Sketching Algorithms for Big Data Fall 2017

Lecture 16 — Oct. 26, 2017

Prof. Piotr Indyk *Scribe: Chi-Ning Chou*

1 Overview

In the last lecture we constructed sparse RIP_1 matrix via expander and showed that it provides sparse recovery. In this lecture we are going to study a new topic: *sparse Fourier transform* (SFT).

The goal of sparse Fourier transform is finding a fast way to **approximate** the Fourier coefficients of a **sparse** signal and the algorithm can be randomized. We will start with some backgrounds in *discrete Fourier transform* in Section 2 and then move on to *sparse Fourier transform* in Section 3. Historical development will be covered in Section 4.

2 Discrete Fourier transform

In this section, we are going to briefly review the *discrete Fourier transform (DFT)*. In the following, we assume the parameter n to be some power of 2, say $n = 2^\ell$. Also, note that most of the indexes in this lecture start from 0 instead of 1.

The input here is a length n sequence of discrete signal $a = (a_0, a_1, \dots, a_{n-1}) \in \mathbb{C}^n$ and the goal is to compute its *Fourier coefficients* $\hat{a} = (\hat{a}_0, \hat{a}_1, \dots, \hat{a}_{n-1})$ defined as follows. For any $u = 0, 1, \dots, n-1$,

$$\hat{a}_u = \frac{1}{n}\sum_{j=0}^{n-1} a_j e^{-\frac{2\pi i}{n}uj}, \tag{1}$$

where $i = \sqrt{-1}$. For convenience, we denote $\omega = \omega_n = e^{\frac{2\pi i}{n}}$ and (1) becomes $\hat{a}_u = \frac{1}{n}\sum_{j=0}^{n-1} a_j \omega^{-uj}$. When one remove the minus sign in the exponent of ω, we call it *inverse Fourier transform.*

Remark 1. *The scaling term in* (1) *has three different conventions:* $1/n$*,* $1/\sqrt{n}$*, and* 1*. Basically it does not matter since here we only care about the largest coefficient.*

Remark 2. *Intuitively, one can think of the set* $\{\omega^u : u = 0, 1, \dots, n-1\}$ *as an* ϵ*-net for the unit sphere on the complex space* $\mathbb{C}$*.*

For convenience, one can write the DFT and the inverse Fourier transform into **matrix form**. Namely, we have $\hat{a} = Fa$ and $a = F^{-1}\hat{a}$ for some matrices $F, F^{-1} \in \mathbb{C}^{n\times n}$ defined as follows. For any $u, j = 0, 1, \dots, n-1$,

$$F_{uj} = \frac{1}{n}\omega^{-uj} \text{ and } F^{-1}_{ju} = \omega^{uj}.$$

There are two important properties about the DFT: the *orthogonality* and the *time-shift/phase-shift theorem.*

Fact 3 (orthogonality)**.** *Both F and F^{-1} are orthogonal up to certain scaling. Specifically, for any $x \in \mathbb{C}^n$, we have*

$$\|Fx\|_2^2 = \frac{1}{n}\|x\|_2^2 \text{ and } \|F^{-1}x\| = n\|x\|_2^2.$$

Theorem 4 (time-shift/phase-shift)**.** *Let $a, \hat{a}$ be defined as above and $a', \hat{a}'$ be new signal and the corresponding Fourier coefficients. For any $b = 0, 1, \ldots, n-1$, we have*

$$\forall j = 0, 1, \ldots, n-1,\ a_j = a_j\omega^{bj} \Leftrightarrow \forall u = 0, 1, \ldots, n-1,\ \hat{a}'_u = \hat{a}_{u-b}.$$

With the above definition and properties of DFT, there are two natural questions as follows.

Question: What are the applications of DFT?

One of the main application of DFT is the filtering techniques in signal/image processing. For instance, given a large image, one first apply DFT and get its Fourier coefficients. Next, throw away the least 90% of Fourier coefficients and apply the inverse Fourier transform. The resulting image is almost the same as the original image with only some blurred effects. See Figure 1 for example.

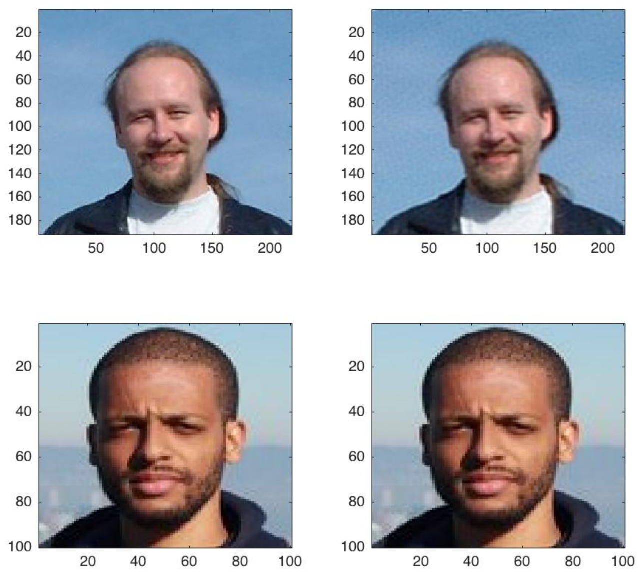

Figure 1: Applying DFT filtering on the photos of the two lecturers. The photos on the left are the original and the photos on the right are after filtering. One can see that the photos after filtering are a little blurred.

Question: How fast can we implement DFT?

This is the main focus for this course. At first glance, one can immediately see that a naive matrix-vector multiplication can give a $O(n^2)$ time algorithm for DFT. Surprisingly, the *fast Fourier transform (FFT)* algorithm achieves $O(n \log n)$ computing time, which is desired for practical usage.

In this lecture, we consider the case where the input signal is k-sparse in the sense that there are k dominant Fourier coefficients and the region $\log n < k \ll n$ is of interest. We call this *sparse Fourier transform.*

3 Sparse Fourier transform

In this section, we are going to study the sparse Fourier transform. We will setup the notations and goal in Section 3.1. In Section 3.2, we will first perform a sanity check on whether it is possible to achieve our goal. Finally, in Section 3.3 we will consider the case where the sparsity is 1 and give a fast algorithm.

3.1 Setup

Let $a \in \mathbb{C}^n$ be the input signal and $\hat{a}$ be its Fourier coefficients defined in (1). For any $k = 1, 2, \dots, n$, define $\hat{a}^{(k)}$ to be the vector that contains the largest[1] k entries in $\hat{a}$. The goal of the sparse Fourier transform (SFT) is to find $\hat{a}'$ such that

$$\|\hat{a} - \hat{a}'\|_2 \leq C \cdot \|\hat{a} - \hat{a}^{(k)}\|_2, \tag{2}$$

where $C > 0$ is some constant.

The final goal is to get a $O(k \log n)$ time randomized algorithm. Note that, when k is much smaller than n, the algorithm might not even read the whole input signal!

3.2 Sanity check: ℓ_1 guarantee

Before we see the algorithm, let's have a sanity check by designing an algorithm with ℓ_1 guarantee instead of the ℓ_2 guarantee in (2). This can actually be achieved by what we learned in the previous lecture.

From Lecture 9, we know that the sub-matrix of the inverse Fourier matrix F^{-1} is RIP. Concretely, let $S \subseteq [n]$ and $|S| = O(\epsilon^{-2} k \log^4 n)$, $A = F_S^{-1}$ satisfies the $(\epsilon, O(k))$-RIP. See the seminal work by Candés and Tao [CT06] for details. Also, from Lecture 13, Jelani showed that if A satisfies RIP, them one can recover an approximation of x given Ax via ℓ_1 minimization. Here, we think of x as the Fourier coefficients $\hat{a}$. Thus, $Ax = F_S^{-1}\hat{a}$, which is the input signal corresponding to the position at S. That is, one only needs to read $|S| = O(\epsilon^{-2} k \log^4 n)$ many samples from the input signal to recover $\hat{a}$. However, the running time of this algorithm is polynomial in n.

3.3 Warm-up: $k = 1$

As a warm-up, in this lecture, we are going to study SFT algorithm where there is only one dominant Fourier coefficient. Concretely, we consider two cases.

[1]In absolute value.

- Noiseless case: There exists $u = 0, 1, \ldots, n-1$ such that for any $u' \neq u$, $\hat{a}_{u'} = 0$. In this case, our goal is to find u and approximate $\hat{a}_u$.
- Noisy case: The goal is to find $\hat{a}'$ such that $\|\hat{a} - \hat{a}'\|_2 \leq C \cdot \|\hat{a} - \hat{a}^{(1)}\|_2$ for some constant $C > 0$.

3.3.1 Noiseless case

The idea is based on the following observation.

Observation 5. *For any $j = 0, 1, \ldots, n-1$, we have $a_j = \hat{a}_u \omega^{uj}$ by inverse Fourier transform and the fact that there is only one non-zero Fourier coefficient.*

As a result, by sampling a_0 and a_1, one can recover u and $\hat{a}_u$ as follows.

$$\hat{a}_u = a_0 \text{ and } u = n \times \text{ the angle between } \frac{a_1}{a_0} \text{ and the real line.} \tag{3}$$

The correctness of the algorithm simply follows from the fact that $a_0 = \hat{a}_u$ and $\frac{a_1}{a_0} = \omega^u$. This algorithm is called the *two-point sampling* and it only takes **constant time**. However, there are two issues for this algorithm.

- It does not scale up well with k since we need to solve a size k linear system, which in general does not have a $\tilde{O}(k)$ algorithm.
- This algorithm cannot handle any noise.

In the rest of this lecture, we are going to solve the second problem while the first one will be partially explained in the next lecture.

Remark 6. *The two-point sampling algorithm is related to OFDM, Pronys method, and matrix pencil.*

3.3.2 Noisy case

Recall that our goal is to find $\hat{a}'$ such that $\|\hat{a} - \hat{a}'\|_2 \leq C \cdot \|\hat{a} - \hat{a}^{(1)}\|_2$ for some $C > 0$. Note that this is only meaningful when $C \cdot \|\hat{a} - \hat{a}^{(1)}\|_2 < \|\hat{a}\|_2$. Otherwise, outputting $\hat{a}' = 0$ suffices. This is equivalent by assuming $\sum_{u' \neq u} \hat{a}_{u'}^2 < \epsilon \cdot \hat{a}_u^2$ for some small $\epsilon = \epsilon(C)$.

In the following, we will start with an algorithm for the noiseless case first with a more robust behavior. Then, we will show that in fact it can handle noisy case with constant probability. As a remark in the beginning, the algorithm is *combinatorial*, which is different from what we saw before. The main idea is to find u bit by bit and then estimate $\hat{a}_u$ in the end.

Bit 0 The idea follows from the following observation in Figure 2. When u is even, then $a_{n/2} = 1$. When u is odd, $a_{n/2} = -1$.

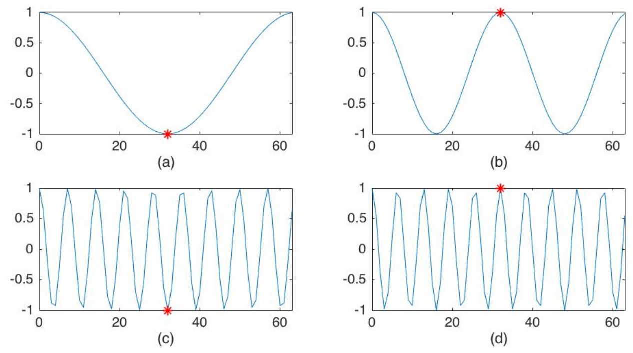

Figure 2: In this example, we pick $n = 64$. In (a), $\hat{a}_1 = 1$. In (b), $\hat{a}_2 = 1$. In (c), $\hat{a}_9 = 1$. In (d), $\hat{a}_{10} = 1$. The red star is the point $u = n/2$.

This observation can be explained by the following analysis. Write $u = 2v + b$ where v is some positive integer and $b = 0, 1$. We have $a_{n/2} = \hat{a}_u \omega^{un/2} = \hat{a}_u \omega^{vn+bn/2}$. As $\omega^{vn} = e^{2\pi iv} = 1$ for any integer v and $\omega^{n/2} = e^{\pi i} = -1$, we have $a_{n/2} = \hat{a}_u \omega^{bn/2} = \hat{a}_u(-1)^b$.

With the above analysis, the following test can help us find out the least significant bit of u. Denote the least significant bit of u as b_0, we have

$$b_0 = 0 \Leftrightarrow |a_0 - a_{n/2}| < |a_0 + a_{n/2}|.$$

This test works well in the noiseless case since if $b_0 = 0$, we have $a_0 = a_{n/2}$ and if $b_0 = 1$, we have $a_0 = -a_{n/2}$. However, what if the noise concentrate on a_0 and $a_{n/2}$ in the noisy case? A simple fix is randomly pick $r = 0, 1, \dots, n - 1$ and do the following test.

$$b_0 = 0 \Leftrightarrow |a_r - a_{r+n/2}| < |a_r + a_{r+n/2}|.$$

For convenience, the $r + n/2$ in the above equation is actually $r + n/2$ modulo n. Observe that $a_{r+n/2} = \hat{a}_u \omega^{r+n/2} = a_r \omega^{n/2} = a_r(-1)^b$. Thus, this new randomized test works for the noiseless case. See Figure 3 for geometric intuition. We will go back to the noisy analysis later.

Note that in this step we sample two points in the signal and the computation time is constant.

Bit 1 and beyond Here we describe the algorithm of estimating the second bit and the rest basically follows the same idea. First, we assume $b_0 = 0$ without loss of generality. The reason is that if $b_0 = 1$, we consider shifted signal a' defined as $a'_j = a_j \omega^j$ for any $j = 0, 1, \dots, n - 1$. From the *time-shift/phase-shift theorem*, we have $\hat{a}'_u = \hat{a}_{u-1}$ and thus yield a signal with least significant bit being 0.

Next, as $u = 2v$, we can think of it as $u = 4v' + b_1$ where $b_1 = 0, 1$. By randomly pick $r = 0, 1, \dots, n - 1$, we have the following test for b_1.

$$b_1 = 0 \Leftrightarrow |a_r - a_{r+n/4}| < |a_r + a_{r+n/4}|.$$

Similarly, the process can be repeated for the rest of the bits of u. Specifically, for the ith least significant bit of u, denoted as b_{i-1}, we have the following test. Randomly pick $r = 0, 1, \dots, n - 1$,

$$b_{i-1} = 0 \Leftrightarrow |a_r - a_{r+n/2^i}| < |a_r + a_{r+n/2^i}|.$$

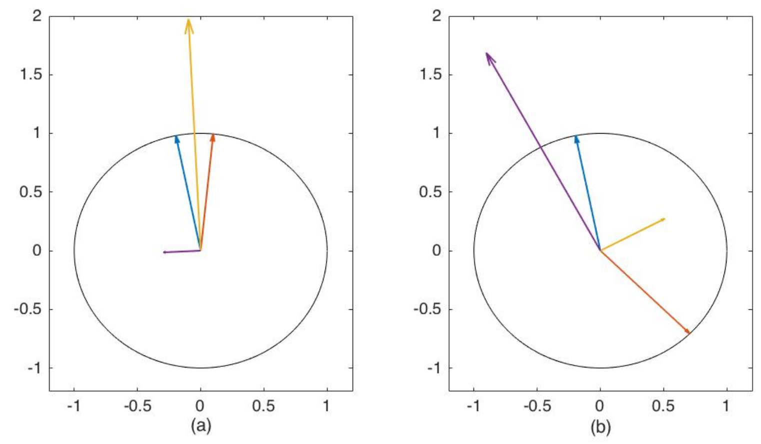

Figure 3: Geometric intuition for the test. The blue vector is a_r and the orange vector is $a_{r+n/2}$. The yellow vector is $a_r + a_{r+n/1}$ and the purple vector is $a_r - a_{r+n/2}$. (a) is the case where $b = 0$ and (b) is the case where $b = 1$.

Note that the number of samples being used is $2 \log n$ and the total computation time is $O(\log n)$.

Dealing with noise So far we have seen the algorithm and the analysis for noiseless case. Now, we are going to show that the algorithm works for noisy case with constant probability. Specifically, let $\hat{a}'$ denote the output, we want to show that for probability at least 2/3, $\|\hat{a} - \hat{a}'\|_2 \le \epsilon \|\hat{a} - \hat{a}^{(1)}\|_2$.

To start with, rewrite the input signal as follows. For each $j = 0, 1, \ldots, n-1$,

$$a_j = \hat{a}_u \omega^{uj} + \sum_{u' \neq u} \hat{a}_u \omega^{uj'} = \hat{a}_u \omega^{uj} + \mu_j. \tag{4}$$

Observe that $a = F^{-1}\hat{a}^{(1)} + \mu$ where the $(j+1)$th entry of μ is μ_j for each $j = 0, 1, \ldots, n-1$. That is, from the orthogonality property of DFT, we have

$$\sum_{j=0}^{n-1} \mu_j^2 = \|\mu\|_2^2 = \|F^{-1}(\hat{a} - \hat{a}^{(1)})\|_2^2 = n\|\hat{a} - \hat{a}^{(1)}\|_2^2 = n \sum_{u' \neq u} \hat{a}_{u'}^2. \tag{5}$$

Recall that we query $2 \log n$ points from the input signal in the algorithm. Denote these points as $j_{1,1}, j_{1,2}, j_{2,1}, j_{2,2} \ldots, j_{\log n,1}, j_{\log n,2}$. The following claim provides a sufficient condition for the algorithm to correctly estimate u's bits.

Claim 7. *For any $t = 1, 2, \ldots, \log n$, as long as $2(|\mu_{t,1}| + |\mu_{t,2}|) < |\hat{a}_u|$, then the estimation of the tth bit of u is correct.*

Proof. Recall that the algorithm estimate the tth bit by comparing $|a_{j_{t,1}} - a_{j_{t,2}}|$ and $|a_{j_{t,1}} + a_{j_{t,2}}|$. We have

$$a_{j_{t,1}} - a_{j_{t,2}} = \hat{a}_u(\omega^{j_{t,1}} - \omega^{j_{t,2}}) + (\mu_{j_{t,1}} - \mu_{j_{t,2}}),$$
$$a_{j_{t,1}} + a_{j_{t,2}} = \hat{a}_u(\omega^{j_{t,1}} + \omega^{j_{t,2}}) + (\mu_{j_{t,1}} + \mu_{j_{t,2}}).$$

As we discussed before, $|\omega^{j_{t,1}} - \omega^{j_{t,2}}|$ will either be 0 or 2. As a result, as long as $|\mu_{j_{t,1}} - \mu_{j_{t,2}}|, |\mu_{j_{t,1}} + \mu_{j_{t,2}}| \leq |\mu_{j_{t,1}}| + |\mu_{j_{t,2}}| \leq 2|\hat{a}_u|$, the comparison between $|a_{j_{t,1}} - a_{j_{t,2}}|$ and $|a_{j_{t,1}} + a_{j_{t,2}}|$ will be correct. That is, the estimation of the tth bit is correct. $\square$

With Claim 7, now it suffices to show that the probability of $2(|\mu_{t,1}| + |\mu_{t,2}|) \geq |\hat{a}_u|$ is small. We show this by the standard Markov+Union+Majority vote argument. Concretely, first compute the expectation of $\mu_{j_t}^2$. As $j_{t,1}, j_{t,2}$ are uniformly distributed in $0, 1, \dots, n-1$ for any $t = 1, 2, \dots, n$, we have

$$\mathbb{E}[\mu_{j_{t,1}}^2] = \mathbb{E}[\mu_{j_{t,2}}^2] = \sum_{u' \neq u} \hat{a}_{u'}^2. \tag{6}$$

By Markov's inequality, for any $t = 1, 2, \dots, n$ and $b = 1, 2$,

$$\mathbb{P}[|\mu_{j_{t,b}}| > \frac{|\hat{a}_u|}{4}] = \mathbb{P}[|\mu_{j_{t,b}}|^2 > \frac{|\hat{a}_u|^2}{16}] \leq \frac{16 \sum_{u' \neq u} \hat{a}_{u'}^2}{\hat{a}_u^2} \leq 16\epsilon, \tag{7}$$

where the last inequality follows the assumption in the beginning of this subsubsection. By picking ϵ small enough such that $32\epsilon < 1/3$, we have that the failure probability of estimating the tth is at most $1/3$. To amplify the probability, repeat the test for each bit by $O(\log\log n)$ times and do the majority vote. This will make the failure probability of each bit becomes at most $1/3\log n$ and thus after union bound, the failure probability of the algorithm is at most $1/3$. Note that $1/3$ is just an arbitrary constant.

Estimating $\hat{a}_u$ Previously, we show that with probability at least $2/3$, the output is u and $|\mu_{j_{t,1}}|, |\mu_{j_{t,1}}| \leq |\hat{a}_u|/4$ for all $t = 1, 2, \dots, n$. As the algorithm approximates $|\hat{a}_u|$ with $a_{j_{t,b}}/\omega^{uj_{t,b}} = \hat{a}_u + \mu_{j_{t,b}}/\omega^{uj_{t,b}}$ for any $t = 1, 2, \dots, n$ and $b = 1, 2$. The error is $|\mu_{j_{t,b}}/\omega^{uj_{t,b}}| = |\mu_{j_{t,b}}| \leq |\hat{a}_u|/4$. That is, the algorithm is a 1/4-approximation for $\hat{a}_u$. By tuning the parameter and some majority vote argument, one can improve this approximation ratio.

To sum up, the algorithm works as $\hat{a}_u$ has constant fraction of mass in $\hat{a}$ with time complexity $O(\log n \log\log n)$.

4 History

Finally, let's take a look at the history of Fourier transform to wrap up this lecture.

- Goldreich and Levin [GL89] and Kushilevitz and Mansour [KM91] started the study of sub-linear sparse Fourier algorithms for the Hadamard transform.
- Mansour [Man92] gave a poly$(k, \log n)$ time assuming the entries of the signal are integers from $-$poly(n) to poly(n).
- Gilbert, Guha, Indyk, Muthukrishnan, and Strauss [GGIMS02] gave a $O(k^2\text{poly}(\log n))$ time algorithm.
- Gilbert, Muthukrishnan, and Strauss [GMS05] gave a $O(k\text{poly}(\log n))$ time algorithm.

- Hassanieh, Indyk, Katabi, and Price [HIKP12] gave a $O(k \log n \log(n/k))$ time algorithm. The run time is $O(k \log n)$ if we consider noiseless case.

There are also several works on *deterministic* Fourier algorithm and closing the gap between randomized and deterministic sparse Fourier transform remains an open problem.

5 Next lecture

Next time we are going to study algorithms for k-sparse Fourier transform under two assumptions: (i) noiseless (ii) the peaks of the Fourier coefficients are uniformly distributed.

References

[CT06] Emmanuel J. Candés and Terence Tao. Near-optimal signal recovery from random projections: universal encoding strategies? *IEEE Trans. Inform. Theory*, 52(12):5406–5425, 2006.

[GL89] O. Goldreich and L. Levin. A hard-corepredicate for allone-way functions. *STOC*, 1989.

[KM91] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum *STOC*, 1991.

[Man92] Yishay Mansour. Randomized Interpolation and Approximation of Sparse Polynomials. *ICALP*, July, 1992.

[GGIMS02] Anna C. Gilbert, Sudipto Guha, Piotr Indyk, S. Muthukrishnan, Martin Strauss. Near-optimal sparse Fourier representations via sampling. *STOC*, May, 2002.

[GMS05] Anna Gilbert, S. Muthukrishnan, and Martin Strauss. Improved time bounds for near-optimal space Fourier representations. *SPIE Wavelets*, August, 2005.

[HIKP12] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly Optimal Sparse Fourier Transform. *STOC*, May, 2012.