A-015

Implementation and Evaluation of a Significant Fourier Transform Algorithm

Kazuki Tsubo† Thomas Zeugmann†

1. Introduction

In mathematics, the Fourier transform is an operation that transforms one complex-valued function of a real variable into another. Nowadays the Fourier transform is used in various fields; so it is a very important operation. In computer science, the **Fast Fourier Transform** (abbr. FFT) algorithm is widely used and it calculates **Discrete Fourier Transform** efficiently in time $O(N \log N)$, where N is the input size. However, for data intensive applications, even the FFT may be too slow. Starting from the observation that in many applications not the entire Fourier transform is needed, but only the τ-significant Fourier transform coefficients, Akavia [1] proposed the **Significant Fourier Transform** (abbr. SFT) algorithm.

That is, we only compute Fourier coefficients whose magnitude is at least τ-fraction (say, 1%) of the energy, i.e., the sum of squared Fourier coefficients. This deterministic algorithm finds the τ-significant Fourier coefficients of functions f over any finite Abelian group G in time polynomial in $log|G|$, $1/\tau$ and $L_1(\hat{f})$ (here $L_1(\hat{f})$ denotes the sum of absolute values of the Fourier coefficients of f). The computing time of SFT does not use N, this point is superior to FFT. This effect becomes visible as N grows. The present abstract reports our experience in implementing the SFT algorithm and verifies that it is indeed much faster than the FFT.

2. Preliminaries

2.1 The Discrete Fourier Transform Problem

To compute the Discrete Fourier Transform (DFT) is well studied problem. We denote by $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$ the additive group of integers modulo N. The *inner product* of complex valued functions f, g over a domain G is $\langle f, g\rangle = \frac{1}{|G|}\Sigma_{x\in G}\ f(x)\overline{g(x)}$. The *Fourier transform* of a complex valued function f over G is the function $\hat{f}: G \to \mathbb{C}$ defined by $\hat{f}(\alpha) = \langle f, \chi_\alpha\rangle$. Here χ_α is a character of $\mathbb{Z}_N$. The *characters* of $\mathbb{Z}_N$ are the functions $\{\chi_\alpha: \mathbb{Z}_N \to \mathbb{C}\}_{\alpha\in\mathbb{Z}_N}$ defined by $\chi_\alpha = \omega_N^{\alpha x}$, where $\omega_N = e^{\frac{2\pi i}{N}}$ is a complex N-th root of unity.

†Graduate School of Information Science and Technology, Hokkaido University.

2.2 Notations and Definitions

Significant Fourier Coefficients. For any $\alpha \in \mathbb{Z}_N$, $val_\alpha \in \mathbb{C}$ and $\tau, \epsilon \in [0, 1]$, we say that α is a τ-significant Fourier coefficient iff $|\hat{f}(\alpha)|^2 \geq \tau||f||_2^2$.

Small biased sets [4]. We say that a set $A \subseteq \mathbb{Z}_N$ is γ-biased in $\mathbb{Z}_N$ if $|\mathbb{E}_{x\in A}[\chi_\alpha(x)]| \leq \gamma$ for every non trivial character χ_α of the group $\mathbb{Z}_N$, $\alpha \neq 0$.

(γ, I)-biased sets [1]. For any Abelian group G and subsets $B, I \subseteq G$, we say that B is $(\gamma, I) - biased$ in G if for every character χ of the group G, $|\mathbb{E}_{x\in B\cap I}[\chi(x)] - \mathbb{E}_{x\in I}[\chi(x)]| \leq \gamma$.

3. Algorithms

3.1 Fast Fourier Transform (FFT)

The FFT is an efficient algorithm to compute the DFT and its inverse. By far the most common method used to compute the FFT is the Cooley-Tukey algorithm [3]. This algorithm is to divide the transform into two pieces of size $N/2$ at each step, and is therefore limited to power-of-two sizes.

3.2 Significant Fourier Transform (SFT)

The SFT algorithm exclusively computes the Fourier coefficients whose magnitude is at least a τ-fraction of the energy. This algorithm is deterministic and efficient. Its running time and query complexity is polynomial in $\log|G|$, $1/\tau$ and $L_1(\hat{f})$. The SFT algorithm is composed of two parts:

1. Queries generating part. A set of entries $S = S(G, \tau, t) \subseteq G$ is chosen, given G, τ, t.
2. Fixed Queries part. The significant Fourier coefficients of a function $f : G \to \mathbb{C}$ such that $L_1(\hat{f}) \leq t$ are found, given G, τ and the restriction to S of f.

Additionally, we also need the so-called Distinguishing Algorithm (cf. Akavia [1]) which takes as inputs $\{a, b\} \in \mathbb{Z}_N \times \mathbb{Z}_N$, $\tau \in \mathbb{R}^+$, $A, B \subseteq \mathbb{Z}_N$ and $\{(x, f(x))\}_{x\in A\setminus B}$. Here, A, and B are small biased set.

The output is 0 or 1, i.e., the Distinguishing Algorithm checks whether or not there exist a τ-significant Fourier coefficient in the range $\{a, b\}$.

The Distinguishing Algorithm is called by the Fixed Queries part.

4. Preparation

We implemented the SFT algorithm in C++. The algorithm is based on the SFT theory given in [1], but two points have been improved. One point is value of γ. The other point concerns the small biased sets. These two points were an important parts in mounting.

4.1 Presumption of γ and $\|f\|_2^2$.

In the first part of the SFT algorithm, we have to get γ. But $\gamma = O(\frac{\tau}{t^2(1+logN)})$ is difficult to compute. So, we adopted binary search here, because we know roughly the range of γ. There remains a problem to obtain the expression precisely which we shall resolve in the future. Right now, we have been mainly interested in the running time.

Furthermore, $||f||_2^2$ is treated as a threshold, but this value is difficult to compute, too. In the SFT algorithm, it is approximated by $||f||_2^2 = \frac{1}{|A|}\Sigma_{x\in A} f(x)^2 + O(\tau||f||_2^2)$. Clearly, the problem here is the error margin of $O(\tau||f||_2^2)$. We resolve this problem by the same method.

4.2 Constructing a Small Biased Set

In the SFT algorithm, we need a γ-biased set. We already know γ from the previous part described above. There is some explicit generating method [4], [5]. The most efficient method is preferable. We implemented the random method proposed by [2]. As stated there:

> A random set S_m of size $|S_m| = O(m/\gamma^2)$ is γ-biased set with very high probability.

5. Results

Here we only report the computing time, i.e., we did not include the search time. This problem can be solved if we theoretically obtain a strict expression for γ and $||f||_2^2$.

The experiments have been executed on a notebook PC (Atom1.66GHz, MEM1GB, WinXP).

$\tau = 0.1$

N	FFT time(s)	SFT time(s)	Set size
1024	0	0	1
2048	0	0	1
4096	0.016	0	1
8192	0.047	0	1
16384	0.078	0.062	1
32768	0.157	0.031	1

$\tau = 0.01$

N	FFT time(s)	SFT time(s)	Set size
1024	0	0	1
2048	0	0	1
4096	0.015	0.015	1
8192	0.032	0.016	1
16384	0.062	0.047	1
32768	0.14	0.046	1

$\tau = 0.001$

N	FFT time(s)	SFT time(s)	Set size
1024	0	0.062	96
2048	0	0.078	44
4096	0.016	0.141	14
8192	0.047	0.016	1
16384	0.078	0.031	1
32768	0.141	0.032	1

6. Conclusions

It has been verified that the SFT is efficiently computable regardless of the size of N. However, when the elements of returned as answers do increase, it might be slower than the FFT. As a future tasks, we plan to improve the accuracy of the presumption. Moreover, we aim to apply the SFT to an actual sample, e.g., a jpeg image, etc. We want to learn whether or not the SFT really applies to such problems that are usually solved by using the FFT but for which the SFT has been designed to reduce the computation time.

References

[1] A. Akavia. Finding significant Fourier transform coefficients deterministically and locally. Technical Report TR08-102, Electronic Colloquium on Computational Complexity, 2008.

[2] A. Akavia and R. Venkatesan. Perturbation codes. In *46th Annual Allerton Conference on Communication, Control, and Computing*, pages 1403–1409. IEEE, 2008.

[3] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[4] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22:838–856, 1993.

[5] A. Razborov, E. Szemerédi, and A. Widgerson. Constructing small sets that are uniform in arithmetic progressions. *Combinatorics, Probability and Computing*, 2:513–518, 1993.