13.06.2023 *ניתוח מידע אדפטיבי*

הרצאה 8: אלגוריתמים אדפטיביים – המשך

Source: Lecture notes by Aaron Roth and Adam Smith

מרצה: אורי שטמר

Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\widetilde{O}\left(\sqrt{m} \cdot \text{Space}(\mathcal{A})\right)$

Algorithm $\mathcal{B}$

Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \dots, r_k) \in (\{0,1\}^*)^k$

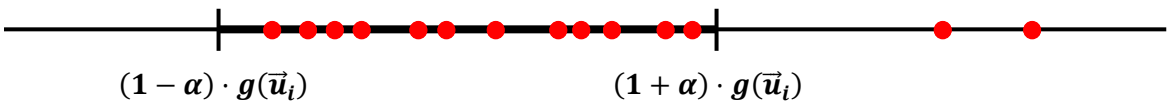
1. Initiate k independent instances $\mathcal{A}_1, \dots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \dots, r_k$
2. For $i = 1, 2, \dots, m$:
   a) Receive next update (u_i, Δ_i)
   b) Insert update (u_i, Δ_i) into each of $\mathcal{A}_1, \dots, \mathcal{A}_k$ and obtain answers $y_{i,1}, \dots, y_{i,k}$
   c) Output $z_i = \text{DP_Stable_Median}(y_{i,1}, \dots, y_{i,k})$ using stability parameter $\varepsilon_0 \approx 1/\sqrt{m}$

Analysis idea:

- $\mathcal{B}$ is DP-stable w.r.t. the collection of strings R
- Fix $i \in [m]$ and let $\vec{u}_i = ((u_1, \Delta_1), \dots, (u_i, \Delta_i))$ denote the first i updates
- Let $\mathcal{A}(r, \vec{u}_i)$ denote the output of $\mathcal{A}$ after the ith update when it is executed with randomness r on stream $\vec{u}_i$
- Consider the function $f_{\vec{u}_i}(r) = \mathbb{1}\{\mathcal{A}(r, \vec{u}_i) \in (1 \pm \alpha) \cdot g(\vec{u}_i)\}$
- Observe that $\vec{u}_i$ is the result of a DP-stable computation on R (post-processing $\mathcal{B}$'s answers), and hence, so is $f_{\vec{u}_i}$
- By the generalization properties of DP-stability we have

$$\frac{1}{k} \cdot \sum_{j=1}^{k} \mathbb{1}\{y_{i,j} \text{ is accurate}\} \approx \frac{1}{k} \cdot \sum_{j=1}^{k} f_{\vec{u}_i}(r_j) \approx \mathbb{E}_r\left[f_{\vec{u}_i}(r)\right] \approx 1$$

- So, most of the $y_{i,j}$'s are accurate, and hence, any approximate median is also accurate



דיון: מצד אחד, ניפוח זיכרון "רק" בפקטור $\sqrt{m}$ זה לא טריוויאלי, ובפרט זה ניפוח תת-לנארי (אז אם לאלג' שהתחלנו איתו יש זיכרון הרבה יותר קטן מ $\sqrt{m}$ נקבל אלג' אדפטיבי עם זיכרון תת לינארי ב- m). מצד שני, $\sqrt{m}$ זה המון ביחס לזיכרון שאנחנו בד"כ רגילים לראות בעולם הסטרימינג הלא-אדפטיבי. בד"כ שואפים לקבל זיכרון שנראה כמו $\text{polylog}(m)$. האם הזיכרון חייב לגדול בצורה פולינומית עם m?

אנחנו נראה כמה תשובות לשאלה הזאת. מצד אחד נראה שיש מקרים/בעיות ספציפיות עבורן ניתן לקבל תוצאות הרבה יותר טובות ממה שקיבלנו בעזרת הטרנספורמציה הנ"ל. מצד שני נראה שיש מקרים שבהם לאדפטיביות אכן יש מחיר גדול מבחינת הזיכרון הדרוש.

אלגוריתמים דטרמניסטיים

<u>בעיית ה *k-Heavy-hitters*:</u>
בהינתן (פרפיקס של) סטרים $S = (s_1, s_2, \dots, s_i)$, כאשר כל $s_\ell \in [n]$, יש להחזיר רשימה בגודל k המכילה את <u>כל</u> האיברים שמופיעים יותר מ- $\frac{i}{k}$ פעמים בסטרים, כלומר את כל האיברים $x \in [n]$ כך ש-

$$\text{weight}_i(x) \triangleq |\{\ell \in [i] : s_\ell = x\}| > \frac{i}{k}$$

(שימו לב, הרשימה יכולה להכיל גם איברים לא-כבדים, וזה בסדר, העיקר שהיא מכילה את כל הכבדים)

<u>משפט:</u> קיים אלג' סטרימינג דטרמניסטי לבעייה זו המשתמש בזיכרון $O\big(k \cdot (\log m + \log n)\big)$.

<u>הוכחה:</u> נתבונן באלג' הבא:

The Misra–Gries Algorithm
אתחול: יהי A מערך ריק. **צעד בשלב ה i (עבור $1 \le i \le m$):** 1. קבל את העדכון הנוכחי s_i 2. $A[s_i] \leftarrow A[s_i] + 1$ 3. אם $\lvert\{x \in [n] : A[x] > 0\}\rvert = k$ אזי: *a.* לכל x עבורו $A[x] > 0$ בצע: $A[s_i] \leftarrow A[s_i] - 1$ 4. החזר את רשימת האיברים $\{x \in [n] : A[x] > 0\}$

<u>ניתוח האלגוריתם – זיכרון:</u> נשים לב כי בכל שלב בריצה ישנם לכל היותר k כניסות במערך ששונות מאפס. מספיק לזכור את האינדקסים של הכניסות האלה (k מספרים בתחום [n]) ואת ערך המערך במקומות האלה (כל ערך הוא מספר שלם בין 1 ל- m). כלומר סה"כ זיכרון $O\big(k \cdot (\log m + \log n)\big)$.

<u>ניתוח האלגוריתם – נכונות:</u>
יהי i שלב כלשהו בריצה ויהי $x^* \in [n]$ איבר כבד כלשהו, כלומר $\text{weight}_i(x^*) > \frac{i}{k}$. נשים לב שכל פעם ש- x^* מופיע בסטרים אז הערך $A[x^*]$ גדל באחד. לכן, על מנת להראות ש- x מוחזר בצעד 4 בסוף השלב ה i, מספיק להראות שהערך של $A[x^*]$ קטן (באחד) לכל היותר $\frac{i}{k}$ פעמים לאורך הריצה. כדי לראות שזה נכון, נסתכל על הניסוח השקול הבא של האלגוריתם. בניסוח השקול נזכור לא רק את הערך של המונה, אלא גם את קבוצת האינדקסים שתרמו למונה הזה. (זה כמובן מצריך יותר זיכרון, אבל אנחנו משתמשים באלג' הזה רק לצורך ההוכחה).

ניסוח שקול של האלג' (זהו רק אלג' מחשבתי לצורך ההוכחה)
אתחול: יהי B מערך ריק **צעד בשלב ה i (עבור $1 \le i \le m$):** 1. קבל את העדכון הנוכחי s_i 2. $B[s_i] \leftarrow B[s_i] \cup \{i\}$ 3. אם $\lvert\{x \in [n] : B[x] \neq \emptyset\}\rvert = k$ אזי: *a.* לכל x עבורו $B[x] \neq \emptyset$ בצע: $B[s_i] \leftarrow B[s_i] \setminus \min(B[s_i])$ (כלומר אנחנו שוכחים את האינדקס של המופע הקדום ביותר של s_i) 4. החזר את רשימת האיברים $\{x \in [n] : B[x] \neq \emptyset\}$

האלג' הזה שקול לאלג' הקודם, כי בכל רגע בריצה, לכל x מתקיים $A[x] = |B[x]|$.

אנחנו רוצים להראות שהערך $A[x^*]$ קטן לכל היותר $\frac{i}{k}$ פעמים בריצה של האלג' הראשון, או לחילופין, שהקבוצה $B[x^*]$ מתכווצת לכל היותר $\frac{i}{k}$ פעמים בריצה של האלג' השני. כעת נשים לב כי בכל פעם שהקבוצה $B[x^*]$ מתכווצת, אז בדיוק k קבוצות מתכווצות כאשר האינדקסים המוסרים מ- k הקבוצות האלה הם אינדקסים ייחודיים (distinct) במובן הזה שאף אינדקס לא יכול להיות מוסר יותר מפעם אחת לאורך הריצה. זו אומר שאם הקבוצה $B[x^*]$ מתכווצת T פעמים לאורך הריצה, אזי ישנם Tk אינדקסים שונים שמוסרים בזמני הכיווץ האלה. מכיוון שהסטרים (עד כה) הוא באורך i, חייב להתקיים ש- $T \leq i/k$.
מ.ש.ל.

<u>מסקנה:</u> לבעיית ה k-Heavy-hitters קיים אלג' סטרימינג <u>במודל האדפטיבי</u> עם זיכרון $O\big(k \cdot (\log m + \log n)\big)$.

Sketch Switching

עכשיו נראה טרנספורמציה עם הבטחות אחרות תחת ההנחה שפונקציית המטרה לא יכולה "לקפוץ" יותר מדי פעמים לאורך הריצה. לדוגמה, נניח שאנחנו מעוניינים להעריך את מספר האיברים השונים מתוך סטרים באורך m ונניח שאין מחיקות בסטרים. זאת היא פונקציה מונוטונית עולה, שערכה חסום ע"י m. כמה פעמים לאורך הריצה הערך של פונקצייה זו יכול לקפוץ בפקטור כפלי של $(1+\alpha)$?

$$(1+\alpha)^t \leq m$$

$$t \cdot \underbrace{\log(1+\alpha)}_{\approx \alpha} \leq \log m$$

$$t \lesssim \frac{1}{\alpha} \cdot \log m$$

נניח שיש לנו חסם λ על מספר הפעמים בהם פונקציית המטרה יכול לקפוץ (לעלות או לרדת) בפקטור כפלי של $(1+\alpha)$ לאורך הריצה, ונניח ש- λ איננו גדול במיוחד. נוכל להשתמש בזה כדי לתכנן אלגוריתמים אדפטיביים בכל מני דרכים: נניח שיש לנו אלגוריתם $\boldsymbol{\mathcal{A}}$ שפועל בעולם הלא-אדפטיבי. ספציפית, בעולם הלא-אדפטיבי אלגוריתם $\boldsymbol{\mathcal{A}}$ מבטיח שבהסתברות לפחות $1-\beta$ כל התשובות שהוא מחזיר לאורך הריצה הן α-מדוייקות. נתבונן באלגוריתם הבא:

Algorithm Switch

1. Initiate $\boldsymbol{\lambda}$ independent instances of the oblivious algorithm $\boldsymbol{\mathcal{A}}$, denoted as $\boldsymbol{\mathcal{A}_1, \dots, \mathcal{A}_\lambda}$
2. Let $\boldsymbol{r = 1}$ and let $\mathbf{OUT} = \mathbf{0}$
3. For $\boldsymbol{i = 1, 2, \dots, m}$:
    a) Receive next update $(\boldsymbol{u_i, \Delta_i})$ and feed it to all copies of algorithm $\boldsymbol{\mathcal{A}}$
    b) Let $\boldsymbol{y_i}$ be the answer returned by $\boldsymbol{\mathcal{A}_r}$
    c) If $\mathbf{OUT} \notin \boldsymbol{(1 \pm \alpha) \cdot y_i}$ then set $\mathbf{OUT} \leftarrow \boldsymbol{y_i}$ and set $\boldsymbol{r \leftarrow r + 1}$ (if $\boldsymbol{r > \lambda}$ then fail)
    d) Output $\mathbf{OUT}$

כעת אנחנו רוצים להראות שכל עוד האלגוריתם לא נכשל, אז בה"ג התשובות שהוא מחזיר הן מדוייקות (גם במודל האדפטיבי).

לצורך הניתוח, קבע $\ell \in [\lambda]$ ונסתכל על האלגוריתם הבא:

Algorithm Switch-ℓ
1. Initiate $\boldsymbol{\lambda}$ independent instances of the oblivious algorithm $\boldsymbol{\mathcal{A}}$, denoted as $\boldsymbol{\mathcal{A}_1, \dots, \mathcal{A}_\lambda}$ 2. Let $\boldsymbol{r = 1}$ and let $\mathbf{OUT} = \mathbf{0}$ 3. For $\boldsymbol{i = 1, 2, \dots, m}$: a) Receive next update $(\boldsymbol{u_i, \Delta_i})$ and feed it to all copies of algorithm $\boldsymbol{\mathcal{A}}$ b) Let $\boldsymbol{y_i}$ be the answer returned by $\boldsymbol{\mathcal{A}_r}$ c) If $(\boldsymbol{r < \ell})$ and $\mathbf{OUT} \notin (\mathbf{1} \pm \boldsymbol{\alpha}) \cdot \boldsymbol{y_i}$ then set $\mathbf{OUT} \leftarrow \boldsymbol{y_i}$ and set $\boldsymbol{r \leftarrow r + 1}$ d) Output $\mathbf{OUT}$

<u>טענה:</u> בהסתברות לפחות $1 - \beta$ כל התשובות ש- $\boldsymbol{\mathcal{A}_\ell}$ מחזיר לאורך הריצה של ***Switch-ℓ*** הן מדוייקות-$\boldsymbol{\alpha}$

<u>הסבר:</u> בעצם באלגוריתם ***Switch-ℓ*** אנחנו לא משתמשים ב $\boldsymbol{\mathcal{A}_\ell}$ ולכן סדרת הקלטים היא בלתי תלויה באקראיות שלו. כלומר עותק זה פועל בעולם הלא-אדפטיבי...

<u>סימון:</u> עבור $\ell \in [\lambda]$ נסמן ב i_ℓ את האינדקס של האיטרציה שבה r מקבל את הערך $\ell + 1$ בפעם הראשונה.

- באלגוריתם ***Switch*** זהו הזמן שבו אנחנו חושפים לראשונה פלט שמתקבל מהעותק $\boldsymbol{\mathcal{A}_\ell}$. באלגוריתם ***Switch-$\ell$*** זהו הזמן שבו היינו רוצים לחשוף זאת, אך לא עשינו את זה בגלל התנאי האדום באלגוריתם.

<u>טענה:</u> נקבע יריב אדפטיבי, נניח בה"כ שהוא דטרמניסטי, ונקבע גם את האקראיות של העותקים של אלגוריתם $\boldsymbol{\mathcal{A}}$. אזי עד **(וכולל)** זמן i_ℓ סדרת הפלטים שתוחזר על ידי בריצה של ***Switch-ℓ*** זהה לסדרת הפלטים שתוחזר בריצה של ***Switch***

<u>הסבר:</u> השינויי שעשינו ב ***Switch-ℓ*** מתחיל להשפיע על הריצה רק <u>בפלט</u> שאנחנו מחזירים באיטרציה i_ℓ. כל הפרפיקס של הריצה לא משתנה. בפרט, הקלט שמתקבל באיטרציה זו לא משתנה. לכן גם הפלט הנוכחי של כל העותקים שאנחנו מתחזקים נשאר זהה.

<u>מסקנה:</u> בהסתברות לפחות $1 - \beta$ עד **(וכולל)** זמן i_ℓ סדרת הפלטים שתוחזר על ידי בריצה של ***Switch*** הם מדוייקים-$\boldsymbol{\alpha}$

אבל נשים לב שאלגוריתם ***Switch*** לא משתמש בעותק $\boldsymbol{\mathcal{A}_\ell}$ לאחר זמן i_ℓ

<u>מסקנה:</u> בהסתברות לפחות $1 - \lambda\beta$ כל פלטי הביניים שאלגוריתם ***Switch*** משתמש בהם הם מדוייקים-$\boldsymbol{\alpha}$

<u>הסבר:</u> נובע מחסם האיחוד על פני כל הבחירות ל- $\ell \in [\lambda]$

<u>מסקנה:</u> בהסתברות לפחות $1 - \lambda\beta$ אלגוריתם ***Switch*** הוא $O(\alpha)$-מדוייק גם במודל האדפטיבי

הפרדה בין מודל הסטרימינג האדפטיבי והלא-אדפטיבי

<u>הגדרה:</u> יהי X דומיין כלשהו ותהי $\vec{u} = \big((x_1, b_1), \dots, (x_\ell, b_\ell)\big) \in (X \times \{\pm 1\})^*$ סדרת עדכונים (או סטרים) כאשר כל עדכון מכיל איבר מ X ומשקל בינארי. עבור איבר $x \in X$ נסמן:

$$weight(\vec{u}, x) = \sum_{i:x_i=x} b_i \neq 0$$

ונסמן

$$count(\vec{u}) = |\{x : \ weight(\vec{u}, x) \neq 0\}|$$

כלומר $count(\vec{u})$ סופר את מספר האיברים (שכרגע) סך המשקלים שהם קיבלו בסטרים שונה מאפס.

<u>הגדרה:</u> בבעיית ה OneFromMany הקלט שלנו הוא סטרים מהצורה הנ"ל. אם $count(\vec{u}) < |X|/100$ אז כל פלט הוא תקין. אחרת, יש להחזיר איבר $x \in X$ כך ש- $weight(\vec{u}, x) \neq 0$

<u>טענה:</u> במודל הלא-אדפטיבי יש אלגוריתם לבעיית OneFromMany עם הסתברות כישלון קבועה (קטנה כרצוננו) וזיכרון $\text{polylog}(m, |X|)$

<u>הסבר:</u> בתחילת הריצה האלגוריתם דוגם תת קבוצה $A \subseteq X$ בגודל $\text{polylog}\, m$. לאורך הריצה האלגוריתם עוקב (במדוייק) אחרי המשקלים של כל האיברים ב A. בכל שלב האלגוריתם מחזיר איבר $a \in A$ עם משקל שונה מאפס אם יש כזה איבר, ואחרת מחזיר איבר שרירותי מ X.

כדי לראות שזה עובד, נשים לב כי לפי הגדרת הבעייה, האלגוריתם לא יכול לטעות בזמנים t בהם יש פחות מ $|X|/100$ איברים עם משקל שונה מאפס. עכשיו נקבע זמן t שבו יש יותר מ $|X|/100$ איברים עם משקל שונה מאפס. לפי חסם צ'רנוף, בה"ג, הקבוצה A מכילה לפחות איבר אחד כזה ואז האלגוריתם מצליח.

הטענה עכשיו נובעת מחסם האיחוד על פני כל נקודות הזמן. מ.ש.ל.