COMS E6998-15: Algorithms for Massive Data (Fall '23) Oct. 18, 2023

Lecture 13: Sparse Fourier Transform and Distribution Testing

Instructor: *Alex Andoni* Scribe: *Eliezer Zimble*

1 Introduction

We continue our discussion of Sparse Fourier Transforms, focusing on the case where $k > 1$.

Our setup has:
$a \in \mathbb{R}^n$ as the "signal".
$\hat{a} \in \mathbb{R}^n$ as the Fourier transform.

Assumption 1. *(noise free):*
We assume $||\hat{a}||_0 \leq k$. We will consider the case in which $\hat{a}$ has k non-zeros.

Goal: we want to recover $\hat{a}$ in time $\approx O\left(k \cdot log^{O(1)}(n)\right)$.

Assumption 2. *(random):*
We assume that the non-zero values of $\hat{a}$ are at random positions.

Main idea: Reduce our problem to the case of $k = 1$. We will achieve this by bucketing the non-zero values such that each bucket isolates a single value.

2 Bucketing algorithm

Theorem 3. *Aliasing: Fix $L \in [n]$ that divides n. For $a' \in \mathbb{R}^{\frac{n}{L}}$, where $a' = (a_0, a_L, a_{2L} \cdots)$ we have:*

$$\hat{a}'_u = \sum_{l=0}^{L-1} \hat{a}_{u+l \cdot \frac{n}{L}}$$

where $u \in [\frac{n}{L}]$.

Our algorithm requires us to choose buckets. We will use the aliasing theorem for our choice of buckets. Namely we define our buckets as:

$$\sum_{l=0}^{L-1} \hat{a}_{u+l \cdot \frac{n}{L}}$$

for $u = 0$ to $\frac{n}{L} - 1$.

Note 1: This is merely a reinterpretation of our input a: for a' the algorithm does not need to perform any calculations but rather only needs to consider certain coordinates of the given input.

Algorithm:
The algorithm proceeds as follows:

1. Set $\frac{n}{L} = 2k^2$ (see analysis ahead for derivation)

2. Define $a' = (a_0, a_L, a_{2L} \cdots)$, such that from the aliasing theorem we have:

$$\hat{a}'_u = \sum_{l=0}^{L-1} \hat{a}_{u+l\cdot\frac{n}{L}}$$

   Likewise, using phase shifting we define $a'' = (a_1, a_{L+1}, a_{2L+1} \cdots)$, such that from the aliasing theorem and phase shift theorem (from lecture 12) we have:

$$\hat{a}''_u = \sum_{l=0}^{L-1} \hat{a}_{u+l\cdot\frac{n}{L}} \cdot \omega_n^{u+l\cdot\frac{n}{L}}$$

   We will use $\hat{a}'_u$ and $\hat{a}''_u$ as bucket u. (with $\hat{a}'_u$ and $\hat{a}''_u$, we can recover the coefficients for the case of $k = 1$ with no error by using the algorithm from lecture 11)

3. Run standard FFT on a' and a'' to output $\hat{a}'$ and $\hat{a}''$.

4. For each bucket $u \in [\frac{n}{L} - 1]$, we use $\hat{a}'_u$ and $\hat{a}''_u$ to recover at most 1 non-zero coefficient of $\hat{a}$.

Implementation of step (4):

- If $\hat{a}_u = 0$, then output "there are no non-zero coordinates in bucket u".
- Otherwise, we have exactly one index v such that $\hat{a}_v \neq 0$ and the coefficient maps to bucket u. We can then recover v as:

$$v = log_\omega \left(\frac{\hat{a}''}{\hat{a}'} \right)$$

3 Proof of correctness

The Bucketing algorithm succeeds when all non-zero coordinates of $\hat{a}$ fall into different buckets. Namely, the algorithm succeeds when there are no collisions. We therefore want to set the number of buckets in order to achieve the desired success probability.

Note 2: Our choice of buckets is not random. We therefore rely on assumption 2 that the non-zero values of $\hat{a}$ are at random positions in order to achieve the effect of randomly tossing the non-zero values into

buckets.

Claim 4. *Setting $\frac{n}{L} = 2k^2$ gives success probability $\geq \frac{1}{2}$.*

Proof. We set $\frac{n}{L} = 2k^2$. Let $S = \{j_1, j_2, \cdots j_k\}$ be the non-zero coordinates of $\hat{a}$. We have:

$$\mathbb{E}[\#\text{ of collisions}] = \mathbb{E}\left[\sum_{\substack{p=1\cdots k \\ q=1\cdots k}} \mathbb{1}\left[\hat{a}_{j_p} \&\ \hat{a}_{j_q} \text{ collide in the same bucket}\right]\right]$$

$$\leq k^2 \cdot Pr[\hat{a}_{j_p} \&\ \hat{a}_{j_q} \text{ collide}]$$

$$\leq k^2 \cdot \frac{L}{n}$$

$$= k^2 \cdot \frac{1}{2k^2}$$

$$= \frac{1}{2}$$

By Markov's bound, we have that:

$$Pr[\text{The number of collisions is } \geq 1] \leq \frac{\mathbb{E}[\#\text{ of collisions}]}{1} = \frac{1}{2}$$

We therefore have that the algorithm succeeds with probability at least $\frac{1}{2}$.

□

4 Runtime analysis:

Claim 5. *The runtime of the bucketing algorithm is $O\left(k^2 log(k) + k^2\right)$.*

Proof. The runtime of the algorithm is from steps (3) and (4):

- Step (3) requires $O(k^2 log(k))$ to perform the FFT on a' and a'' (each of size $\frac{n}{L} = 2k^2$).
- Step (4) requires $O(k^2)$ to iterate through the buckets.

The total runtime is therefore $O\left(k^2 log(k) + k^2\right)$ □

Improved runtimes:
Our algorithm has a factor of k^2 instead of k because we want to guarantee no collisions and therefore require $O(k^2)$ buckets.

There are algorithms that limit the number of buckets to $O(k)$ (e.g. 10k) for which each non-zero value has a constant probability of not colliding. Namely, setting $\frac{n}{L} = O(k)$ achieves:

$$Pr\left[j_1 \text{ does not collide}\right] \geq .9$$

The algorithm would need to detect and filter out buckets in which a collision occurred (possibly requiring further coefficients $\hat{a}'''$). The buckets that survive allow us to recover a constant fraction of non-zeros.

The algorithm can be further improved by shifting the randomness from the location of the non-zero coordinates to the tossing into buckets. The algorithm is then repeated a number of times. Each iteration recovers a random fraction of the non-zeros which are then subtracted out before performing the next iteration. We can perform the subtraction by using the inverse Fourier transform on the non-zeros already found.

The best runtime is $O\left(k \cdot log(n) \cdot log(\frac{n}{k})\right)$ even in the presence of noise from [1].

See also Lecture 17 from [3] and [2] for further discussion.

5 Introduction to Sublinear time algorithms:

Given a string $y \in \mathbb{R}^n$ or a graph, we want to check a property or compute a parameter in time $<<$ than the input size.

We therefore cannot access the entire input, and therefore the algorithms will usually be approximations.

Sparse Fourier transform was a special case because what we are recovering was spread around the input, but in general the problems we consider will not have this property and therefore require some tolerance to error.

Examples of problems on a string include: checking whether a string of integers is monotone, longest increasing subsequence.
Examples of problems on a graphs include: coloring problems, number of connected components.

6 Distribution testing:

Our first subtopic will be distribution testing.
Given a distribution D on $[n]$. Goal: check a property of D using only samples drawn from D.

E.g.

- Is D=Uniform?
- Is D= a fixed distribution Q that we have?
- If D is Gaussian, how many samples do we need to determine the mean and variance?

Note 3: distribution testing differs from the other examples of sublinear time algorithms above. In the other examples, we can choose how to access (portions) of the input as we want. For distribution testing we are not given D and cannot choose which samples to be given. We have examples drawn from D (therefore if a particular example x_i has low probability it will probably not be sampled).

Problem 1: Uniformity testing:

Our first problem is testing whether D is Uniform. Namely, given m samples from distribution D, we want to decide whether D is Uniform or not. How many samples m do we need? (We want to minimize m and usually the runtime will be roughly m).

Without further constraints, we have that $m = \infty$: it is possible that we are unlucky and D is very close to uniform such that we cannot detect the differences.

We therefore need to introduce a tolerance parameter and redefine our problem statement.

Definition 6. *$\epsilon-$far for distributions: D and Q are $\epsilon-$far, $\epsilon > 0$ iff:*

$$||D - Q||_1 > \epsilon$$

Note 4: The definition of $\epsilon-$far from a property will change depending on the context and what results are feasible.

Problem 1′:

Given m samples from D, we want to determine whether D is :

- Uniform

  or

- $\epsilon-$far from Q which is uniform.

Definition 7. *Total variation distance: we define $||D - Q||_{TV}$ as:*

$$\max_{T \subseteq [n]} \left| \Pr_{i \in D}[i \in T] - \Pr_{i \in Q}[i \in T] \right|$$

Intuition: If we are given a single sample i and want to determine whether it is from D or Q our best probability of being able do so is $||D - Q||_{TV}$. then for a single sample

If $||D - Q||_{TV} \leq \epsilon$, then if given a sample either from D or from Q then the behavior will be the same except with ϵ probability.

Fact 8.

$$||D - Q||_{TV} = \frac{1}{2}||D - Q||_1$$

We therefore can justify our new problem statement. If $||D - Q||_1 \leq \epsilon$ such that D is "in the gray zone" between the options of being Uniform and being $\epsilon-$far (i.e. within ϵ-close), we then have:

$$||D - Q||_1 \leq \epsilon$$

such that:

$$||D - Q||_{TV} \leq \frac{\epsilon}{2}$$

such that even if D is "incorrectly" output as Uniform there would be no statistical difference except with $\frac{\epsilon}{2}$ probability.

References

[1] Haitham Hassanieh, Piotr Indyk, Dina Katabi, Eric Price. Nearly Optimal Sparse Fourier Transform *STOC*, 2012

[2] Badih Ghazi, Haitham Hassanieh, Piotr Indyk, Dina Katabi, Eric Price, Lixin Shi. Sample Optimal Average-Case Sparse Fourier Transform in Two Dimensions. *Allerton*, 2013.

[3] `https://www.sketchingbigdata.org/fall17/lec/`