size of the global min-cut in the current graph.

- The algorithm operates in the non-adaptive model. That is, for any pre-determined sequence of updates, whp, $\mathcal{A}$ outputs accurate results at every point in time when queried.

How can we modify the given (oblivious) algorithm $\mathcal{A}$ it to ensure correctness in the adaptive model? For simplicity, assume we must provide an answer after every update.

Algorithm $\mathcal{B}$

Parameters: Let $\boldsymbol{T}$ be a parameter and denote $\boldsymbol{k} \approx \sqrt{\boldsymbol{T}}$. Let $\boldsymbol{\mathcal{A}}$ be an oblivious algorithm

1. Obtain the initial input $\boldsymbol{x_0}$
2. Initiate $\boldsymbol{k}$ independent instances $\boldsymbol{\mathcal{A}_1, \dots, \mathcal{A}_k}$ of the oblivious algorithm $\boldsymbol{\mathcal{A}}$ on the initial input $\boldsymbol{x_0}$
3. For $\boldsymbol{i = 1, 2, \dots, T}$:
   a) Obtain an update $\boldsymbol{u_i}$
   b) Feed the update $\boldsymbol{u_i}$ to all of the copies of $\boldsymbol{\mathcal{A}}$
   c) Sample $\boldsymbol{s}$ of the copies of $\boldsymbol{\mathcal{A}}$ and query them to get intermediate outputs $\boldsymbol{y_{i,1}, \dots, y_{i,s}}$
      % $\boldsymbol{s}$ is the number of points we need in order to find an approximate median in a DP-stable way with parameter $\boldsymbol{\varepsilon_0 \approx \frac{1}{100}}$. For simplicity let's think about it as a constant
   d) Output $\boldsymbol{z_i = \text{DP\_Stable\_Median}(y_{i,1}, \dots, y_{i,k})}$ using stability parameter $\boldsymbol{\varepsilon_0 \approx 1/100}$
4. Reset all copies of $\boldsymbol{\mathcal{A}}$. That is, re-initialize each copy on the current input with fresh randomness, and goto Step 3

Analysis idea:

- We refer to each execution of Step 3 as a phase (consisting of $\boldsymbol{T}$ time steps)
- Fix a specific phase
- Let $\boldsymbol{R}$ denote the collection of random strings used by the copies of $\boldsymbol{\mathcal{A}}$ that are initiated before the beginning of the phase
- Similarly to what we had in the streaming setting, the sequence of all inputs obtained throughout the execution is DP-stable w.r.t. the collection of strings $\boldsymbol{R}$
  % Here we also make use of the amplification lemma we saw before…
- This implies correctness in the adaptive setting, similarly to the streaming setting

What runtime do we get?

The total time needed for a phase ($\boldsymbol{T}$ time steps) is

$$\approx \boldsymbol{k \cdot t_{\text{init}} + T \cdot k \cdot t_{\text{update}} + T \cdot s \cdot t_{\text{query}}}$$
$$\approx \boldsymbol{\sqrt{T} \cdot t_{\text{init}} + T^{1.5} \cdot t_{\text{update}} + T \cdot t_{\text{query}}}$$

And so the average time per step is

$$\approx \frac{\boldsymbol{t_{\text{init}}}}{\sqrt{\boldsymbol{T}}} + \sqrt{\boldsymbol{T}} \cdot \boldsymbol{t_{\text{update}}} + \boldsymbol{t_{\text{query}}}$$

Example of an Application:

Theorem (without proof): There exists an oblivious algorithm for approximating the size of the global min-cut in a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ with runtimes:

$$\boldsymbol{t_{\text{init}}} = \boldsymbol{O}(|\boldsymbol{E}|)$$
$$\boldsymbol{t_{\text{update}}} = \boldsymbol{O}\left(\sqrt{|\boldsymbol{V}|}\right)$$
$$\boldsymbol{t_{\text{query}}} = \boldsymbol{O}(\boldsymbol{1})$$

Conclusion: There exists an algorithm for this problem in the adaptive model with an average runtime per step of

$$\approx \frac{|\boldsymbol{E}| + \boldsymbol{T} \cdot \sqrt{|\boldsymbol{V}|}}{\sqrt{\boldsymbol{T}}}$$

For a choice of $\boldsymbol{T} \approx \frac{|\boldsymbol{E}|}{\sqrt{|\boldsymbol{V}|}}$ we obtain an average runtime per step of $\sqrt{|\boldsymbol{E}|} \cdot |\boldsymbol{V}|^{\boldsymbol{1/4}}$

Note: As of today, no better algorithm is known. This should be compared to the trivial solution, which resolves the problem from scratch at every step, achieving an average runtime of |E| per step...