The Sparse Fourier Transform: Theory & Practice

by

Haitham Al Hassanieh

M.S. in Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2011)
B.Eng. in Computer & Communications Engineering, American University of Beirut (2009)

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2016

Author . . . . . .
Department of Electrical Engineering and Computer Science
December 7, 2015

Certified by . . . . . .
Dina Katabi
Professor
Thesis Supervisor

Accepted by . . . . . .
Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Theses

The Sparse Fourier Transform: Theory & Practice

by

Haitham Al Hassanieh

Submitted to the Department of Electrical Engineering and Computer Science
on December 7, 2015, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The Fourier transform is one of the most fundamental tools for computing the frequency representation of signals. It plays a central role in signal processing, communications, audio and video compression, medical imaging, genomics, astronomy, as well as many other areas. Because of its widespread use, fast algorithms for computing the Fourier transform can benefit a large number of applications. The fastest algorithm for computing the Fourier transform is the FFT (Fast Fourier Transform) which runs in near-linear time making it an indispensable tool for many applications. However, today, the runtime of the FFT algorithm is no longer fast enough especially for big data problems where each dataset can be few terabytes. Hence, faster algorithms that run in sublinear time, *i.e.*, do not even sample all the data points, have become necessary.

This thesis addresses the above problem by developing the Sparse Fourier Transform algorithms and building practical systems that use these algorithms to solve key problems in six different applications.

Specifically, on the theory front, the thesis introduces the Sparse Fourier Transform algorithms: a family of sublinear time algorithms for computing the Fourier transform faster than FFT. The Sparse Fourier Transform is based on the insight that many real-world signals are sparse, *i.e.*, most of the frequencies have negligible contribution to the overall signal. Exploiting this sparsity, the thesis introduces several new algorithms which encompass two main axes:

- **Runtime Complexity:** The thesis presents nearly optimal Sparse Fourier Transform algorithms that are faster than FFT and have the lowest runtime complexity known to date.

- **Sampling Complexity:** The thesis presents Sparse Fourier Transform algorithms with optimal sampling complexity in the average case and the same nearly optimal runtime complexity. These algorithms use the minimum number of input data samples and hence, reduce acquisition cost and I/O overhead.

On the systems front, the thesis develops software and hardware architectures for leveraging the Sparse Fourier Transform to address practical problems in applied fields. Our systems customize the theoretical algorithms to capture the structure of sparsity in each application, and hence maximize the resulting gains. We prototype all of our systems and evaluate them in accordance with

the standard's of each application domain. The following list gives an overview of the systems presented in this thesis.

- **Wireless Networks:** The thesis demonstrates how to use the Sparse Fourier Transform to build a wireless receiver that captures GHz-wide signals without sampling at the Nyquist rate. Hence, it enables wideband spectrum sensing and acquisition using cheap commodity hardware.
- **Mobile Systems:** The thesis uses the Sparse Fourier Transform to design a GPS receiver that both reduces the delay to find the location and decreases the power consumption by $2\times$.
- **Computer Graphics:** Light fields enable new virtual reality and computational photography applications like interactive viewpoint changes, depth extraction and refocusing. The thesis shows that reconstructing light field images using the Sparse Fourier Transform reduces camera sampling requirements and improves image reconstruction quality.
- **Medical Imaging:** The thesis enables efficient magnetic resonance spectroscopy (MRS), a new medical imaging technique that can reveal biomarkers for diseases like autism and cancer. The thesis shows how to improve the image quality while reducing the time a patient spends in an MRI machine by $3\times$ (e.g., from two hours to less than forty minutes).
- **Biochemistry:** The thesis demonstrates that the Sparse Fourier Transform reduces NMR (Nuclear Magnetic Resonance) experiment time by $16\times$ (e.g. from weeks to days), enabling high dimensional NMR needed for discovering complex protein structures.
- **Digital Circuits:** The thesis develops a chip with the largest Fourier Transform to date for sparse data. It delivers a 0.75 million point Sparse Fourier Transform chip that consumes $40\times$ less power than prior FFT VLSI implementations.

Thesis Supervisor: Dina Katabi
Title: Professor

Dedicated to Maha & Nadima Al Hassanieh

Acknowledgments

The work presented in this thesis would not have been possible without the help and support of a large group of people to whom I owe a lot of gratitude.

First and foremost, I am really thankful for my advisor Dina Katabi who has given me this tremendous opportunity to come work with her at MIT. For six years, she has supported me and worked closely with me. She gave me the freedom to work on a wide range of projects outside her research area. She was just as invested in my research and pushed very hard for the Sparse Fourier Transform so we can bring it to completion. Her passion, hard work and dedication to the success of her students is truly inspiring. I could not wish for a better advisor.

I am also really grateful for Piotr Indyk who was like a second advisor to me. He has supported me and guided me through a lot problems. His decision to work with me on the Sparse Fourier Transform has changed and shaped my entire PhD career. I truly admire and respect him.

I would like to thank the rest of my thesis committee members and letter writers: Elfar Adalsteinsson, Victor Bahl and Fredo Durand. They have introduced me to fascinating research areas and have given me a lot of advice and support that helped my career.

I would also like to thank everyone who worked on the Sparse Fourier Transform project. Dina, Piotr and Eric Price were the first people to work with me. They played an indispensable role in developing the theoretical Sparse Fourier Transform algorithms. The next person to work with me was Fadel Adib who helped me take on the hard task of kickstarting the applications. Fadel was like a powerhouse that allowed me to push through my vision for the Sparse Fourier Transform. After that, Lixin Shi helped me bring to life more applications. He worked with me tirelessly for a very long time while making the work process extremely enjoyable. Finally, I would like to thank all of the remaining people who contributed to this thesis: Elfar Adalsteinsson, Fredo Durand, Omid Abari, Ezzeldin Hamed, Abe Davis, Badih Ghazi, Ovidiu Andronesi, Vladislav Orekhov, Abhinav Agarwal and Anantha Chandrakasan.

During my time at MIT, I was lucky to have a large group of friends. I am eternally thankful to my closest friend Fadel Adib who has been there with me every step of the way on this long PhD road and has helped me a lot throughout my PhD career. I am also extremely grateful for Ankur Mani and Ila Sheren who helped me survive six years of MIT and made life so much fun. Jue Wang is another friend who kept me grounded and on my feet during the hard conflicts The rest of my friends and colleagues: Omid, Ezz, Lixin, Abe, Zach, Chen-Yu, Nate, Rahul, Mary, Katina, Kate, Deepak, Swarun, Stephanie, Hongzi and Mingmin are some of the amazing people at MIT who made my time here so wonderful.

I cannot express sufficient gratitude for my parents Najwa and Zuhair and my siblings Dima and Mazen for their endless love, support and advice. I could not have achieved what I did without their help and I owe all my success to them. No matter what I do, I can never repay them. I am lucky to have a group of wonderful women who have helped raise me: Maha, Nadima, May, Nahla, Rima, Nawal and Mona; their wisdom, strength and grace have inspired me to be a much better person. I am also thankful for the other 53+ members of my family who I love and appreciate tremendously.

Last but not least, I have been blessed with a large group of brothers: Mazen, Fadel, Ahmad, Hassane, Majd, Amer, Ali, Assem, Bilal, Majed and Hussien. They are always there for me and I can rely on them for anything. I am also grateful to my favorite person in the world and my best friend: Hiba as well as the rest of my girls: Sarah, Pamela, Jessica, Lamia, Raghid, Hania, Lina, Naela, Mayar and Dima.

Previously Published Material

Chapter 3 revises a previous publication [72]: H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Simple and Practical Algorithm for Sparse Fourier Transform. SODA, 2012 2012.

Chapter 4 revises a previous publication [70]: H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Nearly Optimal Sparse Fourier Transform. STOC, 2012.

Chapter 5 revises and extends a previous publication [56]: B. Ghazi, H. Hassanieh, P. Indyk, D. Katabi, E. Price, and L. Shi. Sample-Optimal Average-Case Sparse Fourier Transform in Two Dimensions. Allerton, 2013.

Chapter 7 revises a previous publication [74]: H. Hassanieh, L. Shi, O. Abari, E. Hamed, and D. Katabi. GHz-Wide Sensing and Decoding Using the Sparse Fourier Transform. INFOCOM, 2014.

Chapter 8 revises a previous publication [69]: H. Hassanieh, F. Adib, D. Katabi, and P. Indyk. Faster GPS via the Sparse Fourier Transform. MOBICOM, 2012.

Chapter 9 revises a previous publication [157]: L. Shi, H. Hassanieh, A. Davis, D. Katabi, F. Durand. Light Field Reconstruction Using Sparsity in the Continuous Fourier Domain. SIGGRAPH, 2015.

Chapter 10 builds on and extends a previous publication [156]: L. Shi, O. Andronesi, H. Hassanieh, B. Ghazi, D. Katabi, E. Adalsteinsson. MRS Sparse-FFT: Reducing Acquisition Time and Artifacts for In Vivo 2D Correlation Spectroscopy, ISMRM, 2013.

Chapter 11 revises a previous publication [73]: H. Hassanieh, M. Mayzel, L. Shi, D. Katabi, and V.Y. Orekhov Fast Multi-dimensional NMR Acquisition and Processing Using the Sparse FFT, Journal of Biomolecular NMR, Springer, 2015.

Appendix G revises a previous publication [3]: O. Abari, E. Hamed, H. Hassanieh, A. Agarwal, D. Katabi, A. Chandrakasan, and V. Stojanovic. A 0.75 Million-Point Fourier Transform Chip for Frequency-Sparse Signals, ISSCC, 2014.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The Fourier transform is one of the most important and widely used computational tasks. It is a foundational tool commonly used to analyze the spectral representation of signals. Its applications include audio/video processing, radar and GPS systems, wireless communications, medical imaging and spectroscopy, the processing of seismic data, and many other tasks [14, 27, 77, 137, 176, 183]. Hence, faster algorithms for computing the Fourier transform can benefit a wide range of applications. The fastest algorithm to compute the Fourier transform today is the Fast Fourier Transform (FFT) algorithm [34]. Invented in 1965 by Cooley and Tukey, the FFT computes the Fourier transform of a signal of size n in $O(n \log n)$ time. This near-linear time of the FFT made it one of the most influential algorithms in recent history [32]. However, the emergence of big data problems, in which the processed datasets can exceed terabytes [154], has rendered the FFT's runtime too slow. Furthermore, in many domains (e.g., medical imaging, computational photography), data acquisition is costly or cumbersome, and hence one may be unable to collect enough measurements to compute the FFT. These scenarios motivate the need for sublinear time algorithms that compute the Fourier transform faster than the FFT algorithm and use only a subset of the input data required by the FFT.

The key insight to enable sublinear Fourier transform algorithms is to exploit the inherit *sparsity* of natural signals. In many applications, most of the Fourier coefficients of the signal are small or equal to zero, i.e., the output of the Fourier transform is *sparse*. For such signals, one does not need to compute the entire output of the Fourier transform; it is sufficient to only compute the large frequency coefficients. Fourier sparsity is in fact very common as it appears in audio/video, medical imaging, computational learning theory, analysis of Boolean functions, similarity search in databases, spectrum sensing, datacenter monitoring, etc [5, 28, 95, 112, 114, 130].

The research presented in this thesis pursues the above insight in the context of both algorithms and systems in order to answer the following two core questions:

How can we leverage sparsity to design faster Fourier transform algorithms?

&

How do we build software and hardware systems that adapt our algorithms to various application domains in order to deliver practical gains?

This thesis answers the above questions by developing the Sparse Fourier Transform algorithms: a family of sublinear algorithms for computing the Fourier transform of frequency-sparse signals faster than FFT and using a small subset of the input samples. The thesis also develops architectures for leveraging sparsity to build practical systems that solve key problems in wireless networks, mobile systems, computer graphics, medical imaging, biochemistry and digital circuits.

This thesis makes both theoretical and systems contributions. The theoretical contributions form the algorithmic foundations of the Sparse Fourier Transform which encompass two main axes:

- **Optimizing the Runtime Complexity:** The thesis presents Sparse Fourier Transform algorithms with the lowest runtime complexity known to date. For exactly sparse signals, we present an algorithm that runs in $O(k \log n)$ time where k is the number of large frequency coefficients (i.e. sparsity) and n is the signal size. This algorithm is optimal if the FFT algorithm is optimal. For approximately sparse signals, which we will formally define in Section 1.1.1, we present an algorithm that runs in $O(k \log n \log (n/k))$ time which is $\log n$ factor away from optimal. Both algorithms improve over FFT for any sparsity $k = o(n)$ and have small "Big-Oh" constants. As a result, they are often faster than FFT in practice and run quickly on very large data sets.

- **Optimizing the Sampling Complexity:** The thesis presents Sparse Fourier Transform algorithms with the *optimal* sampling complexity for average case inputs, *i.e.*, these algorithms use the minimum number of input data samples that would produce a correct answer. Hence, they reduce the acquisition cost, bandwidth and I/O overhead needed to collect, transfer and store the data. Specifically, these algorithms require only $O(k)$ samples for exactly sparse signals and $O(k \log n)$ samples for approximately sparse signals while keeping the same runtime complexity of the aforementioned worst case algorithms. Furthermore, the algorithms naturally extend to multi-dimensional Sparse Fourier Transforms, without incurring much overhead.

The simplicity and practicality of the Sparse Fourier Transform algorithms allowed us to use them to build six new systems that address major challenges in the areas of wireless networks and mobile systems, computer graphics, medical imaging, biochemistry and digital circuits. Table 1.1 summarizes the systems developed in this thesis and our contributions to each application.

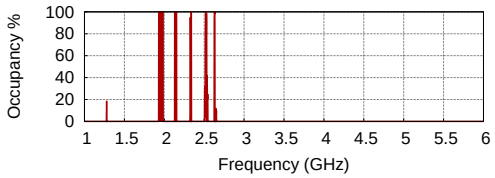
Leveraging the Sparse Fourier Transform to build practical systems, however, is not always straightforward. The Sparse Fourier Transform is a framework of algorithms and techniques for analyzing sparse signals. It inherently depends on the sparsity of the signal which changes from one application to another. Thus, incorporating domain knowledge from each application allows us to deliver much more significant gains. First, different applications exhibit different levels and structure of sparsity. For example, in wireless networks, occupied frequencies in the wireless spectrum are not randomly distributed. They are instead clustered based on transmission regulations set by the FCC. Hence, incorporating the structure of the sparsity into the algorithm and system is essential for achieving good performance gains. In addition, in some applications such as medical imaging, the sparsity of the signal is not apparent, which requires developing methods to sparsify the signal before being able to use the Sparse Fourier Transform. In other applications, sparsity appears only in part of the system and thus we have to redesign the entire system in order to propagate the gains of the Sparse Fourier Transform to other stages and improve the overall system

Wireless Networks

Spectrum Sensing & Acquisition

Realtime GHz-wide spectrum acquisition requires costly and highly customized hardware that consumes high power.

Contribution: Built a receiver that can acquire a bandwidth 6× larger than its digital sampling rate enabling realtime GHz spectrum sensing and decoding using cheap components typically used in WiFi receivers.

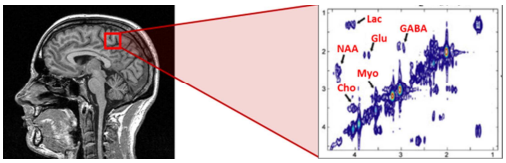


Medical Imaging

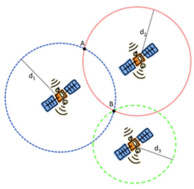
Magnetic Resonance Imaging (MRI)

Magnetic resonance spectroscopy (MRS) detects the biochemical content of each voxel in the brain and can be used to discover disease biomarkers.

Contribution: Delivered a system for processing MRS data that enhances image quality and reduces the time the patient has to spend in the MRI machine by 3×.



Mobile Systems

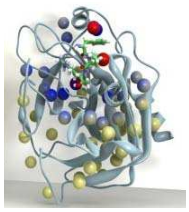


GPS

GPS receivers consume a lot of power on mobile devices which drains the battery.

Contribution: Designed a GPS receiver that reduces the time and power it takes the receiver to lock on its location.

Biochemistry



Nuclear Magnetic Resonance

NMR is used to discover biomolecular structures of proteins.

Contribution: Reduced NMR experiment time by 16× enabling high-dimensional NMR which is needed for discovering complex protein structures.

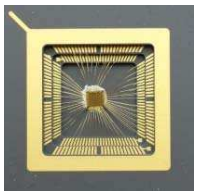
Computer Graphics

Light Field Photography

Light field photography uses a camera array to refocus and change the viewpoint in post-processing.

Contribution: Developed a light field reconstruction system that reduces the camera sampling requirements and improves image reconstruction quality.

Digital Circuits



Sparse Fourier Chip

Massive size Fourier transforms require large silicon area and consume high power.

Contribution: Delivered a 0.75 million point Fourier transform chip for sparse data that consumes 40× less power than prior FFT VLSI implementations.

Table 1.1: **Practical Systems Developed Using the Sparse Fourier Transform**

performance. Hence, adapting the Sparse Fourier Transform into practical applications requires a deep understanding of the application domain and customizing the algorithms to become more in-sync with the system requirements.

The next two sections provide an overview of the theoretical algorithms and the software and hardware systems developed in this thesis.

1.1 Sparse Fourier Transform Algorithms

The existence of Fourier transform algorithms faster than FFT is one of the central questions in the theory of algorithms. The past two decades have witnessed significant advances in sublinear Fourier algorithms for sparse signals. The first such algorithm (for the Hadamard transform) appeared in [102] (building on [63]). Since then, several sublinear algorithms for complex Fourier inputs have been discovered [6, 7, 57, 59, 88, 116]. The main value of these algorithms is that they outperform FFT's runtime for sparse signals. For very sparse signals, the fastest algorithm is due to [59] and has $O(k \log^c(n) \log(n/k))$ runtime, for some $c > 2$. This algorithm outperforms FFT for any k smaller than $\Theta(n/\log^a n)$ for some $a > 1$.

Despite this impressive progress, the prior work suffers from two main limitations. First, none of the existing algorithms improves over FFT's runtime for the whole range of sparse signals, i.e., $k = o(n)$. Second, the aforementioned algorithms are quite complex, and suffer from large "Big-Oh" constants that lead to long runtime in practice. For example, an implementation of the algorithm in [59] can only outperform FFT for extremely sparse signals where $k/n \leq 3.2 \times 10^{-5}$ [89]. The algorithms in [57, 88] require an even sparser signal (i.e., larger n and smaller k). As a result, it has been difficult to incorporate those algorithms into practical systems.

In this section, we give an overview of our Sparse Fourier Transform algorithms, which address the above limitations of prior work. We start by formalizing the problem. We then describe the algorithmic framework underlying all of our Sparse Fourier Transform algorithms. Once we establish this framework, we describe the different techniques that can be used at each step of a Sparse Fourier Transform algorithm. We finally present the various algorithms that result from using these techniques.

1.1.1 Problem Statement

Consider a signal $\mathbf{x}$ of size n whose discrete Fourier transform is $\hat{\mathbf{x}}$ defined by:

$$\hat{\mathbf{x}}(f) = \sum_{t=0}^{n-1} \mathbf{x}(t) \cdot e^{-j2\pi ft/n} \tag{1.1}$$

$\hat{\mathbf{x}}$ is exactly k-sparse if it has exactly k non-zero frequency coefficients while the remaining $n - k$ coefficients are zero. In this case, the goal of the Sparse Fourier Transform is to exactly recover $\hat{\mathbf{x}}$ by finding the frequency positions f and values $\hat{\mathbf{x}}(f)$ of the k non-zero coefficients. For general signals, the Sparse Fourier Transform computes a *k-sparse approximation* $\hat{\mathbf{x}}'$ of $\hat{\mathbf{x}}$. The best *k-sparse approximation* of $\hat{\mathbf{x}}$ can be obtained by setting all but the largest k coefficients of $\hat{\mathbf{x}}$ to 0.

The goal is to compute an approximation $\widehat{\mathbf{x}}'$ in which the error in approximating $\widehat{\mathbf{x}}$ is bounded by the error on the best k-*sparse approximation*. Formally, $\widehat{\mathbf{x}}$ has to satisfy the following ℓ_2/ℓ_2 *guarantee*:

$$\|\widehat{\mathbf{x}} - \widehat{\mathbf{x}}'\|_2 \leq C \min_{k\text{-sparse } \mathbf{y}} \|\widehat{\mathbf{x}} - \mathbf{y}\|_2, \tag{1.2}$$

where C is some approximation factor and the minimization is over exactly k-sparse signals.

In the remainder of this section, we will describe the algorithmic framework and techniques in terms of exactly sparse signals. However, the full details and extensions to the general case of approximately sparse signals can be found in Chapters 3, 4, and 5.

1.1.2 Algorithmic Framework

The Sparse Fourier Transform has three main components: *Frequency Bucketization*, *Frequency Estimation*, and *Collision Resolution*.

1. Frequency Bucketization:

The Sparse Fourier Transform starts by hashing the frequency coefficients of $\widehat{\mathbf{x}}$ into buckets such that the value of the bucket is the sum of the values of the frequency coefficients that hash into the bucket. Since $\widehat{\mathbf{x}}$ is sparse, many buckets will be empty and can be simply discarded. The algorithm then focuses on the non-empty buckets and computes the positions and values of the large frequency coefficients in those buckets in what we call the *frequency estimation* step.

The process of *frequency bucketization* is achieved through the use of filters. A filter suppresses and zeroes out frequency coefficients that hash outside the bucket while passing through frequency coefficients that hash into the bucket. The simplest example of this is the aliasing filter. Recall the following basic property of the Fourier transform: *subsampling in the time domain causes aliasing in the frequency domain*. Formally, let $\mathbf{b}$ be a subsampled version of $\mathbf{x}$, *i.e.*, $\mathbf{b}(i) = \mathbf{x}(i \cdot p)$ where p is a subsampling factor that divides n. Then, $\widehat{\mathbf{b}}$, the Fourier transform of $\mathbf{b}$ is an aliased version of $\widehat{\mathbf{x}}$, *i.e.*:

$$\widehat{\mathbf{b}}(i) = \sum_{m=0}^{p-1} \widehat{\mathbf{x}}(i + m(n/p)). \tag{1.3}$$

Thus, an aliasing filter is a form of bucketization in which frequencies equally spaced by an interval $B = n/p$ hash to the same bucket and there are B such buckets as shown in Figure 1-1. The hashing function resulting from this bucketization can be written as: $h(f) = f \mod n/p$. Further, the value in each bucket is the sum of the values of only the frequency coefficients that hash to the bucket as can be seen from Equation 1.3.

For the above aliasing filter, the buckets can be computed efficiently using a B-point FFT which takes $O(B \log B)$ time. We set $B = O(k)$ and hence bucketization takes only $O(k \log k)$ time and uses only $O(B) = O(k)$ of the input samples of $\mathbf{x}$. In Section 1.1.3, we describe additional types of filters that are used by our Sparse Fourier Transform algorithms to perform *frequency bucketization*.

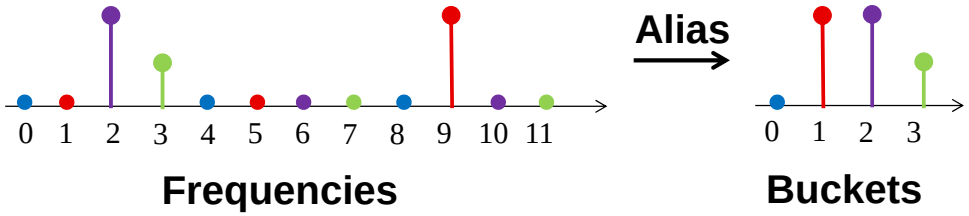


Figure 1-1: **Bucketization Using Aliasing Filter:** Sub-sampling a signal by $3\times$ in the time domain, results in the spectrum aliasing. Specifically, the 12 frequency will alias into 4 buckets. Frequencies that are equally spaced by 4 (shown with the same color) end up in the same bucket.

2. Frequency Estimation:

In this step, the Sparse Fourier Transform estimates the positions and values of the non-zero frequency coefficients which created the energy in each of the non-empty buckets. Since $\widehat{\mathbf{x}}$ is sparse, many of the non-empty buckets will likely have a single non-zero frequency coefficient hashing into them, and only a small number will have a collision of multiple non-zero coefficients. We first focus on buckets with a single non-zero frequency coefficients and estimate the value and the position of this non-zero frequency, *i.e.*, $\widehat{\mathbf{x}}(f)$ and the corresponding f.

In the absence of a collision, the value of the non-zero frequency coefficient is the value of the bucket it hashes to since all other frequencies that hash into the bucket have zero values. Hence, we can easily find the value of the non-zero frequency coefficient in a bucket. However, we still do not know its frequency position f, since *frequency bucketization* mapped multiple frequencies to the same bucket. The simplest way to compute f is to leverage the *phase-rotation property* of the Fourier transform, which states that a shift in time domain translates into phase rotation in the frequency domain [115]. Specifically, we perform the process of bucketization again, after a circular shift of $\mathbf{x}$ by τ samples. Since a shift in time translates into a phase rotation in the frequency domain, the value of the bucket changes from $\widehat{\mathbf{b}}(i) = \widehat{\mathbf{x}}(f)$ to $\widehat{\mathbf{b}}^{(\tau)}(i) = \widehat{\mathbf{x}}(f) \cdot e^{j\Delta\phi}$ where the phase rotation is:

$$\Delta\phi = 2\pi f\tau/n \tag{1.4}$$

Hence, using the change in the phase of the bucket, we can estimate the position of the non-zero frequency coefficient in the bucket. Note that the phase wraps around every 2π and so the shift τ should be 1 to avoid the phase wrapping for large values of f.[1] Since, there are k non-zero frequency coefficients, this *frequency estimation* can be done efficiently using at most $O(k)$ computations. In Section 1.1.3, we describe additional techniques that are used by our Sparse Fourier Transform algorithms to estimate the values and positions of non-zero frequency coefficients.

3. Collision Resolution:

Non-zero frequency coefficients that are isolated in their own bucket can be properly estimated as described above. However, when non-zero frequencies collide in the same bucket, we are unable

[1]Note that for approximately sparse signals, multiple time shifts are used to average the noise and ensure robust estimation as we show in Chapter 4.

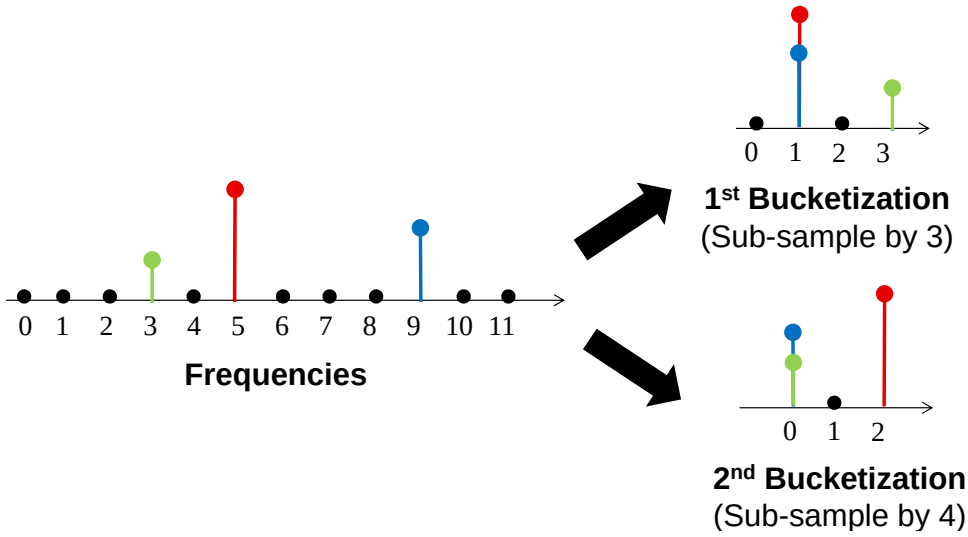


Figure 1-2: **Resolving collisions with Co-prime Aliasing:** Using 2 co-prime aliasing filters, we ensure the frequencies that collide in one filter will not collide in the second. For example, frequencies 5 and 9 collide in the first filter. But frequency 5 dies not collide in the second which allows us to estimate it and subtract it.

to estimate them correctly. Hence, to recover the full frequency spectrum, we need to resolve the collisions.

To resolve collision, we need to repeat the *frequency bucketization* in a manner that ensures that the same non-zero frequencies do not collide with each other every time. The manner in which we achieve this depends on the type of filter used for bucketization. For example, with the aliasing filters described above, we can bucketize the spectrum multiple times using aliasing filters with co-prime sampling rates. This changes the hashing function from $h(f) = f \mod n/p$ to $h'(f) = f \mod n/p'$ where p and p' are co-prime. Co-prime aliasing filters guarantee that any two frequencies that collide in one bucketization will not collide in the other bucketization. To better understand this point, consider the example in Figure 1-2. The first time we bucketize, we use an aliasing filter that sub-samples the time signal by a factor of 3. In this case, the two frequencies labeled in red and blue collide in a bucket whereas the frequency labeled in green does not collide, as shown in the figure. The second time we bucketize, we use an aliasing filter that sub-samples by 4. This time the blue and green frequencies collide whereas the red frequency does not collide. Now we can resolve collisions by iterating between the two bucketizations. For example, we can estimate the green frequency from the first bucketization, where it does not collide.[2] We subtract the green frequency from the colliding bucket in the second bucketization to obtain the blue frequency. We then go back to the first bucketization and subtract the blue frequency from the bucket where it collides to obtain the red frequency.

Iterating between the different bucketizations by estimating frequencies from buckets where

[2]In Chapter 5, we will present techniques to detect collisions. However, accurately detecting collision is not always necessary. Since $\widehat{\mathbf{x}}$ is sparse, the number of collisions will be very small and errors caused by assuming a non-zero frequency is isolated when it is in a collision can be corrected in subsequent iterations of the algorithm.

they do not collide and subtracting them from buckets where they do collide, ensures that each non-zero frequency will be isolated in its own bucket during some iteration of the algorithm. This allows us to estimate each non-zero frequency correctly. Thus, at the end of the of the collision resolution step, we have recovered all non-zero frequencies and hence have successfully computed the Fourier transform of the signal.

1.1.3 Algorithmic Techniques

The previous section established a general framework for computing the Sparse Fourier Transform and gave one example of a technique that can be used in each step of this framework. In this section, we describe a more comprehensive list of techniques that are used by different Sparse Fourier Transform algorithms.

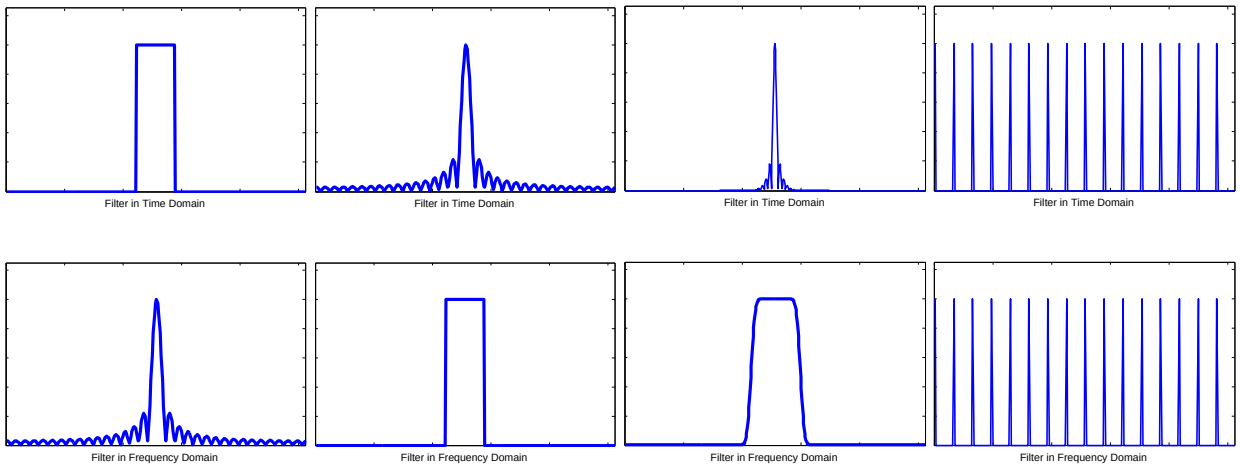
1. Frequency Bucketization Techniques:

As described earlier bucketization is done using filters. The choice of the filter can severely affect the running time of a Sparse Fourier Transform algorithm. Ideally, we would like a filter that uses a small number of input time samples to hash the frequency coefficients into buckets. For example, the rectangular or boxcar filter shown in Figure 1-3(a), uses only B time samples to hash the frequency coefficients into B buckets which is ideal in time domain. However, in the frequency domain, it is equal to the sinc function,[3] which decays polynomially as shown in Figure 1-3(a). This polynomial decay means that the frequency coefficients "leak" between buckets, *i.e.*, the value of the bucket is no longer the sum of n/B coefficients that hash to the bucket. It is a weighted sum of all the n frequency coefficients. Hence, a non-zero frequency coefficient can never be isolated in a bucket and estimated correctly. One the other hand, a rectangular filter is ideal in the frequency domain since it has no leakage as shown in Figure 1-3(b). However, it is sinc function in the time domain and hence requires using all n input samples which take at least $\Omega(n)$ time to process.

In this thesis, we identify several efficient filters that use a small number of samples in time domain and have minimal or no leakage in the frequency domain and as a result can be used to perform fast bucketization.

- **Flat Window Filter:** This filter looks very similar to a rectangle or box in the frequency domain while still using a small number of time samples. An example of such filter is a Gaussian function multiplied by a sinc function in the time domain which is shown in Figure 1-3(c). Since the Gaussian function decays exponentially fast both in time and frequency, the leakage between buckets in this filter is negligible and can be ignored. Similarly, the filter is concentrated in time and hence uses only a small number of time samples. The resulting hash function of such filter can be written as $h(f) = \lceil f/(n/B) \rceil$. Gaussian is only one example of such functions. One can potentially use a Dolph-Chebyshev or a Kaiser-Bessel function as we describe in more detail in Chapter 2.

[3] The sinc function is defined as: $sinc(x) = sin(x)/x$.



(a) Rectangular (b) Sinc (c) Gaussian/Rectangular (d) Aliasing

Figure 1-3: **Filters used for Frequency Bucketization** shown in the time (upper row) and the frequency (lower row) domain.

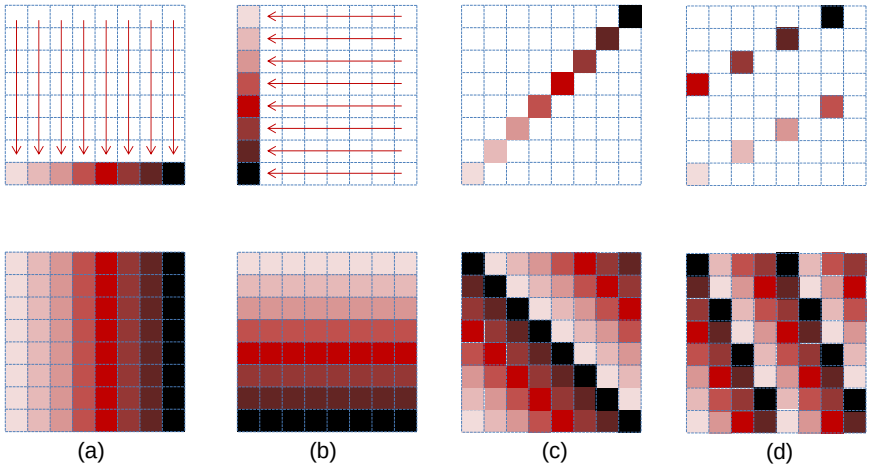


Figure 1-4: **Fourier Projection Filters:** The top row of figures shows the sampled lines of the time signal and the bottom row of figures shows how the spectrum is projected. Frequencies of the same color are projected onto the same point. (a) row projection (b) column projection (c) diagonal projection (d) line with slope = 2.

- **Aliasing Filter:** We presented this filter in the previous section. It is simply a spike-train of period p in time domain since it sub-samples the time domain signal by a factor of p as shown in Figure 1-3(d). It is also a spike-train in the frequency domain since it sums up frequency coefficients that are equally spaced by n/p. The aliasing filter is ideal both in time and in frequency since it only uses B time samples and has zero leakage between buckets. Unfortunately, we will later show that the aliasing filter does not lend itself to powerful randomization techniques and as a result, can only be shown to work for average case input signals as opposed to worst case.

- **Fourier Projection Filter:** This filter is a generalization of the aliasing filter to higher dimensions. It is a direct result of the *Fourier Slice Projection Theorem* which states that taking the Fourier transform of samples along a slice (e.g. a 1D line in 2D time signal or a 2D plane in a 3D signal) results in the orthogonal projection of the frequency spectrum onto the corresponding slice in the Fourier domain. For example, if we sample a row in a 2D time domain signal and then take its 1D Fourier transform, each output point of the Fourier transform will be the sum of the coefficients along one column as shown in Figure 1-4 (a). Alternatively, if we sample a column, each output point will be the sum of the coefficients along one row as shown in Figure 1-4(b). This also holds for any discrete line as shown in Figure 1-4(c,d). Thus, if we sample a line with slope equal to 1 or 2, we get a projection in the frequency domain along lines with slopes equal to -1 or $-1/2$. Note that discrete lines wrap around and hence can generate non-uniform sampling as shown in Figure 1-4(d).

2. Frequency Estimation Techniques:

Recall that the goal of the frequency estimation step is to compute the positions f and values $\widehat{\mathbf{x}}(f)$ of the non-zero frequency coefficients that have been hashed to non-empty buckets. In what follows, we will present the techniques used by the Sparse Fourier Transform algorithms to perform frequency estimation.

- **Time-Shift/Phase-Rotation Approach:** In this approach, which we have previously described in Section 1.1.2, we repeat the bucketization after shifting the time signal $\mathbf{x}$ by a circular shift of τ samples. For buckets with a single isolated non-zero frequency coefficient, this results in a phase rotation $\Delta\phi$ of the complex value of the coefficient. $\Delta\phi = 2\pi f\tau/n$ which we can use to compute the frequency position f. Since the frequency is isolated, its value can be immediately computed from the value of the bucket as described in the previous section. This is an extremely efficient approach since it requires constant $O(1)$ time to estimate each non-zero frequency. For noisy signals, we repeat this process for multiple different time shifts to average the noise. The details of how we average the noise to ensure robust recovery can be found in Chapter 4.

- **Voting Approach:** This is a non-iterative streaming approach where we repeat the bucketization few times while changing the hashing function. For each bucketization, the non-empty buckets vote for all the frequency coefficients that hash into those buckets. Non-zero frequency coefficients will get a vote from every bucketization with high probability. On the other hand, zero and negligible frequencies are unlikely to get many votes. Hence, after repeating the bucketization and this voting procedure a few times, the k non-zero frequency coefficients will have the largest number of votes and can be identified. In Chapter 3, we will show that this approach is more resilient to signal noise. However, this comes at the cost of increased runtime which we prove to be $O(\log n\sqrt{nk\log n})$.

3. Collision Resolution Techniques:

Collision resolution is achieved by repeating the bucketization in a manner that changes the way frequency coefficients hash into buckets so that the same coefficients do not continue to collide.

We can randomize the hashing function if we can perform a random permutation of the positions of the coefficients in $\hat{\mathbf{x}}$ before repeating the bucketization. This can be achieved by rearranging the indices of the input time signal $\mathbf{x}$ and rescaling it in the following manner:[4]

$$\mathbf{x}'(t) = \mathbf{x}(\sigma t \mod n) \cdot e^{-j2\pi\beta t/n} \tag{1.5}$$

where σ is a random integer invertible modulo n and β is a random integer. This results in a random linear mapping of the positions of the frequency coefficients: $f \rightarrow \sigma^{-1}f + \beta \mod n$. The proof of this can be found in Chapter 2. While this randomization is a very powerful collision resolution technique, it only works with the flat window filter and does not work with aliasing filters as we will show in Chapter 3. To resolve collisions using aliasing filters, we need to use co-prime subsampling *i.e.* subsample the signal using different co-prime subsampling rates as explained in the previous section.

1.1.4 Algorithmic Results

In this section, we will present the theoretical results of the Sparse Fourier Transform algorithms that we developed. All the algorithms follow the framework described in Section 1.1.2. However, they use different techniques from Section 1.1.3 and as a result achieve different runtime and sampling complexities as well as different guarantees. Most of the theoretical algorithms presented in this thesis are randomized and succeed with a large constant probability. Table 1.2 summarizes the theoretical Sparse Fourier Transform algorithms presented in this thesis along with their complexity results, guarantees and techniques used.

1.2 Applications of the Sparse Fourier Transform

The second half of this dissertation focuses on developing software and hardware systems that harness the Sparse Fourier Transform to solve practical problems. The thesis presents the design and implementation of six new systems that solve challenges in the areas of wireless networks, mobile systems, computer graphics, medical imaging, biochemistry, and digital circuits. All six systems are prototyped and evaluated in accordance with the standards of each application's field.

Adapting the Sparse Fourier transform to a particular application requires a careful design and deep knowledge of the application domain. The Sparse Fourier transform is a framework of algorithms and techniques for analyzing sparse signals. It is not a single algorithm that can be plugged directly into an application. To apply it to a problem, one has to deeply understand the structure of the sparsity in the application of interest, and customize the framework to the observed sparsity. More generally, real applications add new constraints that are application-dependent, and can deviate from our mathematical abstraction. Below we highlight some of the common themes that arise in mapping the Sparse Fourier Transform to practical systems.

[4]Note that only time samples that will be used by the bucketization filters need to be computed.

Chapter	Algorithm	Runtime / Sampling Complexity	Sparsity Range	Model & Guarantee	Analysis	Bucketization & Estimation Techniques
3	SFT 1.0[*]	$O(\log n\sqrt{nk\log n})$	$O(n/\log n)$	Approximate (ℓ_∞/ℓ_2)	Worst Case	• Flat Window • Voting
	SFT 2.0	$O(\log n\sqrt[3]{nk^2\log n})$	$O(n/\sqrt{\log n})$	Approximate (Heuristic)	Worst Case	• Flat Window & Aliasing • Voting
4	SFT 3.0	$O(k\log n)$	$O(n)$	Exact	Worst Case (Time Optimal)	• Flat Window • Phase Rotation
	SFT 4.0	$O(k\log n\log(n/k))$	$O(n)$	Approximate (ℓ_2/ℓ_2)	Worst Case	• Flat Window • Phase Rotation
	SFT 4.1	$O(k\log^2 n\log(n/k))$	$O(n/\log n\log\log n)$	Approximate (ℓ_2/ℓ_2)	Worst Case, 2D	• Flat Window • Phase Rotation
5	SFT 5.1	$O(k\log k)$ Time $O(k)$ Samples	$O(\sqrt{n})$	Exact	Average Case, 2D (Optimal)	• Projections & Aliasing • Phase Rotation
	SFT 6.0	$O(k\log^2 n)$ Time $O(k\log n)$ Samples	$\Theta(\sqrt{n})$	Approximate (ℓ_2/ℓ_2)	Average Case, 2D (Sample Optimal)	• Projections • Phase Rotation

Table 1.2: **Theoretical Sparse Fourier Transform Algorithms**

[*] The sampling complexity of algorithms SFT 1.0–4.1 is the same as their runtime complexity.

- **Structure of Sparsity:** In most applications, the occupied frequency coefficients are not randomly distributed; they follow a specific structure. For example, as mentioned earlier, in wireless communication, the occupied frequencies in the wireless spectrum are clustered. In computational photography, the occupied frequencies are more likely to be present in part of the Fourier spectrum. On the other hand, in an application like GPS, the occupied coefficient can be anywhere. Understanding the structure of sparsity in each application allows us to design a system that leverages it to achieve the best performance gains.

- **Level of Sparsity:** A main difference between theory and practice is that the Sparse Fourier Transform algorithms operate in the discrete domain. In real and natural signals, however, the frequency coefficients do not necessarily lie on discrete grid points. Simply rounding off the locations of the coefficients to the nearest grid points can create bad artifacts which significantly reduce the sparsity of the signal and damage the quality of the results. This problem occurs in application like medical imaging and light-field photography. Hence, we have to design systems that can sparsify the signals and recover their original sparsity in the continuous domain in order to address the mismatch between the theoretical Sparse Fourier Transform framework and the scenarios in which it is applied.

- **System Requirements:** Different applications have different goals. For example, in medical imaging, the acquisition cost is high since it requires the patient to spend more time in the MRI machine while processing the captured data afterwards is not a problem. Hence, the goal would be to minimize the number of input samples that need to be collected even if it requires additional processing time. On the other hand, in applications like GPS, collecting the samples is very cheap. However, processing them consumes a lot of power. Hence, the goal would be to minimize computational load even if all the input samples are used. Finally, in applications like spectrum acquisition and NMR spectroscopy, the goal would be to minimize both the runtime and the sampling. Hence, understanding the system is essential for adapting the Sparse Fourier Transform techniques to satisfy the requirements of each application.

- **System Architecture:** In some applications, the applicability of the Sparse Fourier Transform to a system architecture might not even be apparent. For example, the Fourier transform of the GPS signal is not sparse and hence applying the Sparse Fourier Transform directly to GPS is not feasible. However, we observed that the GPS receiver correlates its signal with a special code transmitted by the satellite, and the output of the correlation is sparse because it spikes only when the code and the signal are aligned. Hence, we have to map this indirect form of sparsity to the Sparse Fourier Transform framework. We also need to ensure that the gains of the Sparse Fourier Transform are not bottlenecked by other components in the system. Thus, careful system design is essential for propagating these gains along the system pipeline and improving the overall system performance.

- **Signal to Noise Ratio:** In practice, the gains of the Sparse Fourier Transform are constraint by the noise level in the system. It is essential to perform sampling and processing that would be sufficient to bring the signal above the noise floor of the system. For example, although the sparsity that appears in a GPS system is extremely high ($\approx$0.025%), GPS signals are -30 dB

to -20 dB below the noise floor which requires additional computation that upper bounds the performance gains. Thus, understanding the noise level and structure is essential in designing any system that uses the Sparse Fourier Transform.

The following subsections summarize the developed systems in this thesis and how they benefit from the Sparse Fourier Transform.

1.2.1 Spectrum Sensing and Decoding

The ever-increasing demand for wireless connectivity has led to a spectrum shortage which prompted the FCC to release multiple new bands for dynamic spectrum sharing. This is part of a bigger vision to dynamically share much of the currently under-utilized spectrum, creating GHz-wide spectrum superhighways that can be shared by different types of wireless services. However, a major technical obstacle precluding this vision is the need for receivers that can capture and sense GHz of spectrum in real-time in order to quickly identify unoccupied bands. Such receivers consume a lot of power because they need to sample the wireless signal at GigaSample/s.

To overcome this challenge, we leverage the fact that the wireless spectrum is sparsely utilized and use the Sparse Fourier Transform to build a receiver that can capture and recover GHz of spectrum in real-time, while sampling only at MegaSample/s. We use the aliasing filters and phase rotation techniques described in Section 1.1.3 to build the entire receiver using only cheap, low power hardware similar to what is used today by WiFi and LTE in every mobile phone. We implement our design using three software radios, each sampling at 50 MegaSample/s, and produce a device that captures 0.9 GHz – *i.e.,* $6\times$ larger digital bandwidth than the three software radios combined. The details of the system design, implementation and evaluation can be found in Chapter 7.

1.2.2 GPS Receivers

GPS is one of the most widely used wireless systems. In order to calculate its position, a GPS receiver has to lock on the satellite signals by aligning the received signal with each satellite's code. This process requires heavy computation, which consumes both time and power. As a result, running GPS on a phone can quickly drain the battery.

We introduced a new GPS receiver that minimizes the required computation to lock on the satellite's signal hence reducing localization delay and power consumption. Specifically, We observed that GPS synchronization can be reduced into a sparse computation problem by leveraging the fact that only the correct alignment between the received GPS signal and the satellite code causes their correlation to spike. We built on this insight to develop a GPS receiver that exploits the Sparse Fourier Transform to quickly lock on the satellite signal and identify its location. We prototyped this design using software radios. Our empirical results with real satellite signals demonstrate that the new GPS receiver reduces the computational overhead by $2\times$ to $6\times$, which translates into significant reduction in localization delay and power consumption. Chapter 8 presents the analysis and evaluation of the design in detail.

1.2.3 Light Field Photography

Light field photography is an active area in graphics where a 2D array of cameras or lenslets is used to capture the 4D light field of a scene.[5] This enables a user to extract the 3D depth, refocus the scene to any plane, and change the angle from which he views the scene. This is essential for VR (Virtual Reality) systems as well as post processing of images and videos. Capturing light fields, however, is costly since it requires many cameras or lenslets to sample the scene from different viewpoints.

Thus, our goal is to reduce the cost of light field capture by using only few of the cameras in the 2D array and reconstructing the images from the missing cameras . To do this, we leverage the fact that the 4D Fourier transform of a light field is sparse and we use the Sparse Fourier Transform to sub-sample the input and reduce the number of cameras. Once we have computed the Fourier transform, we can invert it back to recover the images from the missing cameras which were not sampled and hence recover the full 2D array. However, as explained earlier, natural signals like light fields are not very sparse in the discrete domain. To address this issue, we developed a light field reconstruction system that optimizes for sparsity in the continuous Fourier domain. This improves the quality of reconstruction while reducing the required number of cameras by $6\times$ to $10\times$. The light field reconstruction algorithm along with the reconstruction results can be found in Chapter 9.

1.2.4 Magnetic Resonance Spectroscopy (MRS)

One of the next frontiers in MRI is Magnetic Resonance Spectroscopy (MRS). MRS enables zooming in and detecting the biochemical content of each voxel in the brain, which can be used to discover disease biomarkers that allow early detection of cancer, Autism, and Alzheimer. MRS tests, however, take prohibitively long time, requiring the patient to stay in the MRI machine for more than two hours. MRS images also suffer from a lot of clutter and artifacts that can mask some disease biomarkers. These two challenges have been a major barrier against adopting these tests in clinical diagnosis. To overcome this barrier, We demonstrated that processing MRS data using the Sparse Fourier Transform enhances image quality by suppressing artifacts and reduces the time the patient has to spend in the machine by $3\times$ (e.g. from 2 hrs to 40 mins). The details of our MRS algorithm, experiments and results can be found in Chapter 10.

1.2.5 Nuclear Magnetic Resonance (NMR)

NMR is a technique that provides the detailed structural properties of chemical compounds, providing the 3D structure of complex proteins and nucleic acids. However, collecting NMR measurements is a very time consuming and costly process that can take from several days up to weeks. This prevents researchers from running high-dimensional NMR experiments which are needed for analyzing more complex protein structures. NMR uses spectral analysis to find the resonance frequencies that correspond to the coupling between different atoms. NMR spectra are sparse. Hence, us-

[5] The four dimensions of a light field correspond to the 2D pixels in each image captured by a camera in the 2D camera array.

ing the Sparse Fourier Transform, we show how to generate the NMR spectra by sub-sampling the NMR measurements. We customized the Sparse Fourier Transform for multi-dimensional NMR and showed that it can reduce the time of an NMR experiment by $16\times$. Chapter 11 describes the Sparse Fourier Transform techniques used for processing our NMR experiments along with our recovery results.

1.2.6 The Sparse Fourier Transform Chip

Traditionally, hardware implementations of FFT have been limited in size to few thousands of points. This is because large FFTs require a huge I/O bandwidth, consume a lot of power, and occupy a large silicon area. The Sparse Fourier Transform naturally addresses these issues due to its low computational and memory cost, enabling very large Fourier transforms. We built the largest Fourier transform VLSI chip to date with nearly a million point Fourier transform while consuming $40\times$ less power than prior FFT VLSI implementations. The hardware architecture and benchmarking of the fabricated chip can be found in Appendix G.

1.3 Thesis Roadmap

This thesis is divided into two parts. Part I describes the theoretical foundations of the Sparse Fourier Transform. It presents the Sparse Fourier Transform algorithms in detail and provides the analysis and proofs of the guarantees of these algorithms. Chapter 2 presents the notation and basic definitions that will be used in this part of the thesis. Chapters 3 and 4 focus on reducing the runtime complexity of the Sparse Fourier Transform algorithms while Chapter 5 focuses on optimizing the sample complexity. Finally, in Chapter 6, we present numerical simulations to evaluate the performance of the Sparse Fourier Transform.

Part II describes the applications and systems designed using the Sparse Fourier Transform. Chapter 7 describes the design and implementation of a wireless receiver that can capture GHz of spectrum in realtime. Chapter 8 presents a GPS receiver design with lower computational overhead. Chapter 9 describes a light field photography reconstruction algorithm that achieves high quality image recovery. Chapter 10 shows how the Sparse Fourier Transform can be used to reduce the time a patient spends in an MRI machine and generate clearer images. Chapter 11 presents the application of the Sparse Fourier Transform to Nuclear Magnetic Resonance in biochemistry.

Finally, in Chapter 12, we conclude and discuss the future work.

Part I

Theory of the Sparse Fourier Transform

Chapter 2

Preliminaries

2.1 Notation

We use $\omega = e^{-2\pi \mathbf{i}/n}$ as the n-th root of unity and $\omega' = e^{-2\pi \mathbf{i}/\sqrt{n}}$ as the $\sqrt{n}$-th root of unity. For any complex number a, we use $\phi(a) \in [0, 2\pi]$ to denote the *phase* of a. For a complex number a and a real positive number b, the expression $a \pm b$ denotes a complex number a' such that $|a - a'| \leq b$.

For a vector $x \in \mathbb{C}^n$, its support is denoted by $\mathrm{supp}(x) \subset [n]$. We use $\|x\|_0$ to denote $|\mathrm{supp}(x)|$, the number of non-zero coordinates of x. Its Fourier spectrum is denoted by $\widehat{x}$, with

$$\widehat{x}_i = \frac{1}{\sqrt{n}} \sum_{j \in [n]} \omega^{ij} x_j.$$

For a vector of length n, indices should be interpreted modulo n, so $x_{-i} = x_{n-i}$. This allows us to define *convolution*

$$(x * y)_i = \sum_{j \in [n]} x_j y_{i-j}$$

and the *coordinate-wise product* $(x \cdot y)_i = x_i y_i$, so $\widehat{x \cdot y} = \widehat{x} * \widehat{y}$.

We use $[n]$ to denote the set $\{1, \ldots, n\}$. All operations on indices in are taken modulo n. Therefore we might refer to an n-dimensional vector as having coordinates $\{0, 1, \ldots, n-1\}$ or $\{0, 1, \ldots, n/2, -n/2+1, \ldots, -1\}$ interchangeably. When $i \in \mathbb{Z}$ is an index into an n-dimensional vector, sometimes we use $|i|$ to denote $\min_{j \equiv i \pmod n} |j|$. Finally, we assume that n is an integer power of 2.

For the case of 2D Fourier transforms which will appear in Chapter 5, we assume that $\sqrt{n}$ is a power of 2. We use $[m] \times [m] = [m]^2$ to denote the $m \times m$ grid $\{(i, j) : i \in [m], j \in [m]\}$. For a 2D matrix $x \in \mathbb{C}^{\sqrt{n} \times \sqrt{n}}$, its support is denoted by $\mathrm{supp}(x) \subseteq [\sqrt{n}] \times [\sqrt{n}]$. We also use $\|x\|_0$ to denote $|\mathrm{supp}(x)|$. Its 2D Fourier spectrum is denoted by $\widehat{x}$, with

$$\widehat{x}_{i,j} = \frac{1}{\sqrt{n}} \sum_{l \in [\sqrt{n}]} \sum_{m \in [\sqrt{n}]} (\omega')^{il+jm} x_{l,m}$$

Finally, if y is in frequency-domain, its inverse is denoted by $\check{y}$.

2.2 Basics

2.2.1 Window Functions

In digital signal processing [138] one defines *window functions* in the following manner:

Definition 2.2.1. *We define a* (ϵ, δ, w) standard window function *to be a symmetric vector* $F \in \mathbb{R}^n$ *with* $\text{supp}(F) \subseteq [-w/2, w/2]$ *such that* $\hat{F}_0 = 1$, $\hat{F}_i > 0$ *for all* $i \in [-\epsilon n, \epsilon n]$, *and* $|\hat{F}_i| < \delta$ *for all* $i \notin [-\epsilon n, \epsilon n]$.

Claim 2.2.2. *For any* ϵ *and* δ, *there exists an* $(\epsilon, \delta, O(\frac{1}{\epsilon}\log(1/\delta)))$ *standard window function.*

Proof. This is a well known fact [160]. For example, for any ϵ and δ, one can obtain a standard window by taking a Gaussian with standard deviation $\Theta(\sqrt{\log(1/\delta)}/\epsilon)$ and truncating it at $w = O(\frac{1}{\epsilon}\log(1/\delta)))$. The Dolph-Chebyshev window function also has the claimed property but with minimal big-Oh constant [160] (in particular, half the constant of the truncated Gaussian). □

The above definition shows that a standard window function acts like a filter, allowing us to focus on a subset of the Fourier coefficients. Ideally, however, we would like the pass region of our filter to be as flat as possible.

Definition 2.2.3. *We define a* $(\epsilon, \epsilon', \delta, w)$ flat window function *to be a symmetric vector* $F \in \mathbb{R}^n$ *with* $\text{supp}(F) \subseteq [-w/2, w/2]$ *such that* $\hat{F}_i \in [1-\delta, 1+\delta]$ *for all* $i \in [-\epsilon' n, \epsilon' n]$ *and* $|\hat{F}_i| < \delta$ *for all* $i \notin [-\epsilon n, \epsilon n]$.

A flat window function (like the one in Figure 2-1) can be obtained from a standard window function by convolving it with a "box car" window function, i.e., an interval. Specifically, we have the following.

Claim 2.2.4. *For any* ϵ, ϵ', *and* δ *with* $\epsilon' < \epsilon$, *there exists an* $(\epsilon, \epsilon', \delta, O(\frac{1}{\epsilon-\epsilon'}\log\frac{n}{\delta}))$ *flat window function.*

Note that in our applications we have $\delta < 1/n^{O(1)}$ and $\epsilon = 2\epsilon'$. Thus the window lengths w of the flat window function and the standard window function are the same up to a constant factor.

Proof. Let $f = (\epsilon - \epsilon')/2$, and let F be an $(f, \frac{\delta}{(\epsilon'+\epsilon)n}, w)$ standard window function with minimal $w = O(\frac{2}{\epsilon-\epsilon'}\log\frac{(\epsilon+\epsilon')n}{\delta})$. We can assume $\epsilon, \epsilon' > 1/(2n)$ (because $[-\epsilon n, \epsilon n] = \{0\}$ otherwise), so $\log\frac{(\epsilon+\epsilon')n}{\delta} = O(\log\frac{n}{\delta})$. Let $\hat{F}'$ be the sum of $1 + 2(\epsilon' + f)n$ adjacent copies of $\hat{F}$, normalized to $\hat{F}'_0 \approx 1$. That is, we define

$$\hat{F}'_i = \frac{\sum_{j=-(\epsilon'+f)n}^{(\epsilon'+f)n} \hat{F}_{i+j}}{\sum_{j=-fn}^{fn} \hat{F}_j}$$

so by the shift theorem, in the time domain

$$F'_a \propto F_a \sum_{j=-(\epsilon'+f)n}^{(\epsilon'+f)n} \omega^j.$$

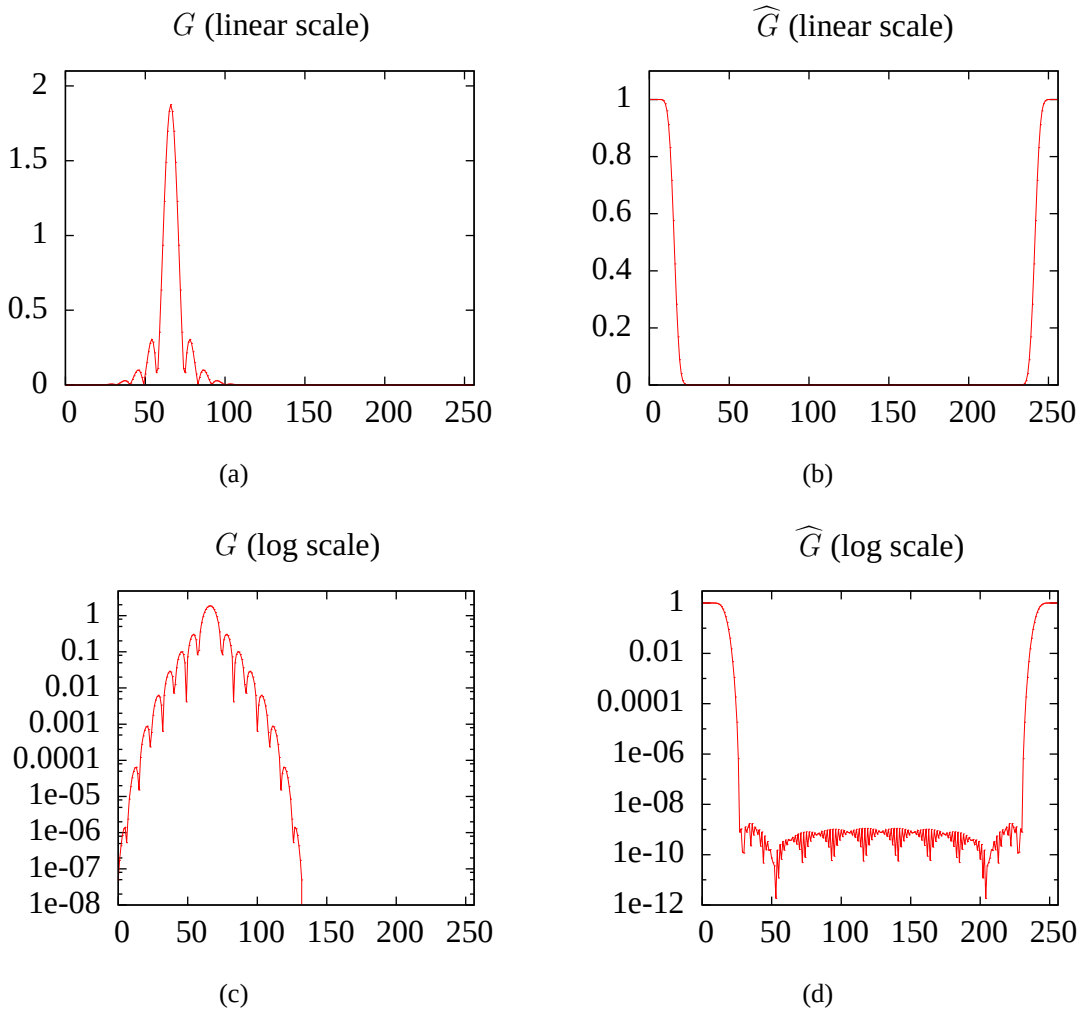


Figure 2-1: **An Example Flat Window Function for** $n = 256$**.** This is the sum of 31 adjacent $(1/22, 10^{-8}, 133)$ Dolph-Chebyshev window functions, giving a $(0.11, 0.06, 2 \times 10^{-9}, 133)$ flat window function (although our proof only guarantees a tolerance $\delta = 29 \times 10^{-8}$, the actual tolerance is better). The top row shows G and $\widehat{G}$ in a linear scale, and the bottom row shows the same with a log scale.

Since $\hat{F}_i > 0$ for $|i| \leq fn$, the normalization factor $\sum_{j=-fn}^{fn} \hat{F}_j$ is at least 1. For each $i \in [-\epsilon' n, \epsilon' n]$, the sum on top contains all the terms from the sum on bottom. The other $2\epsilon' n$ terms in the top sum have magnitude at most $\delta/((\epsilon'+\epsilon)n) = \delta/(2(\epsilon'+f)n)$, so $|\hat{F'}_i - 1| \leq 2\epsilon' n(\delta/(2(\epsilon'+f)n)) < \delta$. For $|i| > \epsilon n$, however, $\hat{F'}_i \leq 2(\epsilon'+f)n\delta/(2(\epsilon'+f)n) < \delta$. Thus F' is an $(\epsilon, \epsilon', \delta, w)$ flat window function, with the correct w. □

2.2.2 Permutation of Spectra

Following [59], we can permute the Fourier spectrum as follows by permuting the time domain:

Definition 2.2.5. *Suppose σ^{-1} exists mod n. We define the* permutation $P_{\sigma,a,b}$ *by*

$$(P_{\sigma,a,b}x)_i = x_{\sigma(i-a)}\omega^{\sigma bi}.$$

We also define $\pi_{\sigma,b}(i) = \sigma(i-b) \bmod n$.

Claim 2.2.6. $\widehat{P_{\sigma,a,b}x}_{\pi_{\sigma,b}(i)} = \hat{x}_i\omega^{a\sigma i}$.

Proof.

$$\begin{aligned}
\widehat{P_{\sigma,a,b}x}_{\sigma(i-b)} &= \frac{1}{\sqrt{n}} \sum_{j\in[n]} \omega^{\sigma(i-b)j}(P_{\sigma,a,b}x)_j \\
&= \frac{1}{\sqrt{n}} \sum_{j\in[n]} \omega^{\sigma(i-b)j} x_{\sigma(j-a)}\omega^{\sigma bj} \\
&= \omega^{a\sigma i}\frac{1}{\sqrt{n}} \sum_{j\in[n]} \omega^{i\sigma(j-a)} x_{\sigma(j-a)} \\
&= \hat{x}_i\omega^{a\sigma i}.
\end{aligned}$$

□

Lemma 2.2.7. *If $j \neq 0$, n is a power of two, and σ is a uniformly random odd number in $[n]$, then* $\Pr[\sigma j \in [-C, C]] \leq 4C/n$.

Proof. If $j = m2^l$ for some odd m, then the distribution of σj as σ varies is uniform over $m'2^l$ for all odd m'. There are thus $2 \cdot \text{round}(C/2^{l+1}) < 4C/2^{l+1}$ possible values in $[-C, C]$ out of $n/2^{l+1}$ such elements in the orbit, for a chance of at most $4C/n$. □

Note that for simplicity, we will only analyze our algorithm when n is a power of two. For general n, the analog of Lemma 2.2.7 would lose an $n/\varphi(n) = O(\log\log n)$ factor, where φ is Euler's totient function. This will correspondingly increase the running time of the algorithm on general n.

Claim 2.2.6 allows us to change the set of coefficients binned to a bucket by changing the permutation; Lemma 2.2.7 bounds the probability of non-zero coefficients falling into the same bucket.

2.2.3 Subsampled FFT

Suppose we have a vector $x \in \mathbb{C}^n$ and a parameter B dividing n, and would like to compute $\hat{y}_i = \hat{x}_{i(n/B)}$ for $i \in [B]$.

Claim 2.2.8. *$\hat{y}$ is the B-dimensional Fourier transform of $y_i = \sum_{j=0}^{n/B-1} x_{i+Bj}$.*
Therefore $\hat{y}$ can be computed in $O(|\mathrm{supp}(x)| + B \log B)$ time.

Proof.

$$\begin{aligned}\hat{x}_{i(n/B)} &= \sum_{j=0}^{n-1} x_j \omega^{ij(n/B)} = \sum_{a=0}^{B-1} \sum_{j=0}^{n/B-1} x_{Bj+a} \omega^{i(Bj+a)n/B} \\ &= \sum_{a=0}^{B-1} \sum_{j=0}^{n/B-1} x_{Bj+a} \omega^{ian/B} = \sum_{a=0}^{B-1} y_a \omega^{ian/B} = \hat{y}_i,\end{aligned}$$

□

2.2.4 2D Aliasing Filter

The aliasing filter presented in Section 1.1.2. The filter generalizes to 2 dimensions as follows:

Given a 2D matrix $x \in \mathbb{C}^{\sqrt{n} \times \sqrt{n}}$, $(\tau_r, \tau_c) \in [\sqrt{n}] \times [\sqrt{n}]$, and B_r, B_c that divide $\sqrt{n}$, then for all $(i, j) \in [B_r] \times [B_c]$ set

$$y_{i,j} = x_{i(\sqrt{n}/B_r)+\tau_r, j(\sqrt{n}/B_c)+\tau_c}$$

Then, compute the 2D DFT $\hat{y}$ of y. Observe that $\hat{y}$ is a folded version of $\hat{x}$:

$$\hat{y}_{i,j} = \sum_{l \in [\frac{\sqrt{n}}{B_r}]} \sum_{m \in [\frac{\sqrt{n}}{B_c}]} \hat{x}_{lB_r+i, mB_c+j} (\omega')^{-\tau_r(i+lB_r)-\tau_c(j+mB_c)}$$

Chapter 3

Simple and Practical Algorithm

3.1 Introduction

In this chapter, we propose a new sub-linear Sparse Fourier Transform algorithm over the complex field. The key feature of this algorithm is its simplicity: the algorithm has a simple structure, which leads to efficient runtime with low big-Oh constant. This was the first algorithm to outperform FFT in practice for a practical range of sparsity as we will show later in Chapter 6.

3.1.1 Results

We present an algorithm which has the runtime of

$$O\left(\log n\sqrt{nk\log n}\right)$$

where n is the size of the signal and k is the number of non-zero frequency coefficients. Thus, the algorithm is faster than FFT for k up to $O(n/\log n)$. In contrast, earlier algorithms required asymptotically smaller bounds on k. This asymptotic improvement is also reflected in empirical runtimes. For example, we show in Chapter 6 that for $n = 2^{22}$, this algorithm outperforms FFT for k up to about 2200, which is an order of magnitude higher than what was achieved by prior work.

The estimations provided by our algorithm satisfy the so-called ℓ_∞/ℓ_2 guarantee. Specifically, let y be the minimizer of $\|\hat{x} - y\|_2$. For a precision parameter $\delta = 1/n^{O(1)}$, and a constant $\epsilon > 0$, our (randomized) algorithm outputs $\hat{x}'$ such that:

$$\|\hat{x} - \hat{x}'\|_\infty^2 \leq \epsilon\|\hat{x} - y\|_2^2/k + \delta\|x\|_1^2 \tag{3.1}$$

with probability $1 - 1/n$. The additive term that depends on δ appears in all past algorithms [6, 7, 57, 59, 88, 116], although typically (with the exception of [90]) it is eliminated by assuming that all coordinates are integers in the range $\{-n^{O(1)} \ldots n^{O(1)}\}$. In this chapter, we keep the dependence on δ explicit.

The ℓ_∞/ℓ_2 guarantee of Equation (3.1) is *stronger* than the ℓ_2/ℓ_2 guarantee of Equation (1.2). In particular, the ℓ_∞/ℓ_2 guarantee with a constant approximation factor C implies the ℓ_2/ℓ_2 guar-

antee with a constant approximation factor C', if one sets all but the k largest entries in $\hat{x}'$ to 0.[1] Furthermore, instead of bounding only the collective error, the ℓ_∞/ℓ_2 guarantee ensures that every Fourier coefficient is well-approximated.

3.1.2 Techniques

Recall from Chapter 1 that the Sparse Fourier Transform algorithms work by binning[2] the Fourier coefficients into a small number of buckets. Since the signal is sparse in the frequency domain, each bucket is likely[3] to have only one large coefficient, which can then be located (to find its position) and estimated (to find its value). For the algorithm to be sublinear, the binning has to be done in sublinear time. Binning the Fourier coefficient is done using an n-dimensional filter vector G that is concentrated both in time and frequency, *i.e.*, G is zero except at a small *number* of time coordinates, and its Fourier transform $\widehat{G}$ is negligible except at a small *fraction* (about $1/k$) of the frequency coordinates (the "pass" region).

Prior work, uses different types of filters. Depending on the choice of the filter G, past algorithms can be classified as: iteration-based or interpolation-based.

Iteration-based algorithms use a filter that has a significant mass outside its pass region [57, 59, 116]. For example, the papers [57, 59] set G to the rectangular filter which was shown in Figure 1-3(a), in which case $\widehat{G}$ is the Dirichlet kernel[4], whose tail decays in an inverse linear fashion. Since the tail decays slowly, the Fourier coefficients binned to a particular bucket "leak" into other buckets. On the other hand, the paper [116] estimates the convolution in time domain via random sampling, which also leads to a large estimation error. To reduce these errors and obtain the ℓ_2/ℓ_2 guarantee, these algorithms have to perform multiple iterations, where each iteration estimates the largest Fourier coefficient (the one least impacted by leakage) and subtracts its contribution to the time signal. The iterative update of the time signal causes a large increase in runtime. The algorithms in [57, 116] perform this update by going through $O(k)$ iterations each of which updates at least $O(k)$ time samples, resulting in an $O(k^2)$ term in the runtime. The algorithm [59], introduced a "bulk sampling" algorithm that amortizes this process but it requires solving instances of a non-uniform Fourier transform, which is expensive in practice.

Interpolation-based algorithms are less common and limited to the design in [88]. This approach uses the aliasing filter presented in Chapter 1, which is a leakage-free filter that allows [88] to avoid the need for iteration. Recall that in this case, the filter G has $G_i = 1$ iff $i \bmod n/p = 0$ and $G_i = 0$ otherwise. The Fourier transform of this filter is a "spike train" with period p and hence this filter does not leak; it is equal to 1 on $1/p$ fraction of coordinates and is zero elsewhere. Unfortunately, however, such a filter requires that p divides n and the algorithm in [88] needs many different values of p. Since in general one cannot assume that n is divisible by all numbers p, the algorithm treats the signal as a continuous function and *interpolates* it at the required points. Interpolation introduces additional complexity and increases the exponents in the runtime.

[1] This fact was implicit in [35]. For an explicit statement and proof see [58], remarks after Theorem 2.

[2] Throughout this thesis, we will use the terms "Binning" and "Bucketization" interchangeably.

[3] One can randomize the positions of the frequencies by sampling the signal in time domain appropriately as we have shown in Section 2.2.2.

[4] The Dirichlet kernel is the discrete version of the sinc function.

Our Approach

The key feature of our algorithm is the use of a different type of filter. In the simplest case, we use a filter obtained by convolving a Gaussian function with a box-car function.[5] Because of this new filter, our algorithm does not need to either iterate or interpolate. Specifically, the frequency response of our filter $\widehat{G}$ is nearly flat inside the pass region and has an *exponential* tail outside it. This means that leakage from frequencies in other buckets is negligible, and hence, our algorithm need not iterate. Also, filtering can be performed using the existing input samples x_i, and hence our algorithm need not interpolate the signal at new points. Avoiding both iteration and interpolation is the key feature that makes this algorithm efficient.

Further, once a large coefficient is isolated in a bucket, one needs to identify its frequency. In contrast to past work which typically uses binary search for this task, we adopt an idea from [145] and tailor it to our problem. Specifically, we simply select the set of "large" bins which are likely to contain large coefficients, and directly estimate all frequencies in those bins. To balance the cost of the bin selection and estimation steps, we make the number of bins somewhat larger than the typical value of $O(k)$. Specifically, we use $B \approx \sqrt{nk}$, which leads to the stated runtime.[6]

3.2 Algorithm

We refer to our algorithm as SFT 1.0 and it is shown in Algorithm 3.2.1. A key element of this algorithm is the *inner loop*, which finds and estimates each "large" coefficient with constant probability. In Section 3.2.1 we describe the inner loop, and in Section 3.2.2 we show how to use it to construct the full algorithm.

3.2.1 Inner Loop

Let B be a parameter that divides n, to be determined later. Let G be a $(1/B, 1/(2B), \delta, w)$ flat window function described in Section 2.2.1 for some δ and $w = O(B \log \frac{n}{\delta})$. We will have $\delta \approx 1/n^c$, so one can think of it as negligibly small.

There are two versions of the inner loop: *location* loops and *estimation* loops. Location loops, described as the procedure LOCATIONINNERLOOP in Algorithm 3.2.1, are given a parameter d, and output a set $I \subset [n]$ of dkn/B coordinates that contains each large coefficient with "good" probability. Estimation loops, described as the procedure ESTIMATIONINNERLOOP in Algorithm 3.2.1, are given a set $I \subset [n]$ and estimate $\widehat{x}_I$ such that each coordinate is estimated well with "good" probability.

By Claim 2.2.8, we can compute $\hat{z}$ in $O(w + B \log B) = O(B \log \frac{n}{\delta})$ time. Location loops thus take $O(B \log \frac{n}{\delta} + dkn/B)$ time and estimation loops take $O(B \log \frac{n}{\delta} + |I|)$ time. Figure 3-1

[5] A more efficient filter can be obtained by replacing the Gaussian function with a Dolph-Chebyshev function. (See Figure 2-1 for an illustration.)

[6] Although it is plausible that one could combine our filters with the binary search technique of [59] and achieve an algorithm with a $O(k \log^c n)$ runtime, our preliminary analysis indicates that the resulting algorithm would be slower. Intuitively, observe that for $n = 2^{22}$ and $k = 2^{11}$, the values of $\sqrt{nk} = 2^{16.5} \approx 92681$ and $k \log_2 n = 45056$ are quite close to each other.

procedure LOCATIONINNERLOOP(x, k, B, d)
Choose σ, τ and b uniformly at random from $[n]$ such that σ is odd.
Compute $\widehat{z}_i = \widehat{y}_{in/B}$ for $j \in [B]$, where $y = G \cdot (P_{\sigma,\tau,b}x)$ ▷ FFT of $z_i = \sum_{j=0}^{\lceil w/B \rceil - 1} y_{i+jB}$
$J \leftarrow$ indices of dk largest coefficients in $\widehat{z}$.
$I = \{i \in [n] | h_{\sigma,b}(i) \in J\}$ where $h_{\sigma,b}(i) = \text{round}(\sigma(i-b)B/n)$ ▷ $h_{\sigma,b} : [n] \to [B]$
return I
procedure ESTIMATIONINNERLOOP(x, B, I)
Choose σ, τ and b uniformly at random from $[n]$ such that σ is odd.
Compute $\widehat{z}_i = \widehat{y}_{in/B}$ for $j \in [B]$, where $y = G \cdot (P_{\sigma,\tau,b}x)$
$\widehat{x}' \leftarrow 0$
for $i \in I$ **do**
$\widehat{x}'_i = \widehat{z}_{h_{\sigma,b}(i)} \omega^{-\tau\sigma i} / \widehat{G}_{o_{\sigma,b}(i)}$ where $o_{\sigma,b}(i) = \sigma(i-b) - h_{\sigma,b}(i)(n/B)$
▷ $o_{\sigma,b} : [n] \to [-n/(2B), n/(2B)]$
return $\widehat{x}'$
procedure NONITERATIVESPARSEFFT(x, k, B, L, d)
$\widehat{x}' \leftarrow 0$
for $r \in \{1, \cdots, L\}$ **do**
$I_r \leftarrow$ LOCATIONINNERLOOP(x, k, B, d).
$I = I_1 \cup \cdots \cup I_L$
for $i \in I$ **do**
$s_i = |\{r | i \in I_r\}|$
$I' = \{i \in I | s_i \geq L/2\}$
for $r \in \{1, \cdots, L\}$ **do**
$\widehat{x}^r_{I'} \leftarrow$ ESTIMATIONINNERLOOP(x, B, I')
for $i \in I'$ **do**
$\widehat{x}'_i = \text{median}(\{\widehat{x}^r_i\})$
return $\widehat{x}'$

3.2.1: SFT 1.0: Non-Iterative Sparse Fourier Transform for $k = o(n/\log n)$

illustrates the inner loop.

For estimation loops, we get the following guarantee:

Lemma 3.2.1. *Let S be the support of the largest k coefficients of $\hat{x}$, and $\hat{x}_{-S}$ contain the rest. Then for any $\epsilon \leq 1$,*

$$\Pr_{\sigma,\tau,b}\left[|\hat{x}'_i - \hat{x}_i|^2 \geq \frac{\epsilon}{k}\|\hat{x}_{-S}\|_2^2 + 3\delta^2\|\hat{x}\|_1^2\right] < O\left(\frac{k}{\epsilon B}\right).$$

Proof. The proof can be found in Appendix A.1. □

Furthermore, since $|\widehat{G}_{o_{\sigma,b}(i)}| \in [1-\delta, 1+\delta]$, $|\hat{z}_{h_{\sigma,b}(i)}|$ is a good estimate for $|\hat{x}_i|$—the division is mainly useful for fixing the phase. Therefore in location loops, we get the following guarantee:

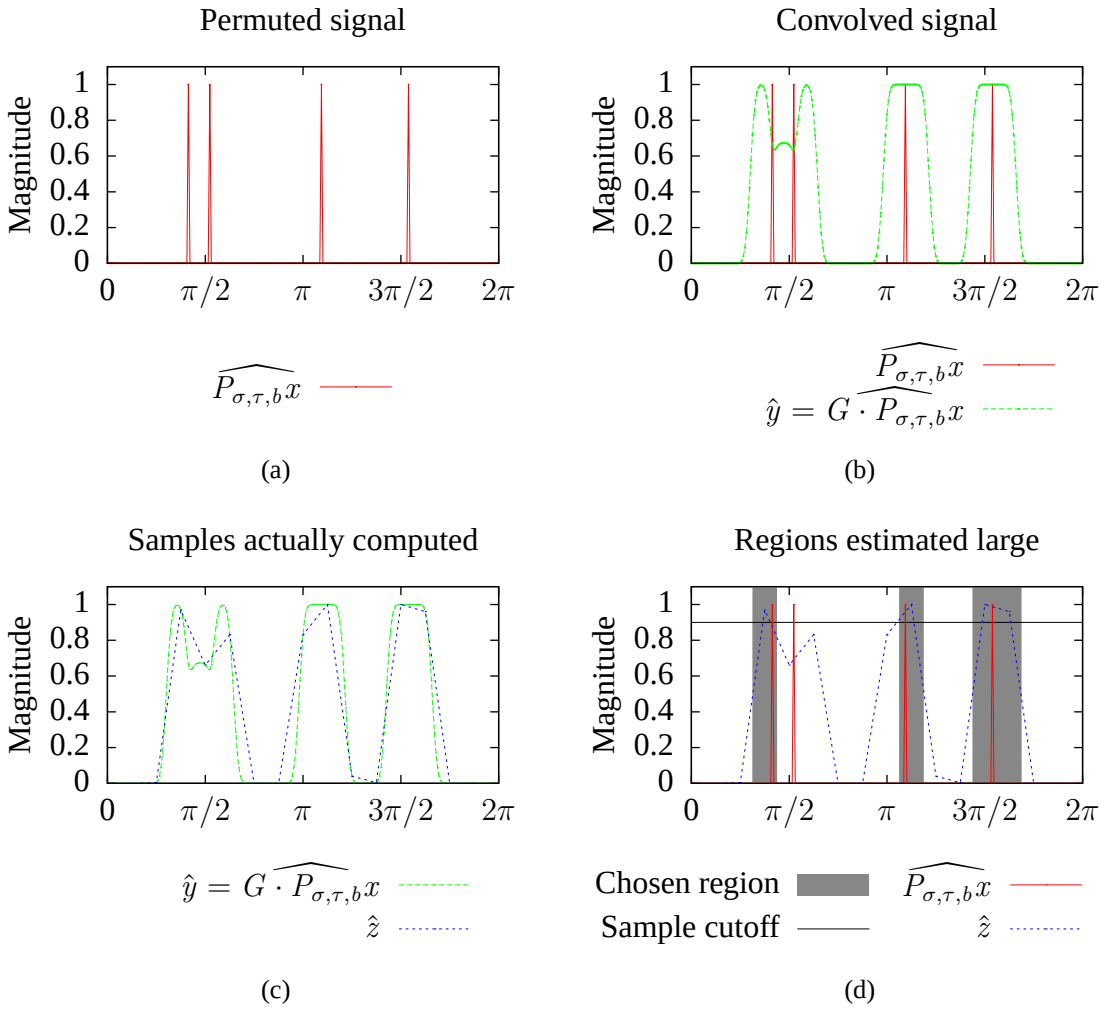


Figure 3-1: **Example Inner Loop of the Algorithm on Sparse Input.** This run has parameters $n = 256$, $k = 4$, G being the $(0.11, 0.06, 2 \times 10^{-9}, 133)$ flat window function in Figure 2-1, and selecting the top 4 of $B = 16$ samples. In part (a), the algorithm begins with time domain access to $P_{\sigma,\tau,b}x$ given by $(P_{\sigma,\tau,b}x)_i = x_{\sigma(i-\tau)}\omega^{\sigma bi}$, which permutes the spectrum of x by permuting the samples in the time domain. In part (b), the algorithm computes the time domain signal $y = G \cdot P_{\sigma,\tau,b}x$. The spectrum of y (pictured) is large around the large coordinates of $P_{\sigma,\tau,b}x$. The algorithm then computes $\hat{z}$, which is the rate B subsampling of $\hat{y}$ as pictured in part (c). During estimation loops, the algorithm estimates $\hat{x}_i$ based on the value of the nearest coordinate in $\hat{z}$, namely $\hat{z}_{h_{\sigma,b}(i)}$. During location loops (part (d)), the algorithm chooses J, the top dk (here, 4) coordinates of $\hat{z}$, and selects the elements of $[n]$ that are closest to those coordinates (the shaded region of the picture). It outputs the set I of preimages of those elements. In this example, the two coordinates on the left landed too close in the permutation and form a "hash collision". As a result, the algorithm misses the second from the left coordinate in its output. Our guarantee is that each large coordinate has a low probability of being missed if we select the top $O(k)$ samples.

Lemma 3.2.2. *Define* $E = \sqrt{\frac{\epsilon}{k}\|\hat{x}_{-S}\|_2^2 + 3\delta^2\|\hat{x}\|_1^2}$ *to be the error tolerated in Lemma 3.2.1. Then for any* $i \in [n]$ *with* $|\hat{x}_i| \geq 4E$*,*

$$\Pr[i \notin I] \leq O\left(\frac{k}{\epsilon B} + \frac{1}{\epsilon d}\right)$$

Proof. The proof can be found in Appendix A.2 □

3.2.2 Non-Iterative Sparse Fourier Transform

Our SFT 1.0 algorithm shown in Algorithm 3.2.1 is parameterized by ϵ and δ. It runs $L = O(\log n)$ iterations of the inner loop, with parameters $B = O(\sqrt{\frac{nk}{\epsilon \log(n/\delta)}})$ [7] and $d = O(1/\epsilon)$ as well as δ.

Lemma 3.2.3. *The algorithm runs in time* $O(\sqrt{\frac{nk\log(n/\delta)}{\epsilon}}\log n)$.

Proof. To analyze this algorithm, note that

$$|I'|\frac{L}{2} \leq \sum_i s_i = \sum_r |I_r| = Ldkn/B$$

or $|I'| \leq 2dkn/B$. Therefore the running time of both the location and estimation inner loops is $O(B\log\frac{n}{\delta} + dkn/B)$. Computing I' and computing the medians both take linear time, namely $O(Ldkn/B)$. Thus the total running time is $O(LB\log\frac{n}{\delta} + Ldkn/B)$. Plugging in $B = O(\sqrt{\frac{nk}{\epsilon\log(n/\delta)}})$ and $d = O(1/\epsilon)$, this running time is $O(\sqrt{\frac{nk\log(n/\delta)}{\epsilon}}\log n)$. We require $B = \Omega(k/\epsilon)$, however; for $k > \epsilon n/\log(n/\delta)$, this would cause the run time to be larger. But in this case, the predicted run time is $\Omega(n\log n)$ already, so the standard FFT is faster and we can fall back on it. □

Theorem 3.2.4. *Running the algorithm with parameters* $\epsilon, \delta < 1$ *gives* $\hat{x}'$ *satisfying*

$$\|\hat{x}' - \hat{x}\|_\infty^2 \leq \frac{\epsilon}{k}\|\hat{x}_{-S}\|_2^2 + \delta^2\|\hat{x}\|_1^2.$$

with probability $1 - 1/n$ *and running time* $O(\sqrt{\frac{nk\log(n/\delta)}{\epsilon}}\log n)$.

Proof. Define

$$E = \sqrt{\frac{\epsilon}{k}\|\hat{x}_{-S}\|_2^2 + 3\delta^2\|\hat{x}\|_1^2}$$

Lemma 3.2.2 says that in each location iteration r, for any i with $|\hat{x}_i| \geq 4E$,

$$\Pr[i \notin I^r] \leq O(\frac{k}{\epsilon B} + \frac{1}{\epsilon d}) \leq 1/4.$$

[7] Note that B is chosen in order to minimize the running time. For the purpose of correctness, it suffices that $B \geq ck/\epsilon$ for some constant c.

Thus $\mathbb{E}[s_i] \geq 3L/4$, and each iteration is an independent trial, so by a Chernoff bound the chance that $s_i < L/2$ is at most $1/2^{\Omega(L)} < 1/n^3$. Therefore by a union bound, with probability at least $1 - 1/n^2$, $i \in I'$ for all i with $|\hat{x}_i| \geq 4E$.

Next, Lemma 3.2.1 says that for each estimation iteration r and index i,

$$\Pr[|\hat{x}_i^r - \hat{x}_i| \geq E] \leq O(\frac{k}{\epsilon B}) < 1/4.$$

Therefore, with probability $1 - 2^{-\Omega(L)} \geq 1 - 1/n^3$, $|\hat{x}_i^r - \hat{x}_i| \leq E$ in at least $2L/3$ of the iterations.

Since real$(\hat{x}_i')$ is the median of the real$(\hat{x}_i^r)$, there must exist two r with $|\hat{x}_i^r - \hat{x}_i| \leq E$ but one real$(\hat{x}_i^r)$ above real$(\hat{x}_i')$ and one below. Hence one of these r has $|\text{real}(\hat{x}_i' - \hat{x}_i)| \leq |\text{real}(\hat{x}_i^r - \hat{x}_i)| \leq E$, and similarly for the imaginary axis. Then

$$|\hat{x}_i' - \hat{x}_i| \leq \sqrt{2}\max(|\text{real}(\hat{x}_i' - \hat{x}_i)|, |\text{imag}(\hat{x}_i' - \hat{x}_i)|) \leq \sqrt{2}E.$$

By a union bound over I', with probability at least $1 - 1/n^2$ we have $|\hat{x}_i' - \hat{x}_i| \leq \sqrt{2}E$ for all $i \in I'$. Since all $i \notin I'$ have $\hat{x}_i' = 0$ and $|\hat{x}_i| \leq 4E$ with probability $1 - 1/n^2$, with total probability $1 - 2/n^2$ we have

$$\|\hat{x}' - \hat{x}\|_\infty^2 \leq 16E^2 = \frac{16\epsilon}{k}\|\hat{x}_{-S}\|_2^2 + 48\delta^2\|\hat{x}\|_1^2.$$

Rescaling ϵ and δ gives our theorem. □

3.2.3 Extension

In this section, we present an extension of the SFT 1.0 algorithm which adds a heuristic to improve the runtime. We refer to this new algorithm as SFT 2.0 and it is shown in Algorithm 3.2.2. The idea of the heuristic is to apply the aliasing filter to restrict the locations of the large coefficients. The algorithm is parameterized by M that divides n. It performs the aliasing filter as a preprocessing step to SFT 1.0 and uses its output to restrict the frequency locations in the set I_r outputted by the location loops as shown in Algorithm 3.2.2.

Observe that $\hat{y}_i = \sum_{j=0}^{n/M} \hat{x}_{Mj+i}\omega^{-\tau(i+Mj)}$. Thus,

$$\mathop{\mathbb{E}}_{\tau}[|\hat{y}_i|^2] = \sum_{i \equiv j \bmod M} |\hat{x}_j|^2.$$

This means that the filter is very efficient, in that it has no leakage at all. Also, it is simple to compute. Unfortunately, it cannot be "randomized" using $P_{\sigma,\tau,b}$: after permuting by σ and b, any two colliding elements j and j' (i.e., such that $j = j' \mod M$) continue to collide. Nevertheless, if $\hat{x}_j$ is large, then $j \mod M$ is likely to lie in T—at least heuristically on random input.

SFT 2.0 assumes that all "large" coefficients j have $j \bmod M$ in T. That is, we restrict our sets I_r to contain only coordinates i with $i \mod M \in T$. We expect that $|I_r| \approx \frac{2k}{M}dkn/B$ rather than the previous dkn/B. This means that our heuristic will improve the runtime of the inner loops from $O(B\log(n/\delta) + dkn/B)$ to $O(B\log(n/\delta) + \frac{k}{M}dkn/B + M + dk)$, at the cost of $O(M \log M)$ preprocessing.

procedure NONITERATIVESPARSEFFT2(x, k, B, L, d, M)
    $\widehat{x}' \leftarrow 0$
    Choose $\tau \in [n]$ uniformly at random
    $y_i = x_{i(n/M)+\tau}$
    Compute $\widehat{y}$
    $T \leftarrow$ indices of the $2k$ largest elements of $\widehat{y}$ ▷ $T \subset [M]$
    for $r \in \{1, \cdots, L\}$ **do**
        $J \leftarrow$ LOCATIONINNERLOOP(x, k, B, d).
        $I_r = \{i \in J | i \mod M \in T\}$
    $I = I_1 \cup \cdots \cup I_L$
    for $i \in I$ **do**
        $s_i = |\{r | i \in I_r\}|$
    $I' = \{i \in I | s_i \geq L/2\}$
    for $r \in \{1, \cdots, L\}$ **do**
        $\widehat{x}'^r_{I'} \leftarrow$ ESTIMATIONINNERLOOP(x, B, I')
    for $i \in I'$ **do**
        $\widehat{x}'_i = \text{median}(\{\widehat{x}'^r_i\})$
    return $\widehat{x}'$

3.2.2: SFT 2.0: Non-Iterative Sparse Fourier Transform with Heuristic for $k = o(n/\sqrt{\log n})$

Note that on worst case input, SFT 2.0 may give incorrect output with high probability. For example, if $x_i = 1$ when i is a multiple of n/M and 0 otherwise, then $y = 0$ with probability $1 - M/n$ and the algorithm will output 0 over $\text{supp}(x)$. However, in practice the algorithm works for "sufficiently random" x.

Claim 3.2.5. *As a heuristic approximation, SFT 2.0 runs in* $O((k^2 n \log(n/\delta)/\epsilon)^{1/3} \log n)$ *as long as* $k \leq \epsilon^2 n \log(n/\delta)$.

Justification. First we will show that the heuristic improves the inner loop running time to $O(B \log(n/\delta) + \frac{k}{M} dkn/B + M + dk)$, then optimize the parameters M and B.

Heuristically, one would expect each of the I_r to be a $\frac{|T|}{M}$ factor smaller than if we did not require the elements to lie in T modulo M. Hence, we expect each of the I_r and I' to have size $\frac{|T|}{M} dkn/B = O(\frac{k}{M} dkn/B)$. Then in each location loop, rather than spending $O(dkn/B)$ time to list our output, we spend $O(\frac{k}{M} dkn/B)$ time—plus the time required to figure out where to start listing coordinates from each of the dk chosen elements J of $\hat{z}$. We do this by sorting J and $\{\sigma i \mid i \in T\}$ (mod M), then scanning through the elements. It takes $O(M + dk)$ time to sort $O(dk)$ elements in $[M]$, so the total runtime of each location loop is $O(B \log(n/\delta) + \frac{k}{M} dkn/B + M + dk)$. The estimation loops are even faster, since they benefit from $|I'|$ being smaller but avoid the $M + dk$ penalty.

The full algorithm does $O(M \log M)$ preprocessing and runs the inner loop $L = O(\log n)$ times with $d = O(1/\epsilon)$. Therefore, given parameters B and M, the algorithm takes $O(M \log M +$

$B \log \frac{n}{\delta} \log n + \frac{k}{M} \frac{kn}{\epsilon B} \log n + M \log n + \frac{k}{\epsilon} \log n)$ time. Optimizing over B, we take

$$O(M \log n + k\sqrt{\frac{n}{M\epsilon} \log(n/\delta)} \log n + \frac{k}{\epsilon} \log n)$$

time. Then, optimizing over M, this becomes

$$O((k^2 n \log(n/\delta)/\epsilon)^{1/3} \log n + \frac{k}{\epsilon} \log n)$$

time. If $k < \epsilon^2 n \log(n/\delta)$, the first term dominates.

Note that this is an $(\frac{n \log(n/\delta)}{\epsilon k})^{1/6}$ factor smaller than the running time of SFT 1.0.

Chapter 4

Optimizing Runtime Complexity

4.1 Introduction

The algorithm presented in Chapter 3 was the first algorithm to outperform FFT in practice for reasonably sparse signals. However, it has a runtime of $O\left(\log n\sqrt{nk\log n}\right)$ which is polynomial in n and only outperforms FFT for k smaller than $\Theta(n/\log n)$.

4.1.1 Results

In this chapter, we address this limitation by presenting two new algorithms for the sparse Fourier transform. We show:

- An $O(k\log n)$-time algorithm for the exactly k-sparse case, and
- An $O(k\log n\log(n/k))$-time algorithm for the general case.

The key property of both algorithms is their ability to achieve $o(n\log n)$ time, and thus improve over the FFT, for *any* $k = o(n)$. These algorithms are the first known algorithms that satisfy this property. Moreover, if one assumes that FFT is optimal and hence the DFT cannot be computed in less than $O(n\log n)$ time, the algorithm for the exactly k-sparse case is *optimal*[1] as long as $k = n^{\Omega(1)}$. Under the same assumption, the result for the general case is at most one $\log\log n$ factor away from the optimal runtime for the case of "large" sparsity $k = n/\log^{O(1)} n$.

For the general case, given a signal x, the algorithm computes a *k-sparse approximation* $\widehat{x}'$ of its Fourier transform, $\widehat{x}$ that satisfies the following ℓ_2/ℓ_2 *guarantee*:

$$\|\widehat{x} - \widehat{x}'\|_2 \leq C \min_{k\text{-sparse } y} \|\widehat{x} - y\|_2, \tag{4.1}$$

where C is some approximation factor and the minimization is over k-sparse signals.

Furthermore, our algorithm for the exactly sparse case is quite simple and has low big-Oh constants. In particular, our implementation of a variant of this algorithm, described in Chapter 6, is faster than FFTW, a highly efficient implementation of the FFT, for $n = 2^{22}$ and $k \leq 2^{17}$ [71].

[1] One also needs to assume that k divides n. See appendix B for more details.

In contrast, for the same signal size, the algorithms in Chapter 3 were faster than FFTW only for $k \leq 2000$.[2]

We complement our algorithmic results by showing that any algorithm that works for the general case must use at least $\Omega(k \log(n/k)/\log\log n)$ samples from x. The proof of this lower bound can be found in Appendix C. The lower bound uses techniques from [146], which shows a lower bound of $\Omega(k \log(n/k))$ for the number of *arbitrary* linear measurements needed to compute the k-sparse approximation of an n-dimensional vector $\widehat{x}$. In comparison to [146], our bound is slightly worse but it holds even for *adaptive* sampling, where the algorithm selects the samples based on the values of the previously sampled coordinates.[3] Note that our algorithms are *non-adaptive*, and thus limited by the more stringent lower bound of [146].

4.1.2 Techniques

Recall from Chapter 3 that we can use the flat window filters coupled with a random permutation of the spectrum to bin/bucketize the Fourier coefficients into a small number of buckets. We can then use that to estimate the positions and values of the large frequency coefficients that were isolated in their own bucket. Here, we use the same filters introduced in Chapter 3. In this case, a filter G have the property that the value of $\widehat{G}$ is "large" over a constant fraction of the pass region, referred to as the "super-pass" region. We say that a coefficient is "isolated" if it falls into a filter's super-pass region and no other coefficient falls into filter's pass region. Since the super-pass region of our filters is a constant fraction of the pass region, the probability of isolating a coefficient is constant.

However, the main difference in this chapter, that allows us to achieve the stated running times, is a fast method for locating and estimating isolated coefficients. Further, our algorithm is iterative, so we also provide a fast method for updating the signal so that identified coefficients are not considered in future iterations. Below, we describe these methods in more detail.

New Techniques: Location and Estimation

Our location and estimation methods depends on whether we handle the exactly sparse case or the general case. In the exactly sparse case, we show how to estimate the position of an isolated Fourier coefficient using only two samples of the filtered signal. Specifically, we show that the phase difference between the two samples is linear in the index of the coefficient, and hence we can recover the index by estimating the phases. This approach is inspired by the frequency offset estimation in orthogonal frequency division multiplexing (OFDM), which is the modulation method used in modern wireless technologies (see [77], Chapter 2).

[2]Note that both numbers ($k \leq 2^{17}$ and $k \leq 2000$) are for the exactly k-sparse case. The algorithm in Chapter 3 can deal with the general case, but the empirical runtimes are higher.

[3]Note that if we allow *arbitrary* adaptive linear measurements of a vector $\widehat{x}$, then its k-sparse approximation can be computed using only $O(k \log\log(n/k))$ samples [86]. Therefore, our lower bound holds only where the measurements, although adaptive, are limited to those induced by the Fourier matrix. This is the case when we want to compute a sparse approximation to $\widehat{x}$ from samples of x.

In order to design an algorithm[4] for the general case, we employ a different approach. Specifically, we can use two samples to estimate (with constant probability) individual bits of the index of an isolated coefficient. Similar approaches have been employed in prior work. However, in those papers, the index was recovered bit by bit, and one needed $\Omega(\log \log n)$ samples per bit to recover *all* bits correctly with constant probability. In contrast, we recover the index one *block of bits* at a time, where each block consists of $O(\log \log n)$ bits. This approach is inspired by the fast sparse recovery algorithm of [62]. Applying this idea in our context, however, requires new techniques. The reason is that, unlike in [62], we do not have the freedom of using arbitrary "linear measurements" of the vector $\hat{x}$, and we can only use the measurements induced by the Fourier transform.[5] As a result, the extension from "bit recovery" to "block recovery" is the most technically involved part of the algorithm. Section 4.3.1 contains further intuition on this part.

New Techniques: Updating the Signal

The aforementioned techniques recover the position and the value of any isolated coefficient. However, during each filtering step, each coefficient becomes isolated only with constant probability. Therefore, the filtering process needs to be repeated to ensure that each coefficient is correctly identified. In Chapter 3, the algorithm simply performs the filtering $O(\log n)$ times and uses the median estimator to identify each coefficient with high probability. This, however, would lead to a running time of $O(k \log^2 n)$ in the k-sparse case, since each filtering step takes $k \log n$ time.

One could reduce the filtering time by subtracting the identified coefficients from the signal. In this way, the number of non-zero coefficients would be reduced by a constant factor after each iteration, so the cost of the first iteration would dominate the total running time. Unfortunately, subtracting the recovered coefficients from the signal is a computationally costly operation, corresponding to a so-called *non-uniform* DFT (see [61] for details). Its cost would override any potential savings.

In this chapter, we introduce a different approach: instead of subtracting the identified coefficients from the *signal*, we subtract them directly from the *bins* obtained by filtering the signal. The latter operation can be done in time linear in the number of subtracted coefficients, since each of them "falls" into only one bin. Hence, the computational costs of each iteration can be decomposed into two terms, corresponding to filtering the original signal and subtracting the coefficients. For the exactly sparse case these terms are as follows:

- The cost of filtering the original signal is $O(B \log n)$, where B is the number of bins. B is set to $O(k')$, where k' is the the number of yet-unidentified coefficients. Thus, initially B is equal to $O(k)$, but its value decreases by a constant factor after each iteration.
- The cost of subtracting the identified coefficients from the bins is $O(k)$.

Since the number of iterations is $O(\log k)$, and the cost of filtering is dominated by the first iteration, the total running time is $O(k \log n)$ for the exactly sparse case.

[4] We note that although the two-sample approach employed in our algorithm works in theory only for the exactly k-sparse case, our preliminary experiments show that using a few more samples to estimate the phase works surprisingly well even for general signals.

[5] In particular, the method of [62] uses measurements corresponding to a random error correcting code.

For the general case, we need to set k' and B more carefully to obtain the desired running time. The cost of each iterative step is multiplied by the number of filtering steps needed to compute the location of the coefficients, which is $\Theta(\log(n/B))$. If we set $B = \Theta(k')$, this would be $\Theta(\log n)$ in most iterations, giving a $\Theta(k \log^2 n)$ running time. This is too slow when k is close to n. We avoid this by decreasing B more slowly and k' more quickly. In the r-th iteration, we set $B = k/\text{poly}(r)$. This allows the total number of bins to remain $O(k)$ while keeping $\log(n/B)$ small—at most $O(\log \log k)$ more than $\log(n/k)$. Then, by having k' decrease according to $k' = k/r^{\Theta(r)}$ rather than $k/2^{\Theta(r)}$, we decrease the number of rounds to $O(\log k/ \log \log k)$. Some careful analysis shows that this counteracts the $\log \log k$ loss in the $\log(n/B)$ term, achieving the desired $O(k \log n \log(n/k))$ running time.

4.2 Algorithm for the Exactly Sparse Case

In this section, we assume $\widehat{x}_i \in \{-L, \ldots, L\}$ for some precision parameter L. To simplify the bounds, we assume $L \leq n^c$ for some constant $c > 0$; otherwise the $\log n$ term in the running time bound is replaced by $\log L$. We also assume that $\widehat{x}$ is exactly k-sparse. We will use the filter G with parameter $\delta = 1/(4n^2L)$.

Definition 4.2.1. *We say that* $(G, \widehat{G'}) = (G_{B,\delta,\alpha}, \widehat{G'}_{B,\delta,\alpha}) \in \mathbb{R}^n \times \mathbb{R}^n$ *is a* flat window function *with parameters* $B \geq 1$, $\delta > 0$, *and* $\alpha > 0$ *if* $|\text{supp}(G)| = O(\frac{B}{\alpha} \log(n/\delta))$ *and* $\widehat{G'}$ *satisfies*

- $\widehat{G'}_i = 1$ *for* $|i| \leq (1-\alpha)n/(2B)$ *and* $\widehat{G'}_i = 0$ *for* $|i| \geq n/(2B)$
- $\widehat{G'}_i \in [0, 1]$ *for all* i
- $\|\widehat{G'} - \widehat{G}\|_\infty < \delta$.

The above notion corresponds to the $(1/(2B), (1-\alpha)/(2B), \delta, O(B/\alpha \log(n/\delta))$-flat window function. In Appendix D, we give efficient constructions of such window functions, where G can be computed in $O(\frac{B}{\alpha} \log(n/\delta))$ time and for each i, $\widehat{G'}_i$ can be computed in $O(\log(n/\delta))$ time. Of course, for $i \notin [(1-\alpha)n/(2B), n/(2B)]$, $\widehat{G'}_i \in \{0, 1\}$ can be computed in $O(1)$ time. The fact that $\widehat{G'}_i$ takes $\omega(1)$ time to compute for $i \in [(1-\alpha)n/(2B), n/(2B)]$ will add some complexity to our algorithm and analysis. We will need to ensure that we rarely need to compute such values. A practical implementation might find it more convenient to precompute the window functions in a preprocessing stage, rather than compute them on the fly.

The algorithm NOISELESSSPARSEFFT (SFT 3.0) is described as Algorithm 4.2.1. The algorithm has three functions:

- HASHTOBINS. This permutes the spectrum of $\widehat{x - z}$ with $P_{\sigma,a,b}$, then "hashes" to B bins. The guarantee will be described in Lemma 4.2.4.
- NOISELESSSPARSEFFTINNER. Given time-domain access to x and a sparse vector $\widehat{z}$ such that $\widehat{x - z}$ is k'-sparse, this function finds "most" of $\widehat{x - z}$.
- NOISELESSSPARSEFFT. This iterates NOISELESSSPARSEFFTINNER until it finds $\widehat{x}$ exactly.

We analyze the algorithm "bottom-up", starting from the lower-level procedures.

procedure HASHTOBINS(x, $\widehat{z}$, $P_{\sigma,a,b}$, B, δ, α)
  Compute $\widehat{y}_{jn/B}$ for $j \in [B]$, where $y = G_{B,\alpha,\delta} \cdot (P_{\sigma,a,b}x)$
  Compute $\widehat{y'}_{jn/B} = \widehat{y}_{jn/B} - (\widehat{G'_{B,\alpha,\delta}} * \widehat{P_{\sigma,a,b}z})_{jn/B}$ for $j \in [B]$
  return $\widehat{u}$ given by $\widehat{u}_j = \widehat{y'}_{jn/B}$.
procedure NOISELESSSPARSEFFTINNER(x, k', $\widehat{z}$, α)
  Let $B = k'/\beta$, for sufficiently small constant β.
  Let $\delta = 1/(4n^2L)$.
  Choose σ uniformly at random from the set of odd numbers in $[n]$.
  Choose b uniformly at random from $[n]$.
  $\widehat{u} \leftarrow$ HASHTOBINS$(x, \widehat{z}, P_{\sigma,0,b}, B, \delta, \alpha)$.
  $\widehat{u}' \leftarrow$ HASHTOBINS$(x, \widehat{z}, P_{\sigma,1,b}, B, \delta, \alpha)$.
  $\widehat{w} \leftarrow 0$.
  Compute $J = \{j : |\widehat{u}_j| > 1/2\}$.
  for $j \in J$ **do**
    $a \leftarrow \widehat{u}_j/\widehat{u}'_j$.
    $i \leftarrow \sigma^{-1}(\text{round}(\phi(a)\frac{n}{2\pi})) \bmod n$. ▷ $\phi(a)$ denotes the phase of a.
    $v \leftarrow \text{round}(\widehat{u}_j)$.
    $\widehat{w}_i \leftarrow v$.
  return $\widehat{w}$
procedure NOISELESSSPARSEFFT(x, k)
  $\widehat{z} \leftarrow 0$
  for $t \in 0, 1, \ldots, \log k$ **do**
    $k_t \leftarrow k/2^t$, $\alpha_t \leftarrow \Theta(2^{-t})$.
    $\widehat{z} \leftarrow \widehat{z} +$ NOISELESSSPARSEFFTINNER$(x, k_t, \widehat{z}, \alpha_t)$.
  return $\widehat{z}$

4.2.1: SFT 3.0: Exact Sparse Fourier Transform for $k = o(n)$

Analysis of NOISELESSSPARSEFFTINNER and HASHTOBINS.

For any execution of NOISELESSSPARSEFFTINNER, define the support $S = \text{supp}(\widehat{x} - \widehat{z})$. Recall that $\pi_{\sigma,b}(i) = \sigma(i - b) \bmod n$. Define $h_{\sigma,b}(i) = \text{round}(\pi_{\sigma,b}(i)B/n)$ and $o_{\sigma,b}(i) = \pi_{\sigma,b}(i) - h_{\sigma,b}(i)n/B$. Note that therefore $|o_{\sigma,b}(i)| \leq n/(2B)$. We will refer to $h_{\sigma,b}(i)$ as the "bin" that the frequency i is mapped into, and $o_{\sigma,b}(i)$ as the "offset". For any $i \in S$ define two types of events associated with i and S and defined over the probability space induced by σ and b:

- "Collision" event $E_{coll}(i)$: holds iff $h_{\sigma,b}(i) \in h_{\sigma,b}(S \setminus \{i\})$, and
- "Large offset" event $E_{off}(i)$: holds iff $|o_{\sigma,b}(i)| \geq (1 - \alpha)n/(2B)$.

Claim 4.2.2. *For any $i \in S$, the event $E_{coll}(i)$ holds with probability at most $4|S|/B$.*

Proof. Consider distinct $i, j \in S$. By Lemma 2.2.7,

$$\begin{aligned}\Pr[h_{\sigma,b}(i) = h_{\sigma,b}(j)] &\leq \Pr[\pi_{\sigma,b}(i) - \pi_{\sigma,b}(j) \bmod n \in [-n/B, n/B]] \\ &= \Pr[\sigma(i-j) \bmod n \in [-n/B, n/B]] \\ &\leq 4/B.\end{aligned}$$

By a union bound over $j \in S$, $\Pr[E_{coll}(i)] \leq 4\,|S|\,/B$. □

Claim 4.2.3. *For any $i \in S$, the event $E_{off}(i)$ holds with probability at most α.*

Proof. Note that $o_{\sigma,b}(i) \equiv \pi_{\sigma,b}(i) \equiv \sigma(i-b) \pmod{n/B}$. For any odd σ and any $l \in [n/B]$, we have that $\Pr_b[\sigma(i-b) \equiv l \pmod{n/B}] = B/n$. Since only $\alpha n/B$ offsets $o_{\sigma,b}(i)$ cause $E_{off}(i)$, the claim follows. □

Lemma 4.2.4. *Suppose B divides n. The output $\widehat{u}$ of* HASHTOBINS *satisfies*

$$\widehat{u}_j = \sum_{h_{\sigma,b}(i)=j} \widehat{(x-z)}_i \widehat{(G'_{B,\delta,\alpha})}_{-o_{\sigma,b}(i)} \omega^{a\sigma i} \pm \delta \|\widehat{x}\|_1.$$

Let $\zeta = |\{i \in \mathrm{supp}(\widehat{z}) \mid E_{off}(i)\}|$. The running time of HASHTOBINS *is $O(\frac{B}{\alpha}\log(n/\delta) + \|\widehat{z}\|_0 + \zeta\log(n/\delta))$.*

Proof. The proof can be found in Appendix A.3. □

Lemma 4.2.5. *Consider any $i \in S$ such that neither $E_{coll}(i)$ nor $E_{off}(i)$ holds. Let $j = h_{\sigma,b}(i)$. Then*

$$\textit{round}(\phi(\widehat{u}_j/\widehat{u}'_j))\frac{n}{2\pi}) = \sigma i \pmod{n},$$

$$\textit{round}(\widehat{u}_j) = \widehat{x}_i - \widehat{z}_i,$$

and $j \in J$.

Proof. The proof can be found in Appendix A.4. □

For each invocation of NOISELESSSPARSEFFTINNER, let P be the the set of all pairs (i, v) for which the command $\widehat{w}_i \leftarrow v$ was executed. Claims 4.2.2 and 4.2.3 and Lemma 4.2.5 together guarantee that for each $i \in S$ the probability that P does not contain the pair $(i, (\widehat{x} - \widehat{z})_i)$ is at most $4|S|/B + \alpha$. We complement this observation with the following claim.

Claim 4.2.6. *For any $j \in J$ we have $j \in h_{\sigma,b}(S)$. Therefore, $|J| = |P| \leq |S|$.*

Proof. Consider any $j \notin h_{\sigma,b}(S)$. From Equation (A.1) in the proof of Lemma 4.2.5 it follows that $|\widehat{u}_j| \leq \delta n L < 1/2$. □

Lemma 4.2.7. *Consider an execution of* NOISELESSSPARSEFFTINNER, *and let $S = \mathrm{supp}(\widehat{x} - \widehat{z})$. If $|S| \leq k'$, then*

$$E[\|\widehat{x} - \widehat{z} - \widehat{w}\|_0] \leq 8(\beta + \alpha)|S|.$$

Proof. Let e denote the number of coordinates $i \in S$ for which either $E_{coll}(i)$ or $E_{off}(i)$ holds. Each such coordinate might not appear in P with the correct value, leading to an incorrect value of $\widehat{w}_i$. In fact, it might result in an arbitrary pair (i', v') being added to P, which in turn could lead to an incorrect value of $\widehat{w}_{i'}$. By Claim 4.2.6 these are the only ways that $\widehat{w}$ can be assigned an incorrect value. Thus we have

$$\|\widehat{x} - \widehat{z} - \widehat{w}\|_0 \leq 2e.$$

Since $E[e] \leq (4|S|/B + \alpha)|S| \leq (4\beta + \alpha)|S|$, the lemma follows. □

Analysis of NOISELESSSPARSEFFT.

Consider the tth iteration of the procedure, and define $S_t = \operatorname{supp}(\widehat{x} - \widehat{z})$ where $\widehat{z}$ denotes the value of the variable at the beginning of loop. Note that $|S_0| = |\operatorname{supp}(\widehat{x})| \leq k$.

We also define an indicator variable I_t which is equal to 0 iff $|S_t|/|S_{t-1}| \leq 1/8$. If $I_t = 1$ we say the the tth iteration was not *successful*. Let $\gamma = 8 \cdot 8(\beta + \alpha)$. From Lemma 4.2.7 it follows that $\Pr[I_t = 1 \mid |S_{t-1}| \leq k/2^{t-1}] \leq \gamma$. From Claim 4.2.6 it follows that even if the tth iteration is not successful, then $|S_t|/|S_{t-1}| \leq 2$.

For any $t \geq 1$, define an event $E(t)$ that occurs iff $\sum_{i=1}^{t} I_i \geq t/2$. Observe that if none of the events $E(1) \ldots E(t)$ holds then $|S_t| \leq k/2^t$.

Lemma 4.2.8. *Let $E = E(1) \cup \ldots \cup E(\lambda)$ for $\lambda = 1 + \log k$. Assume that $(4\gamma)^{1/2} < 1/4$. Then $\Pr[E] \leq 1/3$.*

Proof. Let $t' = \lceil t/2 \rceil$. We have

$$\Pr[E(t)] \leq \binom{t}{t'} \gamma^{t'} \leq 2^t \gamma^{t'} \leq (4\gamma)^{t/2}$$

Therefore

$$\Pr[E] \leq \sum_t \Pr[E(t)] \leq \frac{(4\gamma)^{1/2}}{1 - (4\gamma)^{1/2}} \leq 1/4 \cdot 4/3 = 1/3.$$

□

Theorem 4.2.9. *Suppose $\widehat{x}$ is k-sparse with entries from $\{-L, \ldots, L\}$ for some known $L = n^{O(1)}$. Then the algorithm* NOISELESSSPARSEFFT *runs in expected $O(k \log n)$ time and returns the correct vector $\widehat{x}$ with probability at least $2/3$.*

Proof. The correctness follows from Lemma 4.2.8. The running time is dominated by $O(\log k)$ executions of HASHTOBINS.

Assuming a correct run, in every round t we have

$$\|\widehat{z}\|_0 \leq \|\widehat{x}\|_0 + |S_t| \leq k + k/2^t \leq 2k.$$

Therefore

$$\mathbb{E}[|\{i \in \operatorname{supp}(z) \mid E_{off}(i)\}|] \leq \alpha \|\widehat{z}\|_0 \leq 2\alpha k,$$

so the expected running time of each execution of HASHTOBINS is $O(\frac{B}{\alpha}\log(n/\delta)+k+\alpha k\log(n/\delta)) = O(\frac{B}{\alpha}\log n + k + \alpha k\log n)$. Setting $\alpha = \Theta(2^{-t/2})$ and $\beta = \Theta(1)$, the expected running time in round t is $O(2^{-t/2}k\log n + k + 2^{-t/2}k\log n)$. Therefore the total expected running time is $O(k\log n)$. □

4.3 Algorithm for the General Case

For the general case, we only achieve Equation (4.1) for $C = 1+\epsilon$ if $\|\widehat{x}\|_2 \le n^{O(1)}\cdot\min_{k\text{-sparse } y}\|\widehat{x}-y\|_2$. In general, for any parameter $\delta > 0$ we can add $\delta\|\widehat{x}\|_2$ to the right hand side of Equation (4.1) and run in time $O(k\log(n/k)\log(n/\delta))$.

Pseudocode for the general algorithm SFT 4.0 is shown in Algorithm 4.3.1 and 4.3.2.

4.3.1 Intuition

Let S denote the "heavy" $O(k/\epsilon)$ coordinates of $\widehat{x}$. The overarching algorithm SPARSEFFT (SFT 4.0) works by first "locating" a set L containing most of S, then "estimating" $\widehat{x}_L$ to get $\widehat{z}$. It then repeats on $\widehat{x-z}$. We will show that each heavy coordinate has a large constant probability of both being in L and being estimated well. As a result, $\widehat{x-z}$ is probably nearly $k/4$-sparse, so we can run the next iteration with $k \to k/4$. The later iterations then run faster and achieve a higher success probability, so the total running time is dominated by the time in the first iteration and the total error probability is bounded by a constant.

In the rest of this intuition, we will discuss the first iteration of SPARSEFFT with simplified constants. In this iteration, hashes are to $B = O(k/\epsilon)$ bins and, with $3/4$ probability, we get $\widehat{z}$ so $\widehat{x-z}$ is nearly $k/4$-sparse. The actual algorithm will involve a parameter α in each iteration, roughly guaranteeing that with $1-\sqrt{\alpha}$ probability, we get $\widehat{z}$ so $\widehat{x-z}$ is nearly $\sqrt{\alpha}k$-sparse; the formal guarantee will be given by Lemma 4.3.8. For this intuition we only consider the first iteration where α is a constant.

Location

As in the noiseless case, to locate the "heavy" coordinates we consider the "bins" computed by HASHTOBINS with $P_{\sigma,a,b}$. This roughly corresponds to first permuting the coordinates according to the (almost) pairwise independent permutation $P_{\sigma,a,b}$, partitioning the coordinates into $B = O(k/\epsilon)$ "bins" of n/B consecutive indices, and observing the sum of values in each bin. We get that each heavy coordinate i has a large constant probability that the following two events occur: no other heavy coordinate lies in the same bin, and only a small (i.e., $O(\epsilon/k)$) fraction of the mass from non-heavy coordinates lies in the same bin. For such a "well-hashed" coordinate i, we would like to find its location $\tau = \pi_{\sigma,b}(i) = \sigma(i-b)$ among the $\epsilon n/k < n/k$ consecutive values that hash to the same bin. Let

$$\theta_j^* = \frac{2\pi}{n}(j+\sigma b) \pmod{2\pi}. \tag{4.2}$$

so $\theta^*_\tau = \frac{2\pi}{n}\sigma i$. In the noiseless case, we showed that the difference in phase in the bin using $P_{\sigma,0,b}$ and using $P_{\sigma,1,b}$ is θ^*_τ plus a negligible $O(\delta)$ term. With noise this may not be true; however, we can say for any $\beta \in [n]$ that the difference in phase between using $P_{\sigma,a,b}$ and $P_{\sigma,a+\beta,b}$, as a distribution over uniformly random $a \in [n]$, is $\beta\theta^*_\tau + \nu$ with (for example) $\mathbb{E}[\nu^2] = 1/100$ (all operations on phases modulo 2π). We can only hope to get a constant number of bits from such a "measurement". So our task is to find τ within a region Q of size n/k using $O(\log(n/k))$ "measurements" of this form.

One method for doing so would be to simply do measurements with random $\beta \in [n]$. Then each measurement lies within $\pi/4$ of $\beta\theta^*_\tau$ with at least $1 - \frac{\mathbb{E}[\nu^2]}{\pi^2/16} > 3/4$ probability. On the other hand, for $j \neq \tau$ and as a distribution over β, $\beta(\theta^*_\tau - \theta^*_j)$ is roughly uniformly distributed around the circle. As a result, each measurement is probably more than $\pi/4$ away from $\beta\theta^*_j$. Hence $O(\log(n/k))$ repetitions suffice to distinguish among the n/k possibilities for τ. However, while the number of measurements is small, it is not clear how to decode in polylog rather than $\Omega(n/k)$ time.

To solve this, we instead do a t-ary search on the location for $t = \Theta(\log n)$. At each of $O(\log_t(n/k))$ levels, we split our current candidate region Q into t consecutive subregions $Q_1, \ldots, Q_t$, each of size w. Now, rather than choosing $\beta \in [n]$, we choose $\beta \in [\frac{n}{16w}, \frac{n}{8w}]$. By the upper bound on β, for each $q \in [t]$ the values $\{\beta\theta^*_j \mid j \in Q_q\}$ all lie within $\beta w \frac{2\pi}{n} \leq \pi/4$ of each other on the circle. On the other hand, if $|j - \tau| > 16w$, then $\beta(\theta^*_\tau - \theta^*_j)$ will still be roughly uniformly distributed about the circle. As a result, we can check a single candidate element e_q from each subregion: if e_q is in the same subregion as τ, each measurement usually agrees in phase; but if e_q is more than 16 subregions away, each measurement usually disagrees in phase. Hence with $O(\log t)$ measurements, we can locate τ to within $O(1)$ consecutive subregions with failure probability $1/t^{\Theta(1)}$. The decoding time is $O(t \log t)$.

This primitive LOCATEINNER lets us narrow down the candidate region for τ to a subregion that is a $t' = \Omega(t)$ factor smaller. By repeating LOCATEINNER $\log_{t'}(n/k)$ times, LOCATESIGNAL can find τ precisely. The number of measurements is then $O(\log t \log_t(n/k)) = O(\log(n/k))$ and the decoding time is $O(t \log t \log_t(n/k)) = O(\log(n/k) \log n)$. Furthermore, the "measurements" (which are actually calls to HASHTOBINS) are non-adaptive, so we can perform them in parallel for all $O(k/\epsilon)$ bins, with $O(\log(n/\delta))$ average time per measurement. This gives $O(k \log(n/k) \log(n/\delta))$ total time for LOCATESIGNAL.

This lets us locate every heavy and "well-hashed" coordinate with $1/t^{\Theta(1)} = o(1)$ failure probability, so every heavy coordinate is located with arbitrarily high constant probability.

Estimation

By contrast, estimation is fairly simple. As in Algorithm 4.2.1, we can estimate $\widehat{(x - z)}_i$ as $\widehat{u}_{h_{\sigma,b}(i)}$, where $\widehat{u}$ is the output of HASHTOBINS. Unlike in Algorithm 4.2.1, we now have noise that may cause a single such estimate to be poor even if i is "well-hashed". However, we can show that for a random permutation $P_{\sigma,a,b}$ the estimate is "good" with constant probability. ESTIMATEVALUES takes the median of $R_{est} = O(\log \frac{1}{\epsilon})$ such samples, getting a good estimate with $1 - \epsilon/64$ probability. Given a candidate set L of size k/ϵ, with $7/8$ probability at most $k/8$ of the coordinates are badly estimated. On the other hand, with $7/8$ probability, at least $7k/8$ of the heavy coordinates are both located and well estimated. This suffices to show that, with $3/4$ probability, the largest k

procedure SPARSEFFT(x, k, ϵ, δ)
  $R \leftarrow O(\log k / \log\log k)$ as in Theorem 4.3.9.
  $\widehat{z}^{(1)} \leftarrow 0$
  for $r \in [R]$ **do**
    Choose B_r, k_r, α_r as in Theorem 4.3.9.
    $R_{est} \leftarrow O(\log(\frac{B_r}{\alpha_r k_r}))$ as in Lemma 4.3.8.
    $L_r \leftarrow$ LOCATESIGNAL$(x, \widehat{z}^{(r)}, B_r, \alpha_r, \delta)$
    $\widehat{z}^{(r+1)} \leftarrow \widehat{z}^{(r)} +$ ESTIMATEVALUES$(x, \widehat{z}^{(r)}, 3k_r, L_r, B_r, \delta, R_{est})$.
  return $\widehat{z}^{(R+1)}$
procedure ESTIMATEVALUES(x, $\widehat{z}$, k', L, B, δ, R_{est})
  for $r \in [R_{est}]$ **do**
    Choose $a_r, b_r \in [n]$ uniformly at random.
    Choose σ_r uniformly at random from the set of odd numbers in $[n]$.
    $\widehat{u}^{(r)} \leftarrow$ HASHTOBINS$(x, \widehat{z}, P_{\sigma,a_r,b}, B, \delta)$.
  $\widehat{w} \leftarrow 0$
  for $i \in L$ **do**
    $\widehat{w}_i \leftarrow \mathrm{median}_r\, \widehat{u}^{(r)}_{h_{\sigma,b}(i)} \omega^{-a_r \sigma i}$. ▷ Separate median in real and imaginary axes.
  $J \leftarrow \arg\max_{|J|=k'} \|\widehat{w}_J\|_2$.
  return $\widehat{w}_J$

4.3.1: SFT 4.0: General Sparse Fourier Transform for $k = o(n)$, Part 1/2.

elements J of our estimate $\widehat{w}$ contains good estimates of $3k/4$ large coordinates, so $\widehat{x - z - w_J}$ is close to $k/4$-sparse.

4.3.2 Analysis

Formal definitions

As in the noiseless case, we define $\pi_{\sigma,b}(i) = \sigma(i-b) \bmod n$, $h_{\sigma,b}(i) = \text{round}(\pi_{\sigma,b}(i)B/n)$ and $o_{\sigma,b}(i) = \pi_{\sigma,b}(i) - h_{\sigma,b}(i)n/B$. We say $h_{\sigma,b}(i)$ is the "bin" that frequency i is mapped into, and $o_{\sigma,b}(i)$ is the "offset". We define $h^{-1}_{\sigma,b}(j) = \{i \in [n] \mid h_{\sigma,b}(i) = j\}$.

Define

$$\mathrm{Err}(x,k) = \min_{k\text{-sparse } y} \|x-y\|_2.$$

In each iteration of SPARSEFFT, define $\widehat{x}' = \widehat{x} - \widehat{z}$, and let

$$\begin{aligned}\rho^2 &= \mathrm{Err}^2(\widehat{x}', k) + \delta^2 n \|\widehat{x}\|_1^2 \\ \mu^2 &= \epsilon\rho^2/k \\ S &= \{i \in [n] \mid |\widehat{x}'_i|^2 \geq \mu^2\}\end{aligned}$$

Then $|S| \leq (1+1/\epsilon)k = O(k/\epsilon)$ and $\|\widehat{x}' - \widehat{x}'_S\|_2^2 \leq (1+\epsilon)\rho^2$. We will show that each $i \in S$ is found by LOCATESIGNAL with probability $1 - O(\alpha)$, when $B = \Omega(\frac{k}{\alpha\epsilon})$.

procedure LOCATESIGNAL(x, $\widehat{z}$, B, α, δ)
    Choose uniformly at random $\sigma, b \in [n]$ with σ odd.
    Initialize $l_i^{(1)} = (i-1)n/B$ for $i \in [B]$.
    Let $w_0 = n/B$, $t = \log n$, $t' = t/4$, $D_{max} = \log_{t'}(w_0 + 1)$.
    Let $R_{loc} = \Theta(\log_{1/\alpha}(t/\alpha))$ per Lemma 4.3.5.
    for $D \in [D_{max}]$ **do**
        $l^{(D+1)} \leftarrow$ LOCATEINNER$(x, \widehat{z}, B, \delta, \alpha, \sigma, \beta, l^{(D)}, w_0/(t')^{D-1}, t, R_{loc})$
    $L \leftarrow \{\pi_{\sigma,b}^{-1}(l_j^{(D_{max}+1)}) \mid j \in [B]\}$
    return L

▷ δ, α parameters for G, G'
▷ $(l_1, l_1 + w), \ldots, (l_B, l_B + w)$ the plausible regions.
▷ $B \approx k/\epsilon$ the number of bins
▷ $t \approx \log n$ the number of regions to split into.
▷ $R_{loc} \approx \log t = \log\log n$ the number of rounds to run

procedure LOCATEINNER(x, $\widehat{z}$, B, δ, α, σ, b, l, w, t, R_{loc})
    Let $s = \Theta(\alpha^{1/3})$.
    Let $v_{j,q} = 0$ for $(j, q) \in [B] \times [t]$.
    for $r \in [R_{loc}]$ **do**
        Choose $a \in [n]$ uniformly at random.
        Choose $\beta \in \{\frac{snt}{4w}, \ldots, \frac{snt}{2w}\}$ uniformly at random.
        $\widehat{u} \leftarrow$ HASHTOBINS$(x, \widehat{z}, P_{\sigma,a,b}, B, \delta, \alpha)$.
        $\widehat{u}' \leftarrow$ HASHTOBINS$(x, \widehat{z}, P_{\sigma,a+\beta,b}, B, \delta, \alpha)$.
        for $j \in [B]$ **do**
            $c_j \leftarrow \phi(\widehat{u}_j / \widehat{u}'_j)$
            for $q \in [t]$ **do**
                $m_{j,q} \leftarrow l_j + \frac{q-1/2}{t} w$
                $\theta_{j,q} \leftarrow \frac{2\pi(m_{j,q}+\sigma b)}{n} \bmod 2\pi$
                if $\min(|\beta\theta_{j,q} - c_j|, 2\pi - |\beta\theta_{j,q} - c_j|) < s\pi$ **then**
                    $v_{j,q} \leftarrow v_{j,q} + 1$
    for $j \in [B]$ **do**
        $Q^* \leftarrow \{q \in [t] \mid v_{j,q} > R_{loc}/2\}$
        if $Q^* \neq \emptyset$ **then**
            $l'_j \leftarrow \min_{q \in Q^*} l_j + \frac{q-1}{t} w$
        else
            $l'_j \leftarrow \perp$
    return l'

4.3.2: SFT 4.0: General Sparse Fourier Transform for $k = o(n)$, Part 2/2.

For any $i \in S$ define three types of events associated with i and S and defined over the probability space induced by σ and b:

- "Collision" event $E_{coll}(i)$: holds iff $h_{\sigma,b}(i) \in h_{\sigma,b}(S \setminus \{i\})$;
- "Large offset" event $E_{off}(i)$: holds iff $|o_{\sigma,b}(i)| \geq (1-\alpha)n/(2B)$; and
- "Large noise" event $E_{noise}(i)$: holds iff $\|\widehat{x'}_{h^{-1}_{\sigma,b}(h_{\sigma,b}(i))\setminus S}\|_2^2 \geq \mathrm{Err}^2(\widehat{x'}, k)/(\alpha B)$.

By Claims 4.2.2 and 4.2.3, $\Pr[E_{coll}(i)] \leq 4|S|/B = O(\alpha)$ and $\Pr[E_{off}(i)] \leq 2\alpha$ for any $i \in S$.

Claim 4.3.1. *For any $i \in S$, $\Pr[E_{noise}(i)] \leq 4\alpha$.*

Proof. For each $j \neq i$, $\Pr[h_{\sigma,b}(j) = h_{\sigma,b}(i)] \leq \Pr[|\sigma j - \sigma i| < n/B] \leq 4/B$ by Lemma 2.2.7. Then

$$\mathbb{E}[\|\widehat{x'}_{h^{-1}_{\sigma,b}(h_{\sigma,b}(i))\setminus S}\|_2^2] \leq 4\|\widehat{x'}_{[n]\setminus S}\|_2^2/B$$

The result follows by Markov's inequality. □

We will show for $i \in S$ that if none of $E_{coll}(i)$, $E_{off}(i)$, and $E_{noise}(i)$ hold then SPARSEFFTINNER recovers $\widehat{x}'_i$ with $1 - O(\alpha)$ probability.

Lemma 4.3.2. *Let $a \in [n]$ uniformly at random, B divide n, and the other parameters be arbitrary in*

$$\widehat{u} = \textsc{HashToBins}(x, \widehat{z}, P_{\sigma,a,b}, B, \delta, \alpha).$$

Then for any $i \in [n]$ with $j = h_{\sigma,b}(i)$ and none of $E_{coll}(i)$, $E_{off}(i)$, or $E_{noise}(i)$ holding,

$$\mathbb{E}[\left|\widehat{u}_j - \widehat{x'}_i \omega^{a\sigma i}\right|^2] \leq 2\frac{\rho^2}{\alpha B}$$

Proof. The proof can be found in Appendix A.5. □

Properties of LOCATESIGNAL

In our intuition, we made a claim that if $\beta \in [n/(16w), n/(8w)]$ uniformly at random, and $i > 16w$, then $\frac{2\pi}{n}\beta i$ is "roughly uniformly distributed about the circle" and hence not concentrated in any small region. This is clear if β is chosen as a random real number; it is less clear in our setting where β is a random integer in this range. We now prove a lemma that formalizes this claim.

Lemma 4.3.3. *Let $T \subset [m]$ consist of t consecutive integers, and suppose $\beta \in T$ uniformly at random. Then for any $i \in [n]$ and set $S \subset [n]$ of l consecutive integers,*

$$\Pr[\beta i \bmod n \in S] \leq \lceil im/n \rceil (1 + \lfloor l/i \rfloor)/t \leq \frac{1}{t} + \frac{im}{nt} + \frac{lm}{nt} + \frac{l}{it}$$

Proof. Note that any interval of length l can cover at most $1 + \lfloor l/i \rfloor$ elements of any arithmetic sequence of common difference i. Then $\{\beta i \mid \beta \in T\} \subset [im]$ is such a sequence, and there are at most $\lceil im/n \rceil$ intervals $an + S$ overlapping this sequence. Hence at most $\lceil im/n \rceil (1 + \lfloor l/i \rfloor)$ of the $\beta \in [m]$ have $\beta i \bmod n \in S$. Hence $\Pr[\beta i \bmod n \in S] \leq \lceil im/n \rceil (1 + \lfloor l/i \rfloor)/t$. □

Lemma 4.3.4. *Let $i \in S$. Suppose none of $E_{coll}(i), E_{off}(i)$, and $E_{noise}(i)$ hold, and let $j = h_{\sigma,b}(i)$. Consider any run of* LOCATEINNER *with $\pi_{\sigma,b}(i) \in [l_j, l_j + w]$. Let $f > 0$ be a parameter such that*

$$B = \frac{Ck}{\alpha f \epsilon}.$$

for C larger than some fixed constant. Then $\pi_{\sigma,b}(i) \in [l'_j, l'_j + 4w/t]$ with probability at least $1 - tf^{\Omega(R_{loc})}$,

Proof. The proof can be found in Appendix A.6. □

Lemma 4.3.5. *Suppose $B = \frac{Ck}{\alpha^2 \epsilon}$ for C larger than some fixed constant. The procedure* LOCATESIGNAL *returns a set L of size $|L| \leq B$ such that for any $i \in S$, $\Pr[i \in L] \geq 1 - O(\alpha)$. Moreover the procedure runs in expected time*

$$O((\frac{B}{\alpha}\log(n/\delta) + \|\widehat{z}\|_0(1 + \alpha\log(n/\delta)))\log(n/B)).$$

Proof. The proof can be found in Appendix A.7. □

Properties of ESTIMATEVALUES

Lemma 4.3.6. *For any $i \in L$,*

$$\Pr[\left|\widehat{w}_i - \widehat{x'}_i\right|^2 > \mu^2] < e^{-\Omega(R_{est})}$$

if $B > \frac{Ck}{\alpha\epsilon}$ for some constant C.

Proof. The proof can be found in Appendix A.8. □

Lemma 4.3.7. *Let $R_{est} \geq C\log\frac{B}{\gamma f k}$ for some constant C and parameters $\gamma, f > 0$. Then if* ESTIMATEVALUES *is run with input $k' = 3k$, it returns $\widehat{w_J}$ for $|J| = 3k$ satisfying*

$$\mathrm{Err}^2(\widehat{x'_L} - \widehat{w_J}, fk) \leq \mathrm{Err}^2(\widehat{x'_L}, k) + O(k\mu^2)$$

with probability at least $1 - \gamma$.

Proof. The proof can be found in Appendix A.9. □

Properties of SPARSEFFT

We will show that $\widehat{x} - \widehat{z}^{(r)}$ gets sparser as r increases, with only a mild increase in the error.

Lemma 4.3.8. *Define $\widehat{x}^{(r)} = \widehat{x} - \widehat{z}^{(r)}$. Consider any one loop r of* SPARSEFFT*, running with parameters $(B, k, \alpha) = (B_r, k_r, \alpha_r)$ such that $B \geq \frac{Ck}{\alpha^2\epsilon}$ for some C larger than some fixed constant. Then for any $f > 0$,*

$$\mathrm{Err}^2(\widehat{x}^{(r+1)}, fk) \leq (1+\epsilon)\,\mathrm{Err}^2(\widehat{x}^{(r)}, k) + O(\epsilon\delta^2 n\|\widehat{x}\|_1^2)$$

with probability $1 - O(\alpha/f)$*, and the running time is*

$$O((\|\widehat{z}^{(r)}\|_0(1 + \alpha \log(n/\delta)) + \frac{B}{\alpha} \log(n/\delta))(\log \frac{1}{\alpha\epsilon} + \log(n/B))).$$

Proof. The proof can be found in Appendix A.10. □

Given the above, this next proof follows a similar argument to [86], Theorem 3.7.

Theorem 4.3.9. *With* $2/3$ *probability,* SPARSEFFT *recovers* $\widehat{z}^{(R+1)}$ *such that*

$$\|\widehat{x} - \widehat{z}^{(R+1)}\|_2 \leq (1 + \epsilon) \operatorname{Err}(\widehat{x}, k) + \delta\|\widehat{x}\|_2$$

in $O(\frac{k}{\epsilon} \log(n/k) \log(n/\delta))$ *time.*

Proof. The proof can be found in Appendix A.11. □

4.4 Extension to Two Dimensions

This section outlines the straightforward generalization of SFT 4.0 shown in Algorithm 4.3.1 to two dimensions. We will refer to this algorithm as SFT 4.1.

Theorem 4.4.1. *There is a variant of Algorithm 4.3.1 that will, given* $x, \widehat{z} \in \mathbb{C}^{\sqrt{n} \times \sqrt{n}}$*, return* $\widehat{x'}$ *with*

$$\|\widehat{x} - \widehat{z} - \widehat{x'}\|_2 \leq 2 \cdot \min_{k\text{-sparse } \widehat{x^*}} \|\widehat{x} - \widehat{z} - \widehat{x^*}\|_2^2 + \|\widehat{x}\|_2^2/n^c$$

with probability $1 - \alpha$ *for any constants* $c, \alpha > 0$ *in time*

$$O(k \log(n/k) \log^2 n + |\operatorname{supp}(\widehat{z})| \log(n/k) \log n),$$

using $O(k \log(n/k) \log^2 n)$ *samples of* x*.*

Proof. We need to modify Algorithm 4.3.1 in two ways: by extending it to two dimensions and by allowing the parameter $\widehat{z}$.[6] We will start by describing the adaptation to two dimensions.

The basic idea of Algorithm 4.3.1 is to construct from Fourier measurements a way to "hash" the coordinates in $B = O(k)$ bins. There are three basic components that are needed: a *permutation* that gives nearly pairwise independent hashing to bins; a *filter* that allows for computing the sum of bins using Fourier measurements; and the *location* estimation needs to search in both axes. The permutation is the main subtlety.

[6] We include $\widehat{z}$ so we can use this a subroutine for partial recovery in the following chapter. However, the input $\widehat{z}$ can be set to zero in which case we get the l_2/l_2 guarantee for a 2D version of SFT 4.0.

Permutation Let $\mathcal{M} \subset [\sqrt{n}]^{2\times 2}$ be the set of matrices with odd determinant. For notational purposes, for $v = (i, j)$ we define $x_v := x_{i,j}$.

Definition 4.4.2. *For $M \in \mathcal{M}$ and $a, b \in [\sqrt{n}]^2$, we define the* permutation $P_{M,a,b}\mathbb{C}^{\sqrt{n}\times\sqrt{n}} \to \mathbb{C}^{\sqrt{n}\times\sqrt{n}}$ *by*

$$(P_{M,a,b}x)_v = x_{M(v-a)}(\omega')^{v^T Mb}.$$

We also define $\pi_{M,b}(v) = M(v - b) \pmod{\sqrt{n}}$.

Claim 4.4.3. $\widehat{P_{M,a,b}x}_{\pi_{M^T,b}(v)} = \widehat{x}_v(\omega')^{v^T M^T a}$

Proof.

$$\begin{aligned}
\widehat{P_{M,a,b}x}_{M(v-b)} &= \frac{1}{\sqrt{n}} \sum_{u\in[\sqrt{n}]^2} (\omega')^{u^T M(v-b)} (P_{M,a,b}x)_u \\
&= \frac{1}{\sqrt{n}} \sum_{u\in[\sqrt{n}]^2} (\omega')^{u^T M(v-b)} x_{M(u-a)} (\omega')^{u^T Mb} \\
&= \omega^{v^T M^T a} \frac{1}{\sqrt{n}} \sum_{u\in[\sqrt{n}]^2} (\omega')^{v^T M^T (u-a)} x_{M(u-a)} \\
&= \widehat{x}_i(\omega')^{v^T M^T a}
\end{aligned}$$

where we used that M^T is a bijection over $[\sqrt{n}]^2$ because $\det(M)$ is odd. □

This gives a lemma analogous to Lemma 2.2.7 from Chapter 3.

Lemma 4.4.4. *Suppose $v \in [\sqrt{n}]^2$ is not 0. Then*

$$\Pr_{M\sim\mathcal{M}}[Mv \in [-C, C]^2 \pmod{\sqrt{n}}] \le O(\frac{C^2}{n}).$$

Proof. For any u, define $G(u)$ to be the largest power of 2 that divides both u_0 and u_1. Define $g = G(v)$, and let $S = \{u \in [\sqrt{n}]^2 \mid G(u) = g\}$. We have that Mv is uniform over S: $\mathcal{M}$ is a group and S is the orbit of $(0, g)$.

Because S lies on a lattice of distance g and does not include the origin, there are at most $(2\lfloor C/g\rfloor + 1)^2 - 1 \le 8(C/g)^2$ elements in $S \cap [-C, C]^2$, and $(3/4)n/g^2$ total elements in S. Hence, the probability is at most $(32/3)C^2/n$. □

We can then define the "hash function" $h_{M,b} : [\sqrt{n}]^2 \to [\sqrt{B}]^2$ given by $(h_{M,b}(u)) = \mathrm{round}(\pi_{M,b}(u) \cdot \sqrt{n/B})$; i.e., round to the nearest multiple of $\sqrt{n/B}$ in each coordinate and scale down. We also define the "offset" $o_{M,b}(u) = \pi_{M,b}(u) - \sqrt{n/B}h_{M,b}(u)$. This lets us give results analogous to Claims 4.2.2 and 4.2.3 from Chapter 4:

- $\Pr[h_{M,b}(u) = h_{M,b}(v) < O(1/B)]$ for $u \ne v$. In order for $h(u) = h(v)$, we need that $\pi_{M,b}(u) - \pi_{M,b}(v) \in \left[-2\sqrt{n/B}, 2\sqrt{n/B}\right]^2$. But Lemma 4.4.4 implies this probability is $O(1/B)$.

- $\Pr[o_{M,b}(u) \notin \left[-(1-\alpha)\sqrt{n/B}, (1-\alpha)\sqrt{n/B}]^2\right] < O(\alpha)$ for any $\alpha > 0$. Because of the offset b, $o_{M,b}(u)$ is uniform over $\left[-\sqrt{n/B}, \sqrt{n/B}\right]^2$. Hence the probability is $2\alpha - \alpha^2 + o(1)$ by a volume argument.

which are all we need of the hash function.

Filter Modifying the filter is pretty simple. Specifically, we define a flat-window filter $G \in \mathbb{R}^{\sqrt{n}}$ with support size $O(\sqrt{B}\log n)$ such that $\widehat{G}$ is essentially zero outsize $\left[-\sqrt{n/B}, \sqrt{n/B}\right]$ and is essentially 1 inside $\left[-(1-\alpha)\sqrt{n/B}, (1-\alpha)\sqrt{n/B}\right]$ for constant α. We compute the $\sqrt{B} \times \sqrt{B}$ 2-dimensional DFT of $x'_{i,j} = x_{i,j} G_i G_j$ to sum up the element in each bin. This takes $B \log^2 n$ samples and time rather than $B \log n$, which is the reason for the extra $\log n$ factor compared to the one dimensional case.

Location Location is easy to modify; we simply run it twice to find the row and column separately. We will see how to do this in the following chapter as well.

In summary, the aforementioned adaptations leads to a variant of Algorithm 4.3.1 that works in two dimensions, with running time $O(k \log(n/k) \log^2 n)$, using $O(k \log(n/k) \log^2 n)$ samples.

Adding extra coefficient list $\widehat{z}$ The modification of the Algorithm 4.3.1 (as well as its variant above) is straightforward. The algorithm performs a sequence of iterations, where each iteration involves hashing the frequencies of the signal into bins, followed by subtracting the already recovered coefficients from the bins. Since the algorithm recovers $\Theta(k)$ coefficients in the first iteration, the subtracted list is always of size $\Theta(k)$.

Given the extra coefficient list, the only modification to the algorithm is that the list of the subtracted coefficients needs to be appended with coefficients in $\widehat{z}$. Since this step does not affect the samples taken by the algorithm, the sample bound remains unchanged. To analyze the running time, let k' be the number of nonzero coefficients in $\widehat{z}$. Observe that the total time of the original algorithm spent on subtracting the coefficients from a list of size $\Theta(k)$ was $O(k \log(n/k) \log n)$, or $O(\log(n/k) \log n)$ per list coefficient. Since in our case the number of coefficients in the list is increased from $\Theta(k)$ to $k' + \Theta(k)$, the running time is increased by an additive factor of $O(k' \log(n/k) \log n)$. □

Chapter 5

Optimizing Sample Complexity

5.1 Introduction

The algorithms presented in Chapter 4 achieve very efficient running times. However, they still suffer from important limitations. The main limitation is that their sample complexity bounds are too high. In particular, the sample complexity of the exactly k-sparse algorithm is $\Theta(k \log n)$. This bound is suboptimal by a logarithmic factor, as it is known that one can recover any signal with k nonzero Fourier coefficients from $O(k)$ samples [8], albeit in super-linear time. The sample complexity of the approximately-sparse algorithm is $\Theta(k \log(n) \log(n/k))$. This bound is also a logarithmic factor away from the lower bound of $\Omega(k \log(n/k))$ [146].

Reducing the sample complexity is highly desirable as it typically implies a reduction in signal acquisition time, measurement overhead and communication cost. For example, in medical imaging the main goal is to reduce the sample complexity in order to reduce the time the patient spends in the MRI machine [114], or the radiation dose she receives [158]. Similarly in spectrum sensing, a lower average sampling rate enables the fabrication of efficient analog to digital converters (ADCs) that can acquire very wideband multi-GHz signals [184]. In fact, the central goal of the area of compressed sensing is to reduce the sample complexity.

A second limitation of the previous algorithms is that most of them are designed for one-dimensional signals. We have show in Section 4.4 that a two-dimensional adaptation of the SFT 4.0 algorithm in Chapter 4 has roughly $O(k \log^3 n)$ time and sample complexity. This is unfortunate, since multi-dimensional instances of DFT are often particularly sparse. This situation is somewhat alleviated by the fact that the two-dimensional DFT over $p \times q$ grids can be reduced to the one-dimensional DFT over a signal of length pq [59, 90]. However, the reduction applies only if p and q are relatively prime, which excludes the most typical case of $m \times m$ grids where m is a power of 2.

5.1.1 Results

In this chapter, we present sample-optimal sublinear time algorithms for the sparse Fourier transform over a two-dimensional $\sqrt{n} \times \sqrt{n}$ grid. Our algorithms are analyzed in the average case.

Our input distributions are natural. For the exactly sparse case, we assume the Bernoulli model: each spectrum coordinate is nonzero with probability k/n, in which case the entry assumes an arbitrary value predetermined for that position.[1] For the approximately-sparse case, we assume that the spectrum $\widehat{x}$ of the signal is a sum of two vectors: the signal vector, chosen from the Bernoulli distribution, and the noise vector, chosen from the Gaussian distribution (see Chapter 2 Preliminaries for the complete definition). These or similar[2] distributions are often used as test cases for empirical evaluations of sparse Fourier Transform algorithms [89, 104] or theoretical analysis of their performance [104].

The algorithms succeed with a constant probability. The notion of success depends on the scenario considered. For the exactly sparse case, an algorithm is successful if it recovers the spectrum exactly. For the approximately sparse case, the algorithm is successful if it reports a signal with spectrum $\widehat{z}$ such that:

$$\|\widehat{z} - \widehat{x}\|_2^2 = O(\sigma^2 n) + \|\widehat{x}\|_2^2/n^c, \tag{5.1}$$

where σ^2 denotes the variance of the normal distributions defining each coordinate of the noise vector, and where c is any constant. Note that any k-sparse approximation to $\widehat{x}$ has error $\Omega(\sigma^2 n)$ with overwhelming probability, and that the second term in the bound in Equation (5.1) is subsumed by the first term as long as the signal-to-noise ratio is at most polynomial, i.e., $\|\widehat{x}\|_2 \leq n^{O(1)}\sigma$.

Assuming $\sqrt{n}$ is a power of 2, we present two new Sparse Fourier Transform algorithms:

- An $O(k \log k)$-time algorithm that uses only $O(k)$ samples for the exactly k-sparse case where $k = O(\sqrt{n})$, and
- An $O(k \log^2 n)$-time algorithm that uses only $O(k \log n)$ samples for the approximately sparse case where $k = \Theta(n)$.

The key feature of these algorithms is that their sample complexity bounds are optimal. For the exactly sparse case, the lower bound of $\Omega(k)$ is immediate. For the approximately sparse case, we note that the $\Omega(k \log(n/k))$ lower bound of [146] holds even if the spectrum is the sum of a k-sparse signal vector in $\{0, 1, -1\}^n$ and Gaussian noise. The latter is essentially a special case of the distributions handled by these algorithms.

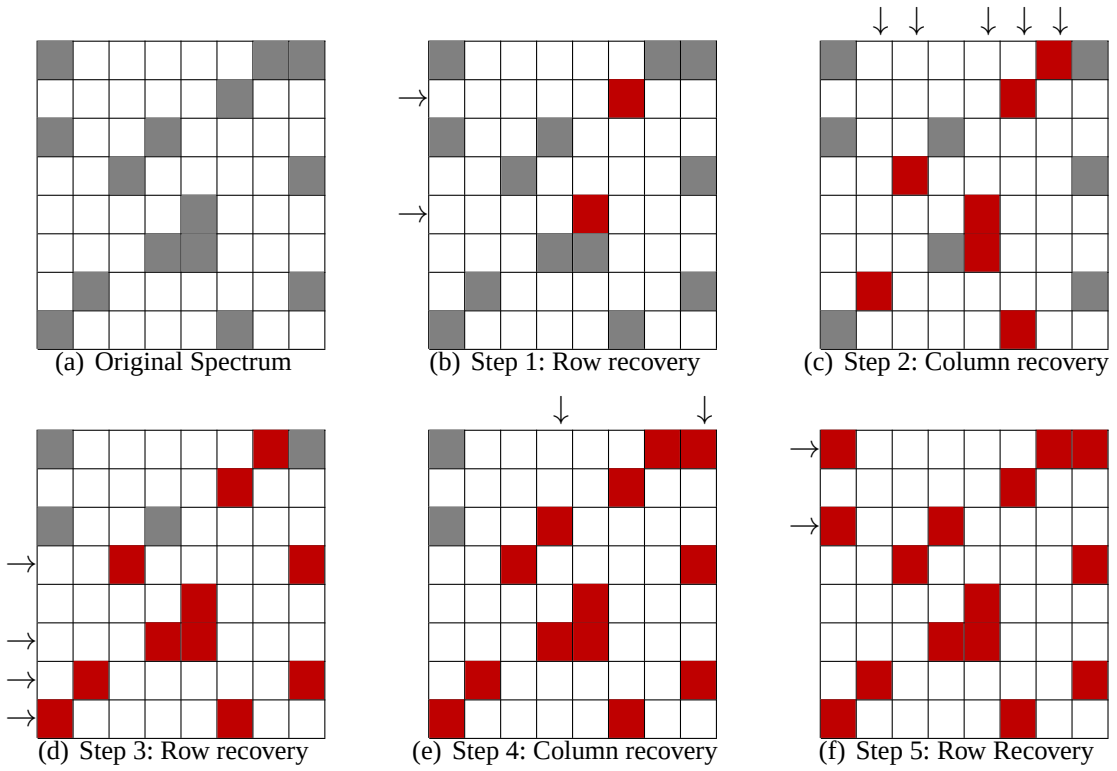
An additional feature of the first algorithm is its simplicity and therefore its low "big-Oh" overhead. As a result, this algorithm is easy to adapt for practical applications.

5.1.2 Techniques

Our first algorithm for k-sparse signals is based on the following observation: The aliasing filter (i.e., uniform sub-sampling) is one of the most efficient ways for mapping the Fourier coefficients

[1] Note that this model subsumes the scenario where the values of the nonzero coordinates are chosen i.i.d. from some distribution.

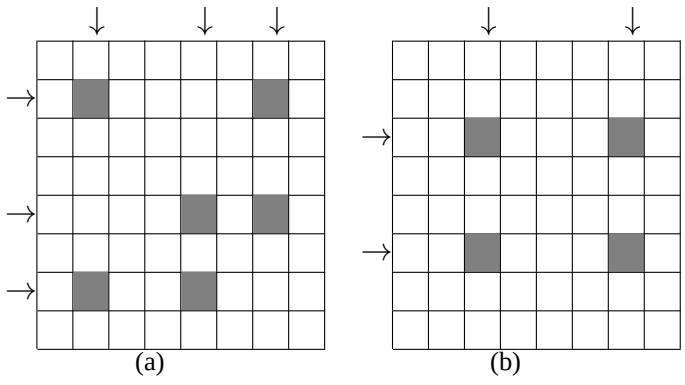
[2] A popular alternative is to use the hypergeometric distribution over the set of nonzero entries instead of the Bernoulli distribution. The advantage of the former is that it yields vectors of sparsity *exactly* equal to k. In this chapter we opted for the Bernoulli model since it is simpler to analyze. However, both models are quite similar. In particular, for large enough k, the actual sparsity of vectors in the Bernoulli model is sharply concentrated around k.



(a) Original Spectrum (b) Step 1: Row recovery (c) Step 2: Column recovery

(d) Step 3: Row recovery (e) Step 4: Column recovery (f) Step 5: Row Recovery

Figure 5-1: **An Illustration of the 2D Sparse Fourier Transform Algorithm.** This illustration shows the "peeling" recovery process on an 8×8 signal with 15 nonzero frequencies. In each step, the algorithm recovers all 1-sparse columns and rows (the recovered entries are depicted in red). The process converges after a few steps.



(a) (b)

Figure 5-2: **Examples of Obstructing Sequences of Non-zero Coefficients.** None of the remaining rows or columns has a sparsity of 1.

into buckets. For one-dimensional signals however, this filter is not amenable to randomization. Hence, when multiple nonzero Fourier coefficients collide into the same bucket, one cannot efficiently resolve the collisions by randomizing the spike-train filter. In contrast, for two-dimensional signals, we naturally obtain two distinct spike-train filters, which correspond to subsampling the columns and subsampling the rows. Hence, we can resolve colliding nonzero Fourier coefficients by alternating between these two filters.

More specifically, recall that one way to compute the two-dimensional DFT of a signal x is to apply the one-dimensional DFT to each column and then to each row. Suppose that $k = a\sqrt{n}$ for $a < 1$. In this case, the expected number of nonzero entries in each row is less than 1. If *every* row contained exactly one nonzero entry, then the DFT could be computed via the following two step process. In the first step, we select the first two columns of x, denoted by $u^{(0)}$ and $u^{(1)}$, and compute their DFTs $\widehat{u}^{(0)}$ and $\widehat{u}^{(1)}$. Let j_i be the index of the unique nonzero entry in the i-th row of $\widehat{x}$, and let a be its value. Observe that $\widehat{u}_i^{(0)} = a$ and $\widehat{u}_i^{(1)} = a(\omega')^{-j_i}$ (where ω' is a primitive $\sqrt{n}$-th root of unity), as these are the first two entries of the inverse Fourier transform of a 1-sparse signal ae_{j_i}. Thus, in the second step, we can retrieve the value of the nonzero entry, equal to $\widehat{u}_i^{(0)}$, as well as the index j_i from the phase of the ratio $\widehat{u}_i^{(1)}/\widehat{u}_i^{(0)}$. (We first introduced this technique in Chapter 4 and we referred to it as the "OFDM trick"). The total time is dominated by the cost of the two DFTs of the columns, which is $O(\sqrt{n}\log n)$. Since the algorithm queries only a constant number of columns, its sample complexity is $O(\sqrt{n})$.

In general, the distribution of the nonzero entries over the rows can be non-uniform –i.e., some rows may have multiple nonzero Fourier coefficients. Thus, our actual algorithm alternates the above recovery process between the columns and rows (see Figure 5-1 for an illustration). Since the OFDM trick works only on 1-sparse columns/rows, we check the 1-sparsity of each column/row by sampling a constant number of additional entries. We then show that, as long as the sparsity constant a is small enough, this process recovers all entries in a logarithmic number steps with constant probability. The proof uses the fact that the probability of the existence of an "obstructing configuration" of nonzero entries which makes the process deadlocked (e.g., see Figure 5-2) is upper bounded by a small constant.

The algorithm is extended to the case of $k = o(\sqrt{n})$ via a reduction. Specifically, we subsample the signal x by the reduction ratio $R = \alpha\sqrt{n}/k$ for some small enough constant α in each dimension. The subsampled signal x' has dimension $\sqrt{m} \times \sqrt{m}$, where $\sqrt{m} = \frac{k}{\alpha}$. Since subsampling in time domain corresponds to "spectrum folding", i.e., adding together all frequencies with indices that are equal modulo $\sqrt{m}$, the nonzero entries of $\widehat{x}$ are mapped into the entries of $\widehat{x'}$. It can be seen that, with constant probability, the mapping is one-to-one. If this is the case, we can use the earlier algorithm for sparse DFT to compute the nonzero frequencies in $O(\sqrt{m}\log m) = O(\sqrt{k}\log k)$ time, using $O(k)$ samples. We then use the OFDM trick to identify the positions of those frequencies.

Our second algorithm works for *approximately* sparse data, at sparsity $\Theta(\sqrt{n})$. Its general outline mimics that of the first algorithm. Specifically, it alternates between decoding columns and rows, assuming that they are 1-sparse. The decoding subroutine itself is similar to that of Algorithm 4.3.1 and uses $O(\log n)$ samples. The subroutine first checks whether the decoded entry is large; if not, the spectrum is unlikely to contain any large entry, and the subroutine terminates.

The algorithm then subtracts the decoded entry from the column and checks whether the resulting signal contains no large entries in the spectrum (which would be the case if the original spectrum was approximately 1-sparse and the decoding was successful). The check is done by sampling $O(\log n)$ coordinates and checking whether their sum of squares is small. To prove that this check works with high probability, we use the fact that a collection of random rows of the Fourier matrix is likely to satisfy the Restricted Isometry Property of [25].

A technical difficulty in the analysis of the algorithm is that the noise accumulates in successive iterations. This means that a $1/\log^{O(1)} n$ fraction of the steps of the algorithm will fail. However, we show that the dependencies are "local", which means that our analysis still applies to a vast majority of the recovered entries. We continue the iterative decoding for $\log \log n$ steps, which ensures that all but a $1/\log^{O(1)} n$ fraction of the large frequencies are correctly recovered. To recover the remaining frequencies, we resort to algorithms with worst-case guarantees.

5.1.3 Extensions

Our algorithms have natural extensions to dimensions higher than 2. We do not include them in this chapter as the description and analysis are rather cumbersome.

Moreover, due to the equivalence between the two-dimensional case and the one-dimensional case where n is a product of different prime powers [59, 90], our algorithms also give optimal sample complexity bounds for such values of n (e.g., $n = 6^t$) in the average case.

5.1.4 Distributions

In the exactly sparse case, we assume a Bernoulli model for the support of $\widehat{x}$. This means that for all $(i,j) \in [\sqrt{n}] \times [\sqrt{n}]$, $\Pr\{(i,j) \in \operatorname{supp}(\widehat{x})\} = k/n$ and thus $\mathbb{E}[|\operatorname{supp}(\widehat{x})|] = k$. We assume an unknown predefined matrix $a_{i,j}$ of values in $\mathbb{C}$; if $\widehat{x}_{i,j}$ is selected to be nonzero, its value is set to $a_{i,j}$.

In the approximately sparse case, we assume that the signal $\widehat{x}$ is equal to $\widehat{x^*} + \widehat{w} \in \mathbb{C}^{\sqrt{n} \times \sqrt{n}}$, where $\widehat{x^*}_{i,j}$ is the "signal" and $\widehat{w}$ is the "noise". In particular, $\widehat{x^*}$ is drawn from the Bernoulli model, where $\widehat{x^*}_{i,j}$ is drawn from $\{0, a_{i,j}\}$ at random independently for each (i,j) for some values $a_{i,j}$ and with $\mathbb{E}[|\operatorname{supp}(\widehat{x^*})|] = k$. We also require that $|a_{i,j}| \geq L$ for some parameter L. $\widehat{w}$ is a complex Gaussian vector with variance σ^2 in both the real and imaginary axes independently on each coordinate; we notate this as $\widehat{w} \sim N_{\mathbb{C}}(0, \sigma^2 I_n)$. We will need that $L = C\sigma\sqrt{n/k}$ for a sufficiently large constant C, so that $\mathbb{E}[\|\widehat{x^*}\|_2^2] \geq C\,\mathbb{E}[\|\widehat{w}\|_2^2]$.

We show in Appendix E that the sample lower bound of $\Omega(k \log(n/k))$ on ℓ_2/ℓ_2 recovery from [146] applies to the above Bernoulli model.

5.2 Algorithm for the Exactly Sparse Case

The algorithm for the noiseless case depends on the sparsity k where $k = \mathbb{E}[|\operatorname{supp}(\widehat{x})|]$ for a Bernoulli distribution of the support.

5.2.1 Exact Algorithm: $k = \Theta(\sqrt{n})$

In this section, we focus on the regime $k = \Theta(\sqrt{n})$. Specifically, we will assume that $k = a\sqrt{n}$ for a (sufficiently small) constant $a > 0$.

The algorithm BASICEXACT2DSFFT (SFT 5.0) is described as Algorithm 5.2.1. The key idea is to fold the spectrum into bins using the aliasing filter defined in Chapter 2 and estimate frequencies which are isolated in a bin. The algorithm takes the FFT of a row and as a result frequencies in the same columns will get folded into the same row bin. It also takes the FFT of a column and consequently frequencies in the same rows will get folded into the same column bin. The algorithm then uses the OFDM trick introduced in Chapter 4 to recover the columns and rows whose sparsity is 1. It iterates between the column bins and row bins, subtracting the recovered frequencies and estimating the remaining columns and rows whose sparsity is 1. An illustration of the algorithm running on an 8×8 signal with 15 nonzero frequencies is shown in Figure 5-1 in Section 5.1. The algorithm also takes a constant number of extra FFTs of columns and rows to check for collisions within a bin and avoid errors resulting from estimating bins where the sparsity is greater than 1. The algorithm uses three functions:

- FOLDTOBINS. This procedure folds the spectrum into $B_r \times B_c$ bins using the aliasing filter described in Chapter 2.

- BASICESTFREQ. Given the FFT of rows or columns, it estimates the frequency in the large bins. If there is no collision, i.e. if there is a single nonzero frequency in the bin, it adds this frequency to the result $\widehat{w}$ and subtracts its contribution to the row and column bins.

- BASICEXACT2DSFFT. This performs the FFT of the rows and columns and then iterates BASICESTFREQ between the rows and columns until is recovers $\widehat{x}$.

Analysis of BASICEXACT2DSFFT

Lemma 5.2.1. *For any constant $\alpha > 0$, if $a > 0$ is a sufficiently small constant, then assuming that all 1-sparsity tests in the procedure* BASICESTFREQ *are correct, the algorithm reports the correct output with probability at least $1 - O(\alpha)$.*

Proof. The algorithm fails if there is a pair of nonzero entries in a column or row of $\widehat{x}$ that "survives" $t_{max} = C \log n$ iterations. For this to happen there must be an "obstructing" sequence of nonzero entries $p_1, q_1, p_2, q_2 \ldots p_t$, $3 \leq t \leq t_{max}$, such that for each $i \geq 1$, p_i and q_i are in the same column ("vertical collision"), while q_i and p_{i+1} are in the same row ("horizontal collision"). Moreover, it must be the case that either the sequence "loops around", i.e., $p_1 = p_t$, or $t > t_{max}$. We need to prove that the probability of either case is less than α. We focus on the first case; the second one is similar.

Assume that there is a sequence $p_1, q_1, \ldots p_{t-1}, q_{t-1}$ such that the elements in this sequence are all distinct, while $p_1 = p_t$. If such a sequence exists, we say that the event E_t holds. The number of sequences satisfying E_t is at most $\sqrt{n}^{2(t-1)}$, while the probability that the entries corresponding

procedure FOLDTOBINS(x, B_r, B_c, τ_r, τ_c)
  $y_{i,j} = x_{i(\sqrt{n}/B_r)+\tau_r, j(\sqrt{n}/B_c)+\tau_c}$ for $(i,j) \in [B_r] \times [B_c]$,
  return $\widehat{y}$, the DFT of y
procedure BASICESTFREQ($\widehat{u}^{(T)}$, $\widehat{v}^{(T)}$, T, IsCol)
  $\widehat{w} \leftarrow 0$.
  Compute $J = \{j : \sum_{\tau \in T} |\widehat{u}_j^{(\tau)}| > 0\}$.
  for $j \in J$ **do**
    $b \leftarrow \widehat{u}_j^{(1)} / \widehat{u}_j^{(0)}$.
    $i \leftarrow \text{round}(\phi(b)\frac{\sqrt{n}}{2\pi}) \bmod \sqrt{n}$. ▷ $\phi(b)$ is the phase of b.
    $s \leftarrow \widehat{u}_j^{(0)}$.
    ▷ Test whether the row or column is 1-sparse
    if $\left(\sum_{\tau \in T} |\widehat{u}_j^{(\tau)} - s(\omega')^{-\tau i}| == 0\right)$ **then**
      if IsCol **then** ▷ whether decoding column or row
        $\widehat{w}_{i,j} \leftarrow s$.
      else
        $\widehat{w}_{j,i} \leftarrow s$.
      for $\tau \in T$ **do**
        $\widehat{u}_j^{(\tau)} \leftarrow 0$
        $\widehat{v}_i^{(\tau)} \leftarrow \widehat{v}_i^{(\tau)} - s(\omega')^{-\tau i}$
  return $\widehat{w}$, $\widehat{u}^{(T)}$, $\widehat{v}^{(T)}$
procedure BASICEXACT2DSFFT(x, k)
  $T \leftarrow [2c]$ ▷ We set $c \geq 6$
  for $\tau \in T$ **do**
    $\widehat{u}^{(\tau)} \leftarrow$ FOLDTOBINS$(x, \sqrt{n}, 1, 0, \tau)$.
    $\widehat{v}^{(\tau)} \leftarrow$ FOLDTOBINS$(x, 1, \sqrt{n}, \tau, 0)$.
  $\widehat{z} \leftarrow 0$
  for $t \in [C \log n]$ **do** ▷ $\widehat{u}^{(T)} := \{\widehat{u}^{(\tau)} : \tau \in T\}$
    $\{\widehat{w}, \widehat{u}^{(T)}, \widehat{v}^{(T)}\} \leftarrow$ BASICESTFREQ$(\widehat{u}^{(T)}, \widehat{v}^{(T)}, T,$ true$)$.
    $\widehat{z} \leftarrow \widehat{z} + \widehat{w}$.
    $\{\widehat{w}, \widehat{v}^{(T)}, \widehat{u}^{(T)}\} \leftarrow$ BASICESTFREQ$(\widehat{v}^{(T)}, \widehat{u}^{(T)}, T,$ false$)$.
    $\widehat{z} \leftarrow \widehat{z} + \widehat{w}$.
  return $\widehat{z}$

5.2.1: SFT 5.0: Exact 2D Sparse Fourier Transform for $k = \Theta(\sqrt{n})$

to the points in a specific sequence are nonzero is at most $(k/n)^{2(t-1)} = (a/\sqrt{n})^{2(t-1)}$. Thus the probability of E_t is at most

$$\sqrt{n}^{2(t-1)} \cdot (a/\sqrt{n})^{2(t-1)} = a^{2(t-1)}.$$

Therefore, the probability that one of the events $E_1, \ldots, E_{t_{max}}$ holds is at most $\sum_{t=3}^{\infty} a^{2(t-1)} = a^4/(1-a^2)$, which is smaller than α for a small enough. □

Lemma 5.2.2. *The probability that any 1-sparsity test invoked by the algorithm is incorrect is at most* $O(1/n^{(c-5)/2})$.

The proof can be found in Appendix A.12.

Theorem 5.2.3. *For any constant* α*, the algorithm* BASICEXACT2DSFFT *uses* $O(\sqrt{n})$ *samples, runs in time* $O(\sqrt{n}\log n)$ *and returns the correct vector* $\widehat{x}$ *with probability at least* $1 - O(\alpha)$ *as long as* a *is a small enough constant.*

Proof. From Lemma 5.2.1 and Lemma 5.2.2, the algorithm returns the correct vector $\widehat{x}$ with probability at least $1 - O(\alpha) - O(n^{-(c-5)/2}) = 1 - O(\alpha)$ for $c > 5$.

The algorithm uses only $O(T) = O(1)$ rows and columns of x, which yields $O(\sqrt{n})$ samples. The running time is bounded by the time needed to perform $O(1)$ FFTs of rows and columns (in FOLDTOBINS) procedure, and $O(\log n)$ invocations of BASICESTFREQ. Both components take time $O(\sqrt{n}\log n)$.

□

5.2.2 Reduction to the Exact Algorithm: $k = o(\sqrt{n})$

Algorithm REDUCEEXACT2DSFFT (SFT 5.1), which is for the case where $k = o(\sqrt{n})$, is described in Algorithm 5.2.2. The key idea is to reduce the problem from the case where $k = o(\sqrt{n})$ to the case where $k = \Theta(\sqrt{n})$. To do that, we subsample the input time domain signal x by the reduction ratio $R = a\sqrt{n}/k$ for some small enough a. The subsampled signal x' has dimension $\sqrt{m} \times \sqrt{m}$, where $\sqrt{m} = \frac{k}{a}$. This implies that the probability that any coefficient in $\widehat{x}'$ is nonzero is at most $R^2 \times k/n = a^2/k = (a^2/k) \times (k^2/a^2)/m = k/m$, since $m = k^2/a^2$. This means that we can use the algorithm BASICNOISELESS2DSFFT in Section 5.2.1 to recover $\widehat{x}'$. Each of the entries of $\widehat{x}'$ is a frequency in $\widehat{x}$ which was folded into $\widehat{x}'$. We employ the same phase technique used in Section 5.2.1 to recover their original frequency position in $\widehat{x}$.

The algorithm uses 2 functions:

- REDUCETOBASICSFFT: This folds the spectrum into $O(k) \times O(k)$ dimensions and performs the reduction to BASICEXACT2DSFFT. Note that only the $O(k)$ elements of x' which will be used in BASICEXACT2DSFFT need to be computed.

- REDUCEEXACT2DSFFT: This invokes the reduction as well as the phase technique to recover $\widehat{x}$.

Analysis of REDUCEEXACT2DSFFT

Lemma 5.2.4. *For any constant* α*, for sufficiently small* a *there is a one-to-one mapping of frequency coefficients from* $\widehat{x}$ *to* $\widehat{x}'$ *with probability at least* $1 - \alpha$.

procedure REDUCETOBASICSFFT(x, R, τ_r, τ_c)
  Define $x'_{ij} = x_{iR+\tau_r, jR+\tau_c}$ ▷ With lazy evaluation
  return BASICEXACT2DSFFT(x', k)
procedure REDUCEEXACT2DSFFT(x, k)
  $R \leftarrow \frac{a\sqrt{n}}{k}$, for some constant $a < 1$ such that $R|\sqrt{n}$.
  $\widehat{u}^{(0,0)} \leftarrow$ REDUCETOBASICSFFT$(x, R, 0, 0)$
  $\widehat{u}^{(1,0)} \leftarrow$ REDUCETOBASICSFFT$(x, R, 1, 0)$
  $\widehat{u}^{(0,1)} \leftarrow$ REDUCETOBASICSFFT$(x, R, 0, 1)$
  $\widehat{z} \leftarrow 0$
  $L \leftarrow \text{supp}(\widehat{u}^{(0,0)}) \cap \text{supp}(\widehat{u}^{(1,0)}) \cap \text{supp}(\widehat{u}^{(0,1)})$
  for $(\ell, m) \in L$ **do**
    $b_r \leftarrow \widehat{u}^{(1,0)}_{\ell,m} / \widehat{u}^{(0,0)}_{\ell,m}$
    $i \leftarrow \text{round}(\phi(b_r)\frac{\sqrt{n}}{2\pi}) \bmod \sqrt{n}$
    $b_c \leftarrow \widehat{u}^{(0,1)}_{\ell,m} / \widehat{u}^{(0,0)}_{\ell,m}$
    $j \leftarrow \text{round}(\phi(b_c)\frac{\sqrt{n}}{2\pi}) \bmod \sqrt{n}$
    $\widehat{z}_{ij} \leftarrow \widehat{u}^{(0,0)}_{\ell,m}$
  return $\widehat{z}$

5.2.2: SFT 5.1: Exact 2D Sparse Fourier Transform for $k = o(\sqrt{n})$

Proof. The probability that there are at least 2 nonzero coefficients among the R^2 coefficients in $\widehat{x}$ that are folded together in $\widehat{x}'$, is at most

$$\binom{R^2}{2}(k/n)^2 < (a^2 n/k^2)^2 (k/n)^2 = a^4/k^2$$

The probability that this event holds for any of the m positions in $\widehat{x}'$ is at most $ma^4/k^2 = (k^2/a^2)a^4/k^2 = a^2$ which is less than α for small enough a. Thus, with probability at least $1 - \alpha$ any nonzero coefficient in $\widehat{x}'$ comes from only one nonzero coefficient in $\widehat{x}$. □

Theorem 5.2.5. *For any constant $\alpha > 0$, there exists a constant $c > 0$ such that if $k < c\sqrt{n}$ then the algorithm* REDUCEEXACT2DSFFT *uses $O(k)$ samples, runs in time $O(k \log k)$ and returns the correct vector $\widehat{x}$ with probability at least $1 - \alpha$.*

Proof. By Theorem 5.2.3 and the fact that each coefficient in $\widehat{x}'$ is nonzero with probability $O(1/k)$, each invocation of the function REDUCETOBASICSFFT fails with probability at most α. By Lemma 5.2.4, with probability at least $1 - \alpha$, we could recover $\widehat{x}$ correctly if each of the calls to REDTOBASICSFFT returns the correct result. By the union bound, the algorithm REDUCEEXACT2DSFFT fails with probability at most $\alpha + 3 \times \alpha = O(\alpha)$.

The algorithm uses $O(1)$ invocations of BASICEXACT2DSFFT on a signal of size $O(k) \times O(k)$ in addition to $O(k)$ time to recover the support using the OFDM trick. Noting that calculating the intersection L of supports takes $O(k)$ time, the stated number of samples and running time then follow directly from Theorem 5.2.3. □

5.3 Algorithm for the General Case

The algorithm for noisy recovery ROBUST2DSFFT (SFT 6.0) is shown in Algorithm 5.3.1. The algorithm is very similar to the exactly sparse case. It first takes FFT of rows and columns using FOLDTOBINS procedure. It then iterates between the columns and rows, recovering frequencies in bins which are 1-sparse using the ROBUSTESTIMATECOL procedure. This procedure uses the function LOCATESIGNAL from Algorithm 4.3.2 to make the estimation of the frequency positions robust to noise.

Preliminaries

Following [25], we say that a matrix A satisfies a *Restricted Isometry Property* (RIP) of order t with constant $\delta > 0$ if, for all t-sparse vectors y, we have $\|Ay\|_2^2/\|y\|_2^2 \in [1-\delta, 1+\delta]$.

Suppose all columns A_i of an $N \times M$ matrix A have unit norm. Let $\mu = \max_{i \neq j} |A_i \cdot A_j|$ be the *coherence* of A. It is folklore[3] that A satisfies the RIP of order t with the constant $\delta = (t-1)\mu$.

Suppose that the matrix A is an $M \times N$ submatrix of the $N \times N$ Fourier matrix F, with each the M rows of A chosen uniformly at random from the rows of F. It is immediate from the Hoeffding bound that if $M = b\mu^2 \log(N/\gamma)$ for some large enough constant $b > 1$ then the matrix A has coherence at most μ with probability $1-\gamma$. Thus, for $M = \Theta(t^2 \cdot t \log N)$, A satisfies the RIP of order t with constant $\delta = 0.5$ with probability $1 - 1/N^t$.

5.3.1 Analysis of Each Stage of Recovery

Here, we show that each step of the recovery is correct with high probability using the following two lemmas. The first lemma shows that with very low probability the ROBUSTESTIMATECOL procedure generates a false negative (misses a frequency), false positive (adds a fake frequency) or a bad update (wrong estimate of a frequency). The second lemma is analogous to Lemma 5.2.2 and shows that the probability that the 1-sparse test fails when there is noise is low.

Lemma 5.3.1. *Consider the recovery of a column/row j in* ROBUSTESTIMATECOL*, where $\widehat{u}$ and $\widehat{v}$ are the results of* FOLDTOBINS *on $\widehat{x}$. Let $y \in \mathbb{C}^{\sqrt{n}}$ denote the jth column/row of $\widehat{x}$. Suppose y is drawn from a permutation invariant distribution $y = y^{head} + y^{residue} + y^{gauss}$, where $\min_{i \in \mathrm{supp}(y^{head})} |y_i| \geq L$, $\|y^{residue}\|_1 < \epsilon L$, and y^{gauss} is drawn from the $\sqrt{n}$-dimensional normal distribution $N_{\mathbb{C}}(0, \sigma^2 I_{\sqrt{n}})$ with standard deviation $\sigma = \epsilon L/n^{1/4}$ in each coordinate on both real and imaginary axes. We do not require that y^{head}, $y^{residue}$, and y^{gauss} are independent except for the permutation invariance of their sum.*

Consider the following bad events:

- *False negative:* $\mathrm{supp}(y^{head}) = \{i\}$ *and* ROBUSTESTIMATECOL *does not update coordinate i.*
- *False positive:* ROBUSTESTIMATECOL *updates some coordinate i but* $\mathrm{supp}(y^{head}) \neq \{i\}$.
- *Bad update:* $\mathrm{supp}(y^{head}) = \{i\}$ *and coordinate i is estimated by b with* $\left|b - y_i^{head}\right| > \|y^{residue}\|_1 + \sqrt{\frac{\log\log n}{\log n}}\epsilon L$.

[3] It is a direct corollary of Gershgorin's theorem applied to any t columns of A.

procedure ROBUSTESTIMATECOL($\widehat{u}$, $\widehat{v}$, T, T', IsCol, J, Ranks)
  $\widehat{w} \leftarrow 0$.
  $S \leftarrow \{\}$ ▷ Set of changes, to be tested next round.
  for $j \in J$ **do**
    continue if Ranks[(IsCol, j)] $\geq \log\log n$.
    $i \leftarrow$ LOCATESIGNAL($\widehat{u}^{(T')}$, T')
      ▷ Procedure from Algorithm 4.3.2: $O(\log^2 n)$ time
    $a \leftarrow \text{median}_{\tau \in T}\, \widehat{u}_j^{\tau}(\omega')^{\tau i}$.
    continue if $|a| < L/2$
      ▷ Nothing significant recovered
    continue if $\sum_{\tau \in T} |\widehat{u}_j^{\tau} - a(\omega')^{-\tau i}|^2 \geq L^2\,|T|\,/10$
      ▷ Bad recovery: probably not 1-sparse
    $b \leftarrow \text{mean}_{\tau \in T}\, \widehat{u}_j^{\tau}(\omega')^{\tau i}$.
    if IsCol **then** ▷ whether decoding column or row
      $\widehat{w}_{i,j} \leftarrow b$.
    else
      $\widehat{w}_{j,i} \leftarrow b$.
    $S \leftarrow S \cup \{i\}$.
    Ranks[(1 − IsCol, i)] += Ranks[(IsCol, j)].
    for $\tau \in T \cup T'$ **do**
      $\widehat{u}_j^{(\tau)} \leftarrow \widehat{u}_j^{(\tau)} - b(\omega')^{-\tau i}$
      $\widehat{v}_i^{(\tau)} \leftarrow \widehat{v}_i^{(\tau)} - b(\omega')^{-\tau i}$
  return $\widehat{w}$, $\widehat{u}$, $\widehat{v}$, S
procedure ROBUST2DSFFT(x, k)
  $T, T' \subset [\sqrt{n}], |T| = |T'| = O(\log n)$
  for $\tau \in T \cup T'$ **do**
    $\widehat{u}^{(\tau)} \leftarrow$ FOLDTOBINS($x, \sqrt{n}, 1, 0, \tau$).
    $\widehat{v}^{(\tau)} \leftarrow$ FOLDTOBINS($x, 1, \sqrt{n}, \tau, 0$).
  $\widehat{z} \leftarrow 0$
  $R \leftarrow 1^{[2]\times[\sqrt{n}]}$ ▷ Rank of vertex (iscolumn, index)
  $S_{col} \leftarrow [\sqrt{n}]$ ▷ Which columns to test
  for $t \in [C \log n]$ **do**
    $\{\widehat{w}, \widehat{u}, \widehat{v}, S_{row}\} \leftarrow$ ROBUSTESTIMATECOL($\widehat{u}, \widehat{v}, T, T'$, true, S_{col}, R).
    $\widehat{z} \leftarrow \widehat{z} + \widehat{w}$.
    $S_{row} \leftarrow [\sqrt{n}]$ if $t = 0$ ▷ Try each row the first time
    $\{\widehat{w}, \widehat{v}, \widehat{u}, S_{col}\} \leftarrow$ ROBUSTESTIMATECOL($\widehat{v}, \widehat{u}, T, T'$, false, S_{row}, R).
    $\widehat{z} \leftarrow \widehat{z} + \widehat{w}$.
  return $\widehat{z}$

5.3.1: SFT 6.0: General 2D Sparse Fourier Transform for $k = \Theta(\sqrt{n})$

For any constant c and ϵ below a sufficiently small constant, there exists a distribution over sets T, T' of size $O(\log n)$, such that as a distribution over y and T, T' we have

- *The probability of a false negative is $1/\log^c n$.*
- *The probability of a false positive is $1/n^c$.*
- *The probability of a bad update is $1/\log^c n$.*

The proof can be found in Appendix A.13.

Lemma 5.3.2. *Let $y \in \mathbb{C}^m$ be drawn from a permutation invariant distribution with $r \geq 2$ nonzero values. Suppose that all the nonzero entries of y have absolute value at least L. Choose $T \subset [m]$ uniformly at random with $t := |T| = O(c^3 \log m)$.*

Then, the probability that there exists a y' with $\|y'\|_0 \leq 1$ and

$$\|(\check{y} - \check{y}')_T\|_2^2 < \epsilon L^2 t/n$$

is at most $c^3(\frac{c}{m-r})^{c-2}$ whenever $\epsilon < 1/8$.

Proof. The proof can be found in Appendix A.14. □

5.3.2 Analysis of Overall Recovery

Recall that we are considering the recovery of a signal $\widehat{x} = \widehat{x^*} + \widehat{w} \in \mathbb{C}^{\sqrt{n}\times\sqrt{n}}$, where $\widehat{x^*}$ is drawn from the Bernoulli model with expected $k = a\sqrt{n}$ nonzero entries for a sufficiently small constant a, and $\widehat{w} \sim N_{\mathbb{C}}(0, \sigma^2 I_n)$ with $\sigma = \epsilon L\sqrt{k/n} = \Theta(\epsilon L/n^{1/4})$ for sufficiently small ϵ.

It will be useful to consider a bipartite graph representation G of $\widehat{x^*}$. We construct a bipartite graph with $\sqrt{n}$ nodes on each side, where the left side corresponds to rows and the right side corresponds to columns. For each $(i, j) \in \mathrm{supp}(\widehat{x^*})$, we place an edge between left node i and right node j of weight $\widehat{x^*}_{(i,j)}$.

Our algorithm is a "peeling" procedure on this graph. It iterates over the vertices, and can with a "good probability" recover an edge if it is the only incident edge on a vertex. Once the algorithm recovers an edge, it can remove it from the graph. The algorithm will look at the column vertices, then the row vertices, then repeat; these are referred to as *stages*. Supposing that the algorithm succeeds at recovery on each vertex, this gives a canonical order to the removal of edges. Call this the *ideal* ordering.

In the ideal ordering, an edge e is removed based on one of its incident vertices v. This happens after all other edges reachable from v without passing through e are removed. Define the *rank* of v to be the number of such reachable edges, and $\mathrm{rank}(e) = \mathrm{rank}(v) + 1$ (with $\mathrm{rank}(v)$ undefined if v is not used for recovery of any edge).

Lemma 5.3.3. *Let c, α be arbitrary constants, and a be a sufficiently small constant depending on c, α. Then with $1 - \alpha$ probability every component in G is a tree and at most $k/\log^c n$ edges have rank at least $\log\log n$.*

Proof. Each edge of G appears independently with probability $k/n = a/\sqrt{n}$. There are at most $\sqrt{n}^t$ cycles of length t. The probability that any cycle of length t exists is at most a^t, so the chance any cycle exists is less than $a^2/(1-a^2) < \alpha/2$ for sufficiently small a.

Each vertex has expected degree $a < 1$. Exploring the component for any vertex v is then a subcritical branching process, so the probability that v's component has size at least $\log\log n$ is $1/\log^c n$ for sufficiently small a. Then for each edge, we know that removing it causes each of its two incident vertices to have component size less than $\log\log n - 1$ with $1 - 1/\log^c n$ probability. Since the rank is one more than the size of one of these components, the rank is less than $\log\log n$ with $1 - 2/\log^c n$ probability.

Therefore, the expected number of edges with rank at least $\log\log n$ is $2k/\log^c n$. Hence, with probability $1 - \alpha/2$ there are at most $(1/\alpha)4k/\log^c n$ such edges; adjusting c gives the result. □

Lemma 5.3.4. *Let* ROBUST2DSFFT' *be a modified* ROBUST2DSFFT *that avoids false negatives or bad updates: whenever a false negative or bad update would occur, an oracle corrects the algorithm. With large constant probability,* ROBUST2DSFFT' *recovers $\widehat{z}$ such that there exists a $(k/\log^c n)$-sparse $\widehat{z}'$ satisfying*

$$\|\widehat{z} - \widehat{x} - \widehat{z}'\|_2^2 \leq 6\sigma^2 n.$$

Furthermore, only $O(k/\log^c n)$ false negatives or bad updates are caught by the oracle.

Proof. The proof can be found in Appendix A.15 □

Lemma 5.3.5. *For any constant $\alpha > 0$, the algorithm* ROBUST2DSFFT *can with probability $1-\alpha$ recover $\widehat{z}$ such that there exists a $(k/\log^{c-1} n)$-sparse $\widehat{z}'$ satisfying*

$$\|\widehat{z} - \widehat{x} - \widehat{z}'\|_2^2 \leq 6\sigma^2 n$$

using $O(k\log n)$ samples and $O(k\log^2 n)$ time.

Proof. To do this, we will show that changing the effect of a single call to ROBUSTESTIMATECOL can only affect $\log n$ positions in the output of ROBUST2DSFFT. By Lemma 5.3.4, we can, with large constant probability, turn ROBUST2DSFFT into ROBUST2DSFFT' with only $O(k/\log^c n)$ changes to calls to ROBUSTESTIMATECOL. This means the outputs of ROBUST2DSFFT and of ROBUST2DSFFT' only differ in $O(k/\log^{c-1} n)$ positions.

We view ROBUSTESTIMATECOL as trying to estimate a vertex. Modifying it can change from recovering one edge (or none) to recovering a different edge (or none). Thus, a change can only affect at most two calls to ROBUSTESTIMATECOL in the next stage. Hence in r stages, at most 2^{r-1} calls may be affected, so at most 2^r edges may be recovered differently.

Because we refuse to recover any edge with rank at least $\log\log n$, the algorithm has at most $\log\log n$ stages. Hence at most $\log n$ edges may be recovered differently as a result of a single change to ROBUSTESTIMATECOL. □

Theorem 5.3.6. *Our overall algorithm can recover $\widehat{x}'$ satisfying*

$$\|\widehat{x} - \widehat{x}'\|_2^2 \leq 12\sigma^2 n + \|\widehat{x}\|_2^2/n^c$$

with probability $1 - \alpha$ *for any constants* $c, \alpha > 0$ *in* $O(k \log n)$ *samples and* $O(k \log^2 n)$ *time, where* $k = a\sqrt{n}$ *for some constant* $a > 0$*.*

Proof. By Lemma 5.3.5, we can recover an $O(k)$-sparse $\widehat{z}$ such that there exists an $(k/\log^{c-1} n)$-sparse $\widehat{z}'$ with

$$\|\widehat{x} - \widehat{z} - \widehat{z}'\|_2^2 \leq 6\sigma^2 n.$$

with arbitrarily large constant probability for any constant c using $O(k \log^2 n)$ time and $O(k \log n)$ samples. Then by Theorem 4.4.1, we can recover a $\widehat{z}'$ in $O(k \log^2 n)$ time and $O(k \log^{4-c} n)$ samples satisfying

$$\|\widehat{x} - \widehat{z} - \widehat{z}'\|_2^2 \leq 12\sigma^2 n + \|\widehat{x}\|_2^2 / n^c$$

and hence $\widehat{x}' := \widehat{z} + \widehat{z}'$ is a good reconstruction for $\widehat{x}$. □

Chapter 6

Numerical Evaluation

In this chapter, we simulate and numerically evaluate the performance of some of our Sparse Fourier Transform algorithms.

6.1 Implementation

We implement the following algorithms: SFT 1.0, SFT 2.0, and a variant of SFT 3.0 which we will refer to as SFT 3.1. We implement them in C++ using the Standard Template Library. The code can be found on the Sparse Fourier Transform webpage: http://www.sparsefft.com.

We evaluate the performance and compare the following six implementations:

1) **SFT 1.0**: This algorithm was presented in Chapter 3 and has a runtime of $O(\log n\sqrt{nk\log n})$.

2) **SFT 2.0**: This algorithm was also presented in Chapter 3 and has a runtime of $O(\log n\sqrt[3]{nk^2\log n})$.

3) **SFT 3.1**:[1] A variant of SFT 3.0 which was presented in Chapter 4 and has a runtime of $O(k\log n)$.

4) **AAFFT 0.9** [87]: This is an implementation of the prior sublinear algorithm which had the fastest theoretical [60] and empirical runtime [89] before our SFT algorithms. The algorithm has a runtime of $O(k\log^c(n)\log(n/k))$ for some $c > 2$.

5) **FFTW 3.2.2** [54]: This is the fastest public implementation for the FFT algorithm which has a runtime of $O(n\log n)$.

6) **FFTW Optimized** [54]: This is an optimized version of FFTW that requires preprocessing, during which the algorithm is tuned to a particular machine hardware. In contrast, our current implementations of SFT algorithms do not perform hardware specific optimizations.

[1] In this variant, an aliasing filter is used in the very first iteration of the algorithm followed by two Gaussian flat window filters as opposed to only using Gaussian filters in SFT 3.0.

6.2 Experimental Setup

The test signals are generated in a manner similar to that in [89]. For the runtime experiments, k frequencies are selected uniformly at random from $[n]$ and assigned a magnitude of 1 and a uniformly random phase. The rest are set to zero. For the tolerance to noise experiments, the test signals are generated as before but they are combined with additive white Gaussian noise, whose variance varies depending on the desired SNR. Each point in the graphs is the average over 100 runs with different instances of test signals and different instances of noise. In all experiments, the parameters of SFT 1.0, SFT 2.0, SFT 3.1 and AAFFT 0.9 are chosen so that the average L^1 error in the absence of noise is between 10^{-7} and 10^{-8} per non-zero frequency.[2] Finally, all experiments are run on a Dual Core Intel 3.0 GHz CPU running Ubuntu Linux 10.04 with a cache size of 6144 KB and 8 GB of RAM.

6.3 Numerical Results

6.3.1 Runtime vs. Signal Size

In this experiment, we fix the sparsity parameter $k = 50$ and report the runtime of the compared algorithms for 12 different signal sizes $n : 2^{14}, 2^{15}, ..., 2^{26}$. We plot, in Figure 6-1, the mean, maximum, and minimum runtimes for SFT 1.0, SFT 2.0, SFT 3.1, AAFFT 0.9, FFTW, and FFTW OPT, over 100 runs. The relative runtimes of AAFFT 0.9 and FFTW are consistent with those reported in [89](see Figure 3.1).

As expected, Figure 6-1 shows that the runtimes of SFT 1.0, SFT 2.0 and FFTW are approximately linear in the log scale as a function of n. However, the slope of the lines for SFT 1.0 and SFT 2.0 is less than the slope for FFTW, which is a result of their sub-linear runtime. On the other hand, SFT 3.1 and AAFFT 0.9 appear almost constant in the log scale as a function of n which is also expected since they only depend logarithmically on n. Further, the figure shows that for signal sizes $n > 16384$, SFT 3.0 has the fastest runtime. It is faster than both variants of FFTW and is $400\times$ faster than AAFFT 0.9. SFT 1.0 and SFT 2.0 are faster than FFTW for $n > 100,000$ and faster than AAFFT for $n < 2^{26}$. Overall, for a large range of signal sizes the SFT algorithms have the fastest runtime.

6.3.2 Runtime vs. Sparsity

In this experiment, we fix the signal size to $n = 2^{22}$ (*i.e.* 4,194,304) and evaluate the runtime verses the number of non-zero frequencies k. For each value of k, the experiment is repeated 100 times. Figure 6-1 illustrates the mean, maximum, and minimum runtimes for the compared algorithms.

Figure 6-1 shows that SFT 1.0 and SFT 2.0 have a faster runtime than basic FFTW for k up to 2000 and 2200, respectively. When compared to the optimized FFTW, the crossing values become 500 and 1000. Thus, SFT's crossing values are around $\sqrt{n}$. In comparison, AAFFT 0.9

[2] For the values of k and n that are close to the ones considered in [89], we use the parameters therein. For other ranges, we follow the guidelines in the AAFFT 0.9 documentation [87].

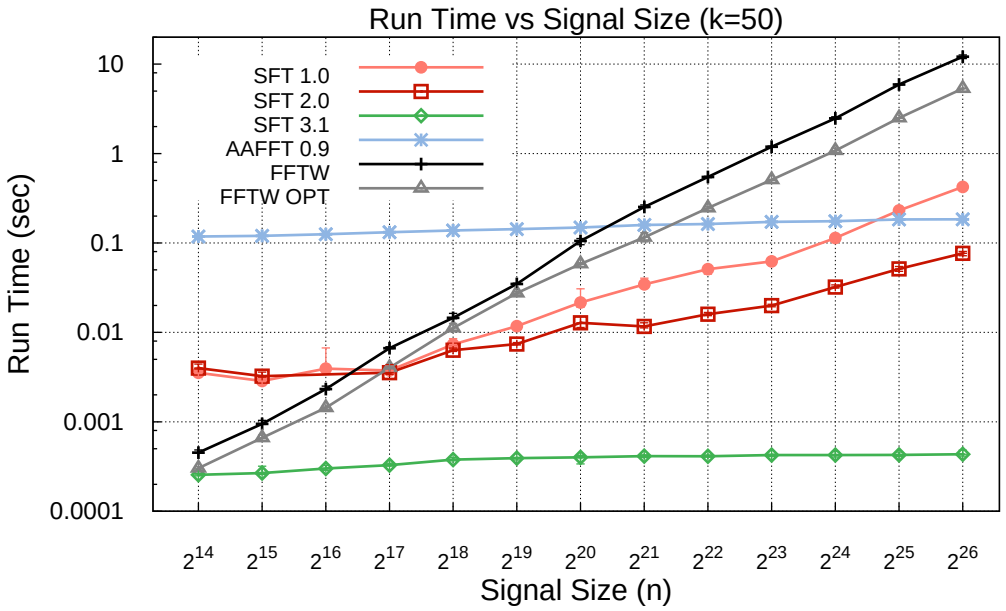


Figure 6-1: **Runtime vs. Signal Size** The figure shows that for a large range of signal sizes, SFT is faster than FFTW and the state-of-the-art sublinear algorithm.

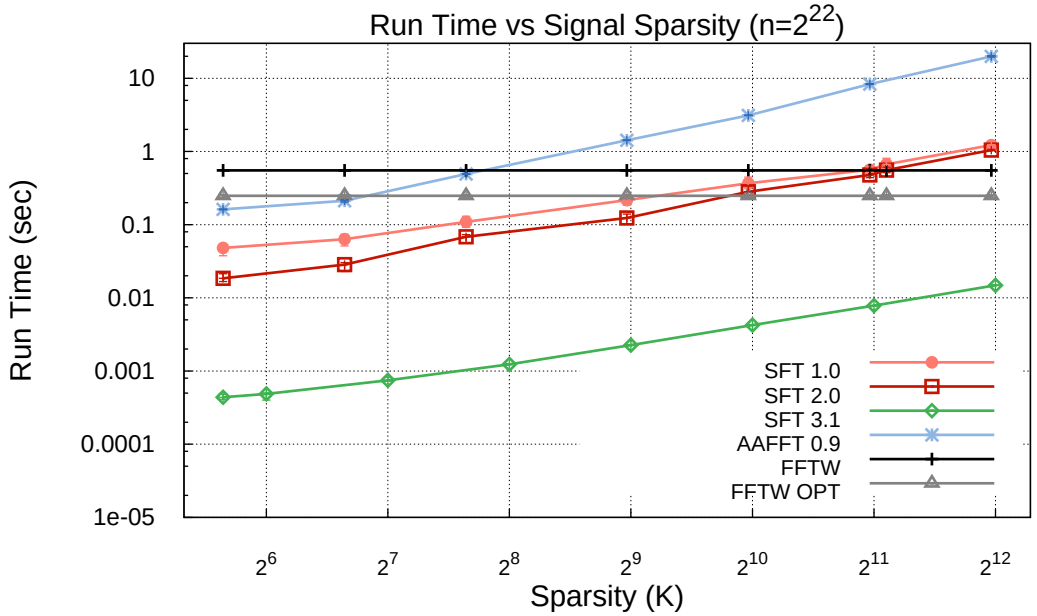


Figure 6-2: **Runtime vs. Signal Sparsity** SFT significantly extends the range of applications for which sparse approximation of Fourier transform is practical, and beats the runtime of FFTW for values of k which are orders of magnitude larger than those achieved by past work.

is faster than FFTW variants for k between 100 and 200. Further, the relative runtimes of AAFFT 0.9, and FFTW 3.2.2 are close to those reported in [89](Figure 3.2). The figure also shows that SFT 3.1 has the fastest runtime among all algorithms. It is almost two orders of magnitude faster than SFT 1.0 and SFT 2.0 and more than $400\times$ faster than AAFFT. Finally, FFTW has a runtime of $O(n\log(n))$, which is independent of the number of non-zero frequencies k, as can be seen in Figure 6-1. Thus, as the sparsity of the signal decreases (i.e., k increases), FFTW eventually becomes faster than all SFT and AAFFT. In fact, the FFTW will become faster than SFT 3.1 for $k > 131072$. Nonetheless, the results show that in comparison with the fastest prior sublinear algorithm [89], the SFT algorithms significantly extend the range of applications for which sparse approximation of Fourier transform is practical.

6.3.3 Robustness to Noise

Last, we would like to check SFT's robustness to noise. Thus, we compare the performance of SFT 1.0 and SFT 2.0 against AAFFT 0.9, for different levels of white Gaussian noise. For this experiment, we exclude SFT 3.1 since it only works for exactly sparse signals. We fix $n = 2^{22}$ and $k = 50$, and experiment with different signal SNRs.[3] We change the SNR by changing the variance of the Gaussian noise. For each noise variance, we run multiple experiments by regenerating new instances of the signal and noise vectors. For each run, we compute the error metric per as the average L_1 error between the output approximation $\hat{x}'$ (restricted to its k largest entries) and the best k-sparse approximation of $\hat{x}$ referred to as $\hat{y}$:

$$\text{Average } L_1 \text{ Error } = \frac{1}{k}\sum_{0<i\leq n} |\hat{x}'_i - \hat{y}_i|.$$

Figure 6-3 plots the average error per non-zero frequency for SFT 1.0, SFT 2.0, and AAFFT 0.9. The figure shows that all three algorithms are stable under noise. Further, SFT variants appear to be more robust to noise than AAFFT 0.9.

[3]The SNR is defined as $SNR = 20\log\frac{||x||_2}{||z||_2}$, where z is an n-dimensional noise vector.

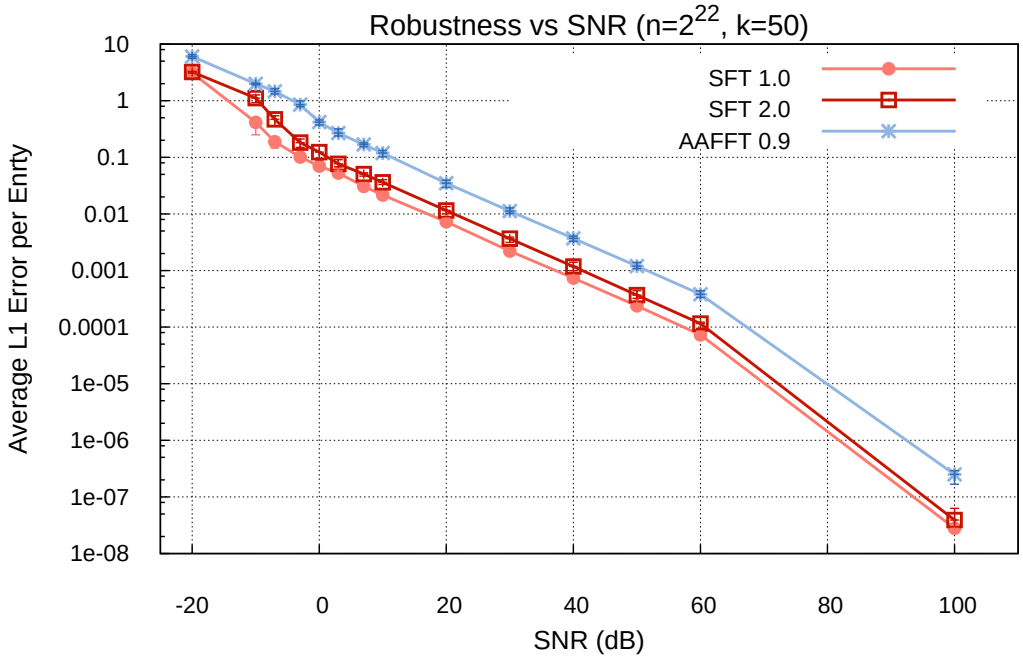


Figure 6-3: **Robustness to Noise Results ($n = 2^{22}, k = 50$).** The figure shows that all three algorithms are stable in the presence of noise but the SFT algorithms have lower errors.

Part II

Applications of the Sparse Fourier Transform

Chapter 7

GHz-Wide Spectrum Sensing and Decoding

7.1 Introduction

The rising popularity of wireless communication and the potential of a spectrum shortage have motivated the FCC to take steps towards releasing multiple bands for dynamic spectrum sharing [48]. The government's interest in re-purposing the spectrum for sharing is motivated by the fact that the actual utilization of the spectrum is sparse in practice. For instance, Figure 7-1 from the Microsoft Spectrum Observatory [123] shows that, even in urban areas, large swaths of the spectrum remain underutilized. To use the spectrum more efficiently, last year, the President's Council of Advisors on Science and Technology (PCAST) [174] has advocated dynamic sharing of much of the currently under-utilized spectrum, creating GHz-wide spectrum superhighways "that can be shared by many different types of wireless services, just as vehicles share a superhighway by moving from one lane to another."

Motivated by this vision, this chapter presents BigBand, a technology that enables realtime GHz-wide spectrum sensing and reception using low-power radios, similar to those in WiFi devices. Making GHz-wide sensing (*i.e.* the ability to detect occupancy) and reception (*i.e.* the ability to decode) available on commodity radios enables new applications:

- In particular, realtime GHz sensing enables highly dynamic spectrum access, where secondary users can detect short sub-millisecond spectrum vacancies and leverage them, thereby increasing the overall spectrum efficiency [16].

- Further, a cheap low-power GHz spectrum sensing technology enables the government and the industry to deploy thousands or such sensors in a metropolitan area for large-scale realtime spectrum monitoring. This will enable a better understanding of spectrum utilization, identification and localization of breaches of spectrum policy, and a more-informed planning of spectrum allocation.

- Beyond sensing, the ability to decode signals in a GHz-wide band enables a single radio to receive concurrent transmissions from diverse parts of the spectrum. This would enable future cell phones to use one radio to concurrently receive Bluetooth at 2.4 GHz, GSM at 1.9 GHz, and CDMA at 1.7 GHz.

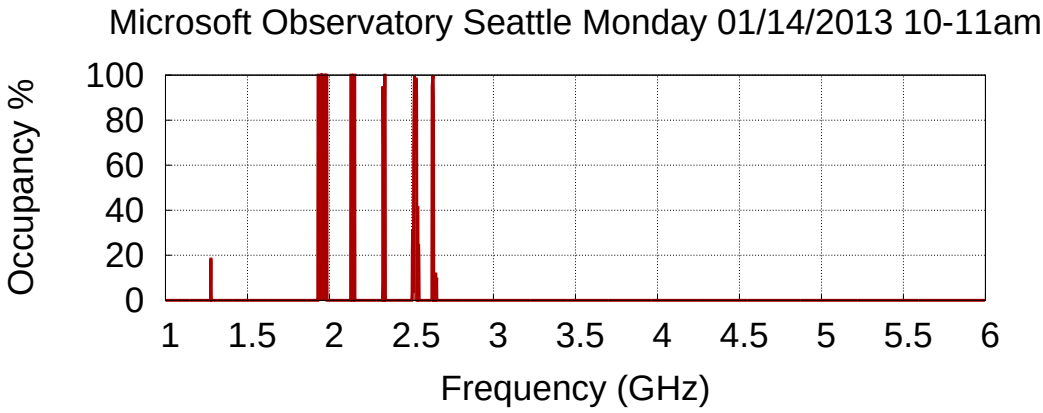


Figure 7-1: **Spectrum Occupancy:** The figure shows the average spectrum occupancy at the Microsoft spectrum observatory in Seattle on Monday January 14, 2013 during the hour between 10 am and 11 am. The figure shows that between 1 GHz and 6 GHz, the spectrum is sparsely occupied.

Realtime GHz signal acquisition, however, is challenging. For example, existing methods for spectrum sensing, like those used in the Microsoft spectrum observatory [123], do not work in realtime. They rely on sequential hopping from one channel to the next, acquiring only tens of MHz at a time [153, 169]. As a result, each band is monitored only occasionally, making it easy to miss short lived signals (*e.g.*, radar).

The key difficulty in capturing GHz of bandwidth in realtime stems from the need for high-speed analog-to-digital converters (ADCs), which are costly, power hungry, and have a low bit resolution [68, 131]. Compare typical low-speed ADCs used in WiFi or cellular phones with the very high speed ADCs needed to capture GHz of bandwidth. A 100 MS/s ADC, like in Wi-Fi receivers, costs a few dollars, consumes a few milli Watts, and has a 12 to 16-bit resolution [40, 131, 166]. In contrast, a high speed ADC that can take multiple giga-samples per second may cost hundreds of dollars, consume multiple orders of magnitude more power, and have only 6 to 8-bits resolution [40, 68, 131].

In this chapter, we explore how one can achieve the best of both worlds. Specifically, we would like to capture GHz of spectrum but using only few ADCs that sample the signal at tens of MS/s.

We introduce BigBand, a technology that can acquire GHz of signal using a few (3 or 4) low-speed ADCs. BigBand can do more than spectrum sensing – the action of detecting occupied bands. It can also decode the signal (i.e., obtain the I and Q components). To achieve its goal, BigBand builds on the Sparse Fourier Transform algorithms, described in Part I of this thesis, to recover the wireless spectrum using only a small subset of samples –i.e., it can recover GHz of spectrum without sampling it at the Nyquist rate.

Some past work has proposed using compressive sensing to acquire GHz signals at sub-Nyquist rate [103, 151, 172, 184]. BigBand builds on this work but differs from it substantially. Approaches based on compressive sensing require random sampling of the signal which cannot be done simply by using standard low-speed ADCs. It needs analog mixing at Nyquist rates [103, 184] and

expensive processing to recover the original signal. Such a design is quite complex and could end up consuming as much power as an ADC that samples at the Nyquist rate [1, 2]. Like the compressive-sensing approaches, BigBand can acquire a wideband signal without sampling it at the Nyquist rate. Unlike compressive sensing, however, BigBand does not need analog mixing or random sampling and can work using commodity radios and standard low-speed ADCs. Further, it computes the Fourier transform of a sparse signal faster than the FFT, reducing baseband processing.

We have built a working prototype of BigBand using USRP software radios. Our prototype uses three USRPs, each of which can capture 50 MHz bandwidth to produce a device that captures 0.9 GHz –*i.e.*, 6× larger bandwidth than the digital bandwidth of the three USRPs combined. We have used our prototype to sense the spectrum between 2 GHz and 2.9 GHz, a 0.9-GHz stretch used by diverse technologies [123]. Our outdoor measurements reveal that, in our metropolitan area,[1] the above band has an occupancy of 2–5%. These results were verified using a spectrum analyzer are in sync with similar measurements conducted at other locations [123]. We further use our prototype to decode 30 transmitters that are simultaneously frequency hopping in a 0.9 GHz band, hence demonstrating that BigBand decodes the signals, not only senses their power.

Finally, we have extended BigBand to perform spectrum sensing (not decoding) even when the spectrum utilization is not sparse. To do so, we leverage the idea that even if the spectrum itself is densely occupied, only a small fraction of the spectrum is likely to change its occupancy over short intervals of a few milliseconds. We build on this basic idea to sense densely occupied spectrum using sub-Nyquist sampling. We also evaluate our design empirically showing that it can detect frequency bands that change occupancy even when the spectrum is 95% occupied.

7.2 Related Work

BigBand is related to signal acquisition via digital and analog compressive sensing [103, 124, 125, 151, 172, 184, 185]. However, compressive sensing needs random sampling and analog mixing at Nyquist rates [103, 125, 184]. These approaches cannot be built using commodity radios and ADCs with regular sampling; they require a custom design and could end up consuming as much power as an ADC that samples at the Nyquist rate [1, 2]. Furthermore, compressive sensing does not directly compute the spectrum representation of the signal and still needs to perform heavy computation to recover the spectrum, which is power consuming.

BigBand is also related to theoretical work in signal processing on co-prime sampling [175, 180, 181]. In [180, 181], co-prime sampling patterns are utilized to sample sparse spectrum. These methods however require k ADCs with co-prime sampling patterns, where k is the number of occupied frequencies. In contrast, using the Sparse Fourier Transform allows us to use only a constant small number of ADCs. Our system is further implemented and shown to work in practice. In [175], co-prime sampling is used to sample linear antenna arrays. This work however assumes the presence of a second dimension where signals can be fully sampled and cross-correlated and hence cannot be used for spectrum acquisition.

[1] MIT campus, Cambridge MA, USA.

Also relevant to our work is the theoretical work on using multicoset sampling to capture the signals in a wideband sparse spectrum with a small number of low speed ADCs [79, 178]. However, in order to recover the original signals from the samples, these techniques require prior knowledge of the locations of occupied frequencies in the spectrum and hence are not useful for spectrum sensing. In contrast, our approach recovers both the locations of the occupied frequencies and the signals in these frequencies and thus can be used for both spectrum sensing and decoding.

Some proposals for test equipment reconstruct wideband periodic signals by undersampling [159, 173]. These approaches however assume that the signal is periodic –*i.e.*, the same signal keeps repeating for very long time – which allows them to take one sample during each period until all samples are recovered and rearranged in the proper order. Though this requires one low speed ADC, it is only applicable to test equipment where the same signal is repeatedly transmitted [173].

There is significant literature about spectrum sensing. Most of this work focuses on narrowband sensing [13, 149, 187]. It includes techniques for detecting the signal's energy [13], its waveform [187], its cyclostationarity [84], or its power variation [149]. In contrast, we focus on wideband spectrum sensing, an area that is significantly less explored. A recent system called QuickSense [186] senses a wideband signal using a hierarchy of analog filters and energy detectors. BigBand differs from QuickSense in that it can recover the signal (obtain the I and Q components) as opposed to only detecting spectrum occupancy. Second, for highly utilized spectrum (*i.e.* not sparse), the approach in [186] reduces to sequentially scanning the spectrum whereas BigBand's extension for the non-sparse case provides a fast sensing mechanism.

Finally, the proposed research complements the geo-location database required by the FCC for identifying the bands occupied by primary users (e.g., the TV stations in the white spaces). The database, however, has no information about frequencies occupied by secondary and unlicensed users in the area. Also, due to the complexity of predicting propagation models, the database provides only long-term predictions, and can be inaccurate, particularly with dynamic access patterns [16, 48].

7.3 BigBand

BigBand is a receiver that can recover sparse signals with sub-Nyquist sampling using low-power commodity radios. BigBand can do more than spectrum sensing – the action of detecting occupied bands. It provides the details of the signals in those bands (I's and Q's of wireless symbols), which enables decoding those signals.

BigBand uses the Sparse Fourier Transform to acquire sparse spectra using low speed ADCs. In this section, we explain how BigBand adapts the Sparse Fourier Transform for the application of spectrum acquisition. We use $\mathbf{x}$ and $\widehat{\mathbf{x}}$ to denote a time signal and its Fourier transform respectively. We also use the terms: the *value* of a frequency and its *position* in the spectrum to distinguish $\widehat{\mathbf{x}}_f$ and f. BigBand discovers the occupied frequency positions f and estimates their values $\widehat{\mathbf{x}}_f$. Once $\widehat{\mathbf{x}}$ is computed, it can recover the time signal $\mathbf{x}$ and decode the wireless symbols.

Following the Sparse Fourier Transform framework set in Section 1.1.2, BigBand's design has three components: frequency *bucketization*, *estimation*, and *collision resolution*. Below we re-explain these components in the context of wireless signal acquisition and processing. A theoretical

analysis of BigBand's Sparse Fourier Transform algorithm follows immediately from Chapter 5.[2]

7.3.1 Frequency Bucketization

BigBand starts by hashing the frequencies in the spectrum into buckets. Since the spectrum is sparsely occupied, many buckets will be empty and can be simply discarded. BigBand then focuses on the non-empty buckets, and computes the values of the frequencies in those buckets in what we call the estimation step.

In order to hash frequencies into buckets, BigBand uses the aliasing filter described in Section 1.1.2. Recall that, if $\mathbf{b}$ is a sub-sampled version of the wireless signal $\mathbf{x}$ of bandwidth BW, *i.e.*, $\mathbf{b}_i = \mathbf{x}_{i \cdot p}$ where p is the sub-sampling factor, then, $\widehat{\mathbf{b}}$, the FFT of $\mathbf{b}$ is an aliased version of $\widehat{\mathbf{x}}$, *i.e.*:

$$\widehat{\mathbf{b}}_i = \sum_{m=0}^{p-1} \widehat{\mathbf{x}}_{i+m(BW/p)} \tag{7.1}$$

Thus, the aliasing filter will hash frequencies equally spaced by an interval BW/p to the same bucket using the hashing function $i = f \mod BW/p$. Further, the value in each bucket is the sum of the values of only the frequencies that hash to the bucket as shown in Equation 7.1. Most importantly, aliasing fits naturally to the problem of spectrum acquisition since it can simply be implemented by sampling the signal using a low-speed ADC slower than the Nyquist rate.

Now that we hashed the frequencies into buckets, we can leverage the fact that the spectrum of interest is sparse and hence most buckets have noise and no signal. BigBand compares the energy (*i.e.*, the magnitude square) of a bucket with the receiver's noise level and considers all buckets whose energy is below a threshold to be empty. It then focuses on the occupied buckets and ignores empty buckets.

7.3.2 Frequency Estimation

Next, for each of the occupied buckets we want to identify which frequencies created the energy in these buckets, and what are the values of these frequencies. If we can do that, we then have recovered a complete representation of the frequencies with non-zero signal values, *i.e.*, we acquired the full signal in the Fourier domain.

Recall that our spectrum is sparse; thus, as mentioned earlier, when hashing frequencies into buckets many buckets are likely to be empty. Even for the occupied buckets, many of these buckets will likely have a single non-zero frequency hashing into them, and only a small number will have a collision of multiple non-zero (or occupied) frequencies. In the next section, we present a mechanism to detect whether a bucket has a collision and resolve such collisions. In this section, we focus on buckets with a single non-zero frequency and estimate the value and the position of this non-zero frequency.

Recall that if there is a single occupied frequency coefficient in the bucket, , then the value of this occupied frequency is the value of the bucket. Said differently, the value of a bucket after

[2]Note that while the algorithms in Chapter 5 are for 2D signals, the analysis holds due to the equivalence between the two-dimensional case and the one-dimensional case where n is a product of different prime powers [59, 90].

aliasing, $\widehat{\mathbf{b}}_i$ is a good estimate of the value of the occupied frequency $\widehat{\mathbf{x}}_f$ in that bucket, since all other frequencies in the bucket have zero signal value (only noise). Although we can easily find the value of the non-zero frequency in a bucket, we still do not know its frequency position f, since aliasing mapped multiple frequencies to the same bucket. Recall from Section 1.1.2, to compute f, we can use the *phase-rotation property* of the Fourier transform, which states that a shift in time domain translates into phase rotation in the frequency domain. We perform the process of bucketization again, after shifting the input signal by τ. Since a shift in time translates into phase rotation in the frequency domain, the value of the bucket of changes from $\widehat{\mathbf{b}}_i = \widehat{\mathbf{x}}_f$ to $\widehat{\mathbf{b}}_i^{(\tau)} = \widehat{\mathbf{x}}_f \cdot e^{2\pi j \cdot f \cdot \tau}$. Hence, using the change in the phase of the bucket, we can estimate our frequency of interest and we can do this for all buckets that do not have collisions.

Two points are worth noting:

- First, recall that the phase wraps around every 2π. Hence, the value of τ has to be small to avoid the phase wrapping around for large values of f. In particular, τ should be on the order of $1/BW$ where BW is the bandwidth of interest. For example, to acquire one GHz of spectrum, τ should be on the order of a nanosecond.[3]

- Second, to sample the signal with a τ shift, we need a second low-speed ADC that has the same sampling rate as the ADC in the bucketization step but whose samples are delayed by τ. This can be achieved by connecting a single antenna to two ADCs using different delay lines (which is what we do in our implementation). Alternatively, one can use different delay lines to connect the clocks to the two ADCs.

7.3.3 Collision Detection and Resolution

We still need to address two questions: how do we distinguish the buckets that have a single non-zero frequency from those that have a collision? and in the case of a collision, how do we resolve the colliding frequencies?

Collision Detection

Again we use the *phase rotation* property of the Fourier transform to determine if a collision has occurred. Specifically, if the bucket contains a single non-zero frequency, *i.e.*, no collision, then performing the bucketization with a time shift τ causes only a phase rotation of the value in the bucket but the magnitude of the bucket does not change –*i.e.*, with or without the time shift, $\|\widehat{\mathbf{b}}_i\| = \|\widehat{\mathbf{b}}_i^{(\tau)}\| = \|\widehat{\mathbf{x}}_f\|$. In contrast, consider the case where there is a collision between, say, two frequencies f and f'. Then the value of the bucket without a time-shift is $\widehat{\mathbf{b}}_i = \widehat{\mathbf{x}}_f + \widehat{\mathbf{x}}_{f'}$ while its value with a time-shift of τ is $\widehat{\mathbf{b}}_i^{(\tau)} = \widehat{\mathbf{x}}_f \cdot e^{2\pi j \cdot f\tau} + \widehat{\mathbf{x}}_{f'} \cdot e^{2\pi j \cdot f'\tau}$. Since the colliding frequencies rotate by different phases, the overall magnitude of the bucket will change. Thus, we can determine whether there is a collision or not by comparing the magnitudes of the buckets with and without the time-shift. Note that even if one occasionally falsely detects a collision when there

[3] In fact, one can prove a looser version of this constraint where large τ are fine. Formally, for τ larger than $1/BW$, the FFT window size must be a non-integer multiple of τ.

is a single frequency, BigBand can still correct this error. This is because the collision resolution step described next will estimate the values of the presumed colliding frequencies to zero.

Collision Resolution

To reconstruct the full spectrum, we need to resolve the collisions –*i.e.*, for each non-zero frequency in a collision we need to estimate its value $\widehat{\mathbf{x}}_f$ and position f. We present two approaches for resolving collisions which may also be combined in case the spectrum is less sparse.

A. Resolving Collisions with Co-prime Aliasing Filters

One approach to resolve collisions is to bucketize the spectrum multiple times using aliasing filters with co-prime sampling rates. As described in Section 1.1.2, co-prime aliasing filters guarantee (by the Chinese remainder theorem) that any two frequencies that collide in one bucketization will not collide in the other bucketizations. Hence, we can resolve collisions by iterating between the two co-prime bucketizations. We can estimate the frequencies that did not collide from the first bucketization and subtract them from the colliding buckets in the second bucketization. This frees some of the colliding frequencies in the second bucketization and allows us to estimate them. We can then go back to the first bucketization and subtract these newly estimated frequencies from the buckets where they collided. We can keep iterating until we have recovered all the occupied frequencies.

Thus, by using co-prime aliasing filters to bucketize and iterating between the bucketizations –*i.e.*, estimating frequencies from buckets where they do not collide and subtracting them from buckets where they do collide– we can recover the spectrum. This suggests that to capture a spectrum bandwidth BW, we can use two ADCs that sample at rates BW/p_1 and BW/p_2 where p_1 and p_2 are co-prime. For example, to recover a 1 GHz spectrum, we can use a 42 MHz ADC [40] along with a 50 MHz ADC. The combination of these two ADCs can capture a bandwidth of 1.05 GHz because 42 MHz $= 1.05$ GHz$/25$ and 50 MHz $= 1.05$ GHz$/21$, where 21 and 25 are co-prime. Note that we also repeat each of these co-prime bucketization with a time shift (as explained in Section 7.3.2, which requires a total of 4 low-speed ADCs.

B. Resolving Collisions without Co-prime Aliasing Filters

Co-prime aliasing filters are an efficient way to resolve collisions, but they are not necessary. Here, we show how to resolve collisions while still using ADCs that sample at the same rate. This means that one can use one type of ADCs for building the whole system. This makes it possible to build BigBand using only software radios like USRPs [47].

We use one type of aliasing filter. However, we perform it for more than twice using multiple different time shifts. To see how this can help resolve collisions, consider again the case where two frequencies f and f' collide in a bucket. If we use two time shifts τ_1 and τ_2, we get three values for each bucket. For the bucket where f and f' collide, these values are:

$$\begin{aligned} \widehat{\mathbf{b}}_i &= \widehat{\mathbf{x}}_f + \widehat{\mathbf{x}}_{f'} \\ \widehat{\mathbf{b}}_i^{(\tau_1)} &= \widehat{\mathbf{x}}_f \cdot e^{2\pi j \cdot f \tau_1} + \widehat{\mathbf{x}}_{f'} \cdot e^{2\pi j \cdot f' \tau_1} \\ \widehat{\mathbf{b}}_i^{(\tau_2)} &= \widehat{\mathbf{x}}_f \cdot e^{2\pi j \cdot f \tau_2} + \widehat{\mathbf{x}}_{f'} \cdot e^{2\pi j \cdot f' \tau_2} \end{aligned} \tag{7.2}$$

If we know the positions of f and f', the above becomes an overdetermined system of equations where the only unknowns are $\widehat{\mathbf{x}}_f$, $\widehat{\mathbf{x}}'_f$. Since only few frequencies hash into each bucket, there is a limited number of possible values of f and f'. For each of these possibilities, the above overdetermined system can be solved to find $\widehat{\mathbf{x}}_f$, $\widehat{\mathbf{x}}'_f$. Hence, we can solve overdetermined system for the possible (f, f') pairs and choose the pair that minimizes the mean square error. While the above does not guarantee that the solution is unique, in case multiple pairs (f,f') satisfy the equations, BigBand can detect that event and report to the user that the values of these frequencies remain unresolved.[4] Our empirical results (in Section 7.7.3) show however that for practical spectrum sparsity (which is about 5%) 3 shifted bucketizations are enough to uniquely resolve the colliding frequencies.

We note that though this method requires more digital computation, we only need to do this for the few buckets that have a collision, and we know the number of collisions is small due to the sparsity of the spectrum. We also note that this method can be combined with the co-prime approach to deal with less sparse spectrum. In this case, one uses this method to resolve collisions of two frequencies while iterating between the co-prime filters.

7.4 Channel Estimation and Calibration

The earlier description of BigBand assumes that the different ADCs can sample exactly the same signal at different time-shifts. However, because the signals experience different channels, they will be scaled differently and the ADCs will not be able to sample exactly the same signal.

To better understand this problem, let us consider the case where we resolve collisions without the co-prime sub-sampling. In this case, we will have 3 ADCs each sampling a signal that is delayed by a time shift. In this case, consider a non-zero frequency f whose value is $\widehat{\mathbf{x}}_f$. If f hashes to bucket i and does not collide, then the value of the bucket at each of the ADCs can be written as:

$$\begin{aligned} \widehat{\mathbf{b}}_i &= h_w(f) \cdot h_1(f) \cdot \widehat{\mathbf{x}}_f \\ \widehat{\mathbf{b}}_i^{(\tau_1)} &= h_w(f) \cdot h_2(f) \cdot \widehat{\mathbf{x}}_f \cdot e^{2\pi j \cdot f \tau_1} \\ \widehat{\mathbf{b}}_i^{(\tau_2)} &= h_w(f) \cdot h_3(f) \cdot \widehat{\mathbf{x}}_f \cdot e^{2\pi j \cdot f \tau_2} \end{aligned} \tag{7.3}$$

where $h_w(f)$ is the channel on the wireless medium, $h_1(f), h_2(f), h_3(f)$ are the hardware channels on each of the radios, and $\cdot(f)$ indicates that these parameters are frequency dependent. We can ensure that $h_w(f)$ is the same in all three bucketizations by connecting the RF frontends to the same antenna. As a result, $h_w(f)$ cancels out once we take the ratios, $\widehat{\mathbf{b}}_i^{(\tau_1)}/\widehat{\mathbf{b}}_i$ and $\widehat{\mathbf{b}}_i^{(\tau_2)}/\widehat{\mathbf{b}}_i$ of the buckets. However, the hardware channels are different for the different bucketizations. We need to estimate them and compensate for them in order to perform frequency estimation and also resolve the collisions.

Furthermore, though it is simple to create time-shifts between the three ADCs as explained in Section 7.3.2, we need to know the values of these time-shifts τ_1, τ_2 in order to perform frequency

[4] Note that theoretically, for a collision of k frequencies, $2k$ samples can guarantee a unique solution in the absence of noise.

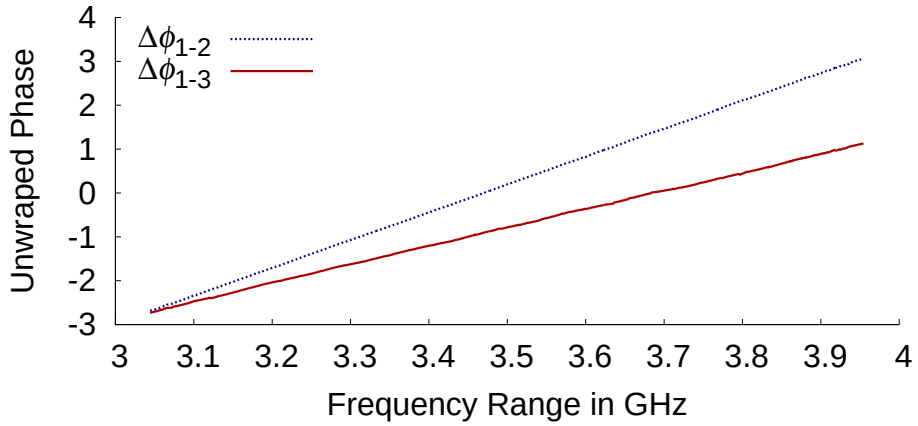


Figure 7-2: **Phase Rotation vs. Frequency:** The figure shows that the phase rotation between the 3 USRPs is linear across the 900 MHz frequency spectrum and can be used to estimate the time shifts.

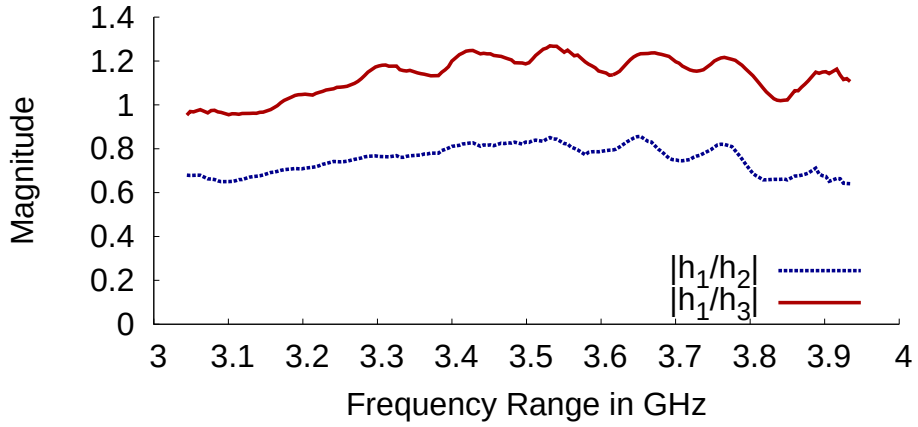


Figure 7-3: **Hardware Channel Magnitude:** The relative channel magnitudes $|h_1(f)/h_2(f)|$ and $|h_1(f)/h_3(f)|$ are not equal to 1 and are not flat across the frequency spectrum. Hence, we need to compensate for these estimates to be able to detect and solve collisions.

estimation based on phase rotation. Hence, we also need a way to estimate these time-shifts.

7.4.1 Estimating the Channels and Time-Shifts

To estimate the channels and the time shifts, we divide the total bandwidth BW that BigBand captures into p consecutive chunks. We then transmit a known signal in each chunk, one by one. Since we only transmit in one chunk at a time, there are no collisions at the receiver after aliasing. We then use Equation 7.3 to estimate the ratios $h_2(f) \cdot e^{2\pi j \cdot f\tau_1}/h_1(f)$ and $h_3(f) \cdot e^{2\pi j \cdot f\tau_2}/h_1(f)$ for each frequency f in the spectrum.

Now that we have the ratios, we need to compute $h_2(f)/h_1(f)$ for each frequency f, and the

delay τ_1. We can estimate this as follows: Both the magnitude and phase of the hardware channel ratio will be different for different frequencies. The magnitude differs with frequency because different frequencies experience different attenuation in the hardware. The phase varies linearly with frequency because all frequencies experience the same delay τ_1, and the phase rotation of a frequency f is simply $2\pi f\tau_1$. We can therefore plot the phase of the ratio as a function of frequency, and compute the delay τ_1 from the slope of the resulting line.

Figure 7-2 shows the phase result of this estimation performed on the USRP software radios used in our implementation described in Section 7.6. As expected, the phase is linear across 900 MHz. Hence, by fitting the points in Figure 7-2 to a line we can estimate the shifts τ_1, τ_2 and the relative phases of the hardware channels (*i.e.* $\angle h_1(f)/h_2(f)$ and $\angle h_1(f)/h_3(f)$). Figure 7-3 also shows the relative magnitudes of the hardware channels on the USRPs (*i.e.* $|h_1(f)/h_2(f)|$ and $|h_1(f)/h_3(f)|$) over the 900 MHz between 3.05 GHz and 3.95 GHz. These hardware channels and time shifts are stable. For our implementation, we estimated them only once at the set up time.

7.5 Differential Sensing of Non-Sparse Spectrum

We extend BigBand's algorithm to sense a non-sparse spectrum. The key idea is that although the spectrum might not be sparse, changes in spectrum usage are typically sparse, i.e., over short intervals, only a small percentage of the frequencies are freed up or become occupied. This makes it possible to estimate the occupancy without sampling the signal at the Nyquist rate. We refer to sparse changes as differential sparsity, and call the extension that deals with such non-sparse spectrum D-BigBand. We note however that unlike in the case where the spectrum is sparse, in the non-sparse setting we only perform spectrum sensing but we cannot recover the I and Q components of the signal. Below we explain how we perform bucketization and estimation in D-BigBand.

7.5.1 Frequency Bucketization

D-BigBand also bucketizes the spectrum using sub-sampling filters. However, since the spectrum is not sparse, it is very likely that all buckets will be occupied. Thus, D-BigBand tries to detect changes in the occupancy of frequencies that hash to each buckets. To do so, D-BigBand computes the average power of the buckets over two consecutive time windows TW by performing the bucketization multiple times during each time window.[5] Since the changes in spectrum occupancies are sparse, only the average power of few buckets would change between the two time windows. D-BigBand can then focus only on the few buckets where the average power changes.

7.5.2 Frequency Estimation

Now that we know in which buckets the average power has changed, we need to estimate which of the frequencies in the bucket is the one whose occupancy has changed. However, we can no longer use the phase rotation property to estimate these frequencies or resolve their collisions since the

[5]The number of times D-BigBand can average is = TW/T where T is the FFT window time.

phase of the bucket now depends on all the frequencies that hash to the bucket and not just the frequency whose occupancy has changed. Thus, to estimate the changing frequencies we are going to use a different method which we refer to as voting which is similar to the voting approach described in Section 1.1.3 and used in Chapter 3. We repeat the bucketization multiple times while randomizing which frequencies hash to which buckets. After that, each bucketization votes for frequencies that hash to buckets where the power changed. Frequencies that get the most number of votes are picked as the ones whose occupancy has changed. To randomize the bucketizations, we simply use co-prime sub-sampling which as described in Section 7.3.3 guarantees that frequencies that hash together in one bucketization can not hash together in the other bucketizations.

As with any differential system, we need to initialize the state of spectrum occupancy. However, an interesting property of D-BigBand is that we can initialize the occupancy of each frequency in the spectrum to unknown. This is because, when we take the difference in power we can tell whether the frequency became occupied or it became empty. Specifically, a negative power difference implies that the corresponding frequency became empty, and a positive power difference implies that the corresponding frequency became occupied. Hence, once the occupancy of a frequency changes, we can tell its current state irrespective of its previous state. This avoids the need for initialization and prevents error propagation.

7.6 A USRP-Based Implementation

7.6.1 Implementing BigBand

As a proof of concept, we implement BigBand using USRP N210 software radios [47]. Since the USRPs use the same ADCs, it is not possible to have co-prime sub-sampling rates. Thus, our implementation relies on resolving collisions without co-prime sub-sampling.

We use three USRP N210 radios with the SBX daughterboards, which can operate in the 400 MHz to 4.4 GHz range. The clocks of the three USRPs are synchronized using an external GPSDO clock [91]. In order to sample the same signal using the three USRPs, we connect the USRPs to the same antenna using a power splitter but with wires of different lengths in order to introduce small time-shifts. We also remove the analog low pass filters on the SBX daughterboards to allow the USRP's ADC to receive the entire bandwidth that its analog front-end circuitry is designed for. The analog circuitry of the USRP front-end can receive at most 0.9 GHz, which puts an upper bound on the digital bandwidth of the system. The three USRP ADCs each samples the signal at 50 MS/s.[6] Thus, our implementation of BigBand captures a bandwidth BW = 900 MHz using only 150 MS/s.

7.6.2 Implementing D-BigBand

D-BigBand's frequency estimation relies on using different co-prime sub-sampling rates and hence we cannot implement D-BigBand directly on USRPs. Thus, to verify that D-BigBand can sense

[6] In principle, the USRP ADC can sample up to 100 MS/s. However, the USRP digital processing chain cannot support this rate and hence the ADC sampling rate can be set to no higher than 50 MS/s.

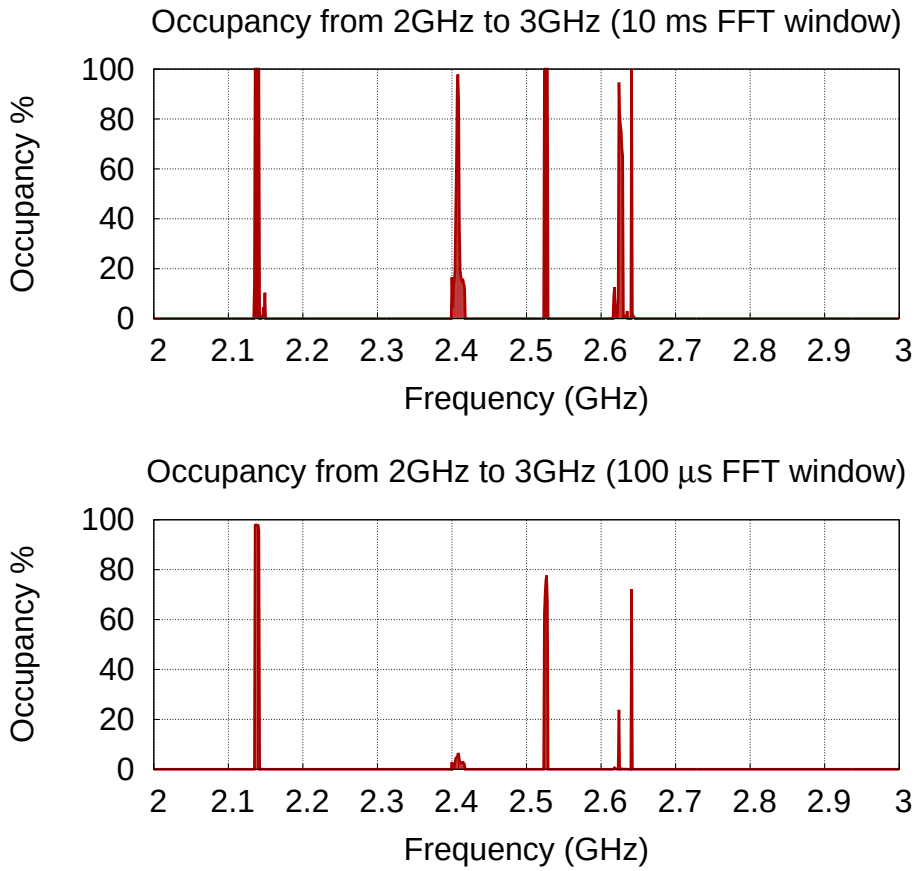


Figure 7-4: **Spectrum Occupancy Results:** The figure shows the average spectrum occupancy at our geographical location on Friday 01/15/2013 between 1-2pm:, as viewed at a 10 ms granularity (top) and 100μs granularity (bottom). It shows that the spectrum is sparsely occupied. Further, the sparsity increases when one computes the occupancy over shorter time windows.

a non-sparse spectrum, we use trace-driven experiments. To collect traces of one GHz of highly occupied spectrum, we use many USRPs to transmit and receive. Since we have a total of 20 USRPs, we divide them into 10 receivers and 10 transmitters and capture 250 MHz at a time. We repeat this 4 times at center frequencies that are 250 MHz apart and stitch them together in the frequency domain to capture the full 1 GHz spectrum. We then perform the inverse FFT to obtain a time signal sampled at 1 GHz. We now sub-sample this time domain signal using three co-prime rates: 1/21, 1/20, and 1/23 GHz. We run D-BigBand using these sub-sampled versions of the signal.

7.7 BigBand's Spectrum Sensing Results

7.7.1 Outdoor Spectrum Sensing

We collect outdoor measurements from the roof top of a 24 floor MIT building. We use BigBand to capture the signal between 2 GHz and 2.9 GHz over 30 minutes. We configure BigBand to compute the spectrum over an FFT window of size W. We report here results for $W = 10$ms and $W = 100\mu$s. We calculate the occupancy of a particular frequency as the percentage of the FFT windows during which the frequency was occupied (*i.e.*, the power at that frequency was at least twice the noise power).

Figure 7-4 shows the fraction of time that each chunk of spectrum between 2 GHz and 2.9 GHz is occupied, as recovered by BigBand. These results were confirmed using a spectrum analyzer. The figure shows that the spectrum is sparsely occupied. In particular, the occupancy is about 5% when considered over FFT windows of 10 ms and drops to about 2%, when viewed over windows of 100 μs. The figure shows that even frequencies that look 100% occupied over 10 ms windows, become less occupied when viewed over shorter intervals. This is because while these frequencies are occupied for some fraction of every 10 ms interval, there is a large number of shorter windows within each 10 ms where these frequencies are not occupied. For example, the WiFi band around 2.4 GHz seems fully utilized when checked over 10 ms windows; yet if one views it over windows that are 100 times shorter (*i.e.*, 100μs), one would discover that the medium is almost always idle. In contrast, the band around 2.1 GHz which is used by cellular technologies is occupied even at very short time scales.

The above implies that the spectrum is sparser at finer time intervals, and provides more opportunities for fine-grained spectrum reuse. This result motivates the need for fast spectrum sensing schemes to exploit these short-term vacancies.

Finally, we note that measurements collected in other locations or on different dates show similar results to those in Figure 7-4 but may differ slightly in which frequencies are occupied. Measurements from higher parts of the spectrum are qualitatively similar but have significantly higher sparsity (we omit the figures for lack of space).

7.7.2 BigBand vs. Spectrum Scanning

Most of today's spectrum sensing equipment relies on scanning. Even expensive, power hungry spectrum analyzers typically capture a 100 MHz bandwidth in one shot, and end up scanning to capture a larger spectrum [169]. The performance of sequentially scanning the spectrum depends mainly on how fast the device can scan a GHz of bandwidth. In the absence of fast scanning, the system can miss radar and other highly dynamic signals. Here, we compare how fast it would take to scan the 900 MHz bandwidth using three techniques: state-of-the-art spectrum monitors like the RFeye [153], which is used in the Microsoft spectrum observatory, 3 USRPs sequentially scanning the 900 MHz, or 3 USRPs using BigBand.

Table 7.1 shows the results for different FFT window sizes. In all cases, BigBand takes exactly the time of the FFT window to acquire the 900 MHz spectrum. The 3 USRPs combined can scan 150 MHz at a time and hence need to scan 6 times to acquire the full 900 MHz. For FFT window

FFT Window	BigBand (900 MHz)	3 USRP Seq. Scan (150 MHz)	RFeye Scan (20 MHz)
1 μs	1 μs	48 ms	22.5 ms
10 μs	10 μs	48 ms	22.5 ms
100 μs	100 μs	48 ms	—
1 ms	1 ms	54 ms	—
10 ms	10 ms	114 ms	—

Table 7.1: **Spectrum Sensing Scanning Time:** BigBand is multiple orders of magnitude faster than other technologies. This allows it to perform real-time sensing to take advantage of even short term spectrum vacancies.

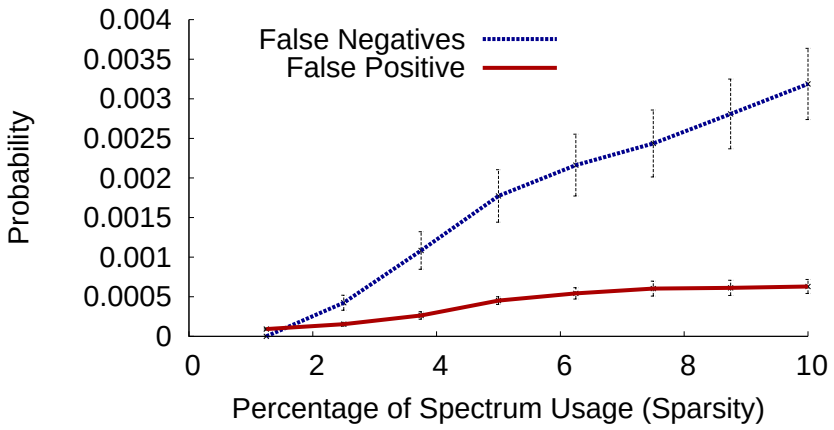


Figure 7-5: **False Negatives and Positives as a Function of Spectrum Sparsity:** BigBand's false positive and false negative rates are extremely low.

sizes lower than 10 ms, the scanning time is about 48 ms. Hence, the USRPs spend very little time actually sensing the spectrum, which will lead to a lot of missed signals. Of course, state of the art spectrum monitors can do much better. The RFeye node has a fast scanning mode of 40 GHz/second [153]. It scans in chunks of 20 MHz and thus will take 22.5 ms to scan 900 MHz. Note that RFeye has a maximum resolution of 20 kHz, and hence does not support FFT windows larger than 50 μs.

Thus, BigBand, which uses off-the-shelf components, is much faster than even expensive scanning based solutions, allowing it to detect short-term spectrum vacancies.

7.7.3 BigBand's Sparsity Range

The primary motivation of BigBand is to be able to sense sparse spectrum. In this section, we verify the range of sparsity for which BigBand works. We run our experiments between 3.05 GHz and 3.95 GHz because this band is effectively empty (see Figure 7-1), and hence enables us to perform controlled experiments. We vary the sparsity in the 3.05 GHz to 3.95 GHz range between 1% and

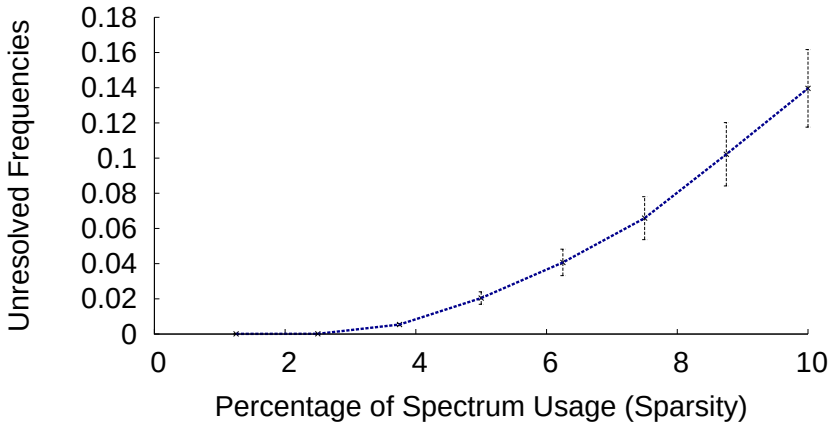


Figure 7-6: **Unresolved Frequencies as a Function of Spectrum Sparsity:** BigBand cannot resolve around 2% of the frequencies with 5% spectrum occupancy, and stays below 14% even when spectrum occupancy grows as large as 10%.

10% by transmitting from 5 different USRPs. Each USRP transmits a signal whose bandwidth is at least 1 MHz and at most 20 MHz. We randomize the bandwidth and the center frequencies of the signals transmitted by the USRPs. For each sparsity level, we repeat the experiment 100 times with different random choices of bandwidth and center frequencies. We run BigBand over a 1 ms FFT window. We consider three metrics:

- **False Negatives**: The fraction of occupied frequencies that BigBand incorrectly reports as empty.
- **False Positives**: The fraction of empty frequencies that BigBand incorrectly reports as occupied.
- **Unresolved Frequencies**: The fraction of total frequencies that BigBand cannot resolve due to unresolved collisions.

Figure 7-5 shows that BigBand's false positives and false negatives rates are extremely low. The probability of false positive stays below 0.0005 even when 10% of the spectrum is occupied. The probability of false negative is less than 0.002 when the spectrum occupancy is less than 5%, and stays within 0.003 even when the spectrum occupancy goes up to 10%.

Figure 7-6 shows that the fraction of unresolved frequencies is less than 0.03 when the spectrum usage is below 5%. This number increases as the spectrum usage increases, but stays below 0.14 when 10% of the spectrum is used. Unresolved frequencies increase as spectrum usage increases because the probability of collision increases. Note however that in contrast to false positive and false negatives, BigBand knows which exact frequencies it could not resolve and reports these frequencies with the label "not-estimated". Thus, unresolved frequencies show lack of information as opposed to errors. The application can decide how to treat unresolved frequencies. For dynamic spectrum access, it can simply avoid the unresolved frequencies.

We also note that real-world spectrum measurements, for instance, in the Microsoft observatory, and our results, reveal that actual spectrum usage is 2–5%. In this regime, BigBand's unre-

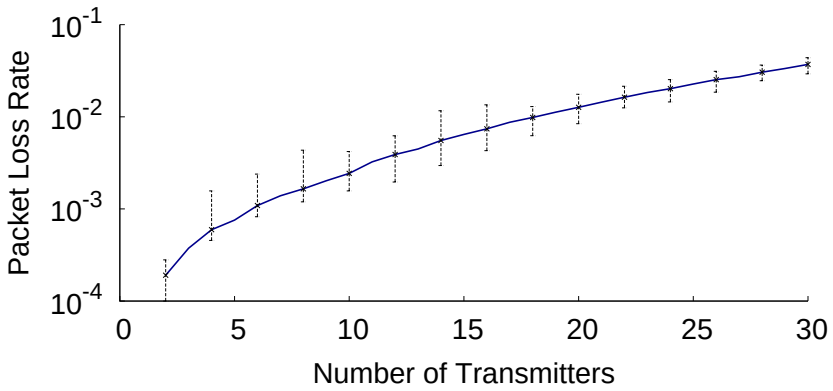


Figure 7-7: **BigBand's Packet Loss as a Function of the Number of Simultaneous Transmitters:** BigBand can decode as many as 30 transmitters spread across a 900 MHz wide band, while keeping the packet loss less than 3.5%.

solved frequencies are less than 0.03. Further, if the occupancy is high, one may use D-BigBand, which deals with high occupancies (see results in Section 7.9.)

7.8 BigBand's Decoding Results

7.8.1 Decoding Multiple Transmitters

In this section, we verify that BigBand can concurrently decode a large number of transmitters from diverse parts of the spectrum. All the transmitters in our implementation use the same technology, but the result naturally generalizes to transmitters using different technologies.

We use 10 USRPs to emulate up to 30 transmitters hopping in a spectrum of 0.9 GHz. At any given time instant, each device uses 1 MHz of spectrum to transmit a BPSK signal. Similar to the Bluetooth frequency hopping standard, we assume that there is a master that assigns a hopping sequence to each device that ensures that no two devices hop to the same frequency at the same time instant. Note however, that the hopping sequence for different devices allows them to hop to frequencies that get aliased to the same bucket at a particular time instant, and hence collide in BigBand's aliasing filters. Like in Bluetooth, each transmitter hops 1, 3, or 5 times per packet, depending on the length of the packet.

Figure 7-7 shows the packet loss rate versus the number of devices hopping in the spectrum. It shows that BigBand can decode the packets from 30 devices spanning a bandwidth of 900 MHz with a packet loss rate less than 3.5%. Decoding all these transmitters without BigBand would either require a wideband 0.9 GHz receiver, or a receiver with 30 RF-frontends, both of which would be significantly more costly and power-hungry.

ADC Quantization	BigBand vs Narrowband RX	
	mean	max
8 bits	-2.73 dB	-2.78 dB
14 bits	-5.68 dB	-5.89 dB

Table 7.2: **Reduction in SNR at Different Quantization Levels**

7.8.2 Signal-to-Noise Ratio

It is expected that BigBand will have more noise than a narrowband receiver since it can capture a much larger bandwidth. This section aims to shed insight on this issue. We note three types of noise: thermal noise, quantization noise and ADC jitter noise [2]. BigBand has higher thermal noise due to bucketization. Specifically, since in our implementation, the 900 MHz bandwidth is aliased into 50 MHz, it is expected that the thermal noise would increase by $18\times$ (12.5 dB). However, quantization noise and ADC jitter noise do not alias, and hence do not increase. The overall increase in noise depends on how the thermal noise compares to these other types of noise.

To understand the impact of thermal noise and quantify the SNR performance of BigBand we compare it with a 50 MHz narrowband receiver that uses the same USRP hardware. We transmit a 10 MHz signal, receive it on BigBand and the narrowband receiver, and compare the resulting SNR. We connect BigBand and the narrowband receiver to the same antenna and ensure that both receivers' rx-gains are set properly so that the received signal amplitude spans the same range on both receivers. We run it for different receive signal strengths and measure the SNR on each. We repeat the measurements for the ADC quantization set to 8 bits and 14 bits to better understand the interaction between thermal noise and quantization noise.

Table 7.2 shows the mean and max reduction in SNR of a signal received on BigBand relative to the narrowband USRP. The result shows that at 8 bit quantization, the reduction is a little less than 3 dB which means that the 12 dB increase in thermal noise only translates to 3 dB reduction in SNR due to quantization and jitter noise. At a quantization of 14 bits, the SNR reduction becomes 6 dB which means that the ADC jitter noise is still significantly higher than thermal noise. Though this reduction in SNR is significant compared to narrowband receivers, one would require using 18 such receivers to capture in realtime the same 900 MHz bandwidth as BigBand which is not practical in terms of cost and bulkiness.

7.9 D-BigBand's Sensing Results

In this section, we evaluate D-BigBand's ability to sense changes in spectrum occupancy independent of sparsity. We implement D-BigBand as described in Section 7.6. We vary the percentage of total occupied frequencies in the spectrum between 1% (sparse) to 95% (almost fully occupied). We then change the number of frequencies that change occupancy every 1 ms by up to 1% (*i.e.*, 10 MHz), and evaluate D-BigBand's accuracy in identifying the frequencies that change occupancy.

As a function of spectrum occupancy, Figure 7-8 shows the false positives (*i.e.*, frequencies

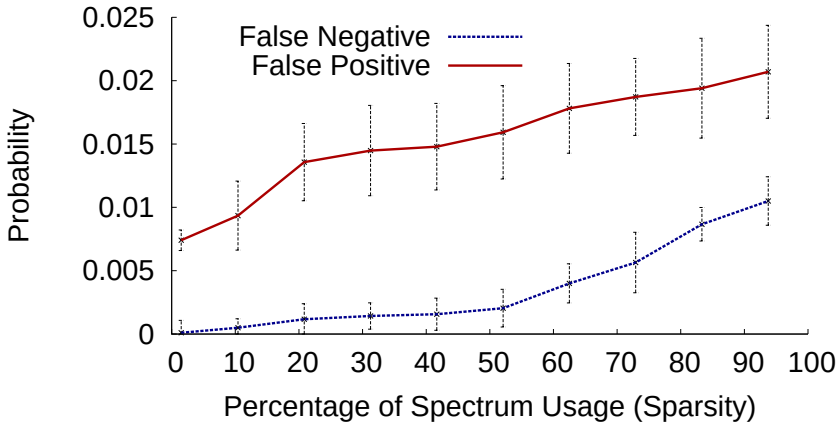


Figure 7-8: **D-BigBand's Effectiveness as a Function of Spectrum Sparsity:** Over a band of 1 GHz, D-BigBand can reliably detect changes in spectrum occupancy even when the spectrum is 95% occupied, as long as the change in spectrum occupancy is less than 1% every ms.

whose occupancy has not changed, but D-BigBand erroneously declared as changed) and false negatives (*i.e.*, frequencies whose occupancy has changed, but D-BigBand erroneously declares as unchanged). We see that D-BigBand robustly identifies changes in occupancy, with both the false positive and the false negative probabilities remaining under 0.02 even for a spectrum occupancy of 95%.

7.10 Conclusion

This chapter presented BigBand, a system that enables GHz-wide sensing and decoding using commodity radios. As a spectrum sensing device, it could sense the occupancy of the spectrum under both sparse and non-sparse cases. As a reception device, it is the first receiver that can decode a sparse signal whose bandwidth is larger than its own digital bandwidth. Empirical evaluation demonstrates that BigBand is able to sense the spectrum stably and dynamically under different sparsity levels; we also demonstrate BigBand's effectiveness as a receiver to decode GHz-wide sparse signals. We believe that BigBand enables multiple applications that would otherwise require expensive and power hungry devices, *e.g.* realtime spectrum monitoring, dynamic spectrum access, concurrent decoding of multiple transmitters in diverse parts of the spectrum.

Chapter 8

Faster GPS Synchronization

8.1 Introduction

The global positioning system (GPS) is one of the most pervasive wireless technologies. It is incorporated in more than one billion smartphones world-wide [78], and embedded in a wide variety of devices, including personal navigation systems [168], sensors [39], digital cameras [135], and even under-the-skin bio-chips [67]. The key functionality of a GPS receiver is to calculate a position, called a fix. Computing a fix involves locking on the GPS satellite signals and decoding satellite orbit and time data. Most GPS receivers, however, are embedded with some other radio (e.g., WiFi, cellular, or ZigBee) and, hence, can download the content of the GPS signal from assisted GPS (A-GPS) servers instead of decoding it from the satellite signals [92].[1] With assisted GPS used widely in phones and other GPS-capable devices [50], the bulk of what a GPS receiver does is to lock on the satellite signal (i.e., synchronize with it). This allows the receiver to calculate the sub-millisecond synchronization delay necessary for computing its position [51]. The importance of locking is further emphasized by the fact that current GPS receivers are typically duty-cycled [22, 152]; hence, they need to re-synchronize with the satellite signals regularly. Synchronizing with the satellite signal, however, is a costly process that requires tens of millions to a few billion digital multiplications [167]. Many GPS-enabled devices (e.g., mobile phones, sensors, etc.) have strict power limitations and would benefit from reducing the complexity of this process.

In this chapter, we aim to reduce the cost of synchronizing with weak signals like GPS. At a high level, GPS synchronization works as follows: each satellite is assigned a CDMA code. For each satellite, the receiver needs to align the corresponding CDMA code with the received signal. The process is complicated because GPS signals are very weak (about 20 dB *below* the noise level [144]). To find the right alignment of each satellite, a GPS receiver conducts a search process. It computes the correlation of the CDMA code with the received signal for all possible shifts of the code with respect to the signal. The correct shift is the one that maximizes the correlation.

So, how does a GPS receiver compute all these shifted correlations? The traditional approach *convolves* the received signal with the CDMA code of each satellite in the time domain. The correct alignment corresponds to the one that maximizes this convolution. This approach has a computa-

[1] The data includes almanac, ephemeris, reference time. AGPS may also provide other optional assistance data [92].

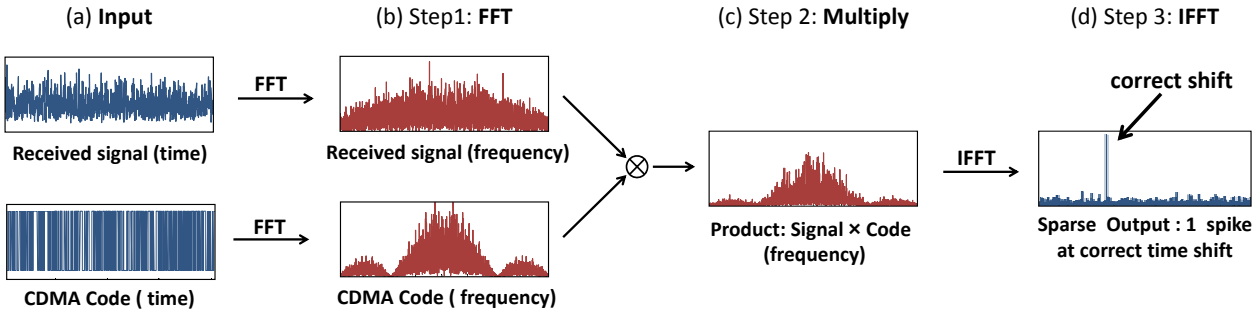


Figure 8-1: **FFT-Based GPS Synchronization Algorithm**. The algorithm multiplies the FFTs of the received signal with the FFT of the code, and takes the IFFT of the resulting signal. The output of the IFFT spikes at the shift that correctly synchronizes the code with the satellite signal.

tional complexity of $O(n^2)$, where n is the number of samples.[2] More recent GPS receivers lock on the satellite using frequency domain computation. This approach leverages the fact that convolution in the time domain corresponds to multiplication in the frequency domain. It proceeds in the following three steps, shown in Figure 8-1: 1) The receiver takes the FFT of the received signal; 2) It multiplies the output of this Fourier transform by the FFT of the CDMA code; and 3) It performs the inverse FFT on the resulting signal. This 3-step process is mathematically equivalent to convolving the signal with the code; thus, the output of the inverse FFT will spike at the correct shift that synchronizes the code with the received signal, as shown in Figure 8-1(d). The computational complexity of this approach is $O(n \log n)$. For the past two decades, this has been the algorithm with the lowest computational complexity for synchronizing a GPS receiver [167].

This chapter introduces the lowest complexity GPS synchronization algorithm to date. Our synchronization algorithm is based on the following observations:

- First, we note that since the output of the synchronization process has a single major spike at the correct shift, as shown in Figure 8-1(d), the inverse FFT is very sparse. We build on the Sparse Fourier Transform algorithms from Part I of this thesis to significantly reduce the runtime of the GPS synchronization algorithm.[3] However, the Sparse Fourier Transform algorithms presented in Part I use relatively complex filters and estimation techniques to deal with the interaction of multiple potential spikes at the output of the transform. In contrast, here, we exploit the fact that the synchronization problem produces only one spike, and design a simple sublinear algorithm that uses only aliasing to filter the signal. This allows us to reduce the complexity of the IFFT step in Figure 8-1(d) to sublinear time.

[2]The CDMA code consists of 1023 chips transmitted at 1.023 MHz. For a GPS receiver that samples at 5 MHz, the computational complexity of the shifted correlation is $(1023 \times 5/1.023)^2$, which is about 25 million multiplications of complex signal samples. The GPS receiver has to repeat this process for multiple satellites (between 4 to 12 satellites) and multiple Doppler shifts (between 21 to 41 shifts) for each satellite, which brings the number of multiplications to over a billion. Further, correlating with one block of the signal may not be sufficient. For weak signals, the receiver may need to repeat this process and sum up the output [96].

[3]Sparse Fourier Transform algorithms are designed for the case where the output of the Fourier Transform contains only a small number of spikes. Hence, they are applicable to both Sparse Fourier Transform and Sparse Inverse Fourier Transform. For a more detailed description see Section 8.3.

- Although the output of the inverse FFT is sparse and can be quickly computed, the GPS signal in the frequency domain is not sparse (Figure 8-1(b)) and, hence, the runtime of the forward FFT cannot be reduced by applying a Sparse Fourier Transform. Thus, simply using Sparse Inverse Fourier Transform does not reduce the overall complexity of the problem (which is still $O(n \log n)$ due to the forward FFT). To address this issue, we note that the FFT in Figure 8-1(b) is just an intermediate step that will be used as an input to the Sparse Inverse Fourier Transform. Since the Sparse Inverse Fourier Transform algorithms operate only on a subset of their input signal, we do not need to compute the values of all frequencies at the output of the forward FFT. We leverage this property to compute only a subset of the frequencies and reduce the complexity of the FFT step.

We provide an algorithm that, for any SNR, is as accurate as the original FFT-based (or convolution-based) algorithm, but reduces the computational complexity from $O(n \log n)$ operations to $O(n\sqrt{\log n})$. Further, when the noise in the received signal can be bounded by $O(n/\log^2 n)$, we prove that the same algorithm has a linear complexity, i.e., $O(n)$.[4]

We implement our design and test it on two datasets of GPS signals: We collected the first dataset in the US using software radios. The second dataset was collected in Europe.[5] The datasets cover both urban and suburban areas. We compare our design against an FFT-based synchronization algorithm. Our design reduces the number of multiplications for detecting the correct shift by a median of $2.2\times$. Since a large fraction of GPS power is consumed by the synchronization process (30% [141] to 75% [142] depending on the required accuracy), we expect the new design to produce a significant reduction in GPS power consumption.

Finally, this chapter makes both algorithmic and systems contributions, which can be summarized as follows:

- It presents the fastest algorithm to date for synchronizing GPS receivers with satellite signals. The algorithm has multiple features: 1) It is adaptive, i.e., it can finish faster if the SNR is higher; 2) it continues to work at very low SNRs; and 3) it is general, i.e., it can be used to synchronize any signal with a random (or pseudo random) code.

- It provides an implementation and an empirical evaluation on real GPS signals, demonstrating that the algorithmic gains translate into a significant reduction in the number of operations performed by a GPS receiver.

8.2 GPS Primer

The key functionality of a GPS receiver is to calculate its position using the signal it receives from the GPS satellites. To do so, the receiver computes the time needed for the signal to travel

[4]Note that n is not a constant and varies across GPS receivers. Specifically, different receivers sample the GPS signal at different rates, hence obtaining a different number of samples per codeword. For example, for a receiver whose sampling rate is 5MHz, n=5000, whereas for a 4MHz receiver, n=4000.

[5]The Europe dataset is courtesy of the GNSS-SDR team [49] at the Centre Tecnologic de Telecomunicacions de Catalunya (CTTC).

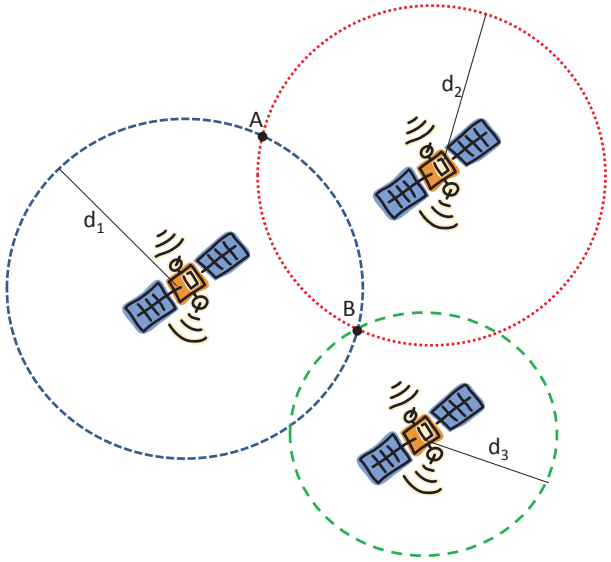


Figure 8-2: **GPS Trilateration:** After determining the distance to different satellites, the receiver can draw spheres centered at each of the satellites and whose radii are the respective distances. These spheres should intersect at the receiver's position. A GPS receiver needs four satellites to uniquely determine its position [96]. Extra satellites can be used to correct for the lack of very tight synchronization between the receiver's clock and those of the satellites.

from each satellite to itself. It then multiplies the computed time by the speed of light to obtain its distance from each satellite. As a result, the receiver knows that it lies on a sphere centered at that satellite and whose radius is the computed distance. It then determines its position as the intersection of several such spheres through a method called trilateration [96] shown in Figure 8-2.

But how does the receiver compute the propagation time from the satellites? The propagation time is obtained using a synchronization algorithm that allows the device to lock on the received signal. Specifically, each satellite has its own CDMA code, called the C/A code, which consists of 1023 chips [96]. Assuming the receiver's and satellites' clocks are perfectly synchronized, a GPS receiver generates the satellites' codes at the same time as the satellites. Due to propagation delay, however, the signal arrives in a shifted version at the receiver by exactly the amount of time it took the signal to travel from the satellite. By correlating with shifted versions of the satellite's code, the receiver calculates the propagation time as the shift at which the correlation spikes [167]. In practice, the receiver's clock is not fully synchronized with that of the satellites; this, however, can be compensated for by increasing the number of satellites used in the trilateration process.[6]

The motion of the satellites introduces a Doppler shift in the received signal. The signal does not correlate with the C/A code unless the Doppler shift is corrected. To deal with this issue, a

[6]All GPS satellites use atomic clocks and are fully synchronized with each other [96]. Hence, a GPS receiver will have the same clock skew with respect to all satellites and all the estimated propagation delays will have the same error ϵ. However, trilateration needs only 4 satellites to estimate the position and thus extra satellites can be used to to estimate and correct ϵ.

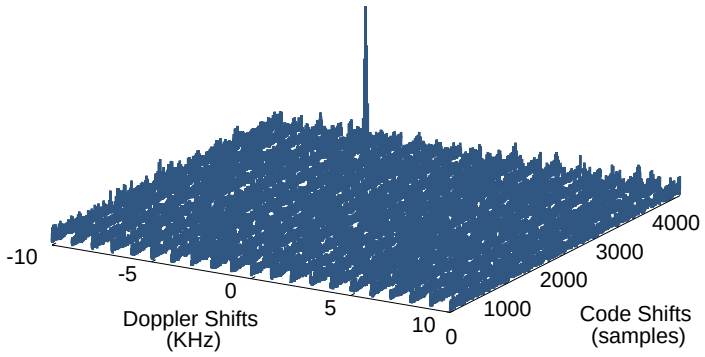


Figure 8-3: **2D Search for Peak Correlation.** The plot shows the result of correlating with a C/A code for a satellite whose signal is present in the received signal. On the x-axis, we search 4000 different code shifts and on the y-axis 21 different Doppler shifts.

GPS device typically performs a 2-dimensional search on the received signal [96]: one for time (code shifts), and one for Doppler shifts. Specifically, the receiver tries all possible code shifts, and 41 equally spaced Doppler shifts within +/-10 kHz of the center frequency [167], as shown in Figure 8-3. Finally, the GPS satellites repeat the code 20 times for each data bit to enable the GPS receiver to decode very weak signals. The receiver tries to use one code repetition to synchronize. However, if the signal is too weak, the receiver repeats the 2D-search for multiple codes and sums the result [96].

8.3 QuickSync

We describe QuickSync, a synchronization algorithm for GPS receivers. The algorithm works in the frequency domain similar to the FFT-based algorithm described in Section 8.1. QuickSync, however, exploits the sparse nature of the synchronization problem, where only the correct alignment between the received GPS signal and the satellite code causes their cross-correlation to spike. QuickSync harnesses this property to perform both the Fourier and inverse Fourier transforms in a time faster than $O(n \log n)$, therefore reducing the overall complexity of GPS synchronization.

The next subsections formalize the problem and detail the algorithm.

8.3.1 Problem Formulation

The synchronization problem can be formulated as follows: Given a spreading code $\mathbf{c} = c_0, \ldots, c_{n-1}$ of size n and a received signal $\mathbf{x} = x_0, \ldots, x_{n-1}$, find the time shift $\hat{t}$ that maximizes the correlation between $\mathbf{c}$ and $\mathbf{x}$, i.e., compute:

$$\hat{t} = \arg\max \mathbf{c}_{-n} \circledast \mathbf{x}, \tag{8.1}$$

where $\circledast$ is a circular convolution and $\mathbf{c}_{-n}$ is the time reversed code; i.e. $\mathbf{c}_{-n} = c_{n-1}, \ldots, c_0$. Computing this convolution in the time domain requires performing n correlations each of size n and thus has complexity $O(n^2)$. However, convolution in the time domain corresponds to element-by-element multiplication in the frequency domain. Therefore computing the convolution in Equation 8.1 can be done more efficiently by performing FFT on each of the code and the signal, multiplying those FFTs, then performing an inverse FFT (IFFT) as shown below:

$$\arg\max_{t} \mathbf{c}_{-n} \circledast \mathbf{x} = \arg\max_{\mathbf{t}} \mathcal{F}^{-\mathbf{1}}\{\mathcal{F}\{\mathbf{c}\}^* \cdot \mathcal{F}\{\mathbf{x}\}\}, \tag{8.2}$$

where $\mathcal{F}(.)$ is the FFT, $\mathcal{F}^{-1}(.)$ is the IFFT, $*$ is the complex conjugate and t is any time sample in the output vector of the convolution. This reduces the complexity of the synchronization process to $O(n\log(n))$. Accordingly, in the remainder of this chapter, we only consider the FFT-based synchronization algorithm as a baseline for evaluating QuickSync's performance.

8.3.2 Basics

Before introducing our synchronization algorithm, we remind the reader of a basic subsampling/aliasing property of the Fourier transform, which we have introduced in Chapter 1 and have previously used in Chapters 3, 5, and 7. However, here we will focus on the dual of this property which states that: *Aliasing a signal in the time domain is equivalent to subsampling it in the frequency domain, and vice versa.* Figure 8-4 illustrates this property.

Formally, let $\mathbf{x}$ be a discrete time signal of length n, and $\mathbf{X}$ its frequency representation. Let $\mathbf{x}'$ be a version of $\mathbf{x}$ in which adjacent windows of size B (where B divides n) are aliased on top of each other (i.e., samples that are $p = n/B$ apart are summed together). Then, for $t = 0 \ldots B-1$:

$$\mathbf{x'}_t = \sum_{j=0}^{n/B-1} \mathbf{x}_{t+jB}. \tag{8.3}$$

Thus, $\mathbf{X}'$, the FFT of $\mathbf{x}'$ is a subsampled version of $\mathbf{X}$, and for $f = 0 \ldots B-1$

$$\mathbf{X'}_f = \mathbf{X}_{pf}, \tag{8.4}$$

where $p = n/B$, and the subscript in $\mathbf{X}_{pf}$ refers to the sample whose index is $p \times f$.

8.3.3 The QuickSync Algorithm

We describe how QuickSync operates on a received GPS signal to synchronize it with an internally generated C/A code. For simplicity, we assume that the input signal neither exhibits a carrier frequency offset nor a Doppler shift; in later sections, we extend the algorithm to deal with these frequency offsets. Furthermore, in this section, we describe the algorithm in the context of synchronizing the GPS receiver with the signal of only one satellite; the algorithm can be easily adapted for synchronizing with multiple satellites.

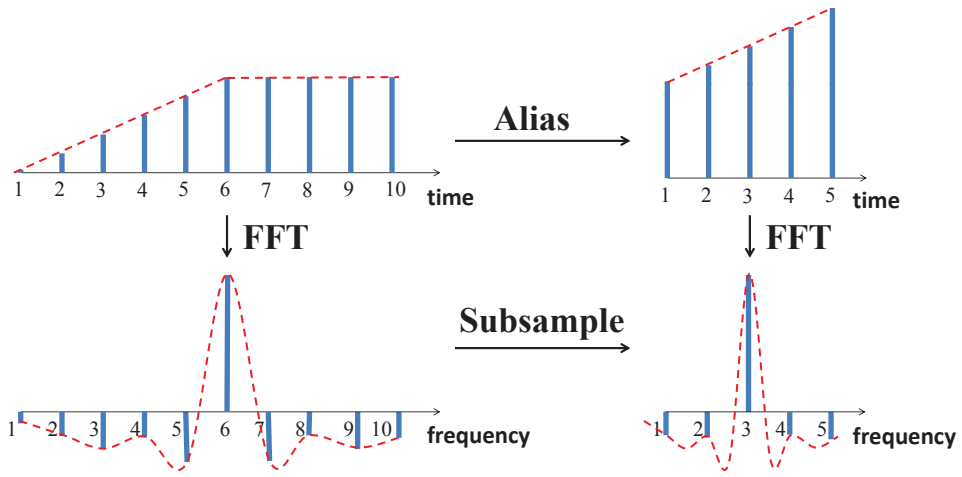


Figure 8-4: **The Duality of Aliasing and Subsampling.** Aliasing in the time domain corresponds to subsampling in the frequency domain and vice versa. Folding (aliasing) the time domain signal in the top left results in the signal in the top right; specifically, time samples 1 and 6 add into sample 1 in the aliased signal, samples 2 and 7 into sample 2, etc. In the Fourier domain, the FFT of the aliased signal is a subsampled version of the FFT of the initial signal; namely, sample 1 in the bottom right signal corresponds to sample 2 in the bottom left, sample 2 corresponds to sample 4, etc.

The key insight to our algorithm is that the IFFT performed in step 3 of the FFT-based synchronization algorithm is sparse in the time domain, i.e., it has only one spike and, hence, can be performed in sub-linear time. Further, a sub-linear time algorithm for computing the Sparse Inverse Fourier Transform would require a sub-linear number of samples as input; thus, there is no need to perform a full $n \log n$ FFT on the received GPS signal and obtain all of its n frequency samples. Rather, we only need to compute the frequency samples that will be used to perform the Sparse Inverse Fourier Transform.

Below, we explain how we exploit these ideas to reduce the complexity of both the IFFT and FFT performed to synchronize the signal with the code. We then put these components together in a complete algorithm.

(a) Sparse Inverse Fourier Transform

We develop a simple algorithm to efficiently perform the IFFT step of GPS synchronization and quickly identify the spike of the correlation between the received signal and the CDMA code. To do so, our algorithm uses a sub-linear number of samples of the signal.

The Sparse Inverse Fourier Transform algorithm proceeds as follows. It first subsamples the frequency domain signal of size n by a factor of p. It then computes the IFFT over these n/p frequency samples. Recall that subsampling in the frequency domain is equivalent to aliasing in the time domain. Thus, the output of our IFFT step is an aliased version of the output in the original IFFT step shown in Figure 8-1. Aliasing here can be viewed as a form of hashing, where

the n original outputs samples, i.e. time shifts, are hashed into n/p buckets. Time shifts which are n/p apart will be summed and hashed together in the same bucket at the output of our IFFT. Since there is only one correlation spike in the output of the IFFT, the magnitude of the bucket it hashes into will be significantly larger than that of other buckets where only noise samples hash to. Hence, the algorithm chooses the bucket with the largest magnitude among the n/p buckets at the output of our IFFT.

Out of the p time shifts that aliased (or hashed) into this chosen bucket, only one is the actual correlation spike. To identify the spike among these p candidate shifts, the algorithm correlates the received signal with each of those p shifts of the CDMA code. The shift that produces the maximum correlation is the right spike.

(b) Subsampled FFT

With the sparse IFFT step in place, the algorithm does not need the whole n-point FFT of the signal. Specifically, all the IFFT requires is a subsampled version of this signal. Thus, rather than taking a full n-point FFT, QuickSync aliases the received signal in the time domain before taking its FFT, as in Equation 8.3 (Said differently, QuickSync sums up blocks of size n/p and then computes a smaller FFT of size n/p.) The output of this FFT, expressed in Equation 8.4, is exactly the samples we need at the input of the sparse IFFT, described above.

A subsampled input to the IFFT (as described in Section 8.3.3(a)) results in an output spike of smaller magnitude relative to the noise bed. To compensate for this loss, we alias $p \times n$ samples instead of n into blocks of size n/p before performing the FFT.

(c) Full Algorithm

The QuickSync algorithm proceeds in the following steps:

1. **Aliasing:** Alias $p \times n$ samples of the GPS signal into $B = n/p$ samples as described in Equation 8.3, where $p = \sqrt{\log n}$.

2. **Subsampled FFT:** Perform an FFT of size n/p on the aliased time signal. This is effectively equivalent to performing an FFT of size pn and subsampling the output by p^2 according to Equation 8.4.

3. **Multiplying with the code:** Subsample the FFT of the satellite CDMA code of length n by p, and multiply the resulting samples by the n/p samples at the output of step 2, above. Note that the algorithm can precompute the FFT of the CDMA code and store it in the frequency domain.

4. **Sparse Inverse Fourier Transform:** Perform an IFFT on the n/p resulting samples. Since the input of this IFFT was subsampled, its output is aliased in the time domain. Specifically, each of the n/p buckets at the output of this stage is effectively the sum of p aliased time samples [7] as described in Section 8.3.3(a).

[7] Note that we only get p candidate shifts (and not p^2) because the actual code is of size n; hence, all shifts mod n are the same. Thus, although the total number of samples is np and they are aliased into n/p buckets, we only have p

5. **Find the unique solution:** Find the bucket with the maximum magnitude among the n/p buckets. Then, check the correlation of each of the p possible time shifts which are aliased into this bucket, and pick the shift that gives the maximum correlation. Checking the correlation can be done using only n/p samples as per Lemma F.3.3; therefore, it takes a total of $p \times n/p = n$ to perform the correlation of the p shifts and pick the one that maximizes the correlation.

(d) Runtime

The running time of the QuickSync algorithm may be computed as follows. Step 1 performs np additions. Step 2 performs an FFT which takes $n/p \log(n/p)$. Step 3 performs n/p multiplications. Step 4 takes $n/p \log(n/p)$ to perform the IFFT, and finally Step 5 performs n operations to compute the correlations and find the solution. Thus, the complexity of QuickSync is $O(pn + (n/p)\log(n/p))$. To minimize this complexity, we set $p = \sqrt{\log n}$ which makes the overall running time of QuickSync $O(n\sqrt{\log n})$.

(e) Scaling with the SNR

If the signal is too weak, GPS receivers repeat the synchronization algorithm on subsequent signal samples and sum up the output to average out the noise [167]. This approach allows the receiver to scale the computation with the SNR of the signal. The approach can be applied independent of the algorithm; hence, we also adopt it for QuickSync. However, QuickSync operates on blocks of size pn whereas the traditional FFT-based algorithm operates on blocks of size n. Both QuickSync and the traditional FFT-based algorithm compare the magnitude squared of the largest spike to the noise variance in the received signal. If the largest spike's squared magnitude exceeds the noise variance by a desired margin, the algorithm terminates the search and declares the time shift corresponding to the largest spike as the correct alignment. Otherwise, the algorithm repeats the same process on the subsequent signal samples, and sums the new output with the previous one. Since the spike corresponding to the correct synchronization is at the same time shift in each run, it becomes more prominent. In contrast, noise spikes are random and hence they tend to average out when combining the output of multiple runs of the synchronization algorithm.

(f) Linear Time Algorithm

The algorithm described in Section 8.3.3(c) above can be made linear-time by modifying Step 1: instead of taking pn samples, we take only n samples and alias them into n/p buckets, where $p = \log n$. The rest of the steps are unmodified. This reduces the complexity of Step 1 to n, and the total complexity of the algorithm to $O(n + (n/p)\log(n/p)) = O(n)$.

This linear-time algorithm has weaker guarantees than the above super-linear algorithm and may not always work at very low SNRs, as detailed in Section 8.4. One can try this algorithm first. If a spike is detected with the required margin, the algorithm terminates. Otherwise, one can fall back to the super-linear algorithm in Section 8.3.3(c).

---

distinct shifts per bucket.

8.4 Theoretical Guarantees

In this section, we analyze the performance of the baseline and QuickSync algorithms (both the linear and super-linear variants), under natural probabilistic assumptions about the input signal $\mathbf{x}$. In particular, we show that both the baseline and the super-linear QuickSync are correct under the same asymptotic assumptions about the variance of the noise in the signal $\mathbf{x}$. At the same time, the running time of our algorithm is equal to $O(pn + (n/p)\log(n/p))$, where p is the number of blocks used. This improves over the baseline algorithm which has $O(n \log n)$ runtime as long as the term pn is smaller than $(n/p)\log(n/p)$. In particular, by setting $p = \sqrt{\log n}$, we achieve the running time of $O(n\sqrt{\log n})$.

8.4.1 Assumptions

Recall that we use $\mathbf{c} = c_0 \dots c_{n-1}$ to denote the spreading code. We use $\mathbf{c}^{(t)}$ to denote the code $\mathbf{c}$ shifted by $t = 0 \dots n-1$, i.e., $c_i^{(t)} = c_{t+i \mod n}$. We have that $\mathbf{x} = \mathbf{c}^{(t)} + \mathbf{g}$ for some shift t, where $\mathbf{g}$ denotes the noise vector. We make the following assumptions:

1. The coordinates $g_0 \dots g_{n-1}$ of the noise vector $\mathbf{g}$ are independent and identically distributed random variables that follow a normal distribution with zero mean and variance σ. That is, we assume additive white Gaussian noise (AWGN).

2. The coordinates $c_0 \dots c_{n-1}$ of the spreading code $\mathbf{c}$ are independent and identically distributed random variables with values in $\{-1, 1\}$, such that $\Pr[c_i = 1] = \Pr[c_i = -1] = 1/2$. This assumption models the fact that the CDMA code, $\mathbf{c}$, is *pseudorandom*.

8.4.2 Combining Multiple Runs

As described in Section 8.3.3(e), both the baseline and our algorithm can sum the output of multiple runs to average out the noise and increase the probability of identifying the correct spike. The analysis of such multi-run scenario can be derived directly from a single run. Specifically, say the algorithm runs L times and sum up the outputs of these L runs. This is equivalent to reducing the noise variance to $\sigma' = \sigma/L$. Therefore, the L-run scenario can be analyzed by reducing it to the case of a single run, with variance divided by L.

8.4.3 Guarantees

Here, we walk the reader through the guarantees. The proofs of the guarantees can be found in Appendix F. We start by stating the sufficient condition for the baseline algorithm to work with probability approaching 1.

Theorem 8.4.1. *Assume that $\sigma \leq c(n)n/\ln n$ for $c(n) = o(1)$. Then the baseline algorithm is correct with probability $1 - o(1)$.*

The proof is in Appendix F.1. The above condition is also tight. Specifically,

Theorem 8.4.2. *There exists a constant $c > 0$ such that for $\sigma \geq cn/\ln n$, the baseline algorithm is incorrect with probability $1 - o(1)$.*

The proof is in Appendix F.2. We then proceed with the analysis of the two variants of the QuickSync algorithm. The first statement holds for the super-linear variant, and shows that the algorithm works with probability approaching 1 under the same condition as the baseline algorithm, while being faster.

Theorem 8.4.3. *Assume that $\sigma \leq c(n)n/\ln n$ for $c(n) = o(1)$, and that $p = o(n^{1/6})$. Then, the QuickSync algorithm that aliases p blocks of size n into n/p buckets is correct with probability $1 - o(1)$. The running time of the algorithm is $O(pn + (n/p)\log(n/p))$, which is $O(n\sqrt{\log n})$ for $p = \sqrt{\log n}$. Moreover, the algorithm performs only $O(n + (n/p)\log(n/p))$ multiplications, for any p.*

Finally, we analyze the linear-time variant of the QuickSync algorithm.

Theorem 8.4.4. *Assume that $\sigma \leq c(n)\frac{n}{p\ln n}$ for $c(n) = o(1)$, and that $p = o(n^{1/6})$. Then, the QuickSync algorithm that aliases one block of n samples into n/p buckets is correct with probability $1 - o(1)$. The running time of the algorithm is $O(n + (n/p)\log(n/p))$, which is $O(n)$ for $p > \log n$.*

The proof of the above two theorems can be found in Appendix F.3.

8.5 Doppler Shift & Frequency Offset

GPS satellites orbit the Earth at very high speeds. Consequently, the GPS signal arrives at the receiver with a Doppler shift. This shift is modeled as a frequency offset f_d which is a function of the relative speed of the satellite (see Chapter 2 in [64] for exact calculations). Furthermore, the discrepancy between the RF oscillators of the GPS satellite and the GPS receiver induces a carrier frequency offset Δf_c. The total frequency offset $\Delta f = f_d + \Delta f_c$ typically ranges from -10 kHz to 10 kHz [167] and is modeled as a phase shift in the received samples. Formally, if $\mathbf{x}$ and $\tilde{\mathbf{x}}$ are respectively the signal without and with a frequency offset then:

$$\tilde{x}_t = x_t e^{j2\pi\Delta f t}, \tag{8.5}$$

where t is time in seconds.

Like past synchronization algorithms, QuickSync must search and correct for the frequency offset in the received GPS signal in order for the correlation to spike at the correct code shift. However, since QuickSync processes $p \times n$ samples as opposed to n samples in past algorithms(see Section 8.3), it needs to deal with larger phase shifts that accumulate over pn samples. In order to overcome this limitation, QuickSync performs a finer grained frequency offset search, which introduces an overhead to the 2D search. This overhead, however, is amortized across all satellites in the GPS signal since correcting for this frequency offset is done on the received signal before it is multiplied by each satellite's C/A code. In Section 8.7.2, we show that despite this overhead,

(a) Sampler (b) Antenna

Figure 8-5: **The SciGe GN3S Sampler.** The sampler is used to collect raw GPS data. It down-coverts the received signal and delivers the I and Q samples to the computer.

QuickSync still provides a significant reduction in the computational complexity of GPS synchronization. Furthermore, the frequency offset changes slowly (see Section 8.7.2); hence, the receiver can cache its value from recent GPS readings, and does not need to search for it for every GPS synchronization event.

8.6 Testing Environment

8.6.1 Data Collection

We test our algorithm on a data set consisting of 40 GPS signal traces captured from urban and suburban areas in US and Europe. The traces in US are collected using the SciGe GN3S Sampler v3 [45] shown in Figure 8-5(a). The GN3S is a form of software radio that collects raw complex GPS signal samples. We set the sampling rate of the GN3S to 4.092 MHz and its carrier frequency to 1575.42 MHz. The traces from Europe are collected using the USRP2 software radio [47] and the DBSRX2 daughterboard, which operates in the 1575.42 MHz range and is capable of powering up active GPS antennas [47]. The Europe traces are collected with a sampling frequency of 4 MHz. We also use a 3V magnetic mount active GPS antenna shown in Figure 8-5(b). These datasets allow us to test the performance of QuickSync in different geographical areas and for different GPS receiver hardware.

8.6.2 Baseline Algorithm

We compare our algorithm against a baseline that uses the traditional FFT-based synchronization [176]. The baseline algorithm operates on blocks of size n. If a spike is not detected after processing the first block, the algorithm repeats the computation on the next block, i.e., the next set of n samples, and sums up the output of the IFFTs. The algorithm keeps processing more blocks until the magnitude of the peak crosses a certain threshold (as described in Section 8.7.1). Note that the algorithm must sum up the magnitudes of the output of the IFFTs rather than the actual complex values; otherwise, samples would combine incoherently due to the accumulated phase caused by the Doppler shift (see Section 8.5).

8.6.3 Implementation

We implement both QuickSync and the FFT-based algorithm in Matlab and run them on the collected GPS traces. Both algorithms use the FFTW [54] implementation internally to compute the Fourier transform (though the baseline computes an n-point transform while QuickSync computes an n/p-point transform).

8.6.4 Metrics

We use two metrics for comparing the algorithms: number of multiplications, and number of floating point operations (FLOPs). We mainly focus on the number of real multiplications executed until an algorithm finds the synchronization offset. This metric is particularly important for hardware-based GPS synchronization, where multiplications are significantly more expensive than additions [148], and serve as standard metric to estimate the complexity of a potential hardware implementation [30, 165].

Some GPS-enabled devices do not have a full fledged GPS receiver hardware to reduce cost and form factor [121]. They use a GPS radio to collect signal samples, but offload the synchronization algorithm to the main CPU of the device, where it is done in software. To evaluate QuickSync's performance on software-based GPS receivers, we count the number of FLOPs executed by both QuickSync and the baseline. FLOPs is a standard metric used to evaluate software implementations of algorithms, including FFTW [54]. It includes both multiplications and additions.

We count the FLOPs using OProfile, a standard profiler for Linux systems [139]. We run the code in Matlab R2011b under Ubuntu 11.10 on a 64-bit machine with Intel i7 processor. We run OProfile from within Matlab in order to profile the part of the code executed by each algorithm, and get a more accurate estimate of the number of FLOPs. We program OProfile to log the counter INST_RETIRED (the number of executed floating point operations on the Intel i7 processor [139]).

8.7 Results

8.7.1 Setting the Synchronization Threshold

As explained in Section 8.3.3(e), both QuickSync and the FFT-based synchronization algorithm check that there is a sufficient margin between the detected maximum spike and the noise level, before accepting the spike as the one that identifies the correct alignment. Specifically, they check that the ratio of the spike's magnitude squared to the noise variance exceeds a particular threshold. This threshold defines how large the spike has to be in comparison to the bed of noise to ensure enough confidence that the spike is not due to noise and is indeed due to the code matching. Hence, the threshold is a measure of the SNR of the spike and is not dependent on the data. In particular, if the GPS data is noisy as in an urban area, the algorithm will continue processing more data until the threshold is crossed (as discussed in Section 8.6). In contrast, if the GPS data is less noisy as in an open suburban area, the algorithm will terminate early on since the spike will cross the threshold after processing one or two blocks.

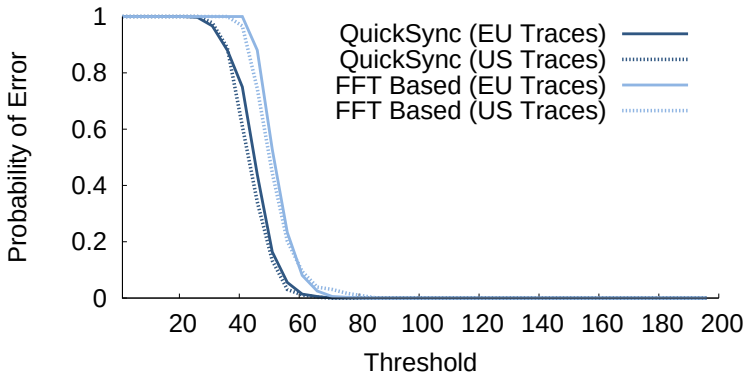


Figure 8-6: **Probability of Error Versus the Threshold.** The plot shows that the probability of error decreases sharply for both algorithms, and that a threshold of 90 for QuickSync and 100 for the baseline produce a zero error probability.

In this section, we aim to verify that there is such a threshold that works for all datasets. Thus, we perform the following experiment. We vary the threshold between a value of 1 and 200, and for each of those values, we run both algorithms on a subset of the GPS traces from both datasets. We define the probability of error as the ratio of runs that output a false positive (i.e., in which the algorithm terminates by returning an invalid shift) to the total number of runs at a given threshold.

Figure 8-6 plots the probability of errors versus the preset threshold. The plot shows that setting the threshold to 90 for QuickSync and 100 for the baseline produces a zero error probability. The baseline has a slightly higher error probability than QuickSync. This is because the baseline takes an n-point IFFT and, hence, has to ensure that none of the $n-1$ noise spikes exceeds the correct spike that corresponds to the proper alignment. In contrast, QuickSync takes an n/p-point IFFT and hence has fewer noise spikes that have to be kept below the threshold.

The figure also shows that the used metric is stable, i.e.: (1) the metric is consistent across traces captured from two continents, and (2) the probability of error decreases monotonically as the threshold increases. This shows that the threshold is independent of the location of the GPS receiver.

In the experiments that follow, we set the thresholds to 90 and 100 for QuickSync and the baseline respectively. We also use a different set of traces from those used in testing for this threshold to ensure separation between testing and training.

8.7.2 Performance in Terms of Hardware Multiplications

We start by evaluating the performance gain of QuickSync over FFT-based synchronization in terms of the number of hardware multiplications. We run each of QuickSync and the FFT-based algorithm on both traces collected in US and Europe. We run the experiment 1000 times; each time taking a different subset of samples from these datasets. We compare the total number of multiplications required by each of the algorithms to synchronize with the signals of satellites present in the GPS traces. Figure 8-7 shows a CDF of the gain. The gain is calculated as the number of mul-

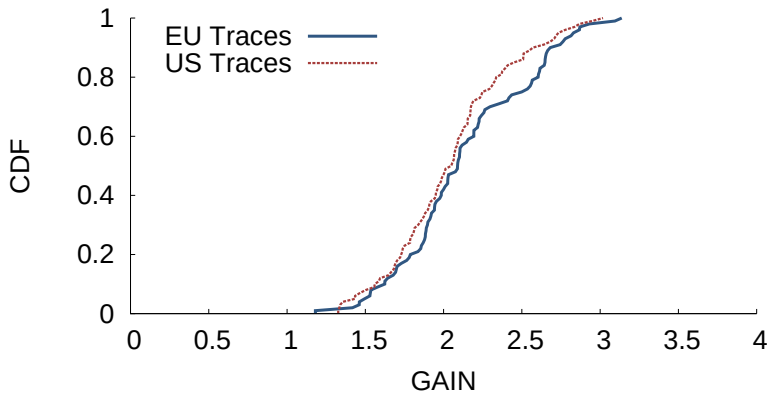


Figure 8-7: **Gain of QuickSync Over the FFT-based Algorithm in Number of Multiplications.** The two curves show the CDFs of the QuickSync's gains for the US and Europe datasets. QuickSync achieves a median gain of around 2.2× and a maximum gain of 3.3×.

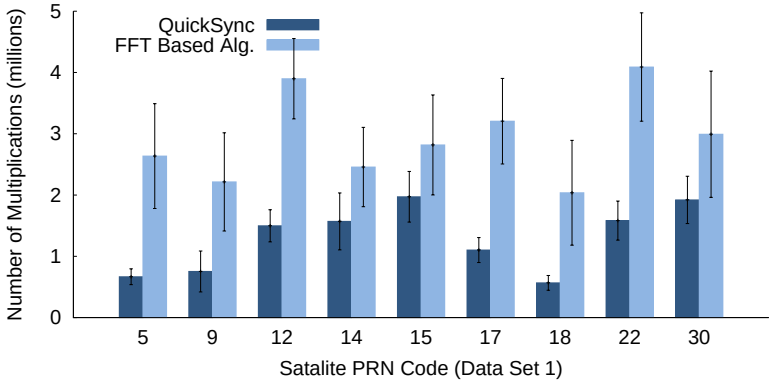


Figure 8-8: **Number of Multiplications on a Per Satellite Basis for the Europe Trace.** The gain of QuickSync over the FFT-based algorithm varies among different satellites and ranges between and 1.5× and 4×.

tiplications needed by the FFT-based algorithm divided by the number of multiplications required by QuickSync. The figure shows that QuickSync always outperforms the FFT-based synchronization on both the US and EU traces with a median gain of 2.2×. This means that QuickSync can save on average twice the number of hardware multiplications.

To better understand the performance of QuickSync we zoom in on the number of multiplications required by each algorithm for each of the satellites. Specifically, each point in the CDFs in Figure 8-7 corresponds to a full GPS reading with all satellites. However, because different satellites have different Doppler shifts and signal strengths, we expect the gains to vary from one satellite to another. Specifically, for each of the satellites detected in the Europe traces, and for each GPS reading, we measure the number of multiplications required by both algorithms to perform the synchronization. We repeat this experiment 1000 times on different subset of the samples and plot the average results in Figure 8-8.

Satellite Code:	9	12	14	15	18	21	22	25	27
Mean (Hz):	75	100	150	175	75	75	175	25	125
Max (Hz):	300	200	300	300	300	200	300	100	300

Table 8.1: **Variation in the Doppler Shift in the US Traces.** For a given satellite, the Doppler shift of the received signal varies very little over a period of 2 hours and in an area of 2-mile diameter.

Figure 8-8 shows that each of the satellites, on average, requires less multiplications using QuickSync. However, the gains vary considerably among those satellites. For example, satellites 5 and 18 have an average gain of 4× whereas satellites 14 and 15 have an average gain of only 1.5×. Examining the Doppler shifts of these satellites we find that satellites 5 and 18 have Doppler shifts of 6000 Hz and 1000 Hz respectively while satellites 14 and 15 have Doppler shifts of 600 Hz and 6800 Hz. This shows that the latter require a finer grain Doppler search as explained in Section 8.5. However, because QuickSync is opportunistic, it first attempts to search at courser grain shifts (the same as the baseline), but falls back to finer resolutions when it fails to detect a peak that passes the threshold. Even in such scenarios, however, it consistently outperforms the baseline as the figure shows.

In many scenarios, the receiver knows the Doppler shift a priori. The reason for this is that the Doppler shift varies only slightly between nearby locations and over a time period of about an hour. In order to test how much the Doppler shift varies, we measure the Doppler shift of satellite signals in the GPS traces captured at different locations within a 2-mile diameter geographical area over a period of 2 hours. For each of those satellites, we calculate the mean and the maximum variation in the Doppler shift of all those signals and record them in Table 8.1. The mean change is around 100 Hz and the maximum is 300 Hz. Accordingly, since GPS receivers are duty cycled, whenever the receiver wakes up, it may use the Doppler shift it calculated before going to sleep rather than performing an exhaustive search for it. Alternatively, assisted GPS receivers may download the measured Doppler shift from an adjacent base station [41]. In both of these situations, the GPS receiver can significantly reduce the overhead of searching for the right Doppler shift.

In order to measure the gains of QuickSync without the Doppler search, we repeat the first experiment but this time by providing each of the synchronization algorithms with the correct Doppler shift for each satellite. Figure 8-9 shows a CDF of QuickSync's gain over the FFT-based algorithm in terms of number of multiplications over all the runs on both traces. For both traces, QuickSync achieves a median gain of 4.8×. This shows that QuickSync's gains increase when the receiver caches the correct Doppler shift across readings. We note that some of the traces benefit from QuickSync much more than others; the reason is that these runs have higher SNRs such that QuickSync can synchronize to their signals using the linear-time algorithm without falling back to the super-linear variant.

8.7.3 Performance on software based GPS receivers

In this section, we test the performance of QuickSync on software based GPS receivers in terms of the number of floating point operations (FLOPs). We run QuickSync and the FFT-based algorithm

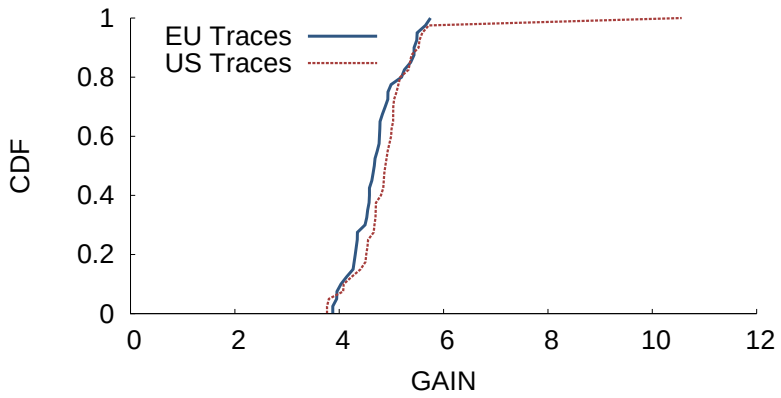


Figure 8-9: **Gain of QuickSync Over the FFT-based Algorithm When the Doppler Shift is Known.** The two curves show the CDFs of the gain in number of multiplications for both of our GPS traces. QuickSync achieves a median gain of 4.8×.

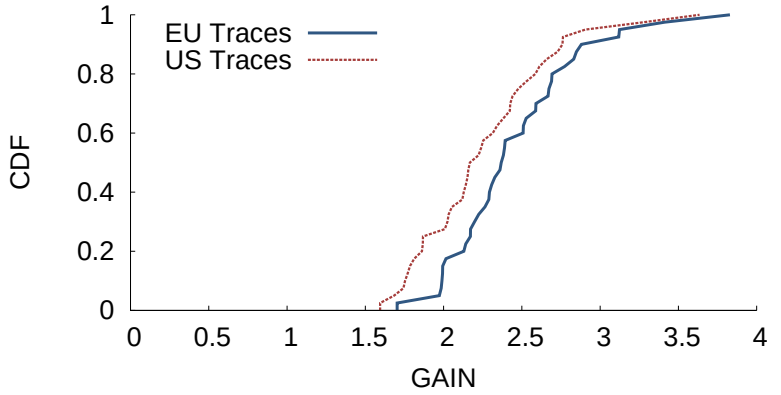


Figure 8-10: **Gain of QuickSync Over the FFT-based Algorithm in FLOPs.** This metric illustrates the gains of QuickSync for a software based implementation. The CDFs show a median gain about 2.2× and a maximum gain of around 3.7×.

on the US and Europe traces and use OProfile to count the number of FLOPs as described in Section 8.6. We run the experiment 1000 times with a different subset samples of the traces and calculate the gain as the ratio of the number of FLOPs required by the FFT-based algorithm to the number of FLOPs required by QuickSync. We do not assume in this experiment that the Doppler shift is known and we let both algorithms search for the right Doppler shift. Figure 8-10 shows a CDF of the gains. QuickSync achieves a median gain of 2× and 2.3× over the FFT-based algorithm for the US and Europe traces respectively. This shows that QuickSync can reduce the number of CPU computation on average by half in software based GPS receivers.

8.8 Related Work

FFT-based GPS synchronization was first proposed by Nee et al. [176] who showed that it reduces synchronization complexity from $O(n^2)$ to $O(nlog(n))$, where n is the number of samples per C/A code. QuickSync builds on this technique and and leverages the Sparse Fourier Transform to further reduce the synchronization complexity.

Our approach is related to past work on GPS block-averaging [128, 167], which sums up consecutive signal blocks before performing the FFT. QuickSync however differs from that work along two axes: First, on the algorithmic front, past work performs a full size FFT of n points. One cannot simply replace this n-point FFT with the Sparse Inverse Fourier Transform because, as explained in Section 8.1, the output of the FFT is not sparse. In contrast, QuickSync introduces a design that can harness the Sparse Fourier Transform. This enables QuickSync to operate with a smaller FFT of size n/p, which provides faster synchronization. Second, past work on block-averaging focuses on weak GPS signals and does not provide an adaptive design that works for the whole range of GPS SNRs. Applying their approach to the whole SNR range can incur unnecessary overhead. This is because they average and pre-process many blocks independent of the SNR. As a result, these schemes increase the synchronization delay for scenarios in which only one (or a few) blocks are sufficient for detecting the spike. In contrast, our algorithm adaptively processes more data when the spike is too low for detection, and hence gracefully scales with the SNR of the received GPS signal.

Signal synchronization is a general problem that applies to other wireless technologies, e.g., WiFi and cellular. However, synchronization in these technologies is simpler because the noise level in the received signals is much lower than in GPS. For example, WiFi receivers can lock on the signal simply by detecting an increase in the received power [76]. This is not possible in GPS since the signal is received at 20 dB *below* the noise floor [144]. Cellular systems also operate at much higher SNRs than GPS, which allows them to synchronize with relatively low overhead [97].

QuickSync is also related to the general class of work on reducing GPS power consumption. The most common approach uses assisted-GPS [92, 150], which involves connecting to an assisted GPS server through a WiFi or cellular network. The server provides the GPS receiver with the GPS data decoded from each satellite signal, which allows the receiver to avoid decoding the GPS signal. The device can also offload GPS computations to the cloud after acquiring the GPS signal [50, 113, 136]. The latter approach, however, reduces the complexity of the device but still requires the device to transmit the GPS signal to the cellular tower (thus consuming transmission power and even bandwidth [50]). Other approaches for reducing GPS power consumption leverage WiFi, sensors, or the cellular signal to localize the receiver [24, 134, 170]. These schemes typically are less accurate than GPS and are more constrained in terms of where they can be deployed. Our work contributes to this effort by tackling the complexity of the GPS receiver itself.

8.9 Conclusion

This chapter presents the fastest synchronization algorithm for GPS receivers to date. The gains of the algorithm are also empirically demonstrated for software GPS implementations as well

as potential hardware architectures. Because synchronization consumes a significant amount of power, we expect the reduced complexity to decrease GPS power consumption. Further, we believe that the sparse synchronization algorithm we introduced has other applications in signal processing and pattern matching. We plan to explore those applications in future work.

Chapter 9

Light Field Reconstruction Using Continuous Fourier Sparsity

9.1 Introduction

Fourier analysis is a critical tool in rendering and computational photography. It tells us how to choose sampling rates, e.g., [42, 43, 75, 126], predict upper bounds on sharpness, e.g., [109, 110, 132], do fast calculations, e.g., [161], model wave optics, e.g., [65, 188], perform light field multiplexing [177], and do compressive sensing, e.g., [26]. In particular, the sparsity of natural spectra, such as those of light fields, makes it possible to reconstruct them from smaller sets of samples, e.g., [106, 177]. This sparsity derives naturally from the continuous Fourier transform, where continuous-valued depth in a scene translates to 2D subspaces in the Fourier domain. However, practical algorithms for reconstruction, including the Sparse Fourier Transform algorithms from Part I of this thesis, usually operate on the Discrete Fourier Transform (DFT). Unfortunately, little attention is usually paid to the impact of going from the continuous Fourier domain to the discrete one, and it is often assumed that the sparsity derived from continuous principles holds for discrete sampling and computation. In this chapter, we make the critical observation that much of the sparsity in continuous spectra is lost in the discrete domain, and that this loss of sparsity can severely limit reconstruction quality. We propose a new approach to reconstruction that recovers a discrete signal by optimizing for sparsity in the continuous domain. We first describe our approach in general terms, then demonstrate its application in the context of 4D light field acquisition and reconstruction, where we show that it enables high-quality reconstruction of dense light fields from fewer samples without requiring extra priors or assumptions such as Lambertian scenes.

The difference between continuous and discrete sparsity is due to the windowing effect. Sampling a signal, such as a light field, inside some finite window is analogous to multiplying that signal by a box function. In the frequency domain, this multiplication becomes convolution by an infinite sinc. If the nonzero frequencies of the spectrum are not perfectly aligned with the resulting discretization of the frequency domain (and therefore the zero crossings of the sinc), this convolution destroys much of the sparsity that existed in the continuous domain. This effect is shown in Figure 9-1(a) which plots a 2D angular slice of the 4D light field spectrum of the Stanford crystal

(a) Sparsity in the discrete vs continuous spectrum

(b) Sampled viewpoint images

(c) Reconstruction of unsampled viewpoint image: blue in (b)

(d) Reconstruction of unsampled viewpoint image: green in (b)

Figure 9-1: **Sparsity in the Discrete vs. Continuous Fourier Domain, and Our Reconstruction Results:** (a) The discrete Fourier transform (top) of a particular 2D angular slice ω_u, ω_v of the crystal ball's light field, and its reconstructed continuous version (bottom). (b) A grid showing the original images from the Stanford light field archive. The images used by our algorithm are highlighted (courtesy of [163]); (c) and (d) Two examples of reconstructed viewpoints showing successful reconstruction of this highly non-Lambertian scene which exhibits caustics, specularities, and nonlinear parallax. The uv locations of (c) and (d) are shown as blue and green boxes in (b).

ball. In practice, natural spectra, including those of light fields, are never so conveniently aligned, and this loss of sparsity is always observed.

We introduce an approach to recover the sparsity of the original continuous spectrum based on nonlinear gradient descent. Starting with some initial approximation of the spectrum, we optimize for sparsity in the continuous frequency domain through careful modeling of the projection of continuous sparse spectra into the discrete domain. The output of this process is an approximation of the continuous spectrum. In the case of a light field, this approximation can be used to reconstruct high quality views that were never captured and even extrapolate to new images outside the aperture of recorded samples.

Our approach effectively reduces the sampling requirements of 4D light fields by recovering the sparsity of the original continuous spectrum. We show that it enables the reconstruction of full 4D light fields from only a 1D trajectory of viewpoints, which could greatly simplify light field capture. We demonstrate a prototype of our algorithm on multiple datasets to show that it is able to accurately reconstruct even highly non-Lambertian scenes. Figures 9-1(b), 9-1(c), and 9-1(d) show our reconstruction of a highly non-Lambertian scene and the 1D trajectory of viewpoints used by our implementation.

We believe that our observations on continuous versus discrete sparsity and careful handling of sampling effects when going from a sparse continuous Fourier transform into the discrete Fourier domain can also have important applications for computational photography beyond light field reconstruction as well as other areas like medical imaging as we will see in Chapter 10.

9.2 Related work

Light field capture is challenging because of the 4D nature of light fields and the high sampling rate they require. Capture can be done with a micro lens array at the cost of spatial resolution, e.g. [4, 55, 133], using robotic gantries [111], using camera arrays [179], or with a handheld camera moved over time around the scene [23, 37, 66]. All these solutions require extra hardware or time, which has motivated the development of techniques that can reconstruct dense light fields from fewer samples.

[106, 108] argue that the fundamental differences between reconstruction strategies can be seen as a difference in prior assumptions made about the light field. Such priors usually assume a particular structure of sparsity in the frequency domain.

Perhaps the most common prior on light fields assumes that a captured scene is made up of Lambertian objects at known depths. Conditioning on depth, the energy corresponding to a Lambertian surface is restricted to a plane in the frequency domain. Intuitively, this means that given a single image and its corresponding depth map we could reconstruct all 4 dimensions of the light field (as is done in many image based rendering techniques). The problems with this approach are that the Lambertian assumption does not always hold and that depth estimation usually involves fragile nonlinear inference that depends on angular information, meaning that sampling requirements are not reduced to 2D in practice. However, paired with a coded aperture [107, 177] or plenoptic camera [18], this prior can be used to recover superresolution for Lambertian scenes in the spatial or angular domain.

Levin and Durand [106] use a Lambertian prior, but do not assume that depth is known. This corresponds to a prior that puts energy in a 3D subspace of the light field spectrum, and reduces reconstruction to a linear inference problem. As a result they require only 3 dimensions of sampling, typically in the form of a focal stack. Like our example application, their technique can also reconstruct a light field from a 1D set of viewpoints. However, they still rely on the Lambertian assumption and the views they reconstruct are limited to the aperture of input views. In contrast, we show how our approach can be used to synthesize higher quality views both inside and outside the convex hull of input images without making the Lambertian assumption. For a comparison, see Section 9.7.

The work of [118] assumes a different kind of structure to the sparsity of light fields. This structure is learned from training data. Specifically, they use sparse coding techniques to learn a dictionary of basis vectors for representing light fields. The dictionary is chosen so that training light fields may be represented as sparse vectors. Their underlying assumption is that new light fields will have similar structure to those in their training data.

9.3 Sparsity in the Discrete vs. Continuous Fourier Domain

In this section, we show how the discretization of a signal that is sparse in the continuous Fourier domain results in a loss of sparsity. We then give an overview of our approach for recovering sparse continuous spectra. In subsequent sections, we will describe in detail one application of this theory, namely, reconstructing full 4 dimensional light fields from a few 1D viewpoint segments. We will

also show results of this application on real light field data.

A signal $x(t)$ of length N is k-sparse in the continuous Fourier domain if it can be represented as a combination of k non-zero continuous frequency coefficients:

$$x(t) = \frac{1}{N} \sum_{i=0}^{k} a_i \exp\left(\frac{2\pi j t \omega_i}{N}\right) \tag{9.1}$$

where $\{\omega_i\}_{i=0}^{k}$ are the continuous positions of frequencies (i.e., each ω_i is not necessarily an integer), and $\{a_i\}_{i=0}^{k}$ are the coefficients or values corresponding to these frequencies. The same signal is sparse in the discrete Fourier domain only if all of the ω_i's happen to be integers. In this case, the output of its N-point DFT has only k non-zero coefficients. Consequently, any signal that is k-sparse in the discrete Fourier domain is also k-sparse in the continuous Fourier domain; however, as we will show next, a signal that is sparse in the continuous Fourier domain is not necessarily sparse in the discrete Fourier domain.

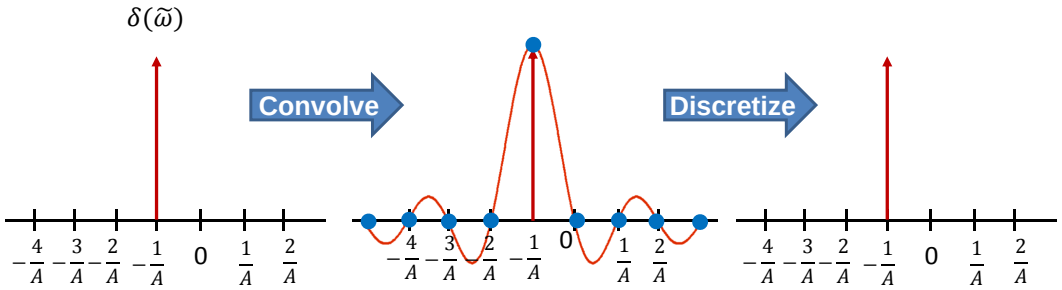
9.3.1 The Windowing Effect

The windowing effect is a general phenomenon that occurs when one computes the discrete Fourier transform (DFT) of a signal using a finite window of samples. Since it is not limited to the light field, we will explain the concept using one-dimensional signals. It naturally extends to higher dimensions.

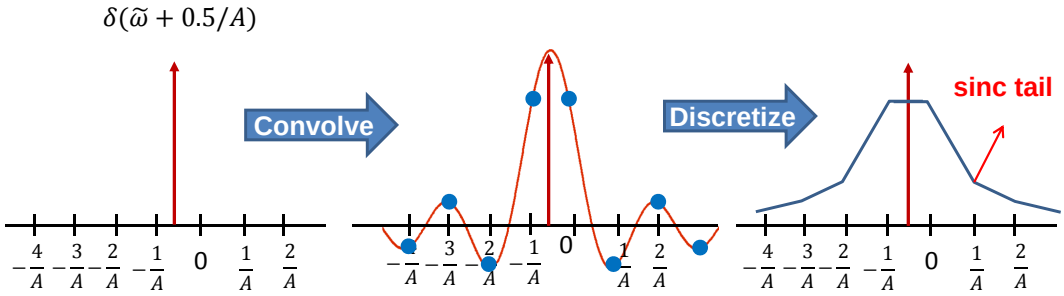
Consider computing the discrete Fourier transform of a time signal $\mathbf{y}(t)$. To do so, we would sample the signal over a time window $[-\frac{A}{2}, \frac{A}{2}]$, then compute the DFT of the samples. Since the samples come from a limited window, it is as if we multiplied the original signal $\mathbf{y}(t)$ by a box function that is zero everywhere outside of this acquisition window. Multiplication in the time domain translates into convolution in the Fourier domain. Since acquisition multiplies the signal by a box, the resulting DFT returns the spectrum of the original signal $\mathbf{y}(t)$ convolved with a sinc function.

Convolution with a sinc, in most cases, significantly reduces the sparsity of the original signal. To see how, consider a simple example where the signal $\mathbf{y}(t)$ is one sinusoid, i.e., $\mathbf{y}(t) = \exp(-2j\pi\tilde{\omega}t)$. The frequency domain of this signal has a single impulse at $\tilde{\omega}$. Say we sample the signal over a window $[-\frac{A}{2}, \frac{A}{2}]$, and take its DFT. The spectrum will be convolved with a sinc, as explained above. The DFT will discretize this spectrum to the DFT grid points located at integer multiples of $\frac{1}{A}$. Because a sinc function of width $\frac{1}{A}$ has zero crossings at multiples of $\frac{1}{A}$, (as can be seen in Figure 9-2(a)), if $\tilde{\omega}$ is an integer multiple of $\frac{1}{A}$ then the grid points of the DFT will lie on the zeros of the sinc$(\cdot)$ function and we will get a single spike in the output of the DFT. However, if $\tilde{\omega}$ is a not an integer multiple of $\frac{1}{A}$, then the output of the DFT will have a sinc tail as shown in Figure 9-2(b).

Like most natural signals, the sparsity of natural light fields is not generally aligned with any sampling grid. Thus, the windowing effect is almost always observed in the DFT of light fields along spatial and angular dimensions. Consider the effect of windowing in the angular domain (which tends to be more limited in the number of samples, and consequently exhibits a stronger windowing effect). Light fields are sampled within a limited 2D window of $\mathbf{uv}$ coordinates. As



(a) Frequency lies on a DFT grid point



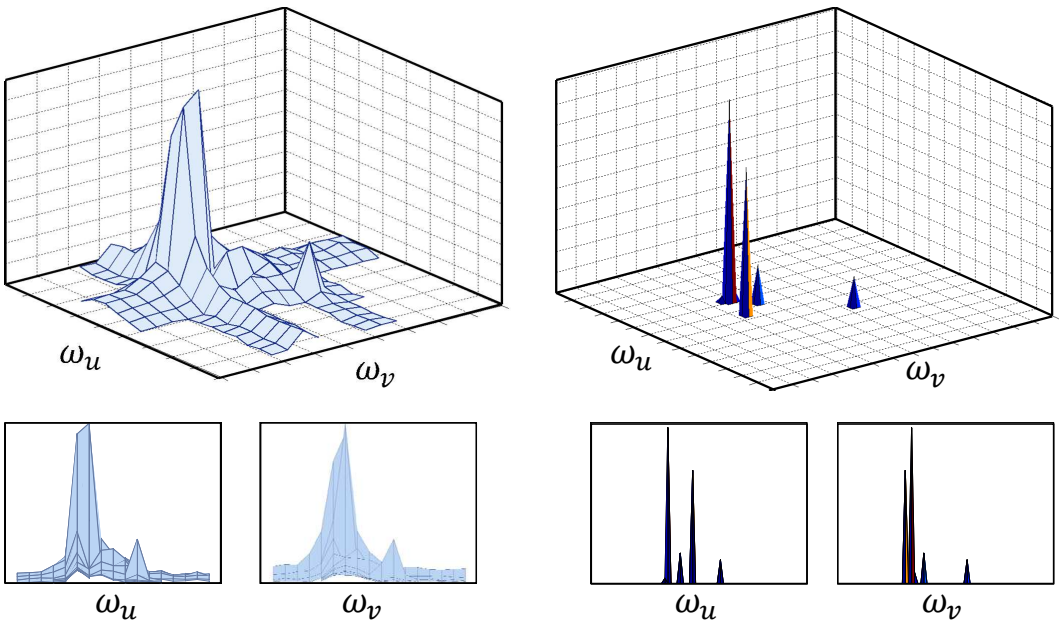
(b) Frequency does not lie on a DFT grid point

Figure 9-2: **The Windowing Effect:** Limiting samples to an aperture A is equivalent to convolving the spectrum with a sinc function. (a) If a frequency spike lies on a DFT grid point then the sinc disappears when it is discretized, and the original sparsity of the spectrum is preserved. (b) If the frequency spike is not on the DFT grid, once we discretize we get a sinc tail and the spectrum is no longer as sparse as in the continuous domain.

a result, the DFT of each 2D angular slice, $\hat{\mathbf{L}}_{\omega_x,\omega_y}(\omega_u, \omega_v)$, is convolved with a 2D sinc function, reducing sparsity. Figure 9-3(a) shows the DFT of an angular slice from the crystal ball light field. As can be seen in the figure, the slice shows a sparse number of peaks but these peaks exhibit tails that decay very slowly. These tails ruin sparsity and make reconstruction more difficult. We propose an approach to light field reconstruction that removes the windowing effect by optimizing for sparsity in the continuous spectrum. Figure 9-3(b) shows a continuous Fourier transform of the same crystal ball slice, recovered using our approach. Note that the peak tails caused by windowing have been removed and the underlying sparsity of light fields has been recovered.

9.3.2 Recovering the Sparse Continuous Fourier Spectrum

From sparse recovery theory we know that signals with sparse Fourier spectra can be reconstructed from a number of time samples proportional to sparsity in the Fourier domain. Most practical sparse recovery algorithms work in the discrete Fourier domain. However, as described above, the non-zero frequency coefficients of most light fields are not integers. As a result, the windowing effect ruins sparsity in the discrete Fourier domain and can cause existing sparse recovery algorithms to



(a) Discrete Spectrum **(b) Continuous Spectrum**

Figure 9-3: **Light Field Spectrum in the Discrete and Continuous Fourier Domains:** A 2D angular slice of the 4D light field spectrum of the Stanford crystal ball for $(\omega_x, \omega_y) = (50, 50)$. (a) In the discrete Fourier domain, we have sinc tails and the spectrum is not very sparse. (b) In the continuous Fourier domain, as reconstructed by our algorithm, the spectrum is much sparser. It is formed of 4 peaks which do not fall on the grid points of the DFT.

fail. Our approach is based on the same principle of sparse recovery, but operates in the continuous Fourier domain where the sparsity of light fields is preserved.

Recall our model in Equation 9.1 of a signal that is sparse in the continuous Fourier domain. Given a set of discrete time samples of $x(t)$, our goal is to recover the unknown positions $\{\omega_i\}_{i=0}^{k}$ and values $\{a_i\}_{i=0}^{k}$ of the non-zero frequency coefficients. From Equation 9.1, we see that this problem is linear in the values $\{a_i\}_{i=0}^{k}$ and non-linear in the positions $\{\omega_i\}_{i=0}^{k}$ of the non-zero coefficients. Thus, to recover the values and positions, we use a combination of a linear and non-linear solver.

Recovering coefficient values $\{a_i\}_{i=0}^{k}$: If we know the positions of non-zero coefficients (i.e. each ω_i) then Equation 9.1 becomes a system of linear equations with unknowns $\{a_i\}_{i=0}^{k}$, and given $> k$ discrete samples of $x(t)$, we can form an over-determined system allowing us to solve for each a_i.

Recovering continuous positions $\{\omega_i\}_{i=0}^{k}$: We use nonlinear gradient descent to find the continuous positions $\{\omega_i\}_{i=0}^{k}$ that minimize the square error between observed discrete samples of $x(t)$ and the reconstruction of these samples given by our current coefficient positions and values. Thus,

the error function we wish to minimize can be written as

$$e = \sum_t \left\| x(t) - \frac{1}{N} \sum_{i=0}^{k} \tilde{a}_i \exp\left(\frac{2\pi j t \tilde{\omega}_i}{N}\right) \right\|^2 \tag{9.2}$$

where $\tilde{a}_i$ and $\tilde{\omega}_i$ are our estimates of a_i and ω_i and the above summation is taken over all the observed discrete samples.

As with any gradient descent algorithm, in practice, we begin with some initial guess of discrete integer positions $\{\omega_i^{(0)}\}_{i=0}^k$. We use the discrete sparse Fourier transform algorithm described in Section 9.5.2 for our initialization. From this initial guess, we use gradient descent on $\{\omega_i\}_{i=0}^k$ to minimize our error function. In practice, the gradient is approximated using finite differences. In other words, we calculate error for perturbed peak locations $\{\omega_i^{(j)} + \delta_i\}$ and update our $\{\omega_i^{(j+1)}\}$ with the ϵ_i that result in the smallest error. We keep updating until the error converges.

Once we have recovered both a_i and ω_i, we can reconstruct the signal $x(t)$ for any sample t using Equation 9.1.

9.4 Light Field Notation

A 4D light field $\mathbf{L}(x, y, u, v)$ characterizes the light rays between two parallel planes: the $\mathbf{uv}$ camera plane and the $\mathbf{xy}$ image plane, which we refer to as angular and spatial dimensions respectively. Each (u, v) coordinate corresponds to the location of the viewpoint of a camera and each (x, y) coordinate corresponds to a pixel location. $\widehat{\mathbf{L}}(\omega_x, \omega_y, \omega_u, \omega_v)$ characterizes the 4D spectrum of this light field. We will use $\widehat{\mathbf{L}}_{\omega_x,\omega_y}(\omega_u, \omega_v)$ to denote a 2D angular slice of this 4D spectrum for fixed spatial frequencies (ω_x, ω_y). Similarly, $\mathbf{L}_{u,v}(x, y)$ denotes the 2D image captured by a camera with its center of projection at location (u, v). Table 9.1 presents a list of terms used throughout this chapter.

9.5 Light Field Reconstruction Algorithm

To demonstrate the power of sparse recovery in the continuous Fourier domain, we show how it can be used to reconstruct light fields from 1D viewpoint trajectories. We choose to work with 1D trajectories because it simplifies the initialization of our gradient descent. However, the continuous Fourier recovery described in the previous section is general and does not require this assumption.

In this section, we describe the Sparse Fourier Transform used for our initialization as well as our sparse continuous Fourier recovery. The reconstruction algorithm described in this section is shown in Algorithm 9.4.1. The initialization is given by the SPARSEDISCRETERECOVERY procedure shown in Algorithm 9.5.1 and the continuous recovery is given by the SPARSECONTINUOUSRECOVERY procedure shown in Algorithm 9.5.2.

Term	Definition
u, v	Angular/camera plane coordinates
x, y	Spatial plane coordinates
ω_u, ω_v	Angular frequencies
ω_x, ω_y	Spatial frequencies
$\mathbf{L}(x, y, u, v)$	4D light field kernel
$\widehat{\mathbf{L}}(\omega_x, \omega_y, \omega_u, \omega_v)$	4D light spectrum
$\widehat{\mathbf{L}}_{\omega_x,\omega_y}(\omega_u, \omega_v)$	A 2D angular slice of the 4D light spectrum
$\widehat{\mathbf{L}}_{\omega_x,\omega_y}(u, v)$	A 2D slice for fixed spatial frequencies
$\mathbf{X}$	2D slice $= \widehat{\mathbf{L}}_{\omega_x,\omega_y}(u, v)$
S	Set of samples (u, v)
$\mathbf{X}_{\|S}$	2D $\mathbf{X}$ with only samples in S
$\mathbf{x}_S$	$\mathbf{X}_{\|S}$ reordered as $1 \times \|S\|$ vector
P	Set of frequency positions (ω_u, ω_v)
$\widehat{\mathbf{x}}_P$	$1 \times \|P\|$ vector of frequency coefficients
F	Set of positions and coefficients (ω_u, ω_v, a)
$[N]$	The set $\{0, 1, \cdots N-1\}$
$\mathbf{y}$	1D signal or line segment
$M \times M$	Number of image pixels in spatial domain
$N \times N$	Number of camera locations

Table 9.1: **Light Field Notation**

procedure SPARSELIGHTFIELD($\mathbf{L}_{|S}$)
  $\widehat{\mathbf{L}}_{u,v}(\omega_x, \omega_y) = \text{FFT}(\mathbf{L}_{u,v}(x, y))$ for $u, v \in S$
  for $\omega_x, \omega_y \in [M]$ **do**
    $\widehat{\mathbf{L}}_{\omega_x,\omega_y}(\omega_u, \omega_v) = \text{2DSPARSEFFT}(\widehat{\mathbf{L}}_{\omega_x,\omega_y}(u, v)_{|S})$
  $\mathbf{L}(x, y, u, v) = \text{IFFT}(\widehat{\mathbf{L}}(\omega_x, \omega_y, \omega_u, \omega_v))$
  return $\mathbf{L}$

procedure 2DSPARSEFFT($\mathbf{X}_{|S}$)
  $P = \text{SPARSEDISCRETERECOVERY}(\mathbf{X}_{|S})$
  $F, e = \text{SPARSECONTINOUSRECOVERY}(\mathbf{X}_{|S}, P)$
  $\mathbf{X}(u, v) = \sum_F a \cdot \exp\left(2j\pi \frac{u\omega_u + v\omega_v}{N}\right)$ for $u, v \in [N]$
  $\widehat{\mathbf{X}} = \text{FFT}(\mathbf{X})$
  return $\widehat{\mathbf{X}}$

9.4.1: Light Field Reconstruction Algorithm

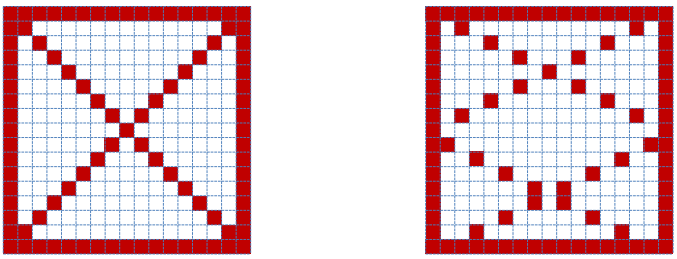


Figure 9-4: **Light Field Sampling Patterns:** Our algorithm samples the (u, v) angular domain along discrete lines. (a) Box and 2 diagonals (b) Box and 2 lines with slopes $= \pm 2$. Note that in this case the discrete line wraps around.

9.5.1 Input

Our input is restricted to 1D viewpoint trajectories that consist of discrete lines. A discrete line in the angular domain is defined by the set of (u, v) points such that:

$$\begin{cases} u &= \alpha_u t + \tau_u \mod N \\ v &= \alpha_v t + \tau_v \mod N \end{cases}, \text{ for } t \in [N]$$
$$\text{where } 0 \leq \alpha_u, \alpha_v, \tau_u, \tau_v < N \text{ and GCD}(\alpha_u, \alpha_v) = 1 \tag{9.3}$$

This lets us use the Fourier projection-slice theorem to recover a sparse discrete spectrum, which we use to initialize our gradient descent. Figure 9-4 shows the specific sampling patterns used in our experiments.

For a light field $\mathbf{L}(x, y, u, v)$, our algorithm operates in the intermediate domain, $\widehat{\mathbf{L}}_{\omega_x,\omega_y}(u, v)$, which describes spatial frequencies as a function of viewpoint. We start by taking the 2D DFT of each input image, which gives us the spatial frequencies (ω_x, ω_y) at a set of viewpoints S consisting of our 1D input lines. We call this set of known samples $\widehat{\mathbf{L}}_{\omega_x,\omega_y}(u, v)_{|S}$. Our task is to recover the 2D angular spectrum $\widehat{\mathbf{L}}_{\omega_x,\omega_y}(\omega_u, \omega_v)$ for each spatial frequency (ω_x, ω_y) from the known samples $\widehat{\mathbf{L}}_{\omega_x,\omega_y}(u, v)_{|S}$. For generality and parallelism we do this at each spatial frequency independently, but one could possibly use a prior on the relationship between different (ω_x, ω_y) to improve our current implementation.

9.5.2 Initialization

The goal of our initialization is to calculate some initial guess for the positions $\{\omega_i^{(0)}\}_{i=0}^{k}$ of our non-zero frequency coefficients. We do this using a Sparse Fourier Transform algorithm that leverages the Fourier projection-slice theorem in a voting scheme similar to a Hough transform. By the projection-slice theorem, taking the DFT of an input discrete line gives us the projection of our light field spectrum onto the line. Each projection gives the sum of several coefficients in our spectrum. Different projections provide us with different sums, and each sum above a given threshold votes for the discrete positions of coefficients that it sums similar to the voting scheme described in Chapter 3. The Sparse Fourier Transform algorithm then selects the discrete positions that receive

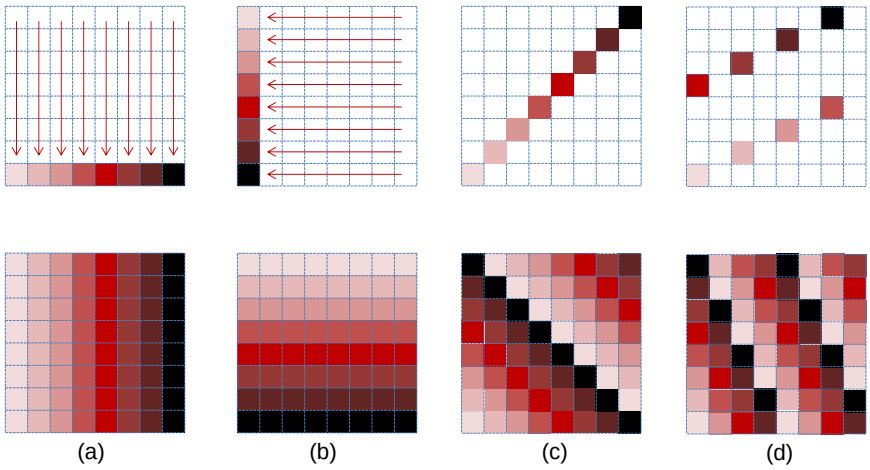


Figure 9-5: **Discrete Fourier Projections:** Computing the DFT of a discrete line of a 2D signal is equivalent to projecting the 2D spectrum onto the line. The top row of figures shows the sampled lines and the bottom row of figures shows how the spectrum is projected. Frequencies of the same color are projected onto the same point. (a) row projection (b) column projection (c) diagonal projection (d) line with slope = 2.

a vote from every input projection and returns these as its initial guess. We refer to this algorithm as DISCRETESPARSERECOVERY and it is shown in Algorithm 9.5.1.

Computing Projections

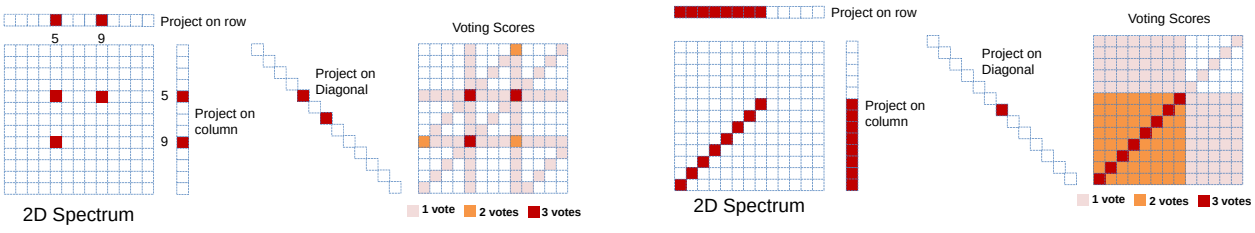
To simplify our discussion of slices, we will use $\mathbf{X}$ to denote the 2D slice $\widehat{\mathbf{L}}_{\omega_x,\omega_y}(u,v)$ in our intermediate domain and $\widehat{\mathbf{X}}$ to denote its DFT, $\widehat{\mathbf{L}}_{\omega_x,\omega_y}(\omega_u,\omega_v)$. Thus, our input is given by a subset of sample slices $\mathbf{X}_{|S}$, where the set S gives the coordinates of our input samples ((u,v) viewpoint positions).

For each slice $\mathbf{X}$ in our input $\mathbf{X}_{|S}$, the views in $\mathbf{X}$ lie on a discrete line. We perform a 1D DFT for each of these discrete lines, which yields the projection of our 2D spectrum onto a corresponding line in the Fourier domain. Specifically, let y be the 1D discrete line corresponding to a 2D slice $\mathbf{X}$, (parameterized by $t \in [N]$)

$$\mathbf{y}(t) = \mathbf{X}(\alpha_u t + \tau_u \bmod N, \alpha_v t + \tau_v \bmod N) \tag{9.4}$$

where $0 \leq \alpha_u, \alpha_v, \tau_u, \tau_v < N$ and GCD$(\alpha_u, \alpha_v) = 1$.

Then, $\widehat{\mathbf{y}}$, the DFT of $\mathbf{y}$, is a projection of $\widehat{\mathbf{X}}$ onto this line. That is, each point in $\widehat{\mathbf{y}}$ is a summation of the N frequencies that lie on a discrete line orthogonal to $\mathbf{y}$, as shown in Figure 9-5. Specifically, the frequencies (ω_u, ω_v) that satisfy $\alpha_u\omega_u + \alpha_v\omega_v = \omega \mod N$ project together onto $\widehat{\mathbf{y}}(\omega)$ (recall that discrete lines may 'wrap around' the input window).



(a) Voting Example 1

(b) Voting Example 2

Figure 9-6: **Voting Based Frequency Estimation:** Two examples of the voting approach used to recover the discrete positions of the large frequencies from projections on discrete lines. The 2D spectrum is projected on a row, a column and a diagonal. Each large projection votes for the frequencies that map to it. Using only projections on a row and column, many frequencies get two votes. By adding a 3rd projection on the diagonal, only the large frequencies get 3 votes. (a) Frequencies (5,5), (5,9), and (9,5) are large and only they get 3 votes. (b) Some frequencies on the diagonal are large and only these frequencies get 3 votes.

Voting

To recover the discrete positions of the nonzero frequency coefficients, we use a voting approach. For each line projection, the projected sums which are above some threshold vote for the frequencies that map to them (similar to a Hough transform). Since the spectrum is sparse, most projected values are very small and only the coefficients of large frequencies receive votes from every line projection. Thus, by selecting frequencies that receive votes from every projection, we get an estimate of the discrete positions of nonzero coefficients.

To better illustrate how voting works, consider the simple example shown in Figure 9-6(a). The 2D spectrum has only 3 large frequencies at $(5,5), (5,9)$ and $(9,5)$. When we project along the rows of our grid the $5th$ and $9th$ entry of the projection will be large and this projection will vote for all frequencies in the $5th$ and $9th$ columns. Similarly, when we project along columns the projection will vote for all frequencies in the $5th$ and $9th$ rows. At this point, frequencies $(5,5), (5,9), (9,5), (9,9)$ have two votes. However, when we project on the diagonal, frequency $(9,9)$ will not get a vote. After 3 projections only the 3 correct frequencies get 3 votes. Another example is shown in Figure 9-6(b).

9.5.3 Optimization in the Continuous Fourier Domain

Recall from Section 9.3.2 that our optimization takes the initial positions $\{\omega_i^{(0)}\}_{i=0}^{k}$ and a subset of discrete samples as input. With both provided by the input and initialization, we now minimize the error function of our reconstruction using the gradient descent approach outlined in Section 9.3.2. This optimization is shown in Algorithm 9.5.2.

procedure SPARSEDISCRETERECOVERY($\mathbf{X}_{|S}$)
$\widehat{\mathbf{y}}_1 = \text{PROJECTLINE}(\mathbf{X}_{|S}, 0, 1, 0, 0)$
$\widehat{\mathbf{y}}_2 = \text{PROJECTLINE}(\mathbf{X}_{|S}, 1, 0, 0, 0)$
$\widehat{\mathbf{y}}_3 = \text{PROJECTLINE}(\mathbf{X}_{|S}, 1, 1, 0, 0)$
$\widehat{\mathbf{y}}_4 = \text{PROJECTLINE}(\mathbf{X}_{|S}, 0, 1, N-1, 0)$
$\widehat{\mathbf{y}}_5 = \text{PROJECTLINE}(\mathbf{X}_{|S}, 1, 0, 0, N-1)$
$\widehat{\mathbf{y}}_6 = \text{PROJECTLINE}(\mathbf{X}_{|S}, 1, -1, 0, N-1)$
$P = \text{RECOVERPOSITIONS}(\widehat{\mathbf{y}}_1, \widehat{\mathbf{y}}_2, \widehat{\mathbf{y}}_3, \widehat{\mathbf{y}}_4, \widehat{\mathbf{y}}_5, \widehat{\mathbf{y}}_6)$
return P

procedure PROJECTLINE($\mathbf{X}_{|S}$, α_u, α_v, τ_u,τ_v)
$\mathbf{y}(i) = \mathbf{X}(i\alpha_u + \tau_u, i\alpha_v + \tau_v)$ for $i \in [N]\}$
$\widehat{\mathbf{y}} = \text{FFT}(\mathbf{y})$
return $\widehat{\mathbf{y}}$

procedure RECOVERPOSITIONS($\widehat{\mathbf{y}}_1, \widehat{\mathbf{y}}_2, \widehat{\mathbf{y}}_3, \widehat{\mathbf{y}}_4, \widehat{\mathbf{y}}_5, \widehat{\mathbf{y}}_6$)
V_1 = VOTE($\widehat{\mathbf{y}}_1, 0, 1, 0, 0, \theta$) ▷ *$\theta$:Power threshold*
V_2 = VOTE($\widehat{\mathbf{y}}_2, 1, 0, 0, 0, \theta$)
V_3 = VOTE($\widehat{\mathbf{y}}_3, 1, 1, 0, 0, \theta$)
V_4 = VOTE($\widehat{\mathbf{y}}_4, 0, 1, N-1, 0, \theta$)
V_5 = VOTE($\widehat{\mathbf{y}}_5, 1, 0, 0, N-1, \theta$)
V_6 = VOTE($\widehat{\mathbf{y}}_6, 1, -1, 0, N-1, \theta$)
$P = V_1 \bigcap V_2 \bigcap V_3 \bigcap V_4 \bigcap V_5 \bigcap V_6$
return P

procedure VOTE($\widehat{\mathbf{y}}, \alpha_u, \alpha_v, \theta$)
$I = \{i : ||\widehat{\mathbf{y}}(i)|| > \theta\}$
$V = \{(\omega_u, \omega_v) : \alpha_u\omega_u + \alpha_v\omega_v = i$ where $i \in I\}$
return V

9.5.1: Sparse Discrete Fourier Recovery Algorithm

Recovering Frequency Coefficients

As we discussed in Section 9.3.2, when we fix the coefficient positions $\{\omega_i^{(0)}\}_{i=0}^k$, Equation 9.1 becomes linear in the coefficient values $\{a_i\}_{i=0}^k$. To solve for the full light field spectrum at each iteration of our gradient descent we express each of our known discrete input samples as a linear combination of the complex exponentials given by our current choice of $\{\omega_i\}_{i=0}^k$.

With the appropriate system of linear equations we can solve for the coefficient values that minimize the error function described in Equation 9.2. To construct our system of linear equations, we concatenate the discrete input (u, v) samples from $\mathbf{X}_{|S}$ into an $|S| \times 1$ vector which we denote as $\mathbf{x}_S$. Given the set P of frequency positions we let $\widehat{\mathbf{x}}_P$ be the $|P| \times 1$ vector of the frequency coefficients we want to recover (with each coefficient in $\widehat{\mathbf{x}}_P$ corresponding to a frequency position in P). Finally, let $\mathbf{A}_P$ be a matrix of $|S| \times |P|$ entries. Each row of $\mathbf{A}_P$ corresponds to a (u, v) sample, each column corresponds to an (ω_u, ω_v) frequency, and the value of each entry is given by a complex exponential:

$$\mathbf{A}_P((u, v), (\omega_u, \omega_v)) = \exp\left(2\pi j \frac{u\omega_u + v\omega_v}{N}\right) \tag{9.5}$$

Now our system of linear equations becomes:

$$\mathbf{x}_S = \mathbf{A}_P \widehat{\mathbf{x}}_P \tag{9.6}$$

We then use the the pseudo-inverse of $\mathbf{A}_P$ to calculate the vector $\widehat{\mathbf{x}}_P^*$ of coefficient values that minimize our error function:

$$\widehat{\mathbf{x}}_P^* = \mathbf{A}_P^\dagger \mathbf{x}_S \tag{9.7}$$

The above is given by the procedure RECOVERCOEFFICIENTS shown in Algorithm 9.5.2.[1]

Gradient Descent

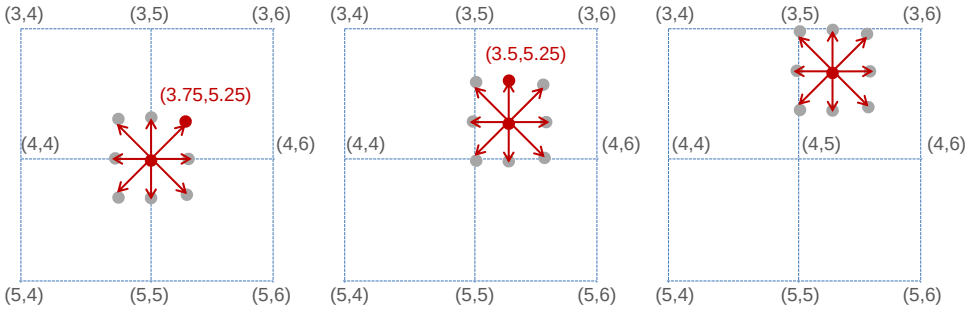
Recall from Section 9.3.2 that our gradient descent algorithm minimizes the error function, which we can now rewrite as:

$$\text{minimize} \;\; e(P) = ||\mathbf{x}_S - \mathbf{A}_P \mathbf{A}_P^\dagger \mathbf{x}_S||^2 \tag{9.8}$$

In the above equation, the frequency positions in the list P are continuous, but the input samples $\mathbf{x}_S$ that we use to compute our error are discrete. Thus, our optimization minimizes error in the *discrete* reconstruction of our light field by finding optimal *continuous* frequency positions.

In our gradient descent, each iteration of the algorithm updates the list of frequency positions P. For each recovered frequency position in P, we fix all other frequencies and shift the position of this frequency by a small fractional step $\delta \ll 1$. We shift it in all 8 directions as shown in Figure 9-7 and compute the new error $e(P)$ given the new position. We then pick the direction that best minimizes the error $e(P)$ and change the position of the frequency in that direction. If none of

[1]Note that we did not specify the threshold used to determine a 'vote' in our initialization. Rather than using a fixed threshold, we choose the smallest threshold such that the system of equations from Equation 9.6 becomes well determined.



(a) Iteration 1 (b) Iteration 2 (c) Iteration 3

Figure 9-7: **Optimizing the Continuous Frequency Positions:** The gradient descent shifts the frequency by a small step in every iteration. The frequency is shifted in the direction that minimizes the error. (a) Frequency (4,5) is shifted to a non-integer position that best reduces the error. (b) The frequency is shifted again to minimize the error. (c) The frequency position converges since shifting in any direction will increase the error.

the directions minimize the error, we do not change the position of this frequency. We repeat this for every frequency position in P.

Our gradient descent ensures that from iteration i to iteration $(i+1)$, we always reduce error, i.e., $e(P^{(i+1)}) < e(P^{(i)})$. The algorithm keeps iterating over the frequencies until the error $e(P)$ falls below a minimum acceptable error ϵ. Once we have a final list of continuous frequency positions, we can recover their coefficients using Equation 9.7. The above gradient descent is given by the procedure GRADIENTSEARCH shown in Algorithm 9.5.2.

9.5.4 Reconstructing the Viewpoints

As explained in Section 9.3.2, once we have the continuous positions and values of our non-zero frequency coefficients, we can reconstruct the missing viewpoints by expressing Equation 9.1 in terms of our data:

$$L_{\omega_x,\omega_y}(u,v) = \sum_{(a,\omega_u,\omega_v)\in F} a \cdot \frac{1}{N} \exp\left(2j\pi \frac{u\omega_u + v\omega_v}{N}\right). \tag{9.9}$$

By setting (u,v) to the missing viewpoints, we are able to reconstruct the full light fields. Figure 9-8 shows a flow chart of the entire reconstruction.

Note that the above equation lets us reconstruct any (u,v) position. We can interpolate between input views and even extend our reconstruction to images that are outside the convex hull of our input. This would not be possible if our sparse coefficients were limited to the discrete Fourier domain, since the above equation would be periodic modulo N. This would create a wrapping effect, and attempting to reconstruct views outside the span of our input would simply replicate views inside the span of our input. In other words, the discrete spectrum assumes that our signal repeats itself outside of the sampling window, but by recovering the continuous spectrum we can relax this assumption and extend our reconstruction to new views.

procedure SPARSECONTINUOUSRECOVERY($\mathbf{X}_{|S}, P$)
    $F^{(0)}, e^{(0)} = \text{RECOVERCOEFFICIENT}(\mathbf{X}_{|S}, P)$
    $i = 0$
    while $e > \epsilon$ **do**
        $F^{(i+1)}, e^{(i+1)} = \text{GRADIENTSEARCH}(\mathbf{X}_{|S}, F^{(i)}, e^{(i)})$
        $i++$
    return $F^{(i)}, e^{(i)}$

procedure GRADIENTSEARCH($\mathbf{X}_{|S}, F, e$)
    $P = \{(\omega_u, \omega_v) : (a, \omega_u, \omega_v) \in F\}$
    for $(\omega_u, \omega_v) \in P$ **do**
        $(\Delta u, \Delta v) = \text{GETGRADIENT}(\mathbf{X}_{|S}, P, e, \omega_u, \omega_v)$
        $(\omega_u, \omega_v) = (\omega_u, \omega_v) + (\delta\Delta u, \delta\Delta v)$
    $F', e' = \text{RECOVERCOEFFICIENT}(\mathbf{X}_{|S}, P)$
    return F', e'

procedure GETGRADIENT($\mathbf{X}_{|S}, P, e, \omega_u, \omega_v$)
    Δ = {(-1,-1), (-1,0), (-1,1), (0,-1), (0,1), (1, -1), (1, 0), (1,1)}
    for $(du, dv) \in \Delta$ **do**
        $P' = P - \{(\omega_u, \omega_v)\}$
        $P' = P \bigcup \{(\omega_u + \delta du, \omega_v + \delta dv)\}$
        $F, e' = \text{RECOVERCOEFFICIENT}(\mathbf{X}_{|S}, P')$
        $\mathrm{d}e_{du,dv} = (e - e')/||(du, dv)||$
    $(du^*, dv^*) = \text{argmax}_{(du,dv)\in\Delta} \mathrm{d}e_{du,dv}$
    return (du^*, dv^*)

procedure RECOVERCOEFFICIENT($\mathbf{X}_{|S}, P$)
    $\mathbf{A} = 0_{|S|\times|P|}$
    $\mathbf{x}_S = 0_{|S|\times 1}$
    for $i \in \{0, \cdots, |S| - 1\}$ **do**
        $(u, v) = S_i$
        $\mathbf{x}_S(i) = \mathbf{X}(u, v)$
        for $k \in \{0, \cdots, |P| - 1\}$ **do**
            $(\omega_u, \omega_v) = P_k$
            $\mathbf{A}(i, k) = \exp\left(2j\pi \frac{u\omega_u + v\omega_v}{N}\right)$
    $\widehat{\mathbf{x}}_P = \mathbf{A}^\dagger \mathbf{x}_S$ ▷ $\mathbf{A}^\dagger$ *is the pseudo-inverse of* $\mathbf{A}$
    $e = ||\mathbf{x}_S - \mathbf{A}\widehat{\mathbf{x}}_P||^2$
    $F = \{(a, \omega_u, \omega_v) \ : \ a = \widehat{\mathbf{x}}_P(k),\ (\omega_u, \omega_v) = P_k\}_{k=0}^{|P|}$
    return F, e

9.5.2: Sparse Continuous Fourier Recovery Algorithm

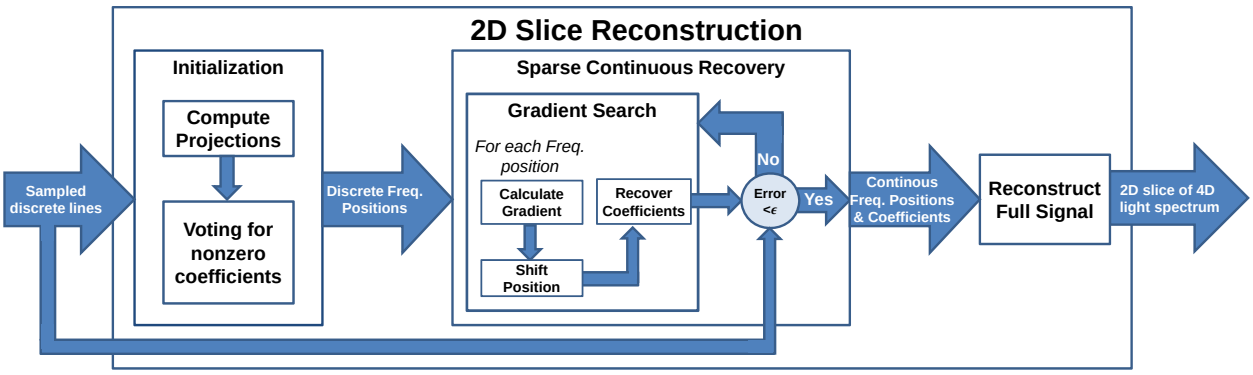


Figure 9-8: **Flow Chart of the 2D Sparse Light Field Reconstruction Algorithm:** The algorithm takes a set of sampled discrete lines. The initialization, Discrete Recovery, has 2 steps: computing the projections and recovering the discrete positions of the large frequency coefficients. In the sparse continuous recovery, the gradient search tries to shift the positions of the frequencies to non-integer locations and recover their coefficients. We keep repeating this gradient search until we get a small enough error. This stage will output a list of continuous frequency positions and their coefficients, which can then be used to reconstruct the full 2D slice.

9.6 Experiments

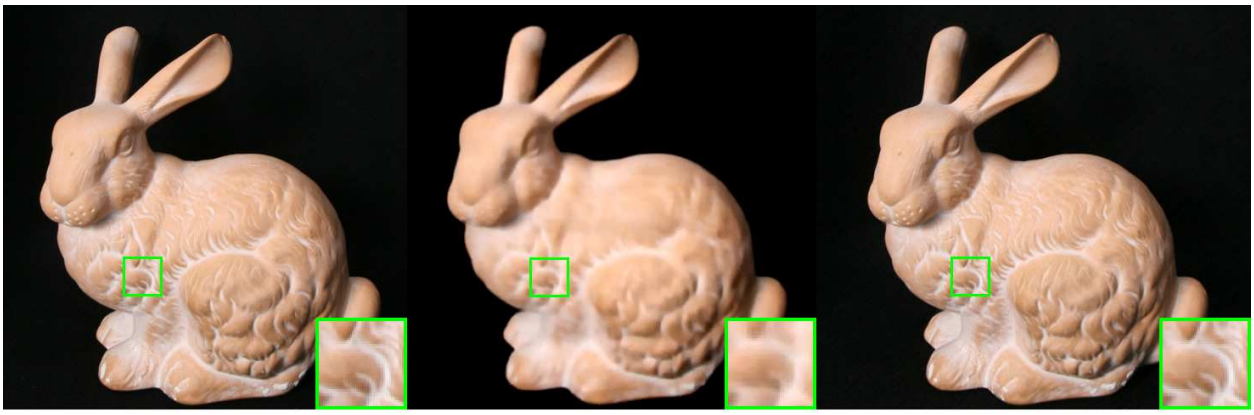
We experimented with several data sets where full 4D coverage of the light field was available for comparison. For each of these data sets we extracted a small number of 1D segments, which we then used to reconstruct the full 2D set of viewpoints. We compare our reconstructed light fields against the complete original data sets in our accompanying videos.

Three of our data sets, the Stanford Bunny, the Amethyst, and the Crystal Ball data sets, were taken from the Stanford light field archive [163]. Each of the Stanford data sets consists of a 17x17 grid of viewpoints and was reconstructed using the box-and-X pattern shown in Figure 9-4(a). On these datasets, we performed our reconstruction in the YUV color space. The U and V channels were reconstructed at half the resolution of the Y channel.

To show how our method scales with the number of input images we are given, we captured a larger dataset (the Gnome) consisting of 51x51 viewpoints. This dataset was captured using a robotic gantry similar to the one from [163], and the double X pattern in Figure 9-4(b) was used to select our input. The total number of input images is the same for both the single X pattern and the double X pattern, as the effective spacing between input views along diagonals is changed. For this dataset our input consists of less than 12% of the original images.

Our code was designed for flexible experimentation and is currently slow. However, the algorithm is highly parallelizable and we run it on a PC cluster. The code is written in C++ using the Eigen library. The ω_u, ω_v slices of a light field are divided among different machines, and the results are collected once all of the machines have finished.

Load balancing, variability in the number of machines used, and different convergence characteristics for different inputs make it difficult to estimate exact run times. Using a cluster of up to 14 machines at a time (averaging 5-6 cores each), typical runtimes ranged from 2 to 3 hours



(a)Reference (b)Levin&Durand (c)Our Reconstruction

Figure 9-9: **Reconstruction of Stanford Bunny Data Set:** On the left and in the middle are the reference figure and the reconstruction from [Levin and Durand 2010](courtesy of Levin). The sampling pattern we used is the box-and-X pattern shown in Figure 9-4(a). Though we used more samples than Levin & Durand used, we did a much better job in terms of less blurring, preserving the textures and having less noise.

for a colored data set (3 channels). There are several ways to accelerate the method - for example, one could leverage the coherence across slices or replace finite differences with a method that converges faster, but we leave this for future work.

9.7 Results

9.7.1 Viewing our results

Our results are best experienced by watching the accompanying videos which are available online through this link: http://netmit.csail.mit.edu/LFSparseRecon/.

9.7.2 The Stanford Bunny

The Stanford Bunny dataset is our simplest test case. The scene is Lambertian and therefore especially sparse in the frequency domain. The spacing between input views is also very narrow, so there is little aliasing. Each image is 512x512 pixels. Our reconstruction of the Bunny is difficult to distinguish from the full light field captured by [163], as shown in Figure 9-9. Figure 9-10 shows that the reconstruction error is small.

9.7.3 Amethyst

The amethyst dataset is highly non-Lambertian. It exhibits both specular reflections and refraction. Again it is difficult to distinguish our reconstruction from the full captured light field, as shown in Figure 9-11. We reconstruct most of the reflected details, with the exception of some undersampled

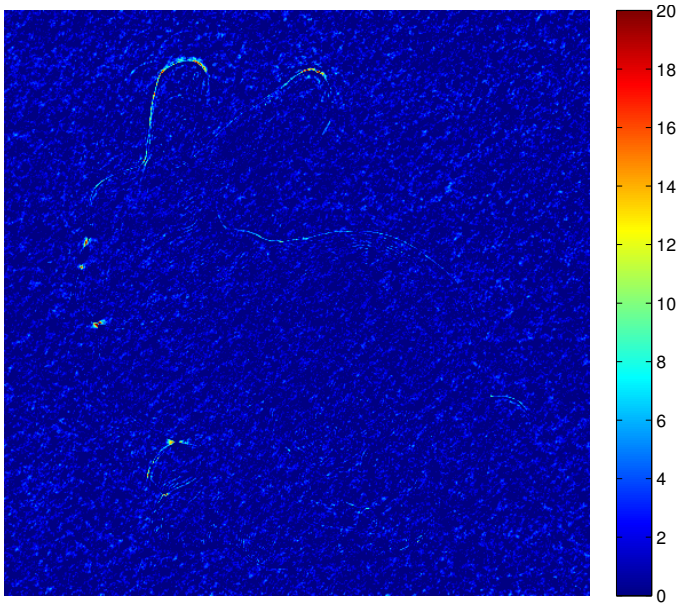


Figure 9-10: **Reconstruction Error:** A color map of the difference in the Y channel between a reference view and a reconstructed view from the Stanford Bunny dataset. In about half of the pixels the difference is zero. There is some unstructured noise, but it is hard to tell whether this comes from the reference figure or our reconstruction. There is also some structured noise on the edge of the bunny, but again it is difficult to tell whether this comes from reconstruction error or an error in the pose estimate of the reference image.

features that move so fast they do not appear at all in the input. Figure 9-12 gives an example of this.

9.7.4 Crystal Ball

The Crystal Ball scene is extremely non-Lambertian, exhibiting caustics, reflections, specularities, and nonlinear parallax. We are able to reproduce most of the complex properties that make this scene shown in Figure 9-13 so challenging, as can be seen in our accompanying video online. If one looks closely, our reconstruction of this light field contains a small amount of structured noise which we discuss in Section 9.8.

9.7.5 Gnome

We acquired a new dataset consisting of 52x52 viewpoints. The resolution of each image is 640x480, and we reconstructed all channels at full resolution.

The Gnome scene is mostly Lambertian with a few specular highlights. In terms of the subject being captured, the difficulty of this scene sits somewhere between the Stanford Bunny and the Amethyst datasets. However, what makes this data more challenging is the level of noise in our input. The captured images of the Gnome have noticeable shot noise, flickering artifacts, and

(a)Reference (b)Levin&Durand (c)Our Reconstruction

Figure 9-11: **Reconstruction of the Amethyst Data Set:** Our reconstruction of Amethyst data set (right), the reference figure (left) and the reconstruction from [106] (middle, courtesy of Levin). We are using the box-and-X sampling pattern shown in Figure 9-4(a), which is more than the number of samples used in [106]. However, we are able to reconstruct this highly non-Lambertian view and it is hard to distinguish our reconstruction from the full captured light field.

registration errors ("camera jitter"). Since these artifacts are not sparse in the frequency domain, our algorithm does not reproduce them in the output shown in Figure 9-14. For most of these artifacts, the result is a kind of denoising, making our output arguably better than the reference images available for comparison. This is especially clear in the case of camera jitter, where the effect of denoising can be seen clearly in an epipolar image shown in Figure 9-16. However, some of the shot noise in our input is reconstructed with greater structure. We have a more general discussion of noise in Section 9.8.

9.7.6 Extending views

Reversing the windowing effect in the second step of our algorithm makes it possible to reconstruct views outside the original window of our input. To demonstrate this we extend each of the u and v dimensions in our Bunny dataset by an additional 4 views, increasing the size of our reconstructed aperture by 53% (see Figure 9-15). These results are best appreciated in our accompanying video.

9.7.7 Informal comparison with Levin and Durand [2010]

Like us, Levin and Durand [106] reconstruct light fields from a 1D set of input images. Their technique is based on a Lambertian prior with unknown depth. We provide an informal comparison with their approach, but the different sampling patterns of the two techniques make it difficult to hold constant the number of input views used by each technique. Levin and Durand's recon-

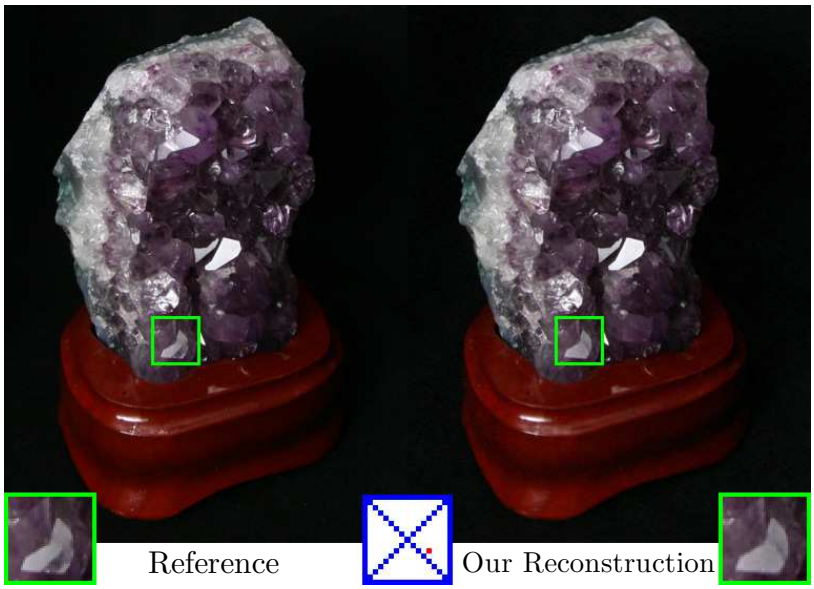


Figure 9-12: **Reconsrution of Specularities:** One example of a reconstructed view from the Amethyst dataset where we lose some reflection details. The missing specular reflection does not appear in any of our input views, so it cannot be recovered.

Figure 9-13: **Reconstruction of the Crystal Ball Data Set:** We show one (u, v) view from the reconstruction of the Crystal Ball dataset. We are using the box plus diagonals sampling pattern (as shown in the blue box in the center). The red dot shows the position of reconstructed view in the angular domain. Despite the fact that the scene is extremely non-Lambertian and has complex structures, we are still able to reconstruct most details of the light field.

struction uses fewer images but is restricted to synthesizing views within the convex hull of input viewpoints. Our sampling patterns use slightly more images, but lets us synthesize views outside

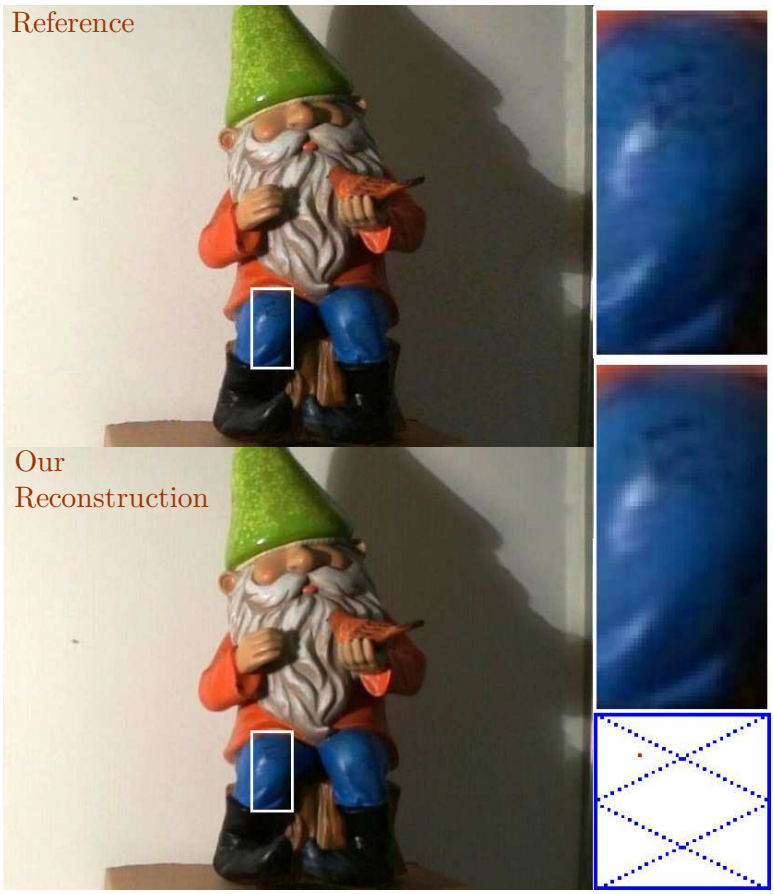


Figure 9-14: **Reconstruction of the Gnome Data Set:** We show one (u, v) view from our reconstruction of the Gnome dataset. We use the sample pattern from Figure 9-4(b), as shown by the blue box in the bottom right. The red marker shows where the view is in the angular domain. Although the captured dataset is noisy, we are still able to reconstruct it in good detail.

the convex hull of our input. Small differences in input aside, the comparison in Figure 9-9 and Figure 9-11 shows that our reconstruction is less blurry and does not have some of the ringing artifacts that appear in their results.

9.8 Discussion

9.8.1 Viewpoint Denoising

One advantage of reconstruction based on a sparse prior is the potential for denoising. Noisy input tends to create low power high frequencies that are not part of our scene. These frequencies make the spectrum less sparse and are usually zeroed out by our algorithm.

Since our reconstruction is based on sparsity in the ω_u, ω_v domain, we remove noise in u, v. This noise corresponds to 'jitter,' usually caused by registration errors or camera shake. We can see

(a) $(u, v) = (-2, -2)$ (b) $(u, v) = (18, 18)$

Figure 9-15: **Extending Views:** We extend our reconstruction of the Stanford Bunny dataset (Figure 9-9) and extend the camera views. The original view is $0 \leq u \leq 16$ and $0 \leq u \leq 16$, and here we show our extension to $(-2, -2)$ and $(18, 18)$.

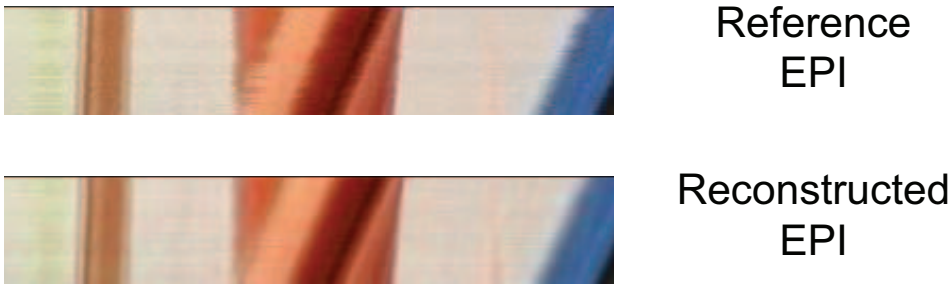


Figure 9-16: **Viewpoint Denoising:** Top: We see noise in the u, v dimensions of our reference data caused by registration errors. This error shows up as camera shake in the reference images. Bottom: Our algorithm effectively removes this noise in the reconstruction, essentially performing camera stabilization.

the effect of this denoising by examining a v, y slice of our light field, like the one in Figure 9-16. These slices are often referred to as epipolar plane images (EPI) in computer vision. To observe the visual effect of this denoising, the reader should watch our accompanying Gnome video. The reconstructed camera motion in this video is much smoother than the reference camera motion. One way to think of this effect is as a kind of video stabilization.

Our ability to denoise in u, v is limited by the number of input slices we have and the sparsity of the spectrum we are reconstructing. If the noise affects the sparsity of our scene too much, some of its power might be projected onto existing spikes from our signal, changing their estimated power. We can see some of this in the Gnome data set, where some of the shot noise in our input is reconstructed with slight structure along the dominant orientation of our scene.

9.8.2 Importance of *Continuous* Fourier Recovery

To better understand how operating in the continuous Fourier domain affects our reconstruction, we examine the impact of our continuous recovery on the reconstructed Bunny light field. We choose this light field because our results are almost indistinguishable from the reference data, so we can reasonably assume that the sparsity estimated by our full algorithm reflects the true sparsity of the captured scene.

We first compare our sparse continuous recovery in Figure 9-17(c) with the sparse discrete recovery used to initialize our gradient descent (shown in Figure 9-17(a)). The error in Figure 9-17(a) shows how existing sparse recovery theory is limited by the the lack of sparsity in the discrete light field spectrum. However, this result does not necessarily isolate the effects of working in the discrete domain. To better isolate these limits, we generate a third reconstruction in Figure 9-17(b) by rounding the coefficients of our final reconstruction Figure 9-17(c) to the nearest discrete frequency positions and removing the sinc tails that result from this rounding. This reconstruction approximates the discrete spectrum that is closest to our continuous spectrum while exhibiting the same sparsity.

As we see in Figure 9-17, the effect of discretization predicted by our experiment is a kind of ghosting. To understand why, recall that the discrete Fourier transform assumes that signals are periodic in the primal domain, and that given a finite number of frequencies our reconstruction will be band limited. As a result, the IDFT will attempt to smooth between images at opposite ends of the primal domain. If we look at Figure 9-18 we can see this effect across the set of viewpoints in our light field. When viewpoints near the center are averaged (smoothed) with their neighbors the artifact is less noticeable because their neighbors are very similar. However, when this smoothing wraps around the edges of our aperture, we average between more dissimilar images and the ghosting becomes more severe.

9.8.3 Potential Applications

Our reconstruction from 1D viewpoint trajectories is directly applicable to capture techniques that seek to acquire a dense 2D sampling of viewpoints on a grid. One could, for instance, use it to significantly reduce the number of cameras needed by a camera array. Alternatively, for applications where light fields are captured by a single moving camera (such as [23, 37, 66]), the algorithm could be used to greatly increase the speed of capture. In both of these cases, the sparse continuous spectrum we recover could also be used as a highly compressed representation of the light field.

The theory of continuous recovery has many potential applications beyond our reconstruction from 1D viewpoint segments. Sparsity in the continuous Fourier domain is a powerful prior that is more general than Lambertianality, making it an exciting new direction for research. While our choice of initialization uses viewpoint sampling patterns that consist of discrete lines, one can imagine different initialization strategies that work with different input. That input could come from plenoptic or mask-based light field cameras, or even some combination of multiview stereo and image based rendering algorithms. However, continuous recovery is not necessarily convex, so proper initialization strategies will be an important and possibly non-trivial part of applying our continuous recovery approach to different types of data.

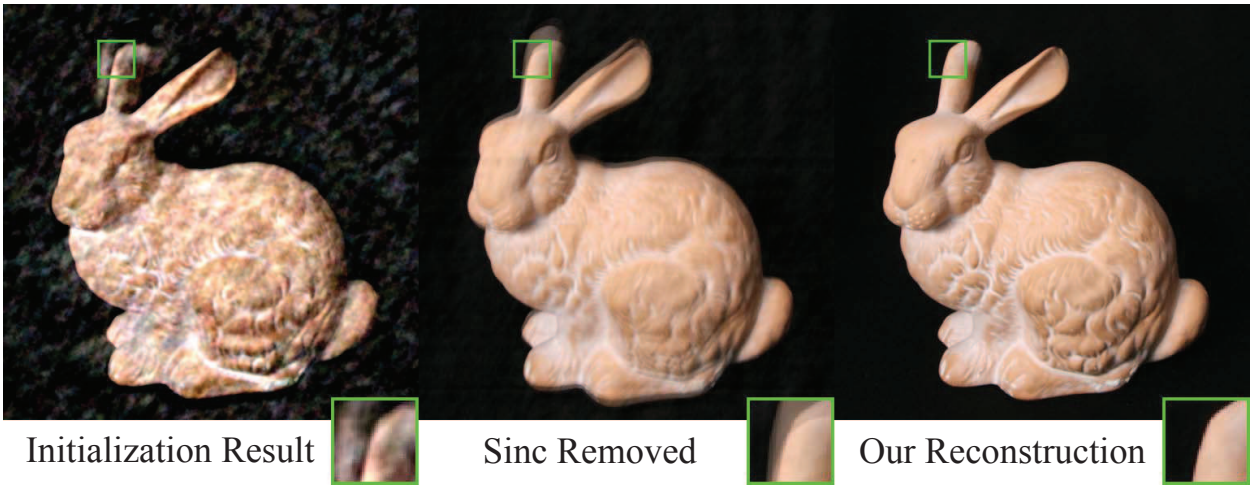


Figure 9-17: **Comparison of Our Final Reconstruction in the Continuous Domain to Two Alternative Reconstructions in the Discrete Domain**: We compare our result (right) with the output of only our initialization (left), as well as the discrete approximation of our result with sinc tails removed (middle).

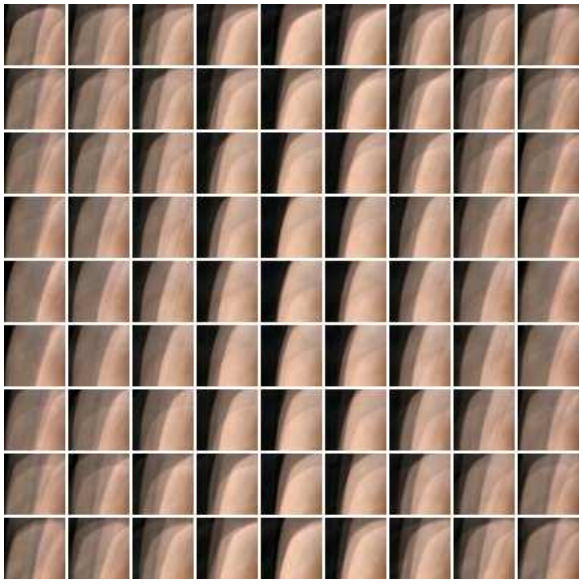


Figure 9-18: **The Ghosting Effect:** A demonstration of the ghosting that happens when we simply remove sinc tails in the frequency domain. We removed the sinc tails from the spectrum of the Stanford Bunny dataset and selected the same inset from each u, v image (we chose the same inset as in Figure 9-17). This figure shows how the inset changes across the (u, v) aperture (note that we subsampled the 17x17 aperture by 2). Ghosting gets worse closer to the edge of the input views.

9.9 Conclusion

In this chapter, we have made the important observation that natural signals like light fields are much sparser in the continuous Fourier domain than in the discrete Fourier domain, and we have shown how this difference in sparsity is the result of a windowing effect. Based on our observations, we presented an approach to light field reconstruction that optimizes for sparsity in the continuous Fourier domain. We then showed how to use this approach to reduce sampling requirements and improve reconstruction quality by applying it to the task of recovering high quality non-Lambertian light fields from a small number of 1D viewpoint trajectories. We believe that our strategy of optimizing for sparsity in the discrete spectrum will lead to exciting new research in light field capture and reconstruction. Furthermore, we hope that our observations on sparsity in the discrete vs. continuous domain will have an impact on areas of computational photography beyond light field reconstruction.

Chapter 10

Fast In-Vivo MRS Acquisition with Artifact Suppression

10.1 Introduction

Magnetic Resonance Spectroscopy (MRS) enables non-invasive analysis of the biomolecular content of the brain. It is an advanced MRI technique that allows us to zoom into specific voxels in the brain and discover the concentration of various metabolites which are used as indicators of diseases like cancer, strokes, seizures, Alzheimer's disease, autism, etc. These metabolites create MR signals that appear as frequency peaks in the MRS spectrum. 2D Correlation Spectroscopy (COSY) is one type of MRS that allows the unambiguous assignment of MR signals in crowded spectra and as such can be used in vivo to detect and disentangle the spectral overlap of metabolites. Several methods were demonstrated to acquire localized COSY spectra in vivo [10, 171] and showed great utility in detecting new molecular biomarkers of diseases [9, 46].

Despite its potential, the in vivo use of 2D COSY is not largely spread due to several challenges and limitations. The two main challenges are long acquisition time which requires the patients to spend a long time in the MRI machine and the presence of artifacts in the output images. These are a direct result of the acquisition of the additional frequency dimension f_1, which requires sufficient evolution time t_1 that significantly prolongs the acquisition time. Typically, because of time limitation with in vivo acquisitions, the indirect t_1 time dimension cannot be sampled long enough and the corresponding f_1 spectral dimension suffers from severe truncation artifacts resulting from the ringing tails of diagonal peaks.[1] Figure 10-1 shows an example of an in vivo 2D COSY spectrum where the strong diagonal peaks of NAA, Choline and Creatine have large ringing tails that obscure cross-diagonal peaks of important metabolites. Signal processing methods such as the use of filtering windows (QSINE) or prediction algorithms (linear prediction) may improve the spectral appearance but significant artifacts are left and the signal to noise ratio (SNR) may be downgraded. Hence, there is a need for a method that recovers the information along f_1 while preserving SNR.

[1] The t_2 time dimension corresponds to the directly measured dimension and hence can be sampled long enough to avoid truncation artifacts along the f_2 frequency dimension. We refer the reader to [137] for additional background on MR.

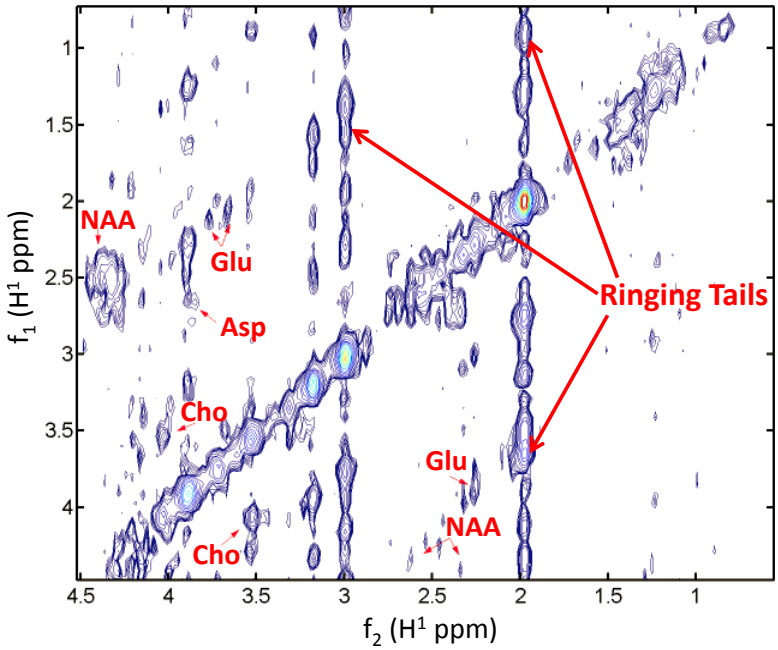


Figure 10-1: **Artifacts in In-Vivo 2D COSY Spectrum:** Truncating the signal in time results ringing tails along the f_1 dimension which create artifacts. There artifacts are hard to distinguish from cross diagonal peaks of important metabolites labeled in red.

2D MRS spectra are highly sparse and the problem of optimizing in-vivo 2D MRS lends itself naturally to the Sparse Fourier Transform. The Sparse Fourier Transform can subsample the 2D COSY signal along the t_1 dimension and hence reduce the acquisition time. However, the artifacts caused by the ringing tails shown in Figure 10-1 significantly reduce the sparsity of the 2D spectrum and hence a straightforward application of the Sparse Fourier Transform would result in a poor reconstruction of the MRS peaks. As we have shown in Section 9.3.1, ringing tails are a direct result of truncating and discretizing the continuous time signal. Since the frequency peaks of natural signals like MR are not aligned with the Fourier discretization grid, *i.e.*, peaks lie on *off-grid* positions, MR signals that are highly sparse in the continuous domain, lose there sparsity in the discrete domain.

In this chapter, we introduce MRS-SFT, a new MRS processing system that adapts the Sparse Fourier Transform to the continuous domain in order to reduce the acquisition time while outputting clearer images with less clutter and artifacts. It estimates and suppresses the ringing tails by recovering the sparsity in the original continuous spectrum while using the minimum number of input samples along the t_1 dimension.

We demonstrated the performance of MRS-SFT on 2D COSY for single and multi-voxel acquisitions in brain phantoms (mixture of non-living samples) and healthy human volunteers. We compared the results obtained by MRS-SFT and other sparse methods such as compressed sensing (CS) to a fully the sampled FFT and a truncated FFT. We show that MRS-SFT reduces the acquisition time by a factor of three on average. It also eliminates the t_1 truncation artifacts which improves the SNR by 8 – 14 dB and the resolution of cross-peaks by 25 – 40% resulting in much clearer and measurable cross diagonal peaks.

10.2 MRS-SFT

10.2.1 Algorithm

We will explain our algorithm for general 1D signals. In the following subsection, we will explain how to use this algorithm to reconstruct a 2D COSY spectrum. Let $x(t)$ be a signal of length N that is k-sparse in the continuous Fourier domain. Then,

$$x(t) = \frac{1}{\sqrt{N}} \sum_{i=0}^{k} a_i \exp\left(\frac{2\pi j f_i t}{N}\right) \tag{10.1}$$

where $\{f_i\}_{i=0}^{k}$ are the continuous positions of frequencies and $\{a_i\}_{i=0}^{k}$ are the complex values corresponding to these frequencies. The same signal is sparse in the discrete Fourier domain only if all of the f_i's happen to be integers, *i.e.*, the large frequency coefficients lie *on-the-grid*.

However, for most natural signals like MRS, the large frequency coefficients are more likely to lie *off-the-grid*. We have shown in Section 9.3.1 that while these *off-grid* frequencies are sparse in the continuous Fourier domain, they are not sparse in the discrete Fourier domain. Recall that truncating the signal in time is equivalent to multiplying it with a rectangular function which results in a convolution with a sinc function in the frequency domain. The sinc function has ringing tails which create truncation artifacts like the ones shown in Figure 10-1. We refer the reader to Section 9.3.1 for a detailed description of this discretization process and the difference in sparsity between the continuous and discrete Fourier domains.

MRS-SFT adapts the Sparse Fourier Transform to optimize for the sparsity in the continuous Fourier domain. The MRS-SFT algorithm has two phases:

- **On-Grid Recovery:** recovers the spectrum assuming frequencies are only located on integer multiples of the discrete Fourier grid. This step only gives an initial estimate of the spectrum and alone cannot give a good quality reconstruction
- **Off-Grid Recovery:** refines the frequencies discovered in the previous stage, allowing them to take non-integer positions. As such it attempts to approximate the continuous spectrum.

As input, the algorithm takes only subsamples of the signal $\mathbf{x}$ which for MRS translates into a reduction in acquisition time. In both of the above phases, the algorithm outputs a list of the positions $\tilde{f}_i$ and the complex values $\tilde{a}_i$ of the non-zero frequency coefficients. In the first phase, the $\tilde{f}_i$'s are integers, *i.e.*, the coefficient lie *on-the-grid*, whereas in the second phase they can be non-integers, *i.e.*, they lie *off-the-grid*.

PHASE 1: On-Grid Recovery

Our on-grid recovery can be divided into three steps:

1. Frequency Bucketization: Recall from previous chapters that in this step, the Sparse Fourier Transform by hashes the frequency coefficients in the spectrum into buckets. Since the spectrum is sparsely occupied, many buckets will be empty and can be simply discarded without additional processing. To hash frequency coefficients into buckets, MRS-SFT uses the aliasing filter presented

in Section 1.1.2. Recall that, this filter works by subsampling the signal in the time domain which results in aliasing the frequency domain. Aliasing is a form of hashing in which frequencies equally spaced by N/p map to the same bucket where p is the subsampling factor.

2. Recovering Frequency Positions: In this step, we aim to find the positions of the non-zero frequency coefficients. For each of the occupied buckets, we want to discover out of all the frequency coefficients that map into the bucket, which one is the non-zero coefficient *i.e.*, the one that has energy.

To do this, we bucketize the frequencies with different aliasing filters, so that different frequency coefficients will hash to different buckets each time. In other words, we repeat the subsampling again at a different subsampling rate. For example, if we bucketized using a subsampling factor p_1, we repeat the process using a different factor p_2. This randomizes the hashing since a frequency coefficient f will hash to bucket $f \mod N/p_1$ in the first bucketization and to different bucket $f \mod N/p_2$ in the second bucketization. Recall that the best choice of subsampling is to use co-prime factor. Co-prime aliasing aliasing guarantees that any two frequencies that collide in one bucketization will not collide in the other bucketization, which best randomizes the voting.

We then use the voting based approach presented in Section 1.1.2 where occupied buckets vote for the frequencies that map to them. Since the spectrum is sparse, most of the buckets do not have energy and hence only few frequencies get votes each time. Non-zero frequency coefficients will always get votes since they create energy in the buckets they map to. The number of bucketizations needed typically depends on the sparsity of the signal but after performing a few bucketizations, the non-zeros frequency coefficients will have the largest number of votes. Hence, this gives us a list of the large frequency coefficients, *i.e.*, $\{\tilde{f}_i\}_{i=0}^{k}$.

3. Recovering the Frequency Values: Now that we have a list of positions of non-zero frequency coefficients $\{\tilde{f}_i\}_{i=0}^{k}$, we want to find their complex values $\{a_i\}_{i=0}^{k}$. Given our model in Equation 10.1 of a sparse signal: if we know the positions of non-zero frequencies (*i.e.*, $\tilde{f}_i$), then Equation 10.1 becomes a system of linear equations with unknowns $\{a_i\}_{i=0}^{k}$. Given $s > k$ discrete input samples of $\mathbf{x}$, we can form an overdetermined system allowing us to solve for each a_i.

To construct the system of linear equations, we concatenate the input time samples into an $s \times 1$ vector which we denote as $\mathbf{x}_S$. We let $\widehat{\mathbf{x}}_K$ be a $k \times 1$ vector of the frequency coefficients which we want to recover. Each coefficient in $\widehat{\mathbf{x}}_K$ corresponds to one position $\tilde{f}_i$ of the non-zero frequencies. Finally, let $\mathbf{A}$ be a matrix of $s \times k$ entries. Each row corresponds to an input time sample and each column corresponds to a non-zero frequency coefficient and the value of each entry will be a complex exponential:

$$\mathbf{A}(t, \tilde{f}_i) = \frac{1}{\sqrt{N}} \exp\left(j \frac{2\pi t \tilde{f}_i}{N}\right) \tag{10.2}$$

Thus, our system of linear equations becomes:

$$\mathbf{x}_S = \mathbf{A}\widehat{\mathbf{x}}_K \tag{10.3}$$

The standard minimal square error solver is to multiply by the pseudo-inverse of $\mathbf{A}$:

$$\widehat{\mathbf{x}}_K^* = \mathbf{A}^\dagger \mathbf{x}_S \tag{10.4}$$

where $\mathbf{A}^\dagger$ is the pseudo-inverse of $\mathbf{A}$. Once we calculate $\widehat{\mathbf{x}}_K^*$, each coefficient will correspond to the position of a non-zero frequency $\tilde{f}_i$. This procedure of recovering frequency coefficients given their positions does not assume that the frequencies are integers and hence we can use it again in second phase.

PHASE 2: Off-Grid Recovery

The off-grid recovery process refines the frequencies discovered in the on-grid stage, allowing them to take non-integer positions.The algorithm formulates this as an optimization problem and uses nonlinear gradient descent to estimate the continuous positions $\{\tilde{f}_i\}_{i=0}^k$ that minimize the square error between observed discrete samples of $x(t)$ and the reconstruction of these samples given by our current coefficient positions and values. Thus, the error function we wish to minimize can be written as:

$$e = \sum_t \left\| x(t) - \frac{1}{\sqrt{N}} \sum_{i=0}^{k} \tilde{a}_i \exp\left(j \frac{2\pi t \tilde{f}_i}{N}\right) \right\|^2 \tag{10.5}$$

where a $\tilde{a}_i$ and $\tilde{f}_i$ are our estimates of a_i and f_i, and the above summation is taken over all the observed discrete samples. We can rewrite the error of this optimization problem using our vector notation from Equation 10.3 as:

$$e = \left\| \mathbf{x}_S - \mathbf{A}\mathbf{A}^\dagger \mathbf{x}_S \right\|^2 \tag{10.6}$$

To solve the above optimization problem we use a gradient descent algorithm based on finite differences. Each iteration of the algorithm updates the list of frequency positions $\{\tilde{f}_i\}$. For each recovered frequency position in $\{\tilde{f}_i\}$, we fix all other frequencies and shift the position of this frequency by a small fractional step $\delta \ll 1$. We shift it in the positive and negative directions and compute the new error e given the new position. We then pick the direction that best minimizes the error e and change the position of the frequency in that direction.[2] We repeat this for every frequency position $\{\tilde{f}_i\}$. Figure 10-2 shows an example of the gradient descent algorithm.

The gradient descent ensures that from iteration i to iteration $i+1$, we are always reducing the residual error, *i.e.*, $e^{(i+1)} < e^{(i)}$ where $e^{(i)}$ denotes the error in iteration i. The algorithm keeps iterating over the frequencies until the error falls below a minimum acceptable error ϵ. Once we have a final list of positions, we can use the same procedure described in the On-Grid recovery to recover the values of these frequency coefficients.

[2]It is possible that none of the directions can minimize error in which case we do not change the position of this frequency.

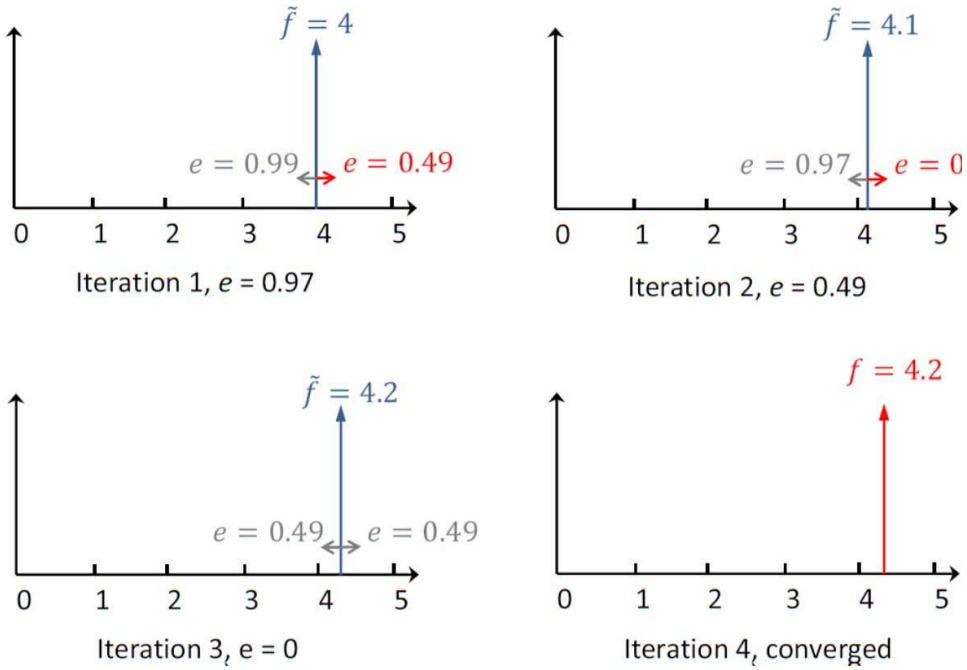


Figure 10-2: **Off-Grid Recovery Using Gradient Descent:** Consider a continuous spectrum with a single peak at position $f = 4.2$. The on-grid recovery stage estimates the initial peak position to the nearest integer of $f = 4$. We set the fractional step size δ of the gradient descent to 0.1. In iteration 1, we calculate the error $e = 0.97$. If we shift the peak to the left, the error becomes $e = 0.99$ and if we shift it to the right, $e = 0.49$. Since shifting the peak to the right will reduce the error the, we shift the peak position to $f = 4.1$. We do the same thing in iteration 2 and shift the peak to the right again. In iteration 3, $f = 4.2$ and neither shifting the peak to the left or right will reduce the error. At this stage the algorithm converges and the off-grid position of the peak has been recovered correctly.

10.2.2 Reconstructing the 2D COSY Spectrum

In the previous section, we introduced a general algorithm for 1D signals. In this section, we will show how we use this algorithm to reconstruct 2D COSY spectra. A 2D COSY signal has two time dimensions:

- t_2 **dimension:** The t_2 dimension is directly measured, *i.e.*, it is fully acquired and hence we can compute the full 1D FFT along this dimension. Furthermore, since we often have sufficient t_2 samples, the truncation artifacts are not as severe as in the t_1 dimension. We can simply multiple the t_2 samples by the QSINE weighting functions to suppress the tails. Thus, for each value of t_1, we multiply the t_2 samples by the QSINE function and take a 1D FFT along t_2.[3] At the end of this step, we get a signal in an intermediate domain $\mathbf{x}'(t_1, f_2)$.

[3]Note that to compute a full Fourier transform of a 2D matrix $\mathbf{x}$, the FFT algorithm computes 1D FFTs along the rows of $\mathbf{x}$ followed by 1D FFTs of the columns of $\mathbf{x}$.

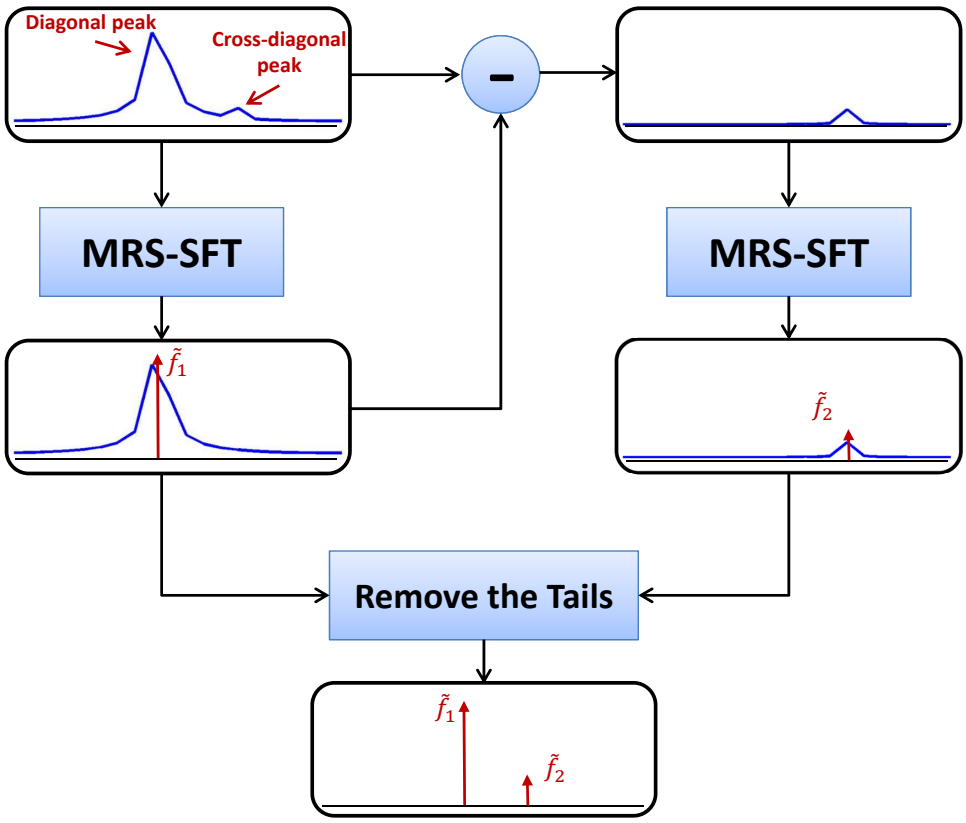


Figure 10-3: **Iterating Between Recovering Diagonal Peaks and Cross-Diagonal Peaks:** MRS-SFT starts by recovering the large diagonal peaks. It then subtracts these peaks and their ringing tails from the signal which allows it to recover the smaller cross diagonal peaks. Once both peaks are recovered, MRS-SFT reconstructs the spectrum by removing the ringing tails resulting from the off-grid positions of the peaks.

- t_1 **dimension:** Since acquiring t_1 increments can be very time consuming, we only acquire a subset of the t_1 samples in order to reduce the acquisition time. We run our MRS-SFT algorithm only along the t_1 dimension. Thus, for each value of f_2, we run the 1D MRS-SFT algorithm over $\mathbf{x}'(t_1, f_2)$ to recover the 2D COSY frequency spectrum $\widehat{\mathbf{x}}(f_1, f_2)$.

Directly applying the MRS-SFT algorithm for each value of f_2, however, does not work due to the huge amplitude difference between the diagonal and cross-diagonal peaks. The cross-diagonal peaks might be immersed in the tails of the diagonal peaks, and simply running MRS-SFT will only recover the diagonal peaks but treat the cross-diagonal peaks as noises. To cope with this, we need to recover the diagonal peaks first. We start by running the MRS-SFT algorithm but restrict our recovery to the diagonal peaks. Once we have the Off-Grid positions and values of the diagonal peaks, we can reconstruct the truncation artifacts and subtract them from our discrete input signal. After subtracting the diagonal peaks and their tails, the cross diagonal peaks emerge. We can then run the MRS-SFT algorithm to recover the cross-diagonal peaks. Figure 10-3 demonstrates this process.

The question that remains is: How to construct and suppress the truncation artifacts caused by the *off-grid* frequency peaks? Given a list of continuous peaks $\{(\tilde{f}_{11}, \tilde{f}_{21}), \cdots, (\tilde{f}_{1k}, \tilde{f}_{2k})\}$ and their values $\{\tilde{a}_1, \cdots, \tilde{a}_k\}$, we can compute the discrete Fourier spectrum at integer positions (f_1, f_2):

$$\widehat{\mathbf{x}}(f_1, f_2) = \sum_{i=0}^{k} \tilde{a}_i \times \Psi_{N_1,N_2}\left(f_1 - \tilde{f}_{1i}, f_2 - \tilde{f}_{2i}\right) \tag{10.7}$$

where N_1 and N_2 are the total number of increments before subsampling along t_1 and t_2 respectively. $\Psi_{N_1,N_2}(\cdot,\cdot)$ is the discretized sinc function which creates the artifacts. It can be written as:

$$\Psi_{N_1,N_2}(f_1, f_2) = \frac{sin(\pi f_1)}{sin\left(\frac{\pi f_1}{N_1}\right)} \frac{sin(\pi f_2)}{sin\left(\frac{\pi f_2}{N_2}\right)} \exp\left(-\pi j \left(\frac{N_1 - 1}{N_1} f_1 + \frac{N_2 - 1}{N_2} f_2\right)\right) \tag{10.8}$$

In our final reconstruction of the output spectrum, we simply suppress the sinc tails by truncating the sinc function:

$$\Psi^*_{N_1,N_2}(f_1, f_2) = \Psi_{N_1,N_2}(f_1, f_2) \times \text{RECT}(f_1, f_2) \tag{10.9}$$

where $\text{RECT}(f_1, f_2)$ is defined to be 1 in the rectangle $-1 \leq f_1, f_2 \leq 1$. The output of our final reconstruction without ringing tails can be written as:

$$\widehat{\mathbf{x}}(f_1, f_2) = \sum_{i=0}^{k} \tilde{a}_i \times \Psi^*_{N_1,N_2}\left(f_1 - \tilde{f}_{1i}, f_2 - \tilde{f}_{2i}\right) \tag{10.10}$$

10.3 Methods

10.3.1 Single Voxel Experiments

We performed experiments on a whole-body 3T MR scanner (Tim Trio Siemens, Erlangen). We used the COSY-LASER sequence (TR = 1.5 s, TE = 30 ms) [10] to acquire 2D COSY spectra on 3 volunteers and a brain phantom. The acquired MRS data was post-processed in MATLAB using four methods:

- **MRS-SFT:** The method proposed in this chapter uses 60 t_1 increments on volunteers and 64 t_1 increments on brain phantom.
- **FFT:** A truncated FFT which uses 60 t_1 increments on volunteers and 64 t_1 increments on brain phantom.
- **Full-FFT:** Longer sampled FFT which uses 160 t_1 increments on volunteers and 180 t_1 increments on brain phantom.
- **Compressive Sensing (CS):**[4] An iterative thresholding based compressive sensing algorithm which uses 60 t_1 increments on volunteers and 64 t_1 increments on brain phantom.

[4]We experimented with several compressive sensing algorithms and we present here the best compressive sensing results which we obtained.

For FFT, Full FFT, and CS, we pre-process the samples by multiplying with a QSINE window and using linear prediction to improve the cross peaks and reduce artifacts.

10.3.2 Multi Voxel Experiments

Correlation chemical shift imaging was acquired with an adiabatic spiral COSY sequence [11] which was here improved with MRS-SFT. The experiments were performed on a whole-body 3T MR scanner (Tim Trio, Siemens, Erlangen). Acquisition parameters included: TR = 1.8 s, TE = 30 ms, 72 sparse t_1 samples out of 252 consecutive samples, 380 points in t_2 zero filled to 512, 10 ppm spectral window in both f_1 and f_2 dimensions, 4 averages, a 16×16 voxel matrix, FOV= 200×200 mm, and an acquisition time of 17 min and 32 s. The acquired MRS data was post-process in MATLAB using the MRS-SFT with 72 t_1 samples and the Full-FFT with 252 t_1 samples.

The f_1 dimension of 2D COSY is also particularly susceptible to scan instabilities arising from frequency drifts and patient motion. Post-processing methods have limited ability to recover spectral quality in particular downgraded water suppression due to frequency drift or change in metabolite signal amplitude due to shift of voxel position across a time series. To alleviate these problems, we incorporated into our 2D COSY acquisition a recently developed real-time motion correction and shim update sequence module.

10.4 MRS Results

10.4.1 Single Voxel Results

The result for phantom data is shown in Figure 10-4 for the four methods described above (MRS-SFT, FFT, Full-FFT, and CS). The figure shows the 2D MRS spectrum recovered by each method. As can be seen, MRS-SFT is able to eliminate the truncation artifacts while maintaining the cross diagonal peaks. For example, the metabolite Myo-inositol is clearly present only in the MRS-SFT and Full-FFT results. However, the Full-FFT data requires 180/64 = 2.8× more time to acquire all the measurements. The figure also shows that despite using the QSINE window and linear prediction, the other techniques still suffer from significant truncation artifacts. Note that all four spectra have the same scaling along the iso-contour lines and hence changing the window setting to suppress the artifacts would also eliminate the cross diagonal peaks.

In the case of in-vivo data, *i.e.*, experiments ran on human volunteers, the truncation artifacts are even more severe as shown in Figure 10-5. In this case, the ringing tails are highly prominent in the FFT, Full-FFT and CS results and hence cause a lot of truncation artifacts which are hard to distinguish from the actual cross diagonal peaks. MRS-SFT, however, can model and subtract these ringing tails making the cross diagonal peaks much more apparent. For example, Choline (Cho) is much clearer in the MRS-SFT result as can be seen in Figure 10-5. In this case, even the Full-FFT, which requires 160/60 = 2.67× longer measurement time than MRS-SFT, continues to suffer from artifacts.

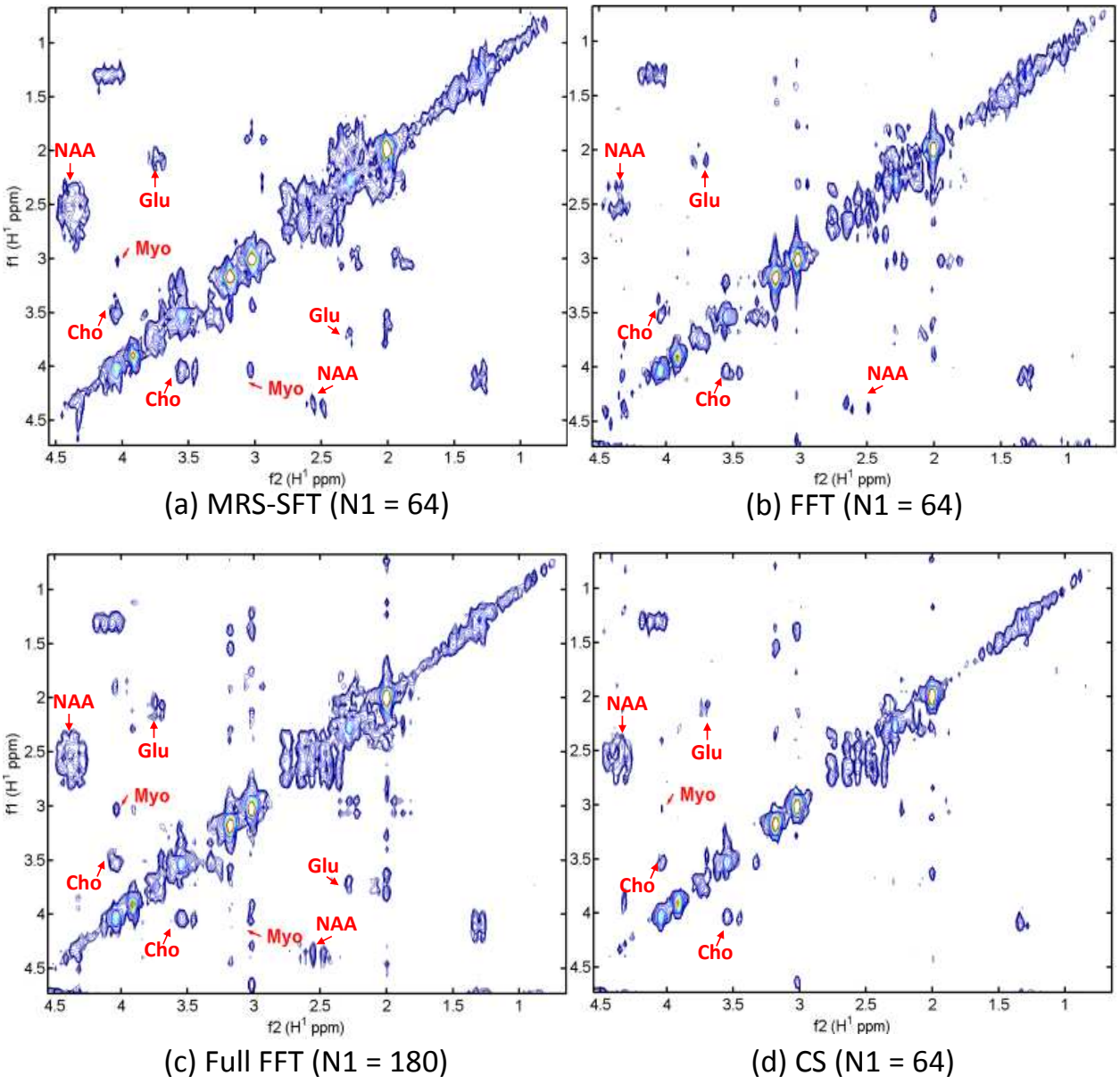


Figure 10-4: **MRS-SFT Recovery Results on Brain Phantom Data:** The figure shows that MRS-SFT can recover all the metabolites similar to a Full FFT while using 180/64=2.8× less samples. For the same number of samples, a truncated FFT or CS loose some of the metabolites (e.g. Myo) and degrade the quality of the peaks. The following metabolites have been labeled for reference: Choline (Cho), N-Acetylaspartate (NAA), Myo-Inositol (Myo), and Glutamate (Glu).

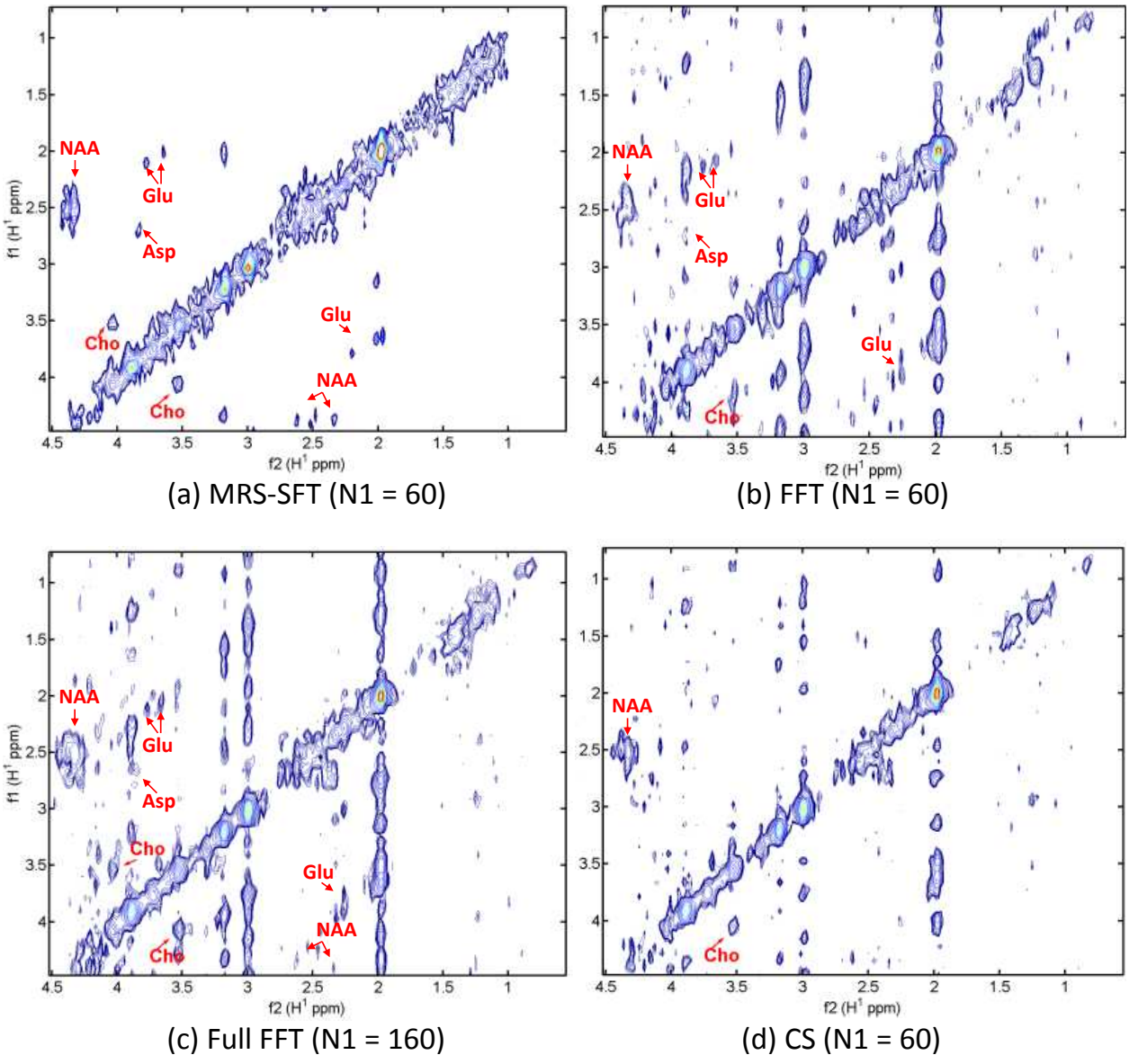


Figure 10-5: **MRS-SFT Recovery Results on In-Vivo Data:** The figure shows that MRS-SFT has significantly less clutter making it much easier to distinguish artifacts from cross diagonal peaks. For Full FFT, FFT and CS, it is very hard to distinguish the metabolites from the artifacts and for FFT and CS, some of the metabolites are lost. The following metabolites have been labeled for reference: Choline (Cho), N-Acetylaspartate (NAA), Aspartate (Asp), and Glutamine (Glu).

	Phantom	In Vivo
MRS-SFT	13.78	2.05
Full-FFT	4.29	-11.7
FFT	0.84	-11.6
CS	-0.65	-13.3

Table 10.1: **Signal to Artifact Ratio (dB)**

	Phantom	In Vivo
MRS-SFT	0.17	0.15
Full-FFT	0.23	0.24
FFT	0.15	0.15
CS	0.21	0.13

Table 10.2: **Line Width of NAA (ppm)**

To quantify the above results, we compute the Signal to Artifact Ratio of the cross diagonal peaks recovered by each of the four methods. The results are shown in Table 10.1. As can be seen, MRS-SFT has the highest signal to artifact ratio and which is 8 dB higher than Full-FFT for phantom data and 14 dB higher for in vivo data. MRS-SFT also reduces the average FWHM (Full Width at Half Maximum) of cross-peaks by 40% for in vivo and 25% for phantom compared to the Full-FFT. For example, the line width of NAA (N-acetyl aspartate) is reduced from 0.23 ppm to 0.17 ppm in phantom data and from 0.24 ppm to 0.15 ppm in in vivo data as shown in Table 10.2.

10.4.2 Multi Voxel Results

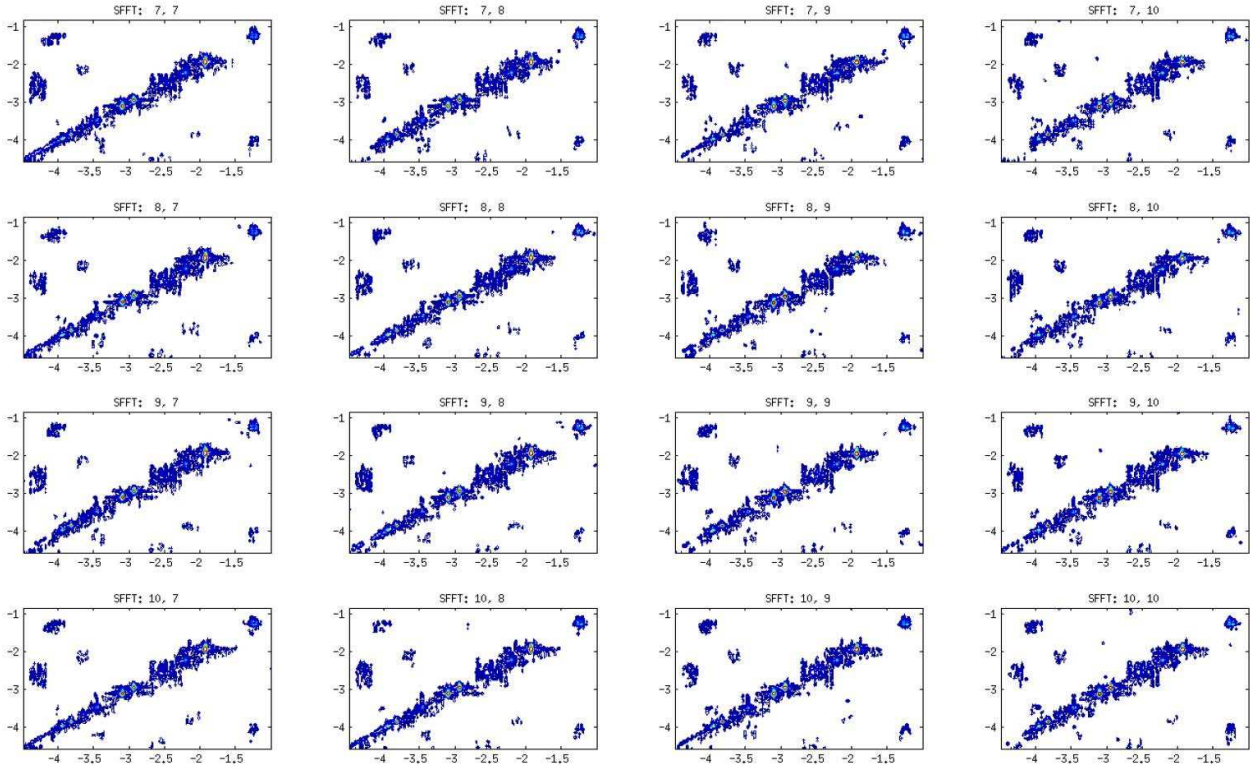
In the case of correlation chemical shift imaging, we demonstrate that by using MRS-SFT combined with real-time motion correction, we can recover multi-voxel MRS spectra which does not suffer from artifacts and ringing tails. Figures 10-6(a) and (b) show the resulting spectra of a 4×4 matrix sampled from the 16×16 voxels which were acquired and processed using MRS-SFT and Full-FFT respectively. For all voxels, MRS-SFT can eliminate the t_1 truncation artifacts while reducing the measurement time by a factor of $3.5\times$ (252/72).

The above single voxel and multi-voxel results demonstrate that by using MRS-SFT can:

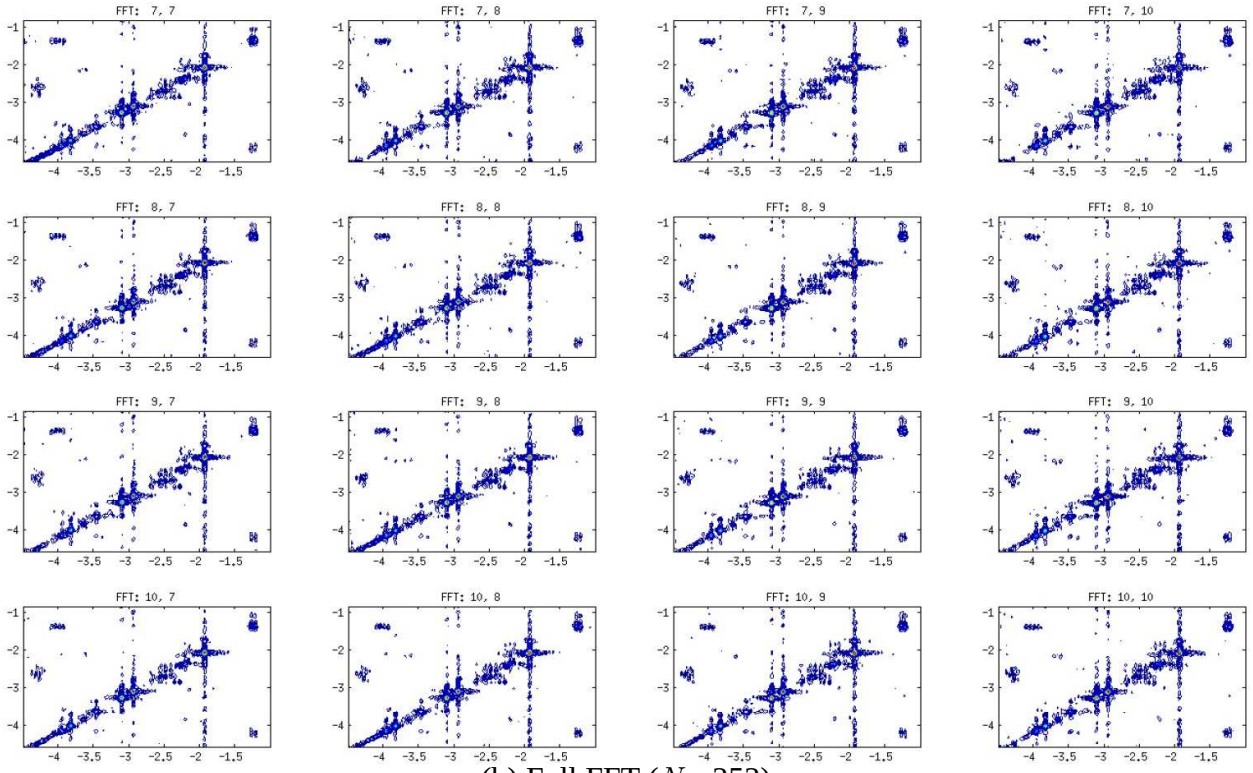
1) Reduce the measurement time by almost a factor of three.
2) Eliminate the t1 truncation artifacts resulting from the ringing tails of the diagonal.
3) Improve the SNR and resolution of cross-peaks.

10.5 Conclusion

In this chapter, we presented MRS-SFT; a system that leverages the Sparse Fourier Transform to reduce the time the patient spends in an MRI machine while generating clearer images for 2D COSY MRS experiments. Our results indicate the MRS-SFT can reduce the acquisition time by a factor of 3 and suppress truncation artifacts. MRS-SFT is also superior to Compressed Sensing in terms of reducing the f_1 ringing and enhancing the SNR. Typically, 2D COSY spectra reconstructed with conventional FFT use windowing functions such as QSINE and linear prediction to improve cross-peaks and reduce the artifacts. However, QSINE windowing selectively enhances only some of the cross-peaks and linear prediction reduces the SNR and introduces spiking artifacts. The Sparse Fourier Transform is less biased in finding the cross-peaks and, hence provides a more robust method in dealing with the limitations of in vivo MRS.



(a) MRS-SFT (N_1=72)



(b) Full FFT (N_1=252)

Figure 10-6: **Multi-Voxel Recovery Results for MRS-SFT vs. Full FFT**

Chapter 11

Fast Multi-Dimensional NMR Acquisition and Processing

11.1 Introduction

Multi-dimensional NMR (Nuclear Magnetic Resonance) spectroscopy is an invaluable biophysical tool in chemistry, structural biology, and many other applications. However, from its introduction in 1970s, the technique is impeded by long measurement times, heavy computations and large data storage requirements. These problems stem from the huge number of data points needed for quantifying the frequency domain spectrum with the required resolution.

With the traditional systematic data sampling in the time domain, the duration of an NMR experiment increases exponentially with spectral dimensionality and polynomially with resolution. The rapid development in the field of fast spectroscopy with non-uniform sampling reduces the measurement time by decreasing number of the acquired data points [17, 33, 85, 140]. Non-uniform sampling (NUS) has enabled the acquisition and analysis of practical high-resolution experiments of dimensionality up to 7D [81, 100, 129]. Success of the NUS techniques is explained by the notion that the NMR spectrum is sparse in the frequency domain, i.e. only a small fraction of the spectrum contains signals, while the rest contains only baseline noise. Moreover, typically the higher the spectrum dimensionality and resolution are, the sparser the frequency domain spectrum is. While the numerous NUS spectra reconstruction algorithms differ in their underlying assumptions, the common theme is that all information about the spectral signals can be obtained from a relatively small number of measurements, which is linear to the number of signals and nearly independent on the spectrum dimensionality and resolution.

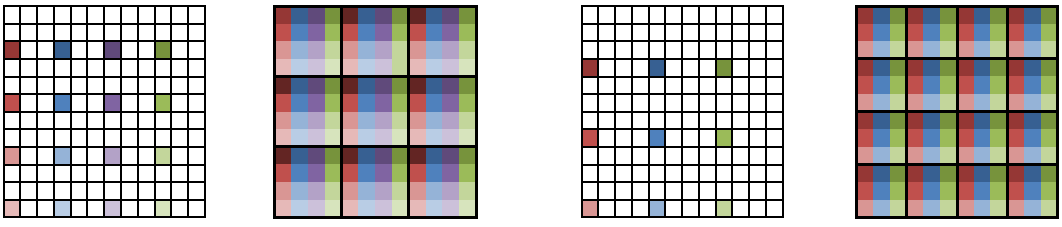
NMR measurements are performed in the time domain and, in the case of traditional Fourier spectroscopy, the time signal is converted to the frequency spectrum by the Discrete Fourier Transforms (DFT). For a d-dimensional spectrum with N points for each spectral dimension, we need to sample N^d experimental points, perform DFT with $O(N^d \log N^d)$ elementary mathematical operations and allocate $O(N^d)$ bytes for spectrum processing, storage, and analysis. For example, a moderate-resolution 5D spectrum with N=256 for all dimensions requires 4 TB of storage. Even if such spectrum can be computed, it cannot be easily handled in the downstream analysis. Algo-

rithms used for reconstructing the complete spectrum from the NUS data, require at least the same and often significantly larger computations and storage than the traditional Fourier based approach. For example, for the Compressed Sensing (CS) [83, 99], storage is $O(N^d)$ and the amount of calculations is polynomial on N^d. Moreover, these algorithms are iterative and thus are impractical, when data do not fit into computer operative memory. Modern computers meet the computational and storage requirements for 2D, 3D, and relatively low-resolution 4D spectra. Yet, spectra of higher dimensionality and/or resolution are still beyond reach, unless the analysis is performed in low dimensional projections or is reduced to small regions restricted in several spectral dimensions. In this work, we demonstrate for the first time a new approach allowing reconstruction, storage, and handling of high dimensionality and resolution spectra.

Reconstructing a spectrum with computational complexity and storage, which are sub-linear in respect to the number of points in the full spectrum (N^d) may only work by using NUS in the time domain and by computing a sparse representation of the spectrum, i.e. without producing the complete spectrum at any stage of the procedure. The latter requirement excludes powerful non-parametric NUS processing algorithms designed to reconstruct the full spectrum, such as Maximum Entropy (ME) [15, 82], Projection Reconstruction (PR) [52], Spectroscopy by Integration of Frequency and Time Domain Information (SIFT) [53, 120], Compressed Sensing [83, 99], and Low Rank reconstruction [147]. The parametric methods such as Bayesian [21], maximum likelihood [31], and multidimensional decomposition (MDD) [93] approximate the spectrum using a relatively small number of adjustable parameters, and thus are not limited in spectral dimensionality and resolution. However, due to the intrinsic problems of choosing the right model and convergence, the parametric algorithms cannot guaranty the detection of all significant signals in a large spectrum. Another approach Multidimensional Fourier Transform (MFT) [98] for large spectra exploits prior knowledge about the signal positions in some or all of the spectral dimensions. MFT reconstructs only small regions around known spectral peaks and thus requires less computations and storage. The Signal Separation Algorithm (SSA) [162], represents a combination of the parametric and MFT methods and to some extent inherits strong and weak points of both. Notably, the SSA also avoids dealing with the full spectrum matrices in time and frequency domains and can deal with large spectra. The method was demonstrated for high-resolution 4D spectra with the corresponding full sizes of tens of gigabytes.

The Sparse Fourier Transform is the first non-parametric algorithm capable of producing a high quality sparse representation for high resolution and dimensionality spectra. The Sparse Fourier Transform offers fast processing and requires manageable data storage, which are sub-linear to the total number of points in the frequency spectrum. It also allows reconstruction of complete high quality ND spectra of any size and dimensionality. It is most useful for high-resolution spectra of four and more dimensions, where methods like CS require too much computations and storage.

In this chapter, we will describe the multi-dimensional version of the Sparse Fourier Transform algorithm which we use for NMR spectra. We will also demonstrate its effectiveness as recovering NMR spectra from only 6.4% of the input samples through experiments on a 4D BEST-HNCOCA spectrum of ubiquitin.



(a) Subsampling by 3 (b) Subsampling by 4

Figure 11-1: **2D Bucketization Using Co-prime Aliasing** on a 12×12 signal. (a) Subsampling by a factor of 3 folds (aliases) the spectrum by 3. Frequencies with the same color sum up together in the same bucket. (b) Subsampling by a factor of 4, which is co-prime to 3 ensures that the frequencies will be bucketized differently avoiding collisions.

11.2 Multi-Dimensional Sparse Fourier Transform

In this section, we adapt the Sparse Fourier Transform algorithms for multi-dimensional NMR spectra. We will describe the Sparse Fourier Transform algorithm for 2D signals of size $N \times N$. However, the algorithm can be easily extended to any dimension. We will use $\mathbf{X}$ to denote the 2D time signal and $\widehat{\mathbf{X}}$ to denote its 2D discrete Fourier transform (DFT). We will use (f_1, f_2) to denote a frequency position and $\widehat{\mathbf{X}}(f_1, f_2)$ to denote the spectral value at this frequency position. For simplicity, we will refer to frequencies that have no signal energy, *i.e.*just noise, as the zero frequencies and the frequencies that have signal energy as the non-zero frequencies.

Recall the two key components of the Sparse Fourier Transform operates : bucketization and estimation. The bucketization step divides the frequency spectrum into buckets where the value of each bucket represents the sum of the values of frequencies that map to that bucket. Since the spectrum is sparse, many buckets will be empty and can be discarded. The algorithm then focuses on the non-empty buckets and computes the frequencies with large values in those buckets in the estimation step. Below we describe in details the bucketization and estimation techniques we use for NMR. Some of the concepts below have been introduced in previous chapters. However, here we formalize them for multi-dimensional signals and put them in the context of NMR experimentation.

11.2.1 Multi-Dimensional Frequency Bucketization

Bucketization Using Co-Prime Aliasing

Bucketization through co-prime aliasing previously appeared in Chapters 1 and 7. Here, we formalize it for multi-dimensional signals. Let $\mathbf{B}$ be a sub-sampled version of $\mathbf{X}$, *i.e.*, $\mathbf{B}(t_1, t_2) = \mathbf{X}(p \cdot t_1, p \cdot t_2)$ where p is the sub-sampling factor. Then, $\widehat{\mathbf{B}}$, the FFT of $\mathbf{B}$, is an aliased version of $\widehat{\mathbf{X}}$, *i.e.*:

$$\widehat{\mathbf{B}}(b_1, b_2) = \sum_{i=0}^{p-1} \sum_{j=0}^{p-1} \widehat{\mathbf{X}}(b_1 + i \cdot N/p, b_2 + j \cdot N/p) \tag{11.1}$$

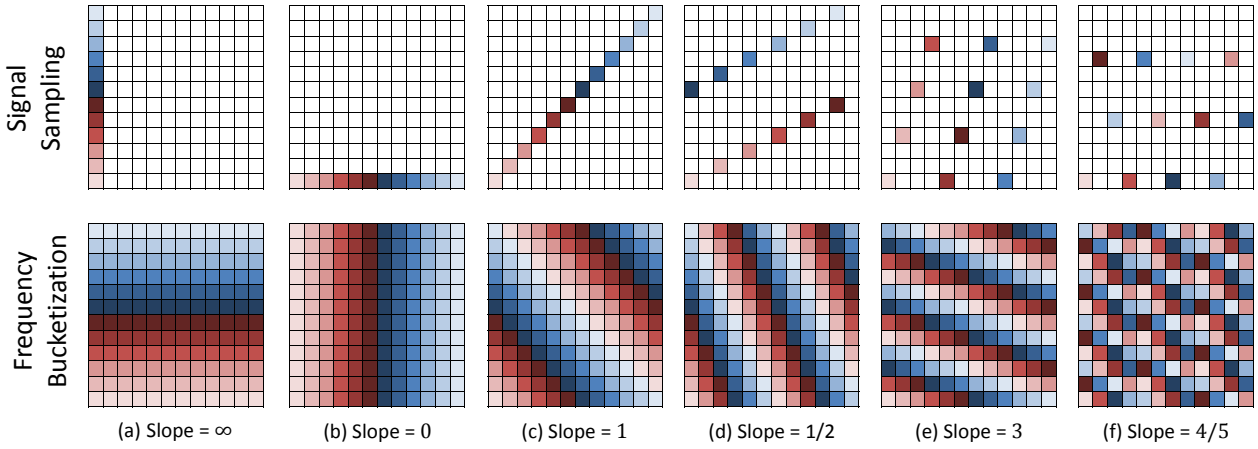


Figure 11-2: **Bucketization Using Discrete Line Projections**. The top shows the discrete line, which was sampled and the bottom shows how the frequencies are projected. Frequencies with the same color will sum up together in the same bucket. This is shown for different slope (a)-(f). Since the lines are discrete, they wrap around and can result in pseudo random sampling and projection patterns as can be seen in (f).

Thus, frequencies equally spaced by an interval N/p along each dimension map to the same bucket, *i.e.*, frequency (f_1, f_2) maps to bucket number (b_1, b_2) such that:

$$(b_1, b_2) = (f_1 \mod N/p, f_2 \mod N/p) \tag{11.2}$$

Further, recall that the value of each bucket is the sum of the values of only the frequencies that map to the bucket as can be seen from Equation 11.1. Now that we mapped the frequencies into buckets, we can leverage the fact that the spectrum of interest is sparse and hence most buckets have noise and no signal. We compare the energy (i.e., the magnitude squared) of a bucket with the noise level and considers all buckets whose energy is below a threshold to be empty. We then focus on the occupied buckets and ignore empty buckets.

Recall that most of the occupied buckets will have a single non-zero frequency. However, some buckets will have more than one non-zero frequency i.e., collisions. Recall, that we can resolve collisions by repeating the bucketization with a different sampling factor p' that is co-prime with p. Co-prime aliasing guarantees that any two frequencies that collide in the first bucketization will not collide in the second bucketization. Figure 11-1 shows an example of bucketization using co-prime aliasing of a 2D spectrum.

Bucketization Using Discrete Line Projections

Bucketization using discrete line projections previously appeared in Chapters 1 and 9. We repeat the formalization of this form of bucketization here and put in the context of multi-dimensional NMR spectra. Recall that, performing a 1D DFT of a discrete line, yields the projection of the spectrum onto a corresponding line in the Fourier domain. Specifically, let $\mathbf{y}$ be the 1D discrete line corresponding to a 2D signal $\mathbf{X}$, parameterized by $t \in [0, \cdots, N-1]$:

$$\mathbf{y}(t) = \mathbf{X}(\alpha_1 t \mod N, \alpha_2 t \mod N) \tag{11.3}$$

where α_1,α_2 are integers whose greatest common divisor is invertible modulo N such that $0 \leq \alpha_1, \alpha_2 < N$. α_1/α_2 represents the slope of the line. Then $\hat{\mathbf{y}}$, the DFT of $\mathbf{y}$, is a projection of $\widehat{\mathbf{X}}$ onto this line. That is each point in $\hat{\mathbf{y}}$ is a summation of the N frequencies that lie on a discrete line orthogonal to $\mathbf{y}$ as shown in Figure 11-2. Specifically, the frequencies (f_1, f_2) that satisfy $\alpha_1 f_1 + \alpha_2 f_2 = f \mod N$ will project together into the same bucket f and sum up to $\hat{\mathbf{y}}(f)$. Figure 11-2 shows some examples of discrete lines and their projections. Note that discrete lines from Equation 11.3 wrap around as can be seen in Figures 11-2(d),(e),(f) and hence bucketization can result in a pseudo random non-uniform sampling as shown in Figure 11-2(f). Also note that this can be extended to any number of dimensions. In that case, we can take projections of discrete lines, planes or hyper-planes.

The above procedure is based on the Fourier projection-slice theorem [20] and thus bears resemblance to the reduced dimensionality techniques and radial sampling [19, 33, 80, 164]. The important difference, however, is that the sampling defined by Equation 11.3 is performed on the Nyquist time domain grid of the full multidimensional experiment, while the traditional radial sampling is off-grid. As it is described in the next section, having all sampled point on the grid allows direct frequency estimation without resorting to the often problematic inverse Radon transform used in the traditional projection reconstruction [101].

Further, discrete projections can benefit from the complex virtual echo representation [122], which improves the sparsity. Specifically, for this representation, once we sample a discrete line passing through the origin (0, 0), we automatically obtain the samples of another discrete line which is symmetric to the first line with respect to one of the axes i.e. if we sample a line with slope α_1/α_2 , we directly get the line with slope $-\alpha_1/\alpha_2$. For higher dimensions the gain is larger. If we sample a discrete line in a d-dimensional signal, we automatically get the samples of $2^d - 1$ other discrete lines. For example, in 3D for a discrete line with slope $(\alpha_1, \alpha_2, \alpha_3)$, we get the samples of three other discrete lines which are $(-\alpha_1, \alpha_2, \alpha_3), (\alpha_1, -\alpha_2, \alpha_3), (\alpha_1, \alpha_2, -\alpha_3)$. Note that $(-\alpha_1, -\alpha_2, \alpha_3)$ and $(\alpha_1, \alpha_2, -\alpha_3)$ define the same projections and thus only one of these is needed.

Choosing the Bucketization and Number of Buckets in NMR

The choice of bucketization and number of buckets depends on the sparsity. If the signal has k non-zero frequency peaks, then the number of buckets in each bucketization should be at least $O(k)$ or larger. The discrete projections and aliasing approaches give us a lot of flexibility in choosing the number of buckets. For example, in a 4D signal, if k is very large we can project on 2D discrete planes to get N^2 buckets or 3D discrete hyper-planes to get N^3 buckets. If k is small, we can project on 1D discrete lines to get N buckets. We can also combine discrete projections with aliasing to accommodate almost any value of k. For example, we can project on sub-sampled lines as shown in Figures 11-3(a),(b) to get $N/2$ or $N/3$ buckets. We can also project on sub-sampled plane as shown in Figure 11-3(c) to get $2N$ buckets.

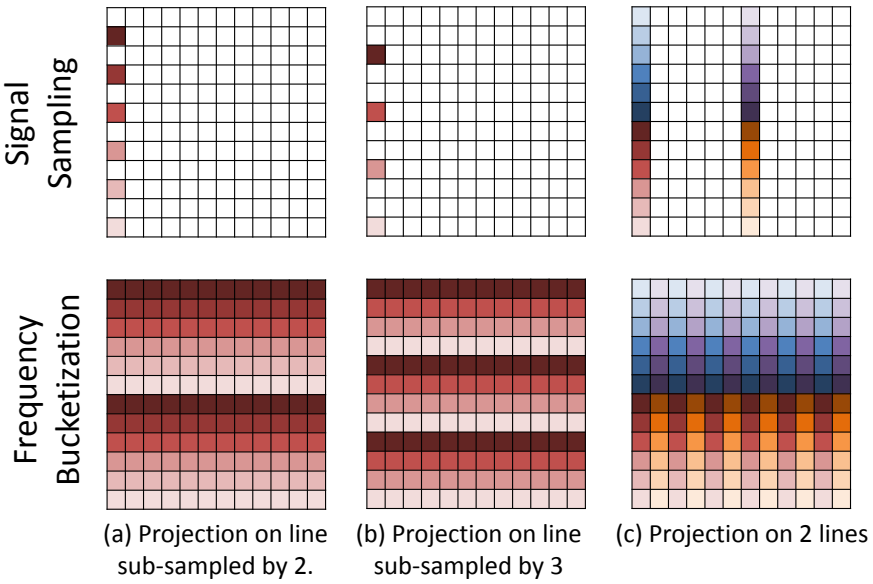


Figure 11-3: **Combining Discrete Projections with Aliasing.** (a, b) Projection on sub-sampled discrete line gives number of buckets less than N. (c) Projection on two lines (i.e. sub-sampled plane) gives number of buckets larger than N. Frequencies with the same color sum up together in the same bucket.

11.2.2 Multi-Dimensional Frequency Estimation

Here we describe the frequency estimation step we use in the context of NMR. Recall that in this step, for each of the occupied buckets we want to identify which frequencies created the energy in these buckets, and what are the values of these frequencies.

To identify the frequencies, we will use the voting based approach which we previously introduced in Chapters 1, 3 and 9. In this approach, occupied buckets vote for the frequencies that map to them. Since the spectrum is sparse, most of the buckets are empty and hence only few frequencies get votes each time. Because by definition the non-zero frequencies will end up in occupied buckets, they will get a vote every time we perform a new bucketization. In practice, a non-zero frequency may miss the votes in some of the bucketizations. This may happen when the corresponding spectral peak is very weak and/or is cancelled by superposition with a peak of the opposite sign. Such negative peaks may be present in the spectrum, for example, in case of the peak aliasing when the acquisition is started from half-dwell time. Nevertheless, after performing a few random bucketizations by using co-prime aliasing or discrete lines with different slopes, the non-zero frequencies will have the largest number of votes, which allows the algorithm to identify these frequencies. An illustrative example of this voting based estimation can be found in Section 9.5.2.

Now that we have a list of non-zero frequencies (f_1, f_2), we want to estimate the values $\widehat{\mathbf{X}}(f_1, f_2)$ of these frequencies. We may use a simple approach analogous to those used in the method of projection reconstruction [101]. It would estimate the value of each non-zero frequency as the median value of the different buckets to which this frequency was mapped across the different bucketizations. However, this approach may yield a poor reconstruction in the presence of noise

and significant signal overlap. Instead, we can compute better estimates of the values of the non-zero frequencies by harnessing the fact that all these frequencies are defined at the same Nyquist grid. The values of the occupied buckets can be viewed as linear combinations of the values of the non-zero frequencies. Hence, we can construct a linear system of equations:

$$A\mathbf{x} = \mathbf{b} \tag{11.4}$$

where the unknown vector $\mathbf{x}$ corresponds to the values of the non-zero frequencies and the known vector $\mathbf{b}$ corresponds to the values of the buckets. The matrix A is a sparse binary matrix that specifies which frequency values contribute to which buckets. In general, this system is over-determined since the number of occupied buckets can be as large as the number of non-zero frequencies times the number of bucketizations. Hence, the solution that minimizes the mean square error of the frequency values is:

$$\mathbf{x}^* = A^\dagger \mathbf{b} \tag{11.5}$$

where $A^\dagger$ is the is the pseudo inverse of A. This approach of computing the values of non-zero frequencies is more robust to noise and can correct for errors by estimating the falsely presumed non-zero frequencies to near zero values. This comes at the cost of the additional computational complexity associated with computing the pseudo inverse. However, since the number of non-zero frequencies is small, the size of the matrix A is still small.

11.3 Materials and Methods

The 4D fully sampled BEST-HNCOCA [105] spectrum of 1.7 mM human ubiquitin sample (H2O/D2O 9:1, pH 4.6) was acquired at 25 °C on 800 MHz Bruker AVANCE III HD spectrometer equipped with 5 mm CP-TCI probe with the Nyquist grid of $16 \times 16 \times 16$ complex time points (acquisition times 12 ms, 10 ms and 4.4 ms) for the ^{15}N, ^{13}CO and ^{13}Cα spectral dimensions, respectively. The amide region of the full reference 4D spectrum (9.7 -7.0 ^{1}H ppm, 174 points) was processed using NMRPipe software [38]. For the Sparse Fourier Transform processing, only the directly detected dimension was processed in NMRPipe followed by extraction of the same amide region.

The hyper-complex time domain data were converted to the complex virtual echo (VE) representation [122] with dimensions $174 \times 32 \times 32 \times 32$. Unlike the original hyper-complex data representation, the VE is directly amenable for the multi-dimensional Sparse Fourier Transform processing and improves the result of the reconstruction from the NUS data. However, the VE relies on the prior knowledge of the phase and requires the linear phase correction in the indirectly detect dimensions of the ND spectrum to be multiple of π (i.e. 0, π, 2π, $\cdots$).

Two independent sampling tables were generated using Discrete Line Projections given by Equation 11.3. Each of the tables contained 6 orthogonal projections, i.e. (1, 0, 0), (0 1 0), (1 1 0), (1 0 1), (0 1 1), (1 1 1), and 16 projections obtained by random combinations of prime numbers less than 32 (i.e. 0 1 2 3 5 7 11 13 17 23 29 31). As described in the theory, these 6 + 16 unique line projections were augmented by 7 + 48 symmetric projections, respectively, which resulted in total 77 line projections in the Sparse Fourier Transform calculations. In total, each

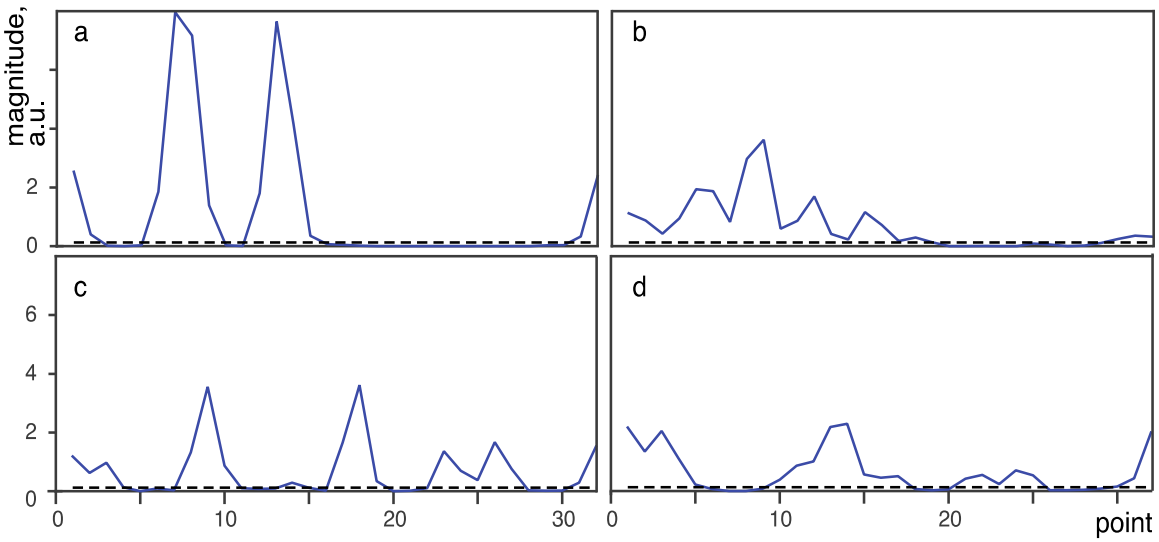


Figure 11-4: **Discrete Line Projections Obtained on 4D BEST-HNCOCA Spectrum of Ubiquitin:** at 1H frequency of 8.05 ppm for with prime numbers for line slopes $[\alpha_1, \alpha_2, \alpha_3]$: (a) [1,0,31], (b) [17,1,23], (c) [31,7,3], (d) [11,1,29]. Horizontal dashed lines in each panel indicate the Sparse Fourier Transform adaptive threshold used for frequency identification.

NUS data set included 2096 (6.4%) out of total 32K complex time domain points in the indirectly detected dimensions. Figure 11-4 shows a few examples of the obtained discrete line projections. The number of used samples or discrete lines depends on the sparsity of the 3D spectra. Since, for all 174 directly detected points, the same indirectly detected points are used, the minimum number of samples needed is bounded by the 3D sub-spectrum with the lowest sparsity, i.e. the largest part occupied by signals.

Although, the same discrete lines are used in all 3D sub-spectra, the cut-off threshold for selecting frequencies varies for different directly detected points. The Sparse Fourier Transform adaptively sets the cut-off threshold by ensuring that the system of linear equations in Equation 11.4 is well determined. This allows lowering the cut-off and thus improving sensitivity for regions with small number of signals. Finally, the Sparse Fourier Transform calculations were performed in MATLAB with the resulting spectrum exported to NMRPipe format for comparison with the reference spectrum.

11.4 Results

We demonstrate the Sparse Fourier Transform ability to reconstruct 4D HNCOCA spectrum using a NUS data set extracted from the complete experiment, which is acquired with $512 \times 16 \times 16 \times 16$ complex time points for the ^{1}H, ^{15}N, $^{1}3$CO and ^{13}Cα spectral dimensions, respectively. After conventional Fourier processing of the directly detected ^{1}H dimension and extraction of the amide region 9.7 - 7.0 ^{1}H ppm (174 points), the Discrete Line Projections which were used for bucketization selected 262 (6.4%) hyper complex time domain points in the indirectly detected dimensions.

In a real experiment, of course, only these selected data points need to be acquired, thus reducing the measurement time to 6.4% of the full experiment. The selected hyper-complex data points were converted to the complex virtual echo representation [53, 122], which contained 174×2096 points out of the full complex array with dimensions $174 \times 32 \times 32 \times 32$. Then, in the frequency domain, the Sparse Fourier Transform voting algorithm identified 10588 non-zero points, which correspond to approximately 100 peaks with 100 data points per peak in the 4D spectrum. In the resulting spectrum, only these non-zero intensities were stored, which constitute to less than 0.2 % of the full reference spectrum.

The running time of the Sparse Fourier Transform is dominated by the time to compute the projections and perform the pseudo inverse. For the current experiment, the time to compute all projections in MATLAB is 0.25 ms and the time to perform the pseudoinverse is around 150 ms. CS based algorithms like IST would require between 10-100 iteration while performing a full FFT on 3D spectra and hence take between 30-300 ms. The computational advantage of the Sparse Fourier Transform is expected to increase for higher resolution and dimensions. However, a more thorough analysis of runtime would require implementing the Sparse Fourier Transform algorithm in C/ C++ and is thus is left for future work.

A few points are worth noting. First, the pseudoinverse matrix is computed separately for each point in the directly detected dimension. Thus, the size of this matrix depends on the number of peaks in each of the 3D spectra of indirectly detected dimensions as opposed to the number of peaks in the entire 4D spectrum. The pseudoinverse of the matrix used in our work (ca 2000×250), takes 0.15 sec. Hence, calculating the pseudoinverse fits well in to a desktop computer memory. Even for a more demanding case of quadruple matrix size required for a large system or NOESY type spectrum, the calculation will take less than 40 seconds per point in the directly detected spectral dimension. Second, the Sparse Fourier Transform can naturally benefit from prior knowledge about the dark regions in the spectrum in a similar manner to the SIFT method. For example, we can compute the pseudoinverse only for the peaks, which we want to estimate. We can also avoid collecting votes for frequencies we know do not contain energy.

Figure 11-5 illustrates the Sparse Fourier Transform reconstructed spectrum using two different approaches for the evaluation of the frequency values. Comparison of panels (a),(b) and (c),(d) in Figure 11-5 shows that the spectrum obtained using the matrix inversion from Equation 11.5 is very similar to the full reference spectrum. This visual impression is corroborated by the correlation in Figure 11-5(e) of the cross-peak intensities between the full reference and Sparse Fourier Transform reconstructed spectrum. It can be seen that most of the peaks found in the reference spectrum (red circles) are faithfully reproduced in the Sparse Fourier Transform reconstruction.

Results of spectral reconstructions from NUS may vary for different sampling schedules [12]. In order to check this, we calculated the Sparse Fourier Transform spectrum with an alternative set of randomly selected projections. The two independent Sparse Fourier Transform spectral reconstructions had comparable quality. Pairwise correlations between the peak intensities in the reference spectrum and in the two independent Sparse Fourier Transform reconstructions were very similar as can be seen in Figure 11-6. 98 peaks were detected in the reference spectrum using the peak-picker program from NMRPipe software [38] with the noise level of 0.01 (in the scale used in Figure 11-5(e),(f)) and peak detection threshold 0.05. Thanks to the high spectral sensitiv-

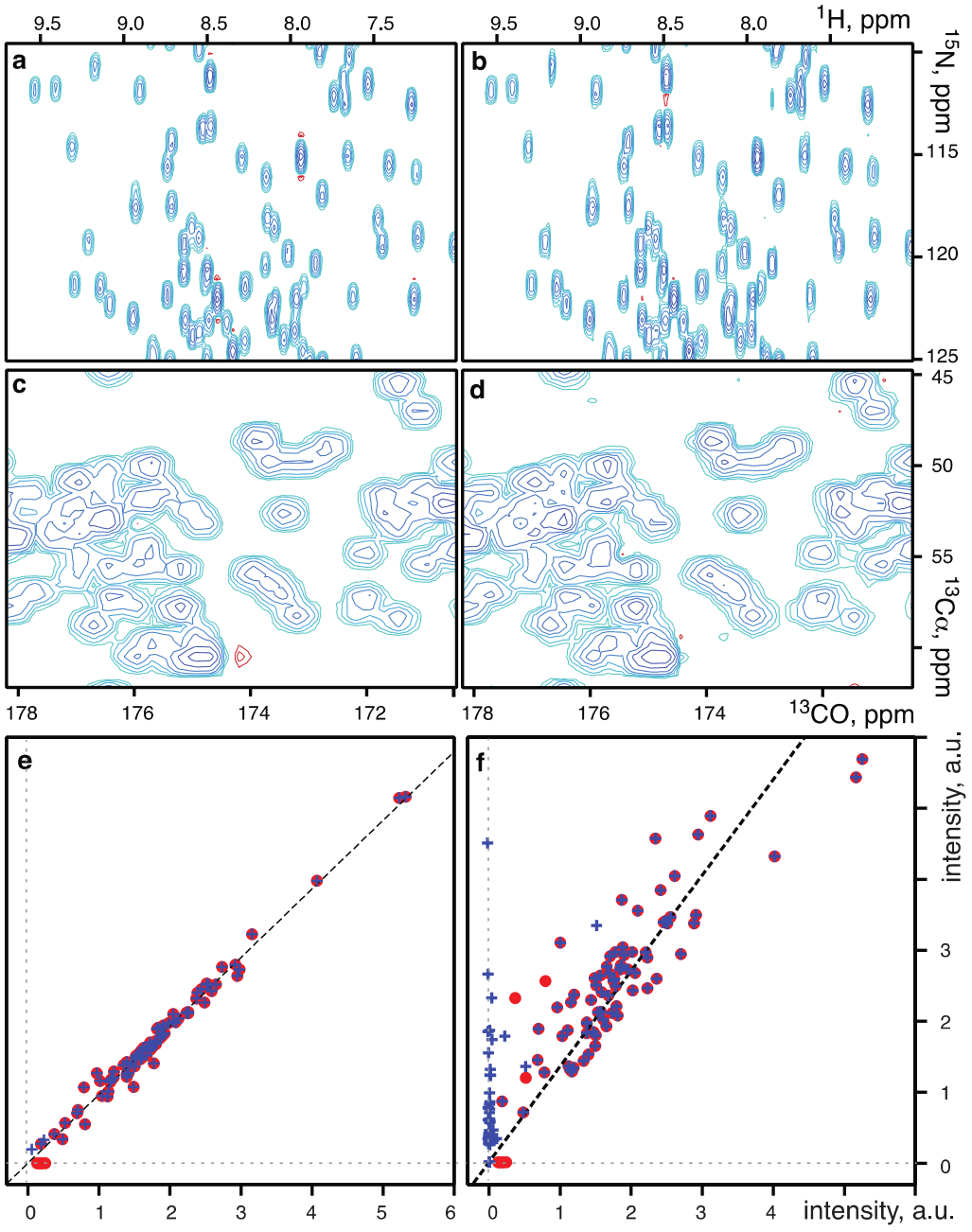


Figure 11-5: **NMR Reconstruction Results:** 4D BEST-HNCOCA spectrum of ubiquitin. Orthogonal ^{1}H/^{15}N (a, b) and (c, d) ^{13}CO/^{13}Cα projections of fully sampled and FFT processed (a, c) and 6.4% NUS processed with the Sparse Fourier Transform (b, d). (e, f) Correlation of peak intensities measured in the full reference spectrum (abscissa) and the Sparse Fourier Transform reconstruction (ordinate) using the matrix inversion (e) and median estimation (f). Dark red circles and blue crosses show intensities measured at the positions of peaks picked in the reference and Sparse Fourier Transform spectra, respectively.

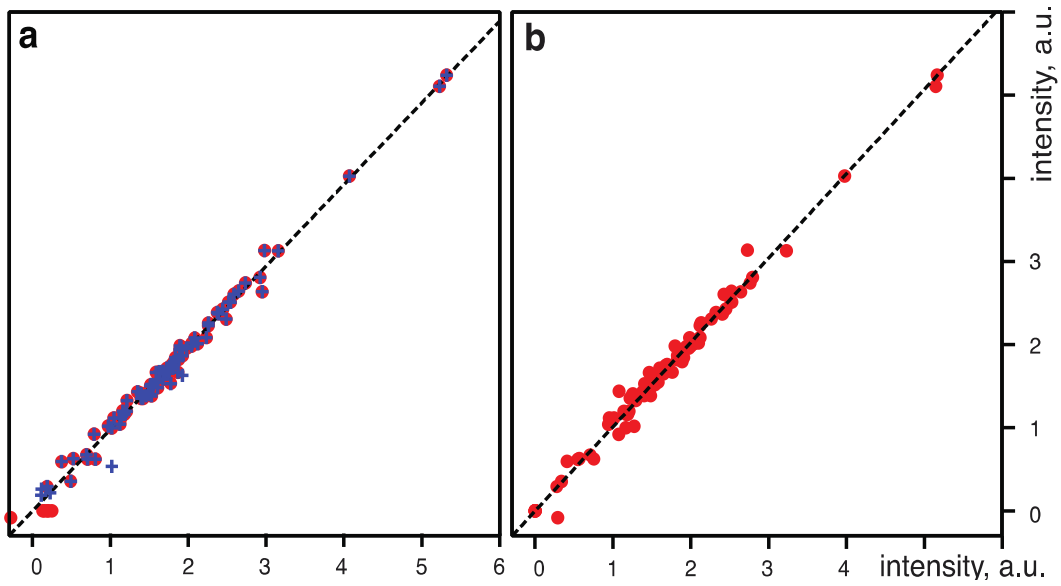


Figure 11-6: **Correlation of Peak Intensities in 4D BEST-HNCOCA Spectrum of Ubiquitin.** (a) the peak intensities were measured in the full reference spectrum (abscissa) and the Sparse Fourier Transform reconstruction (ordinate) using the matrix inversion method obtained using a different set of samples from Figure 11-5. Dark red circles and blue crosses show intensities measured at the positions of peaks picked in the reference and Sparse Fourier Transform spectra, respectively. (b) Correlation of peak intensities measured in two Sparse Fourier Transform reconstructions calculated using different set of randomly selected projections.

ity for the 1.7 mM ubiquitin sample, the signal dynamic range in the reference spectrum reached 1:50, which covers the range typically found in the triple resonance experiments for assignment and approaches the dynamic range in 4D NOESY spectra.

11.5 Discussion

Practical sensitivity in a spectrum can be defined as a level of reliable signal detection, i.e. separation of true signals from noise and spectral artefacts. The weakest detected peak in the reference spectrum has intensity 0.12. Out of total 98, five weakest peaks with intensity up to 0.25 were lost in the Sparse Fourier Transform reconstruction. No peaks above this level were missing. The observed decrease of the sensitivity seems reasonable considering that duration of the Sparse Fourier Transform experiment is only 6.4% of the reference and thus up to four times drop of sensitivity is expected for the Sparse Fourier Transform reconstruction.

In the Sparse Fourier Transform voting algorithm, the frequency detection is limited by the sensitivity in the individual projections, whose measurement time was 1/77 of the total Sparse Fourier Transform experiment time. On the other hand, combined analysis of many projections provides efficient cross-validation of the selected frequencies and allows lowering of the detection threshold in the individual projections as can be seen in Figure 11-6. Similar to the geometric

analysis of projections (GAPRO) algorithm in APSY [80], the Sparse Fourier Transform voting procedure, recovers large part of the sensitivity that is lost due to short measurement time of the projections. It should be noted also that in Sparse Fourier Transform, the purpose of the frequency identification voting algorithm is not to find peaks but to select frequencies, which are worth for the evaluation. The detection limit corresponds to a trade-off between the number of points selected for the evaluation and the requirements for low computational complexity and data storage. Thus, lowering of the detection threshold does not lead to many additional false peaks but only increases the computational complexity and storage. Whatever threshold level is used, the weakest peaks are inevitably lost at the frequency identification step of the Sparse Fourier Transform algorithm and consequently have zero intensities in the Sparse Fourier Transform reconstruction. Thus, the Sparse Fourier Transform, as well as many other NUS processing methods, should be used with caution for spectra with high dynamic range and when detection of the peaks close to the signal-to-nose limit is important, e.g. for NOESY's.

The correlation for the peaks picked in the Sparse Fourier Transform spectrum is shown in Figure 11-5(e), (f) with blue crosses. The peaks were detected by NMRPipe peak-picker with the same noise and detection threshold parameters as for the reference spectrum. This plot is intended for revealing peaks in the Sparse Fourier Transform reconstruction that are not present in the reference spectrum, i.e. false positives. As it is seen in Figure 11-5(e),(f), while the median algorithm for the frequency evaluation resulted in many false peaks, no false peaks were detected when the Sparse Fourier Transform reconstruction was obtained using the matrix inversion method. As expected, the median method also provided less accurate peak intensities. Notably, both methods evaluate intensity of the same set of frequencies, which are defined at the common frequency identification step of the Sparse Fourier Transform algorithms. Thus, the matrix inversion method effectively suppresses the false positive peaks.

Apart from the signals, which were identified by the peak-picker as peaks, the Sparse Fourier Transform spectrum obtained by the matrix inversion method contained a number of signals with intensities lower than the peak detection cut-off. In addition, there were several relatively low intensity (< 0.3) signals, which didn't pass the peak quality checks as can be seen in the example in Figure 11-7. In most cases such signals were represented by only one point in two or more spectral dimensions. The reduced dimensionality data collection used by the Sparse Fourier Transform may be prone to false peak artefacts that are not, in general, the result of a method used to compute spectra, but are intrinsic for this type of data sampling [127], especially in the case of signals with high dynamic range. Thus, it is unlikely that the Sparse Fourier Transform will be able to produce reconstructions for small and medium size spectra that are better than the modern NUS-based techniques, such as CS. On the other hand, the computational and storage efficiency of the Sparse Fourier Transform are well suited for the large spectra, i.e. 4Ds and above, where the full spectral reconstructions and computationally demanding algorithms often fail while methods based on the radial sampling (e.g. APSY) are efficiently used. For example we envisage that Sparse Fourier Transform will be instrumental in high-dimensional spectra of the Intrinsically Disordered Proteins that often exhibit long transverse relaxation times and heavy peak overlap [129].

Typically, the number of cross-peaks does not increase with spectrum dimensionality and resolution. Consequently, the number of non-zero frequencies, which is related to the number of cross-

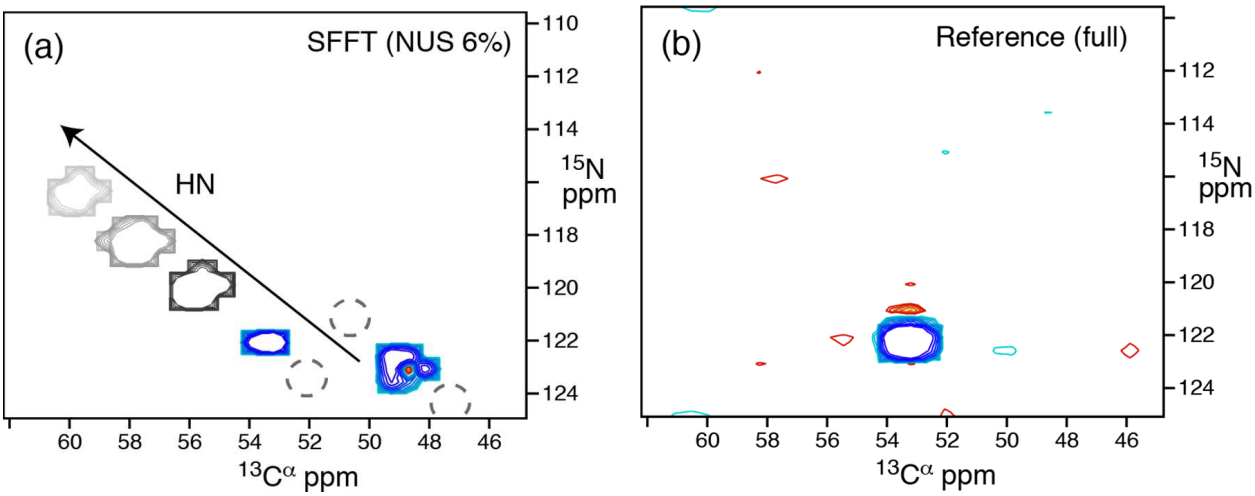


Figure 11-7: **Cα-N Plane from the 4D BEST-HNCOCA Experiment:** (a) the Sparse Fourier Transform reconstruction, (b) the reference spectrum. The planes are plotted at the same contour level. The first contour is drawn at the level of 0.01 at the scale of Figure 11-5(e),(f). In (a), two colored peaks exemplify true (left) and false (right) signals. In (a), the peaks appearance in the preceding and subsequent planes in the directly detected HN dimension is indicated by gray contours. The dashed gray circles indicate absence of the peaks in the adjacent planes. The false peak, which has relatively low intensity (0.2), was not picked by the peak-picker, because it has the distorted line shape and is represented only by a single point in the directly detected dimension. The true peak has the maximum in the second subsequent plane and was picked there in both the reference and Sparse Fourier Transform spectra.

peaks, only moderately increases proportionally to dimensionality and resolution of the spectrum. This makes it possible for the Sparse Fourier Transform to handle very large spectra.

Another good feature of the technique is that the data sampling using the discrete line projections and voting algorithm used by the Sparse Fourier Transform for the identification of non-zero frequencies are fully compatible with the optimization by incremental data collection and analysis [44, 94]. For example we can envisage an approach where an experiment is continued until the list of identified frequencies stabilizes and reaches a plateau. Thus, the number of projections can be adjusted for every experiment.

11.6 Conclusion

From the NMR perspective, the Sparse Fourier Transform for the first time combines the best features of so far distinctly different approaches known as reduced dimensionality and compressed sensing. The former is robust and computationally very efficient, the later provides highest quality spectral reconstructions. In this chapter, we presented the NMR tailored version of the Sparse Fourier Transform algorithm and demonstrated its performance for 4D BEST-HNCOCA spectrum of ubiquitin.

Chapter 12

Conclusion

The widespread use of the Fourier transform coupled with the emergence of big data applications has generated a pressing need for new algorithms that compute the Fourier transform in sublinear time, faster than the FFT algorithm. This thesis addresses this need by developing the Sparse Fourier Transform algorithms and building practical systems that use these algorithms to solve key problems in the areas of wireless networks, mobile systems, computer graphics, medical imaging, biochemistry and digital circuits.

The Sparse Fourier Transform algorithms are a family of sublinear time algorithms that leverage the sparsity of the signals in the frequency domain to compute the Fourier transform faster than the FFT. The thesis presents state-of-the-art algorithms with the lowest runtime complexity known to date. Specifically, these algorithms run in $O(k \log n)$ time for exactly sparse signals and $O(k \log n \log (n/k))$ time for approximately sparse signals. The algorithms are faster than FFT, which has a runtime of $O(n \log n)$, for any sparsity $k = o(n)$. The thesis also presents algorithms with the *optimal* sampling complexity for average case inputs. These algorithms use the minimum number of input samples, *i.e.*, $O(k)$ samples for exactly sparse signals and $O(k \log n)$ samples for approximately sparse signals, and hence reduce acquisition cost.

The thesis also develops software and hardware systems that leverage the Sparse Fourier Transform to address challenges in practical applications. Specifically, in the area of wireless networks, the thesis shows how to use commodity hardware to build a wireless receiver that captures GHz-wide signals that are $6\times$ larger than its digital sampling rate. The thesis also show how to design a GPS receiver that consumes $2\times$ lower power on mobile systems. In the area of computer graphics, the thesis demonstrates that reconstructing light field images using the Sparse Fourier Transform reduces sampling requirements and improves image reconstruction quality. The thesis also shows how the Sparse Fourier Transform can be used to generate clearer MRS images while reducing the time the patient spends in an MRI machine. In the area of biochemistry, the thesis demonstrates that the Sparse Fourier Transform can reduce an NMR experiment time by $16\times$. Finally, the thesis presents a Sparse Fourier Transform chip that delivers the largest Fourier transform chip to date for sparse data while consuming $40\times$ less power than traditional FFT chips.

This thesis lays the foundational grounds of the Sparse Fourier Transform. It develops a theoretical and practical framework which researchers can use and build on to improve the performance of their specific applications.

12.1 Future Directions

Looking forward, the Sparse Fourier Transform can help address further challenges in building practical systems that benefit many more applications, beyond what has been demonstrated in this thesis. Below we highlight some of the future applications of the Sparse Fourier Transform.

- *Discovering the Brain's Connectivity:* Understanding the structure and function of the connections between neurons in the brain is a major ongoing research project. High dimensional MRI tests like 6D diffusion MRI enable discovering the communication channels between different parts of the brain. However, going to higher dimensions requires collecting more data, which translates into the patient spending much longer time in the MRI machine. Similar to MRS, diffusion MRI data is sparse in the frequency domain. Hence, the Sparse Fourier Transform can help enable high dimensional MRI tests using very few sub-samples of the data to reduce the time the patient spends in the machine.

- *Optical Coherence Tomography (OCT):* OCT is an established medical imaging technique that uses optical scattering in biological tissues to provide diagnostic images of the eye, the retina, the coronary arteries, the face, and the finger tips. OCT generates a lot of data which makes it difficult to provide the images to doctors in realtime. However, OCT images are generated in the frequency domain which is typically sparse and hence can benefit from the Sparse Fourier Transform to quickly process the data and generate the medical images in realtime.

- *DNA Sequencing:* The post processing of DNA sequencing data is typically very computational intensive and there are many attempts in computational biology to create faster processing tools. Applications like antibody sequence alignment require finding the right genes that match the DNA sequence and discovering where each gene starts in the sequence. This problem is very similar to the GPS code alignment problem from Chapter 8 and hence it can benefit from the Sparse Fourier Transform to speed up the sequence alignment problem.

- *Radio Astronomy:* The Square Kilometer Array (SKA) is a radio telescope in development in Australia and South Africa. It spreads over an area of one square kilometer providing the highest resolution images ever captured in astronomy. To generate images of the sky, SKA performs a Fourier transform over the sampled data. However, the amount of incoming data will be larger than terabytes per second which is hard to process with today's computational power. The output images are sparse since at any point in time there are few events occurring in the universe. Thus, the Sparse Fourier Transform can help generate these images much faster than FFT which significantly reduces the required computational load.

- *Wireless Localization:* Antenna arrays have become the key component in localizing RF devices in a variety of applications such as indoor navigation and location-based marketing. In these systems, a Fourier transform is computed across the samples from the antennas in order to identify the spatial direction along which a wireless signal arrives. Large antenna arrays provide high resolution in measuring the angle of arrival but are very costly since they require a large number of synchronized RF-receivers. The direction of arrival of wireless signals is

typically sparse and hence we can use the Sparse Fourier Transform to subsample the antennas and reduce the number of receivers needed compute the direction of arrival.

- *Radar Systems:* Radar chirps are signals in which the frequency sweeps over the bandwidth of operation during a given time period. Detecting and acquiring chirps transmitted by an unknown source is a challenging problem in radar. Since the parameters of the chirp are unknown, it is hard to capture the signal. For example, if we do not know the frequency which the chirp is sweeping, the chirp can be anywhere in a multi-GHz spectrum but only sweeping over few MHz of bandwidth. Moreover, if we do not know the sweeping function of chirp (e.g. the slope of linear sweep), we cannot pick the right time window over which we should perform the Fourier transform. Chirps, however, are extremely sparse in frequency domain. Exploiting this sparsity would allow us to design radar acquisition systems with lower computational overhead and the high accuracy.

In addition to the above applications, the Sparse Fourier Transform can be leveraged to speed up processes like feature extraction in speech recognition, optical proximity correction (OPC) in computational lithography, seismic data analysis in oil and gas exploration, analysis of boolean functions, proximity queries in databases, anomaly detection in data centers, as well as many more applications.

Appendix A

Proofs

A.1 Proof of Lemma 3.2.1

Lemma 3.2.1 Let S be the support of the largest k coefficients of $\hat{x}$, and $\hat{x}_{-S}$ contain the rest. Then for any $\epsilon \leq 1$,

$$\Pr_{\sigma,\tau,b}\left[|\hat{x}'_i - \hat{x}_i|^2 \geq \frac{\epsilon}{k}\|\hat{x}_{-S}\|_2^2 + 3\delta^2\|\hat{x}\|_1^2\right] < O\left(\frac{k}{\epsilon B}\right).$$

Proof. By the definitions in Section 2.2,

$$\widehat{x}'_i = \widehat{z}_{h_{\sigma,b}(i)}\omega^{-\tau\sigma i}/\widehat{G}_{o_{\sigma,b}(i)} = \widehat{y}_{\sigma(i-b)-o_{\sigma,b}(i)}\omega^{-\tau\sigma i}/\widehat{G}_{o_{\sigma,b}(i)}.$$

Consider the case that $\hat{x}$ is zero everywhere but at i, so $\mathrm{supp}(\widehat{P_{\sigma,\tau,b}x}) = \{\sigma(i-b)\}$. Then $\hat{y}$ is the convolution of $\widehat{G}$ and $\widehat{P_{\sigma,\tau,b}x}$, and $\widehat{G}$ is symmetric, so

$$\begin{aligned}\hat{y}_{\sigma(i-b)-o_{\sigma,b}(i)} &= (\widehat{G} * \widehat{P_{\sigma,\tau,b}x})_{\sigma(i-b)-o_{\sigma,b}(i)} = \widehat{G}_{-o_{\sigma,b}(i)}\widehat{P_{\sigma,\tau,b}x}_{\sigma(i-b)} \\ &= \widehat{G}_{o_{\sigma,b}(i)}\hat{x}_i\omega^{\tau\sigma i}\end{aligned}$$

which shows that $\hat{x}'_i = \hat{x}_i$ in this case. But $\hat{x}'_i - \hat{x}_i$ is linear in $\hat{x}$, so in general we can assume $\hat{x}_i = 0$ and bound $|\hat{x}'_i|$. Since $|\hat{x}'_i| = |\hat{z}_{h_{\sigma,b}(i)}/\widehat{G}_{o_{\sigma,b}(i)}| < |\hat{z}_{h_{\sigma,b}(i)}|/(1-\delta) = |\hat{y}_{\sigma(i-b)-o_{\sigma,b}(i)}|/(1-\delta)$, it is sufficient to bound $|\hat{y}_{\sigma(i-b)-o_\sigma(i)}|$.

Define $T = \{j \in [n] \mid \sigma(i-b-j) \in [-2n/B, 2n/B]\}$. We have that

$$\begin{aligned}
|\hat{y}_{\sigma(i-b)-o_\sigma(i)}|^2 &= \left|\sum_{t=0}^{n-1} (\widehat{P_{\sigma,\tau,b} x})_t \widehat{G}_{\sigma(i-b)-t-o_{\sigma,b}(i)}\right|^2 \\
&= \left|\sum_{j=0}^{n-1} (\widehat{P_{\sigma,\tau,b} x})_{\sigma j} \widehat{G}_{\sigma(i-b-j)-o_{\sigma,b}(i)}\right|^2 \\
&\leq 2\left|\sum_{j\in T} (\widehat{P_{\sigma,\tau,b} x})_{\sigma j} \widehat{G}_{\sigma(i-b-j)-o_{\sigma,b}(i)}\right|^2 + 2\left|\sum_{j\notin T} (\widehat{P_{\sigma,\tau,b} x})_{\sigma j} \widehat{G}_{\sigma(i-b-j)-o_{\sigma,b}(i)}\right|^2 \\
&\leq 2(1+\delta)^2\left|\sum_{j\in T} (\widehat{P_{\sigma,\tau,b} x})_{\sigma j}\right|^2 + 2\delta^2\left|\sum_{j\notin T} (\widehat{P_{\sigma,\tau,b} x})_{\sigma j}\right|^2
\end{aligned}$$

In the last step, the left term follows from $|\widehat{G_a}| \leq 1+\delta$ for all a. The right term is because for $j \notin T$, $|\sigma(i-b-j) - o_{\sigma,b}(i)| \geq |\sigma(i-b-j)| - |o_{\sigma,b}(i)| > 2n/B - n/(2B) > n/B$, and $|\widehat{G_a}| \leq \delta$ for $|a| > n/B$. By Claim 2.2.6, this becomes:

$$|\hat{y}_{\sigma(i-b)-o_{\sigma,b}(i)}|^2 \leq 2(1+\delta)^2\left|\sum_{j\in T} \hat{x}_j \omega^{\tau\sigma j}\right|^2 + 2\delta^2\|\hat{x}\|_1^2.$$

Define $V = \left|\sum_{j\in T} \hat{x}_j \omega^{\tau\sigma j}\right|^2$. As a choice over τ, V is the energy of a random Fourier coefficient of the vector $\hat{x}_T$ so we can bound the expectation over τ:

$$\mathop{\mathbb{E}}_{\tau}[V] = \|\hat{x}_T\|_2^2.$$

Now, for each coordinate $j \neq i-b$, $\Pr_{\sigma,b}[j \in T] \leq 8/B$ by Lemma 2.2.7. Thus $\Pr_{\sigma,b}[S \cap T \neq \emptyset] \leq 8k/B$, so with probability $1 - 8k/B$ over σ and b we have

$$\|\hat{x}_T\|_2^2 = \|\hat{x}_{T\setminus S}\|_2^2.$$

Let E be the event that this occurs, so E is 0 with probability $8k/B$ and 1 otherwise. We have

$$\mathop{\mathbb{E}}_{\sigma,b,\tau}[EV] = \mathop{\mathbb{E}}_{\sigma,b}\left[E\|\hat{x}_T\|_2^2\right] = \mathop{\mathbb{E}}_{\sigma,b}\left[E\|\hat{x}_{T\setminus S}\|_2^2\right] \leq \mathop{\mathbb{E}}_{\sigma,b}\left[\|\hat{x}_{T\setminus S}\|_2^2\right].$$

Furthermore, we know

$$\begin{aligned}
\mathop{\mathbb{E}}_{\sigma,b,\tau}[EV] &\leq \mathop{\mathbb{E}}_{\sigma,b}\left[\|\hat{x}_{T\setminus S}\|_2^2\right] = \sum_{j\notin S} |\hat{x}_j|^2 \Pr_{\sigma,b}[\sigma(i-b-j) \in [-2n/B, 2n/B]] \\
&\leq \frac{8}{B}\|\hat{x}_{-S}\|_2^2
\end{aligned}$$

by Lemma 2.2.7. Therefore, by Markov's inequality and a union bound, we have for any $C > 0$

that

$$\Pr_{\sigma,b,\tau}\left[V > 8\frac{C}{B}\|\hat{x}_{-S}\|_2^2\right] \leq \Pr_{\sigma,b,\tau}\left[E = 0 \cup EV > C\,\mathop{\mathbb{E}}_{\sigma,b,\tau}[EV]\right] \leq 8\frac{k}{B} + 1/C.$$

Hence

$$\Pr_{\sigma,b,\tau}\left[|\hat{y}_{\sigma(i-b)-o_{\sigma,b}(i)}|^2 \geq 16\frac{C}{B}(1+\delta)^2\|\hat{x}_{-S}\|_2^2 + 2\delta^2\|\hat{x}\|_1^2\right] < 8\frac{k}{B} + 1/C.$$

Replacing C with $\epsilon B/(32k)$ and using $|\hat{x}'_i - \hat{x}_i| < |\hat{y}_{\sigma(i-b)-o_{\sigma,b}(i)}|/(1-\delta)$ gives

$$\Pr_{\sigma,b,\tau}\left[|\hat{x}'_i - \hat{x}_i|^2 \geq \frac{\epsilon}{2k}\left(\frac{1+\delta}{1-\delta}\right)^2\|\hat{x}_{-S}\|_2^2 + \frac{2}{1-\delta}\delta^2\|\hat{x}\|_1^2\right] < (8 + 32/\epsilon)\frac{k}{B}.$$

which implies the result. □

A.2 Proof of Lemma 3.2.2

Lemma 3.2.2 Define $E = \sqrt{\frac{\epsilon}{k}\|\hat{x}_{-S}\|_2^2 + 3\delta^2\|\hat{x}\|_1^2}$ to be the error tolerated in Lemma 3.2.1. Then for any $i \in [n]$ with $|\hat{x}_i| \geq 4E$,

$$\Pr[i \notin I] \leq O\left(\frac{k}{\epsilon B} + \frac{1}{\epsilon d}\right)$$

Proof. With probability at least $1 - O(\frac{k}{\epsilon B})$, $|\hat{x}'_i| \geq |\hat{x}_i| - E \geq 3E$. In this case $|\hat{z}_{h_{\sigma,b}(i)}| \geq 3(1-\delta)E$. Thus it is sufficient to show that, with probability at least $1 - O(\frac{1}{\epsilon d})$, there are at most dk locations $j \in [B]$ with $|\hat{z}_j| \geq 3(1-\delta)E$. Since each $\hat{z}_j$ corresponds to n/B locations in $[n]$, we will show that with probability at least $1 - O(\frac{1}{\epsilon d})$, there are at most dkn/B locations $j \in [n]$ with $|\hat{x}'_j| \geq 3(1-\delta)^2E$.

Let $U = \{j \in [n] \mid |\hat{x}_j| \geq E\}$, and $V = \{j \in [n] \mid |\hat{x}'_j - \hat{x}_j| \geq E\}$. Therefore $\{j \in [n] \mid |\hat{x}'_j| \geq 2E\} \subseteq U \cup V$. Since $2 \leq 3(1-\delta)^2$, we have

$$\left|\{j \mid |\hat{x}'_j| \geq 3(1-\delta)^2E\}\right| \leq |U \cup V|.$$

We also know

$$|U| \leq k + \frac{\|x_{-S}\|_2^2}{E^2} \leq k(1 + 1/\epsilon)$$

and by Lemma 3.2.1,

$$\mathbb{E}[|V|] \leq O(\frac{kn}{\epsilon B}).$$

Thus by Markov's inequality $\Pr[|V| \geq \frac{1}{2}dk\frac{n}{B}] \leq O(\frac{1}{\epsilon d})$, or

$$\Pr[|U \cup V| \geq \frac{1}{2}dk\frac{n}{B} + k(1+1/\epsilon)] \leq O(\frac{1}{\epsilon d}).$$

Since the RHS is only meaningful for $d = \Omega(1/\epsilon)$ we can assume $\frac{1}{2}dk\frac{n}{B} > k(1+1/\epsilon)$. Therefore

$$\Pr[|U \cup V| \geq dk\frac{n}{B}] \leq O\left(\frac{1}{\epsilon d}\right).$$

and thus

$$\Pr[|\{j \in [B] \mid |\hat{z}_j| \geq 3(1-\delta)E\}| > kd] \leq O(\frac{1}{\epsilon d}).$$

Hence a union bound over this and the probability that $|\hat{x}'_i - \hat{x}_i| < E$ gives

$$\Pr[i \notin I] \leq O\left(\frac{k}{\epsilon B} + \frac{1}{\epsilon d}\right)$$

as desired. □

A.3 Proof of Lemma 4.2.4

Lemma 4.2.4 Suppose B divides n. The output $\widehat{u}$ of HASHTOBINS satisfies

$$\widehat{u}_j = \sum_{h_{\sigma,b}(i)=j} \widehat{(x-z)}_i \widehat{(G'_{B,\delta,\alpha})}_{-o_{\sigma,b}(i)} \omega^{a\sigma i} \pm \delta\|\widehat{x}\|_1.$$

Let $\zeta = |\{i \in \mathrm{supp}(\widehat{z}) \mid E_{\mathit{off}}(i)\}|$. The running time of HASHTOBINS is $O(\frac{B}{\alpha}\log(n/\delta) + \|\widehat{z}\|_0 + \zeta\log(n/\delta))$.

Proof. Define the flat window functions $G = G_{B,\delta,\alpha}$ and $\widehat{G'} = \widehat{G'}_{B,\delta,\alpha}$. We have

$$\widehat{y} = \widehat{G \cdot P_{\sigma,a,b}x} = \widehat{G} * \widehat{P_{\sigma,a,b}x}$$
$$\widehat{y'} = \widehat{G'} * \widehat{P_{\sigma,a,b}(x-z)} + (\widehat{G} - \widehat{G'}) * \widehat{P_{\sigma,a,b}x}$$

By Claim 2.2.6, the coordinates of $\widehat{P_{\sigma,a,b}x}$ and $\widehat{x}$ have the same magnitudes, just different ordering and phase. Therefore

$$\|(\widehat{G} - \widehat{G'}) * \widehat{P_{\sigma,a,b}x}\|_\infty \leq \|\widehat{G} - \widehat{G'}\|_\infty \|\widehat{P_{\sigma,a,b}x}\|_1 \leq \delta\|\widehat{x}\|_1$$

and hence

$$\begin{aligned}\widehat{u}_j = \widehat{y'}_{jn/B} &= \sum_{|l|<n/(2B)} \widehat{G'}_{-l}\widehat{(P_{\sigma,a,b}(x-z))}_{jn/B+l} \pm \delta\|\widehat{x}\|_1 \\ &= \sum_{\left|\pi_{\sigma,b}(i)-jn/B\right|<n/(2B)} \widehat{G'}_{jn/B-\pi_{\sigma,b}(i)}\widehat{(P_{\sigma,a,b}(x-z))}_{\pi_{\sigma,b}(i)} \pm \delta\|\widehat{x}\|_1 \\ &= \sum_{h_{\sigma,b}(i)=j} \widehat{G'}_{-o_{\sigma,b}(i)}\widehat{(x-z)}_i\omega^{a\sigma i} \pm \delta\|\widehat{x}\|_1\end{aligned}$$

as desired.

We can compute HASHTOBINS via the following method:

1. Compute y with $\|y\|_0 = O(\frac{B}{\alpha}\log(n/\delta))$ in $O(\frac{B}{\alpha}\log(n/\delta))$ time.
2. Compute $v \in \mathbb{C}^B$ given by $v_i = \sum_j y_{i+jB}$.
3. Because B divides n, by the definition of the Fourier transform (see also Claim 2.2.8 from Chapter 3) we have $\widehat{y}_{jn/B} = \widehat{v}_j$ for all j. Hence we can compute it with a B-dimensional FFT in $O(B\log B)$ time.
4. For each coordinate $i \in \mathrm{supp}(\widehat{z})$, decrease $\widehat{y}_{\frac{n}{B}h_{\sigma,b}(i)}$ by $\widehat{G'}_{-o_{\sigma,b}(i)}\widehat{z}_i\omega^{a\sigma i}$. This takes $O(\|\widehat{z}\|_0 + \zeta\log(n/\delta))$ time, since computing $\widehat{G'}_{-o_{\sigma,b}(i)}$ takes $O(\log(n/\delta))$ time if $E_{\mathit{off}}(i)$ holds and $O(1)$ otherwise.

□

A.4 Proof of Lemma 4.2.5

Lemma 4.2.5 Consider any $i \in S$ such that neither $E_{\mathit{coll}}(i)$ nor $E_{\mathit{off}}(i)$ holds. Let $j = h_{\sigma,b}(i)$. Then

$$\mathrm{round}(\phi(\widehat{u}_j/\widehat{u}'_j))\frac{n}{2\pi}) = \sigma i \pmod n,$$
$$\mathrm{round}(\widehat{u}_j) = \widehat{x}_i - \widehat{z}_i,$$

and $j \in J$.

Proof. We know that $\|\widehat{x}\|_1 \le k\|\widehat{x}\|_\infty \le kL < nL$. Then by Lemma 4.2.4 and $E_{\mathit{coll}}(i)$ not holding,

$$\widehat{u}_j = \widehat{(x-z)}_i\widehat{G'}_{-o_{\sigma,b}(i)} \pm \delta nL.$$

Because $E_{\mathit{off}}(i)$ does not hold, $\widehat{G'}_{-o_{\sigma,b}(i)} = 1$, so

$$\widehat{u}_j = \widehat{(x-z)}_i \pm \delta nL. \tag{A.1}$$

Similarly,

$$\widehat{u}'_j = \widehat{(x-z)}_i\omega^{\sigma i} \pm \delta nL$$

Then because $\delta nL < 1 \leq \left|\widehat{(x-z)}_i\right|$, the phase is

$$\phi(\widehat{u}_j) = 0 \pm \sin^{-1}(\delta nL) = 0 \pm 2\delta nL$$

and $\phi(\widehat{u}'_j) = -\sigma i \frac{2\pi}{n} \pm 2\delta nL$. Thus $\phi(\widehat{u}_j/\widehat{u}'_j) = \sigma i \frac{2\pi}{n} \pm 4\delta nL = \sigma i \frac{2\pi}{n} \pm 1/n$ by the choice of δ. Therefore

$$\text{round}(\phi(\widehat{u}_j/\widehat{u}'_j)\frac{n}{2\pi}) = \sigma i \pmod{n}.$$

Also, by Equation (A.1), $\text{round}(\widehat{u}_j) = \widehat{x}_i - \widehat{z}_i$. Finally, $|\text{round}(\widehat{u}_j)| = |\widehat{x}_i - \widehat{z}_i| \geq 1$, so $|\widehat{u}_j| \geq 1/2$. Thus $j \in J$. □

A.5 Proof of Lemma 4.3.2

Lemma 4.3.2 Let $a \in [n]$ uniformly at random, B divide n, and the other parameters be arbitrary in

$$\widehat{u} = \text{HASHTOBINS}(x, \widehat{z}, P_{\sigma,a,b}, B, \delta, \alpha).$$

Then for any $i \in [n]$ with $j = h_{\sigma,b}(i)$ and none of $E_{coll}(i)$, $E_{off}(i)$, or $E_{noise}(i)$ holding,

$$\mathbb{E}[\left|\widehat{u}_j - \widehat{x'}_i \omega^{a\sigma i}\right|^2] \leq 2\frac{\rho^2}{\alpha B}$$

Proof. Let $\widehat{G'} = \widehat{G'}_{B,\delta,\alpha}$. Let $T = h^{-1}_{\sigma,b}(j) \setminus \{i\}$. We have that $T \cap S = \{\}$ and $\widehat{G'}_{-o_{\sigma,b}(i)} = 1$. By Lemma 4.2.4,

$$\widehat{u}_j - \widehat{x'}_i \omega^{a\sigma i} = \sum_{i' \in T} \widehat{G'}_{-o_\sigma(i')} \widehat{x'}_{i'} \omega^{a\sigma i'} \pm \delta \|\widehat{x}\|_1.$$

Because the $\sigma i'$ are distinct for $i' \in T$, we have by Parseval's theorem

$$\mathop{\mathbb{E}}_{a}\left|\sum_{i' \in T} \widehat{G'}_{-o_\sigma(i')} \widehat{x'}_{i'} \omega^{a\sigma i'}\right|^2 = \sum_{i' \in T} (\widehat{G'}_{-o_\sigma(i')} \widehat{x'}_{i'})^2 \leq \|\widehat{x'_T}\|_2^2$$

Since $|X + Y|^2 \leq 2|X|^2 + 2|Y|^2$ for any X, Y, we get

$$\begin{aligned}
\mathop{\mathbb{E}}_{a}[\left|\widehat{u}_j - \widehat{x'}_i \omega^{a\sigma i}\right|^2] &\leq 2\|x'_T\|_2^2 + 2\delta^2 \|\widehat{x}\|_1^2 \\
&\leq 2\,\text{Err}^2(\widehat{x'}, k)/(\alpha B) + 2\delta^2 \|\widehat{x}\|_1^2 \\
&\leq 2\rho^2/(\alpha B).
\end{aligned}$$

□

A.6 Proof of Lemma 4.3.4

Lemma 4.3.4 Let $i \in S$. Suppose none of $E_{coll}(i)$, $E_{off}(i)$, and $E_{noise}(i)$ hold, and let $j = h_{\sigma,b}(i)$. Consider any run of LOCATEINNER with $\pi_{\sigma,b}(i) \in [l_j, l_j + w]$. Let $f > 0$ be a parameter such that

$$B = \frac{Ck}{\alpha f \epsilon}.$$

for C larger than some fixed constant. Then $\pi_{\sigma,b}(i) \in [l'_j, l'_j + 4w/t]$ with probability at least $1 - tf^{\Omega(R_{loc})}$,

Proof. Let $\tau = \pi_{\sigma,b}(i) \equiv \sigma(i - b) \pmod n$, and for any $j \in [n]$ define

$$\theta^*_j = \frac{2\pi}{n}(j + \sigma b) \pmod{2\pi}$$

so $\theta^*_\tau = \frac{2\pi}{n}\sigma i$. Let $g = \Theta(f^{1/3})$, and $C' = \frac{B\alpha\epsilon}{k} = \Theta(1/g^3)$.

To get the result, we divide $[l_j, l_j + w]$ into t "regions", $Q_q = [l_j + \frac{q-1}{t}w, l_j + \frac{q}{t}w]$ for $q \in [t]$. We will first show that in each round r, c_j is close to $\beta\theta^*_\tau$ with $1 - g$ probability. This will imply that Q_q gets a "vote," meaning $v_{j,q}$ increases, with $1 - g$ probability for the q' with $\tau \in Q_{q'}$. It will also imply that $v_{j,q}$ increases with only g probability when $|q - q'| > 3$. Then R_{loc} rounds will suffice to separate the two with $1 - f^{-\Omega(R_{loc})}$ probability. We get that with $1 - tf^{-\Omega(R_{loc})}$ probability, the recovered Q^* has $|q - q'| \le 3$ for all $q \in Q^*$. If we take the minimum $q \in Q^*$ and the next three subregions, we find τ to within 4 regions, or $4w/t$ locations, as desired.

In any round r, define $\widehat{u} = \widehat{u}^{(r)}$ and $a = a_r$. We have by Lemma 4.3.2 and that $i \in S$ that

$$\begin{aligned}\mathbb{E}[\left|\widehat{u}_j - \omega^{a\sigma i}\widehat{x'}_i\right|^2] &\le 2\frac{\rho^2}{\alpha B} = \frac{2k}{B\alpha\epsilon}\mu^2 \\ &= \frac{2}{C'}\mu^2 \le \frac{2}{C'}|\widehat{x'}_i|^2.\end{aligned}$$

Note that $\phi(\omega^{a\sigma i}) = -a\theta^*_\tau$. Thus for any $p > 0$, with probability $1 - p$ we have

$$\left|\widehat{u}_j - \omega^{a\sigma i}\widehat{x'}_i\right| \le \sqrt{\frac{2}{C'p}}\left|\widehat{x'}_i\right|$$

$$\|\phi(\widehat{u}_j) - (\phi(\widehat{x'}_i) - a\theta^*_\tau)\|_\bigcirc \le \sin^{-1}(\sqrt{\frac{2}{C'p}})$$

where $\|x - y\|_\bigcirc = \min_{\gamma\in\mathbb{Z}}|x - y + 2\pi\gamma|$ denotes the "circular distance" between x and y. The analogous fact holds for $\phi(\widehat{u'}_j)$ relative to $\phi(\widehat{x'}_i) - (a + \beta)\theta^*_\tau$. Therefore with at least $1 - 2p$

probability,

$$\begin{aligned}\|c_j - \beta\theta^*_\tau\|_\bigcirc &= \|\phi(\widehat{u}_j) - \phi(\widehat{u'}_j) - \beta\theta^*_\tau\|_\bigcirc \\ &= \left\| \left(\phi(\widehat{u}_j) - (\phi(\widehat{x'}_i) - a\theta^*_\tau)\right) - \left(\phi(\widehat{u'}_j) - (\phi(\widehat{x'}_i) - (a+\beta)\theta^*_\tau)\right) \right\|_\bigcirc \\ &\leq \|\phi(\widehat{u}_j) - (\phi(\widehat{x'}_i) - a\theta^*_\tau)\|_\bigcirc + \|\phi(\widehat{u'}_j) - (\phi(\widehat{x'}_i) - (a+\beta)\theta^*_\tau)\|_\bigcirc \\ &\leq 2\sin^{-1}(\sqrt{\frac{2}{C'p}})\end{aligned}$$

by the triangle inequality. Thus for any $s = \Theta(g)$ and $p = \Theta(g)$, we can set $C' = \frac{2}{p\sin^2(s\pi/4)} = \Theta(1/g^3)$ so that

$$\|c_j - \beta\theta^*_\tau\|_\bigcirc < s\pi/2 \tag{A.2}$$

with probability at least $1 - 2p$.

Equation (A.2) shows that c_j is a good estimate for i with good probability. We will now show that this means the appropriate "region" $Q_{q'}$ gets a "vote" with "large" probability.

For the q' with $\tau \in [l_j + \frac{q'-1}{t}w, l_j + \frac{q'}{t}w]$, we have that $m_{j,q'} = l_j + \frac{q'-1/2}{t}w$ satisfies

$$|\tau - m_{j,q'}| \leq \frac{w}{2t}$$

so

$$|\theta^*_\tau - \theta_{j,q'}| \leq \frac{2\pi}{n}\frac{w}{2t}.$$

Hence by Equation (A.2), the triangle inequality, and the choice of $B \leq \frac{snt}{2w}$,

$$\begin{aligned}\|c_j - \beta\theta_{j,q'}\|_\bigcirc &\leq \|c_j - \beta\theta^*_\tau\|_\bigcirc + \|\beta\theta^*_\tau - \beta\theta_{j,q'}\|_\bigcirc \\ &< \frac{s\pi}{2} + \frac{\beta\pi w}{nt} \\ &\leq \frac{s\pi}{2} + \frac{s\pi}{2} \\ &= s\pi.\end{aligned}$$

Thus, $v_{j,q'}$ will increase in each round with probability at least $1 - 2p$.

Now, consider q with $|q - q'| > 3$. Then $|\tau - m_{j,q}| \geq \frac{7w}{2t}$, and (from the definition of $\beta > \frac{snt}{4w}$) we have

$$\beta\,|\tau - m_{j,q}| \geq \frac{7sn}{8} > \frac{3sn}{4}. \tag{A.3}$$

We now consider two cases. First, suppose that $|\tau - m_{j,q}| \leq \frac{w}{st}$. In this case, from the definition of β it follows that

$$\beta\,|\tau - m_{j,q}| \leq n/2.$$

Together with Equation (A.3) this implies

$$\Pr[\beta(\tau - m_{j,q}) \bmod n \in [-3sn/4, 3sn/4]] = 0.$$

On the other hand, suppose that $|\tau - m_{j,q}| > \frac{w}{st}$. In this case, we use Lemma 4.3.3 with parameters $l = 3sn/2$, $m = \frac{snt}{2w}$, $t = \frac{snt}{4w}$, $i = (\tau - m_{j,q})$ and $n = n$, to conclude that

$$\begin{aligned}
\Pr[\beta(\tau - m_{j,q}) \bmod n \in [-3sn/4, 3sn/4]] &\leq \frac{4w}{snt} + 2\frac{|\tau - m_{j,q}|}{n} + 3s + \frac{3sn}{2}\frac{st}{w}\frac{4w}{snt} \\
&\leq \frac{4w}{snt} + \frac{2w}{n} + 9s \\
&< \frac{6}{sB} + 9s \\
&< 10s
\end{aligned}$$

where we used that $|i| \leq w \leq n/B$, the assumption $\frac{w}{st} < |i|$, $t \geq 1$, $s < 1$, and that $s^2 > 6/B$ (because $s = \Theta(g)$ and $B = \omega(1/g^3)$).

Thus in either case, with probability at least $1 - 10s$ we have

$$\|\beta\theta_{j,q} - \beta\theta^*_\tau\|_\bigcirc = \|\frac{2\pi\beta(m_{j,q} - \tau)}{n}\|_\bigcirc > \frac{2\pi}{n}\frac{3sn}{4} = \frac{3}{2}s\pi$$

for any q with $|q - q'| > 3$. Therefore we have

$$\|c_j - \beta\theta_{j,q}\|_\bigcirc \geq \|\beta\theta_{j,q} - \beta\theta^*_\tau\|_\bigcirc - \|c_j - \beta\theta^*_\tau\|_\bigcirc > s\pi$$

with probability at least $1 - 10s - 2p$, and $v_{j,q}$ is not incremented.

To summarize: in each round, $v_{j,q'}$ is incremented with probability at least $1 - 2p$ and $v_{j,q}$ is incremented with probability at most $10s + 2p$ for $|q - q'| > 3$. The probabilities corresponding to different rounds are independent.

Set $s = g/20$ and $p = g/4$. Then $v_{j,q'}$ is incremented with probability at least $1 - g$ and $v_{j,q}$ is incremented with probability less than g. Then after R_{loc} rounds, if $|q - q'| > 3$,

$$\Pr[v_{j,q} > R_{loc}/2] \leq \binom{R_{loc}}{R_{loc}/2} g^{R_{loc}/2} \leq (4g)^{R_{loc}/2} = f^{\Omega(R_{loc})}$$

for $g = f^{1/3}/4$. Similarly,

$$\Pr[v_{j,q'} < R_{loc}/2] \leq f^{\Omega(R_{loc})}.$$

Hence with probability at least $1 - tf^{\Omega(R_{loc})}$ we have $q' \in Q^*$ and $|q - q'| \leq 3$ for all $q \in Q^*$. But then $\tau - l'_j \in [0, 4w/t]$ as desired.

Because $\mathbb{E}[|\{i \in \mathrm{supp}(\widehat{z}) \mid E_{off}(i)\}|] = \alpha\|\widehat{z}\|_0$, the expected running time is $O(R_{loc}Bt + R_{loc}\frac{B}{\alpha}\log(n/\delta) + R_{loc}\|\widehat{z}\|_0(1 + \alpha\log(n/\delta)))$. □

A.7 Proof of Lemma 4.3.5

Lemma 4.3.5 Suppose $B = \frac{Ck}{\alpha^2\epsilon}$ for C larger than some fixed constant. The procedure LOCATESIGNAL returns a set L of size $|L| \leq B$ such that for any $i \in S$, $\Pr[i \in L] \geq 1 - O(\alpha)$. Moreover the procedure runs in expected time

$$O((\frac{B}{\alpha}\log(n/\delta) + \|\widehat{z}\|_0(1 + \alpha\log(n/\delta)))\log(n/B)).$$

Proof. Consider any $i \in S$ such that none of $E_{coll}(i), E_{off}(i)$, and $E_{noise}(i)$ hold, as happens with probability $1 - O(\alpha)$.

Set $t = \log n, t' = t/4$ and $R_{loc} = O(\log_{1/\alpha}(t/\alpha))$. Let $w_0 = n/B$ and $w_D = w_0/(t')^{D-1}$, so $w_{D_{max}+1} < 1$ for $D_{max} = \log_{t'}(w_0 + 1) < t$. In each round D, Lemma 4.3.4 implies that if $\pi_{\sigma,b}(i) \in [l_j^{(D)}, l_j^{(D)} + w_D]$ then $\pi_{\sigma,b}(i) \in [l_j^{(D+1)}, l_j^{(D+1)} + w_{D+1}]$ with probability at least $1 - \alpha^{\Omega(R_{loc})} = 1 - \alpha/t$. By a union bound, with probability at least $1 - \alpha$ we have $\pi_{\sigma,b}(i) \in [l_j^{(D_{max}+1)}, l_j^{(D_{max}+1)} + w_{D_{max}+1}] = \{l_j^{(D_{max}+1)}\}$. Thus $i = \pi_{\sigma,b}^{-1}(l_j^{(D_{max}+1)}) \in L$.

Since $R_{loc}D_{max} = O(\log_{1/\alpha}(t/\alpha)\log_t(n/B)) = O(\log(n/B))$, the running time is

$$\begin{aligned} &O(D_{max}(R_{loc}\frac{B}{\alpha}\log(n/\delta) + R_{loc}\|\widehat{z}\|_0(1 + \alpha\log(n/\delta)))) \\ &= O((\frac{B}{\alpha}\log(n/\delta) + \|\widehat{z}\|_0(1 + \alpha\log(n/\delta)))\log(n/B)). \end{aligned}$$

□

A.8 Proof of Lemma 4.3.6

Lemma 4.3.6 For any $i \in L$,

$$\Pr[\left|\widehat{w}_i - \widehat{x'}_i\right|^2 > \mu^2] < e^{-\Omega(R_{est})}$$

if $B > \frac{Ck}{\alpha\epsilon}$ for some constant C.

Proof. Define $e_r = \widehat{u}_j^{(r)}\omega^{-a_r\sigma i} - \widehat{x'}_i$ in each round r. Suppose none of $E_{coll}^{(r)}(i), E_{off}^{(r)}(i)$, and $E_{noise}^{(r)}(i)$ hold, as happens with probability $1 - O(\alpha)$. Then by Lemma 4.3.2,

$$\mathop{\mathbb{E}}_{a_r}[|e_r|^2] \leq 2\frac{\rho^2}{\alpha B} = \frac{2k}{\alpha\epsilon B}\mu^2 < \frac{2}{C}\mu^2$$

Hence with $3/4 - O(\alpha) > 5/8$ probability in total,

$$|e_r|^2 < \frac{8}{C}\mu^2 < \mu^2/2$$

for sufficiently large C. Then with probability at least $1 - e^{-\Omega(R_{est})}$, both of the following occur:

$$\left|\operatorname*{median}_r \text{real}(e_r)\right|^2 < \mu^2/2$$
$$\left|\operatorname*{median}_r \text{imag}(e_r)\right|^2 < \mu^2/2.$$

If this is the case, then $\left|\text{median}_r\ e_r\right|^2 < \mu^2$. Since $\widehat{w}_i = \widehat{x'}_i + \text{median}\ e_r$, the result follows. □

A.9 Proof of Lemma 4.3.7

Lemma 4.3.7 Let $R_{est} \geq C \log \frac{B}{\gamma fk}$ for some constant C and parameters $\gamma, f > 0$. Then if ESTIMATEVALUES is run with input $k' = 3k$, it returns $\widehat{w_J}$ for $|J| = 3k$ satisfying

$$\text{Err}^2(\widehat{x'_L} - \widehat{w_J}, fk) \leq \text{Err}^2(\widehat{x'_L}, k) + O(k\mu^2)$$

with probability at least $1 - \gamma$.

Proof. By Lemma 4.3.6, each index $i \in L$ has

$$\Pr[\left|\widehat{w}_i - \widehat{x'}_i\right|^2 > \mu^2] < \frac{\gamma fk}{B}.$$

Let $U = \{i \in L \mid \left|\widehat{w}_i - \widehat{x'}_i\right|^2 > \mu^2\}$. With probability $1-\gamma$, $|U| \leq fk$; assume this happens. Then

$$\|(\widehat{x'} - \widehat{w})_{L\setminus U}\|_\infty^2 \leq \mu^2. \tag{A.4}$$

Let T contain the top $2k$ coordinates of $\widehat{w}_{L\setminus U}$. By the analysis of Count-Sketch (most specifically, Theorem 3.1 of [146]), the ℓ_∞ guarantee in Equation (A.4) means that

$$\|\widehat{x'}_{L\setminus U} - \widehat{w}_T\|_2^2 \leq \text{Err}^2(\widehat{x'}_{L\setminus U}, k) + 3k\mu^2. \tag{A.5}$$

Because J is the top $3k > (2+f)k$ coordinates of $\widehat{w_L}$, $T \subset J$. Let $J' = J \setminus (T \cup U)$, so $|J'| \leq k$. Then

$$\begin{aligned}
\text{Err}^2(\widehat{x'_L} - \widehat{w_J}, fk) &\leq \|\widehat{x'_{L\setminus U}} - \widehat{w_{J\setminus U}}\|_2^2 \\
&= \|\widehat{x'}_{L\setminus(U\cup J')} - \widehat{w_T}\|_2^2 + \|(\widehat{x'} - \widehat{w})_{J'}\|_2^2 \\
&\leq \|\widehat{x'}_{L\setminus U} - \widehat{w_T}\|_2^2 + |J'|\,\|(\widehat{x'} - \widehat{w})_{J'}\|_\infty^2 \\
&\leq \text{Err}^2(\widehat{x'}_{L\setminus U}, k) + 3k\mu^2 + k\mu^2 \\
&= \text{Err}^2(\widehat{x'}_{L\setminus U}, k) + O(k\mu^2)
\end{aligned}$$

where we used Equations A.4 and A.5. □

A.10 Proof of Lemma 4.3.8

Lemma 4.3.8 Define $\widehat{x}^{(r)} = \widehat{x} - \widehat{z}^{(r)}$. Consider any one loop r of SPARSEFFT, running with parameters $(B, k, \alpha) = (B_r, k_r, \alpha_r)$ such that $B \geq \frac{Ck}{\alpha^2\epsilon}$ for some C larger than some fixed constant. Then for any $f > 0$,

$$\mathrm{Err}^2(\widehat{x}^{(r+1)}, fk) \leq (1+\epsilon)\,\mathrm{Err}^2(\widehat{x}^{(r)}, k) + O(\epsilon\delta^2 n\|\widehat{x}\|_1^2)$$

with probability $1 - O(\alpha/f)$, and the running time is

$$O((\|\widehat{z}^{(r)}\|_0(1+\alpha\log(n/\delta)) + \frac{B}{\alpha}\log(n/\delta))(\log\frac{1}{\alpha\epsilon} + \log(n/B))).$$

Proof. We use $R_{est} = O(\log\frac{B}{\alpha k}) = O(\log\frac{1}{\alpha\epsilon})$ rounds inside ESTIMATEVALUES.

The running time for LOCATESIGNAL is

$$O((\frac{B}{\alpha}\log(n/\delta) + \|\widehat{z}^{(r)}\|_0(1+\alpha\log(n/\delta)))\log(n/B)),$$

and for ESTIMATEVALUES is

$$O((\frac{B}{\alpha}\log(n/\delta) + \|\widehat{z}^{(r)}\|_0(1+\alpha\log(n/\delta)))\log\frac{1}{\alpha\epsilon})$$

for a total running time as given.

Recall that in round r, $\mu^2 = \frac{\epsilon}{k}(\mathrm{Err}^2(\widehat{x}^{(r)}, k) + \delta^2 n\|\widehat{x}\|_1^2)$ and $S = \{i \in [n] \mid \left|\widehat{x}_i^{(r)}\right|^2 > \mu^2\}$. By Lemma 4.3.5, each $i \in S$ lies in L_r with probability at least $1 - O(\alpha)$. Hence $|S \setminus L| < fk$ with probability at least $1 - O(\alpha/f)$. Then

$$\begin{aligned}
\mathrm{Err}^2(\widehat{x}^{(r)}_{[n]\setminus L}, fk) &\leq \|\widehat{x}^{(r)}_{[n]\setminus(L\cup S)}\|_2^2 \\
&\leq \mathrm{Err}^2(\widehat{x}^{(r)}_{[n]\setminus(L\cup S)}, k) + k\|\widehat{x}^{(r)}_{[n]\setminus(L\cup S)}\|_\infty^2 \\
&\leq \mathrm{Err}^2(\widehat{x}^{(r)}_{[n]\setminus L}, k) + k\mu^2. \qquad \text{(A.6)}
\end{aligned}$$

Let $\widehat{w} = \widehat{z}^{(r+1)} - \widehat{z}^{(r)} = \widehat{x}^{(r)} - \widehat{x}^{(r+1)}$ by the vector recovered by ESTIMATEVALUES. Then $\mathrm{supp}(\widehat{w}) \subset L$, so

$$\begin{aligned}
\mathrm{Err}^2(\widehat{x}^{(r+1)}, 2fk) &= \mathrm{Err}^2(\widehat{x}^{(r)} - \widehat{w}, 2fk) \\
&\leq \mathrm{Err}^2(\widehat{x}^{(r)}_{[n]\setminus L}, fk) + \mathrm{Err}^2(\widehat{x}^{(r)}_L - \widehat{w}, fk) \\
&\leq \mathrm{Err}^2(\widehat{x}^{(r)}_{[n]\setminus L}, fk) + \mathrm{Err}^2(\widehat{x}^{(r)}_L, k) + O(k\mu^2)
\end{aligned}$$

by Lemma 4.3.7. But by Equation (A.6), this gives

$$\begin{aligned}\operatorname{Err}^2(\widehat{x}^{(r+1)}, 2fk) &\le \operatorname{Err}^2(\widehat{x}^{(r)}_{[n]\setminus L}, k) + \operatorname{Err}^2(\widehat{x}^{(r)}_L, k) + O(k\mu^2)\\ &\le \operatorname{Err}^2(\widehat{x}^{(r)}, k) + O(k\mu^2)\\ &= (1 + O(\epsilon))\operatorname{Err}^2(\widehat{x}^{(r)}, k) + O(\epsilon\delta^2 n\|\widehat{x}\|_1^2).\end{aligned}$$

The result follows from rescaling f and ϵ by constant factors. □

A.11 Proof of Theorem 4.3.9

Theorem 4.3.9 With $2/3$ probability, SPARSEFFT recovers $\widehat{z}^{(R+1)}$ such that

$$\|\widehat{x} - \widehat{z}^{(R+1)}\|_2 \le (1+\epsilon)\operatorname{Err}(\widehat{x}, k) + \delta\|\widehat{x}\|_2$$

in $O(\frac{k}{\epsilon}\log(n/k)\log(n/\delta))$ time.

Proof. Define $f_r = O(1/r^2)$ so $\sum f_r < 1/4$. Choose R so $\prod_{r\le R} f_r < 1/k \le \prod_{r<R} f_r$. Then $R = O(\log k/\log\log k)$, since $\prod_{r\le R} f_r < (f_{R/2})^{R/2} = (2/R)^R$.

Set $\epsilon_r = f_r\epsilon$, $\alpha_r = \Theta(f_r^2)$, $k_r = k\prod_{i<r} f_i$, $B_r = O(\frac{k}{\epsilon}\alpha_r f_r)$. Then $B_r = \omega(\frac{k_r}{\alpha_r^2\epsilon_r})$, so for sufficiently large constant the constraint of Lemma 4.3.8 is satisfied. For appropriate constants, Lemma 4.3.8 says that in each round r,

$$\operatorname{Err}^2(\widehat{x}^{(r+1)}, k_{r+1}) = \operatorname{Err}^2(\widehat{x}^{(r+1)}, f_r k_r) \le (1+f_r\epsilon)\operatorname{Err}^2(\widehat{x}^{(r)}, k_r) + O(f_r\epsilon\delta^2 n\|\widehat{x}\|_1^2) \qquad \text{(A.7)}$$

with probability at least $1 - f_r$. The error accumulates, so in round r we have

$$\operatorname{Err}^2(\widehat{x}^{(r)}, k_r) \le \operatorname{Err}^2(\widehat{x}, k)\prod_{i<r}(1+f_i\epsilon) + \sum_{i<r} O(f_r\epsilon\delta^2 n\|\widehat{x}\|_1^2)\prod_{i<j<r}(1+f_j\epsilon)$$

with probability at least $1 - \sum_{i<r} f_i > 3/4$. Hence in the end, since $k_{R+1} = k\prod_{i\le R} f_i < 1$,

$$\|\widehat{x}^{(R+1)}\|_2^2 = \operatorname{Err}^2(\widehat{x}^{(R+1)}, k_{R+1}) \le \operatorname{Err}^2(\widehat{x}, k)\prod_{i\le R}(1+f_i\epsilon) + O(R\epsilon\delta^2 n\|\widehat{x}\|_1^2)\prod_{i\le R}(1+f_i\epsilon)$$

with probability at least $3/4$. We also have

$$\prod_i (1+f_i\epsilon) \le e^{\epsilon\sum_i f_i} \le e$$

making

$$\prod_i (1+f_i\epsilon) \le 1 + e\sum_i f_i\epsilon < 1 + 2\epsilon.$$

Thus we get the approximation factor

$$\|\widehat{x} - \widehat{z}^{(R+1)}\|_2^2 \le (1+2\epsilon)\operatorname{Err}^2(\widehat{x}, k) + O((\log k)\epsilon\delta^2 n\|\widehat{x}\|_1^2)$$

with at least $3/4$ probability. Rescaling δ by poly(n), using $\|\widehat{x}\|_1^2 \leq n\|\widehat{x}\|_2$, and taking the square root gives the desired

$$\|\widehat{x} - \widehat{z}^{(R+1)}\|_2 \leq (1+\epsilon)\,\mathrm{Err}(\widehat{x}, k) + \delta\|\widehat{x}\|_2.$$

Now we analyze the running time. The update $\widehat{z}^{(r+1)} - \widehat{z}^{(r)}$ in round r has support size $3k_r$, so in round r

$$\|\widehat{z}^{(r)}\|_0 \leq \sum_{i<r} 3k_r = O(k).$$

Thus the expected running time in round r is

$$\begin{aligned}
&O((k(1+\alpha_r \log(n/\delta)) + \frac{B_r}{\alpha_r}\log(n/\delta))(\log\frac{1}{\alpha_r\epsilon_r} + \log(n/B_r))) \\
&= O((k + \frac{k}{r^4}\log(n/\delta) + \frac{k}{\epsilon r^2}\log(n/\delta))(\log\frac{r^2}{\epsilon} + \log(n\epsilon/k) + \log r)) \\
&= O((k + \frac{k}{\epsilon r^2}\log(n/\delta))(\log r + \log(n/k)))
\end{aligned}$$

We split the terms multiplying k and $\frac{k}{\epsilon r^2}\log(n/\delta)$, and sum over r. First,

$$\begin{aligned}
\sum_{r=1}^{R}(\log r + \log(n/k)) &\leq R\log R + R\log(n/k) \\
&\leq O(\log k + \log k \log(n/k)) \\
&= O(\log k \log(n/k)).
\end{aligned}$$

Next,

$$\sum_{r=1}^{R}\frac{1}{r^2}(\log r + \log(n/k)) = O(\log(n/k))$$

Thus the total running time is

$$O(k\log k\log(n/k) + \frac{k}{\epsilon}\log(n/\delta)\log(n/k)) = O(\frac{k}{\epsilon}\log(n/\delta)\log(n/k)).$$

□

A.12 Proof of Lemma 5.2.2

Lemma 5.2.2 The probability that any 1-sparsity test invoked by the algorithm is incorrect is at most $O(1/n^{(c-5)/2})$.

To prove Lemma 5.2.2, we first need the following lemma.

Lemma A.12.1. *Let $y \in \mathbb{C}^m$ be drawn from a permutation invariant distribution with $r \geq 2$ nonzero values. Let $T = [2c]$. Then the probability that there exists a y' such that $\|y'\|_0 \leq 1$ and*

$(\widehat{y} - \widehat{y}')_T = 0$ *is at most* $c\left(\frac{c}{m-r}\right)^{c-2}$.

Proof. Let $A = F_T$ be the first $2c$ rows of the inverse Fourier matrix. Because any $2c \times 2c$ submatrix of A is Vandermonde and hence non-singular, the system of linear equations

$$Az = b$$

has at most one c-sparse solution in z, for any b.

If $r \leq c-1$, then $\|y - y'\|_0 \leq c$ so $A(y - y') = 0$ implies $y - y' = 0$. But $r \geq 2$ so $\|y - y'\|_0 > 0$. This is a contradiction, so if $r < c$ then the probability that $(\widehat{y} - \widehat{y}')_T = 0$ is zero. Henceforth, we assume $r \geq c$.

When drawing y, first place $r - (c-1)$ coordinates into u then place the other $c-1$ values into v, so that $y = u + v$. Condition on u, so v is a permutation distribution over $m - r + c - 1$ coordinates. We know there exists at most one c-sparse vector w with $Aw = -Au$. Then,

$$\begin{aligned}
&\Pr_y[\exists y' : A(y - y') = 0 \text{ and} \|y'\|_0 \leq 1] \\
=&\Pr_v[\exists y' : A(v - y') = -Au \text{ and} \|y'\|_0 \leq 1] \\
\leq&\Pr_v[\exists y' : v - y' = w \text{ and} \|y'\|_0 \leq 1] = \Pr_v[\|v - w\|_0 \leq 1] \\
\leq&\Pr_v[|\mathrm{supp}(v) \triangle \mathrm{supp}(w)| \leq 1] \\
<&\frac{m - r + c - 1}{\binom{m-r+c-1}{c-1}} < c\left(\frac{c}{m-r}\right)^{c-2}
\end{aligned}$$

where the penultimate inequality follows from considering the cases $\|w\|_0 \in \{c-2, c-1, c\}$ separately. □

We now proceed with the proof of Lemma 5.2.2 .

Proof. W.L.O.G. consider the row case. Let y be the jth row of $\widehat{x}$. Note that $\widehat{u}_j^{(\tau)} = \widehat{y}_\tau$. Observe that with probability at least $1 - 1/n^c$ we have $\|y\|_0 \leq r$ for $r = c \log n$. Moreover, the distribution of y is permutation-invariant, and the test in BASICESTFREQ corresponds to checking whether $(\widehat{y} - \widehat{y}')_T = 0$ for some 1-sparse $y' = ae_i$. Hence, Lemma A.12.1 with $m = \sqrt{n}$ implies the probability that any specific test fails is less than $c(2c/\sqrt{n})^{c-2}$. Taking a union bound over the $\sqrt{n} \log n$ total tests gives a failure probability of $4c^3 \log n (2c/\sqrt{n})^{c-4} < O(1/n^{(c-5)/2})$. □

A.13 Proof of Lemma 5.3.1

Lemma 5.3.1 Consider the recovery of a column/row j in ROBUSTESTIMATECOL, where $\widehat{u}$ and $\widehat{v}$ are the results of FOLDTOBINS on $\widehat{x}$. Let $y \in \mathbb{C}^{\sqrt{n}}$ denote the jth column/row of $\widehat{x}$. Suppose y is drawn from a permutation invariant distribution $y = y^{head} + y^{residue} + y^{gauss}$, where $\min_{i \in \mathrm{supp}(y^{head})} |y_i| \geq L$, $\|y^{residue}\|_1 < \epsilon L$, and y^{gauss} is drawn from the $\sqrt{n}$-dimensional normal distribution $N_{\mathbb{C}}(0, \sigma^2 I_{\sqrt{n}})$ with standard deviation $\sigma = \epsilon L / n^{1/4}$ in each coordinate on both real

and imaginary axes. We do not require that y^{head}, $y^{residue}$, and y^{gauss} are independent except for the permutation invariance of their sum.

Consider the following bad events:

- False negative: $\mathrm{supp}(y^{head}) = \{i\}$ and ROBUSTESTIMATECOL does not update coordinate i.
- False positive: ROBUSTESTIMATECOL updates some coordinate i but $\mathrm{supp}(y^{head}) \neq \{i\}$.
- Bad update: $\mathrm{supp}(y^{head}) = \{i\}$ and coordinate i is estimated by b with $\left|b - y_i^{head}\right| > \|y^{residue}\|_1 + \sqrt{\frac{\log\log n}{\log n}}\epsilon L$.

For any constant c and ϵ below a sufficiently small constant, there exists a distribution over sets T, T' of size $O(\log n)$, such that as a distribution over y and T, T' we have

- The probability of a false negative is $1/\log^c n$.
- The probability of a false positive is $1/n^c$.
- The probability of a bad update is $1/\log^c n$.

Proof. Let $\check{y}$ denote the 1-dimensional inverse DFT of y. Note that

$$\widehat{u}_j^{(\tau)} = \check{y}_\tau$$

by definition. Therefore, the goal of ROBUSTESTIMATECOL is simply to perform reliable 1-sparse recovery with $O(\log n)$ queries. Fortunately, Algorithm 4.3.2 solved basically the same problem, although with more false positives than we want here.

We choose T' according to the LOCATEINNER procedure from Algorithm 4.3.2; the set T is chosen uniformly at random from $[\sqrt{n}]$. We have that

$$\widehat{u}_j^{(\tau)} = \sum_{i \in [\sqrt{n}]} y_i \omega^{-\tau i}.$$

This is exactly what the procedure HASHTOBINS of Algorithm 4.3.2 approximates up to a small error term. Therefore, the same analysis goes through (Lemma 4.3.5) to get that LOCATESIGNAL returns i with $1 - 1/\log^c n$ probability if $|y_i| \geq \|y_{-i}\|_2$, where we define $y_{-i} := y_{[\sqrt{n}]\setminus\{i\}}$.

Define $A \in \mathbb{C}^{|T| \times \sqrt{n}}$ to be the rows of the inverse Fourier matrix indexed by T, normalized so $|A_{i,j}| = 1$. Then $\widehat{u}_j^{(\tau)} = (Ay)_\tau$.

First, we prove

$$\|y^{residue} + y^{gauss}\|_2 = O(\epsilon L) \tag{A.8}$$

with all but n^{-c} probability. We have that $\mathbb{E}[\|y^{gauss}\|_2^2] = 2\epsilon^2 L^2$, so $\|y^{gauss}\|_2 \leq 3\epsilon L$ with all but $e^{-\Omega(\sqrt{n})} < 1/n^c$ probability by concentration of chi-square variables. We also have that $\|y^{residue}\|_2 \leq \|y^{residue}\|_1 \leq \epsilon L$.

Next, we show

$$\|A(y^{residue} + y^{gauss})\|_2 = O(\epsilon L\sqrt{|\,T|}) \tag{A.9}$$

with all but n^{-c} probability. We have that Ay^{gauss} is drawn from $N_{\mathbb{C}}(0, \epsilon^2 L^2 I_{|\,T|})$ by the rotation invariance of Gaussians, so

$$\|Ay^{gauss}\|_2 \leq 3\epsilon L\sqrt{|\,T|} \tag{A.10}$$

with all but $e^{-\Omega(|\,T|)} < n^{-c}$ probability. Furthermore, A has entries of magnitude 1 so $\|Ay^{residue}\|_2 \leq \|y^{residue}\|_1\sqrt{|\,T|} = \epsilon L\sqrt{|\,T|}$.

Consider the case where $\mathrm{supp}(y^{head}) = \{i\}$. From Equation (A.8) we have

$$\|y_{-i}\|_2^2 \leq \|y^{gauss} + y^{residue}\|_2^2 \leq O(\epsilon^2 L^2) < L^2 \leq \|y_i\|_2^2 \tag{A.11}$$

so i is located with $1 - 1/\log^c n$ probability by LOCATESIGNAL.

Next, we note that for any i, as a distribution over $\tau \in [\sqrt{n}]$,

$$\mathop{\mathbb{E}}_{\tau}[|\widehat{u}_j^{(\tau)} - y_i\omega^{-\tau i}|^2] = \|y_{-i}\|_2^2$$

and so (analogously to Lemma 4.3.6 from Chapter 4 and for any i), since $a = \mathrm{median}_{\tau\in T}\,\widehat{u}_j^{(\tau)}\omega^{\tau i}$ we have

$$|a - y_i|^2 \leq 5\|y_{-i}\|_2^2 \tag{A.12}$$

with probability $1 - e^{-\Omega(|\,T|)} = 1 - 1/n^c$ for some constant c. Hence if $\{i\} = \mathrm{supp}(y^{head})$, we have $|a - y_i|^2 \leq O(\epsilon^2 L^2)$ and therefore $|a| > L/2$, passing the first check on whether i is valid.

For the other check, we have that with $1 - 1/n^c$ probability

$$\begin{aligned}
(\sum_{\tau\in T}\left|\widehat{u}_j^{(\tau)} - a\omega^{-\tau i}\right|^2)^{1/2} &= \|A(y - ae_i)\|_2 \\
&\leq \|A(y^{gauss} + y^{residue} + (y_i^{head} - a)\,e_i)\|_2 \\
&\leq \|A(y^{gauss} + y^{residue})\|_2 + \left|y_i^{head} - a\right|\sqrt{|\,T|} \\
&\leq \|A(y^{gauss} + y^{residue})\|_2 + (\left|y_i^{residue} + y_i^{gauss}\right| + |y_i - a|)\sqrt{|\,T|} \\
&\leq O(\epsilon L\sqrt{|\,T|}).
\end{aligned}$$

where the last step uses Equation (A.9). This gives

$$\sum_{\tau\in T}\left|\widehat{u}_j^{(\tau)} - a\omega^{-\tau i}\right|^2 = O(\epsilon^2 L^2\,|\,T|) < L^2\,|\,T|\,/10$$

so the true coordinate i passes both checks. Hence the probability of a false negative is $1/\log^c n$ as desired.

Now we bound the probability of a false positive. First consider what happens to any other coordinate $i' \neq i$ when $\left|\mathrm{supp}(y^{head})\right| = \{i\}$. We get some estimate a' of its value. Since $A/\sqrt{|T|}$ satisfies an RIP of order 2 and constant $1/4$, by the triangle inequality and Equation (A.9) we have that with $1 - n^{-c}$ probability,

$$\begin{aligned}\|A(y - a'e_{i'})\|_2 &\geq \|A(y_i^{head} e_i - a'e_{i'})\|_2 - \|A(y^{gauss} + y^{residue})\|_2 \\ &\geq y_i^{head}\sqrt{|T|} \cdot (3/4) - O(\epsilon L\sqrt{|T|}) \\ &> L\sqrt{|T|}/2.\end{aligned}$$

Hence the second condition will be violated, and i' will not pass. Thus if $\left|\mathrm{supp}(y^{head})\right| = 1$, the probability of a false positive is at most n^{-c}.

Next, consider what happens to the result i of LOCATESIGNAL when $\left|\mathrm{supp}(y^{head})\right| = 0$. From Equation (A.8) and Equation (A.9) we have that with $1 - n^{-c}$ probability:

$$|a - y_i|^2 \leq 5\|y_{-i}\|_2^2 \leq O(\epsilon^2 L^2).$$

Therefore, from Equation (A.8),

$$|a| \leq |y_i| + |a - y_i| \leq \|y^{residue} + y^{gauss}\|_2 + |a - y_i| = O(\epsilon L) < L/2$$

so the first check is not passed and i is not recovered.

Now suppose $|\mathrm{supp}(y_{head})| > 1$. Lemma 5.3.2 says that with $1 - n^{-c}$ probability over the permutation, no (i, a) satisfies

$$\|A(y^{head} - ae_i)\|_2^2 < L^2 |T| /5.$$

But then, from Equation (A.10)

$$\begin{aligned}\|A(y - ae_i)\|_2 &\geq \|A(y^{head} - ae_i)\|_2 - \|Ay^{gauss}\|_2 \\ &> L\sqrt{|T| /5} - O(\epsilon L\sqrt{|T|}) \\ &> L\sqrt{|T| /10}\end{aligned}$$

so no i will pass the second check. Thus, the probability of a false positive is $1/n^c$.

Finally, consider the probability of a bad update. We have that

$$b = \underset{\tau \in T}{\mathrm{mean}}(Ay)_\tau \omega^{\tau i} = y_i^{head} + \underset{\tau \in T}{\mathrm{mean}}(Ay^{residue} + Ay^{gauss})_\tau \omega^{\tau i}$$

and so

$$\left|b - y_i^{head}\right| \leq \left|\underset{\tau \in T}{\mathrm{mean}}(Ay^{residue})_\tau \omega^{\tau i}\right| + \left|\underset{\tau \in T}{\mathrm{mean}}(Ay^{gauss})_\tau \omega^{\tau i}\right|.$$

We have that

$$\left|\underset{\tau\in T}{\text{mean}}(Ay^{residue})_\tau\omega^{\tau i}\right| \leq \max_{\tau\in T}\left|(Ay^{residue})_\tau\right| \leq \|y^{residue}\|_1$$

.

We know that Ay^{gauss} is $N_{\mathbb{C}}(0,\epsilon^2L^2I_{|T|})$. Hence its mean is a complex Gaussian with standard deviation $\epsilon L/\sqrt{|T|}$ in both the real and imaginary axes. This means the probability that

$$\left|b - y_i^{head}\right| > \|y^{residue}\|_1 + t\epsilon L/\sqrt{|T|}$$

is at most $e^{-\Omega(t^2)}$. Setting $t = \sqrt{\log\log^c n}$ gives a $1/\log^c n$ chance of a bad update, for sufficiently large $|T| = O(\log n)$. □

A.14 Proof of Lemma 5.3.2

Lemma 5.3.2 Let $y \in \mathbb{C}^m$ be drawn from a permutation invariant distribution with $r \geq 2$ nonzero values. Suppose that all the nonzero entries of y have absolute value at least L. Choose $T \subset [m]$ uniformly at random with $t := |T| = O(c^3 \log m)$.

Then, the probability that there exists a y' with $\|y'\|_0 \leq 1$ and

$$\|(\check{y} - \check{y}')_T\|_2^2 < \epsilon L^2 t/n$$

is at most $c^3(\frac{c}{m-r})^{c-2}$ whenever $\epsilon < 1/8$.

Proof. Let $A = \sqrt{1/t}F_{T\times *}$ be $\sqrt{1/t}$ times the submatrix of the Fourier matrix with rows from T, so

$$\|(\check{y} - \check{y}')_T\|_2^2 = \|A(y - y')\|_2^2 t/n.$$

By a coherence bound (see Section 5.3), with $1 - 1/m^c$ probability A satisfies the RIP of order $2c$ with constant 0.5. We would like to bound

$$P := \Pr[\exists y' : \|A(y - y')\|_2^2 < \epsilon L^2 \text{ and} \|y'\|_0 \leq 1]$$

If $r \leq c - 1$, then $y - y'$ is c-sparse and

$$\begin{aligned}\|A(y - y')\|_2^2 &\geq \|y - y'\|_2^2/2 \\ &\geq (r-1)L^2/2 \\ &> \epsilon L^2\end{aligned}$$

as long as $\epsilon < 1/2$, giving $P = 0$. Henceforth, we can assume $r \geq c$. When drawing y, first place $r-(c-1)$ coordinates into u then place the other $c-1$ values into v, so that $y = u+v$. Condition on u, so v is a permutation distribution over $m - r + c - 1$ coordinates. We would like to bound

$$P = \Pr_v[\exists y' : \|A(u + v - y')\|_2^2 < \epsilon L^2 \text{ and} \|y'\|_0 \leq 1].$$

Let w be any c-sparse vector such that $\|A(u+w)\|_2^2 < \epsilon L^2$ (and note that if no such w exists, then since $v - y'$ is c-sparse, $P = 0$). Then recalling that for any norm $\|\cdot\|$, $\|a\|^2 \leq 2\|b\|^2 + 2\|a+b\|^2$ and hence $\|a+b\|^2 \geq \|a\|^2/2 - \|b\|^2$,

$$\begin{aligned}\|A(u+v-y')\|_2^2 &\geq \|A(v-y'-w)\|_2^2/2 - \|A(u+w)\|_2^2 \\ &\geq \|v-y'+w\|_2^2/4 - \epsilon L^2.\end{aligned}$$

Hence

$$P \leq \Pr_v[\exists y' : \|v-y'+w\|_2^2 < 8\epsilon L^2 \text{ and} \|y'\|_0 \leq 1].$$

Furthermore, we know that $\|v-y'+w\|_2^2 \geq L^2(|\mathrm{supp}(v) \setminus \mathrm{supp}(w)| - 1)$. Thus if $\epsilon < 1/8$,

$$\begin{aligned}P &\leq \Pr_v[|\mathrm{supp}(v) \setminus \mathrm{supp}(w)| \leq 1] \\ &\leq \frac{c + (m-r+c-1)c(c-1)/2}{\binom{m-r+c-1}{c-1}} \\ &< c^3(\frac{c}{m-r})^{c-2}\end{aligned}$$

as desired. □

A.15 Proof of Lemma 5.3.4

Lemma 5.3.4 Let ROBUST2DSFFT' be a modified ROBUST2DSFFT that avoids false negatives or bad updates: whenever a false negative or bad update would occur, an oracle corrects the algorithm. With large constant probability, ROBUST2DSFFT' recovers $\widehat{z}$ such that there exists a $(k/\log^c n)$-sparse $\widehat{z}'$ satisfying

$$\|\widehat{z} - \widehat{x} - \widehat{z}'\|_2^2 \leq 6\sigma^2 n.$$

Furthermore, only $O(k/\log^c n)$ false negatives or bad updates are caught by the oracle.

Proof. One can choose the random $\widehat{x^*}$ by first selecting the topology of the graph G, and then selecting the random ordering of the columns and rows of the matrix. Note that reordering the vertices only affects the ideal ordering by a permutation within each stage of recovery; the set of edges recovered at each stage in the ideal ordering depends only on the topology of G. Suppose that the choice of the topology of the graph satisfies the thesis of Lemma 5.3.3 (which occurs with large constant probability). We will show that with large constant probability (over the space of random permutations of the rows and columns), ROBUST2DSFFT' follows the ideal ordering and the requirements of Lemma 5.3.1 are satisfied at every stage.

For a recovered edge e, we define the "residue" $\widehat{x^*}_e - \widehat{z}_e$. We will show that if e has rank r, then $|\widehat{x^*}_e - \widehat{z}_e| \leq r\sqrt{\frac{\log\log n}{\log n}}\epsilon L$.

During attempted recovery at any vertex v during the ideal ordering (including attempts on vertices which do not have exactly one incident edge), let $y \in \mathbb{C}^{\sqrt{n}}$ be the associated column/row of $\widehat{x} - \widehat{z}$. We split y into three parts $y = y^{head} + y^{residue} + y^{gauss}$, where y^{head} contains the elements of

$\widehat{x^*}$ not in $\operatorname{supp}(\widehat{z})$, $y^{residue}$ contains $\widehat{x^*} - \widehat{z}$ over the support of $\widehat{z}$, and y^{gauss} contains $\widehat{w}$ (all restricted to the column/row corresponding to v). Let $S = \operatorname{supp}(y^{residue})$ contain the set of edges incident on v that have been recovered so far. We have by the inductive hypothesis that $\|y^{residue}\|_1 \leq \sum_{e \in S} \text{rank}(e) \sqrt{\frac{\log\log n}{\log n}} \epsilon L$. Since the algorithm verifies that $\sum_{e \in S} \text{rank}(e) \leq \log\log n$, we have

$$\|y^{residue}\|_1 \leq \sqrt{\frac{\log^3 \log n}{\log n}} \epsilon L < \epsilon L.$$

Furthermore, y is permutation invariant: if we condition on the values and permute the rows and columns of the matrix, the algorithm will consider the permuted y in the same stage of the algorithm.

Therefore, the conditions for Lemma 5.3.1 hold. This means that the chance of a false positive is $1/n^c$, so by a union bound, this never occurs. Because false negatives never occur by assumption, this means we continue following the ideal ordering. Because bad updates never occur, new residuals have magnitude at most

$$\|y^{residue}\|_1 + \sqrt{\frac{\log\log n}{\log n}} \epsilon L.$$

Because $\|y^{residue}\|_1 / \left(\sqrt{\frac{\log\log n}{\log n}} \epsilon L \right) \leq \sum_{e \in S} \text{rank}(e) = \text{rank}(v) = \text{rank}(e) - 1$, each new residual has magnitude at most

$$\text{rank}(e) \sqrt{\frac{\log\log n}{\log n}} \epsilon L \leq \epsilon L. \tag{A.13}$$

as needed to complete the induction.

Given that we follow the ideal ordering, we recover every edge of rank at most $\log\log n$. Furthermore, the residue on every edge we recover is at most ϵL. By Lemma 5.3.3, there are at most $k/\log^c n$ edges that we do not recover. From Equation (A.13), the squared ℓ_2 norm of the residues is at most $\epsilon^2 L^2 k = \epsilon^2 C^2 \sigma^2 n / k \cdot k < \sigma^2 n$ for ϵ small enough. Since $\|\widehat{w}\|_2^2 < 2\sigma^2 n$ with overwhelming probability, there exists a $\widehat{z}'$ so that

$$\|\widehat{z} - \widehat{x} - \widehat{z}'\|_2^2 \leq 2\|\widehat{z} - \widehat{x^*} - \widehat{z}'\|_2^2 + 2\|w\|_2^2 \leq 6\sigma^2 n.$$

Finally, we need to bound the number of times the oracle catches false negatives or bad updates. The algorithm applies Lemma 5.3.1 only $2\sqrt{n} + O(k) = O(k)$ times. Each time has a $1/\log^c n$ chance of a false negatives or bad update. Hence the expected number of false negatives or bad updates is $O(k/\log^c n)$. □

Appendix B

The Optimality of the Exactly k-Sparse Algorithm 4.2.1

If we assume that FFT is optimal and hence the DFT cannot be computed in less than $O(n \log n)$ time, then Algorithm 4.2.1 described in Section 4.2 for the exactly k-sparse case is *optimal* as long as $k = n^{\Omega(1)}$. Under the In this appendix, we show that by reducing a k-dimensional DFT to the an exact k-sparse n-dimensional DFT.

Assume that k divides n.

Lemma B.0.1. *Suppose that there is an algorithm A that, given an n-dimensional vector y such that $\hat{y}$ is k-sparse, computes $\hat{y}$ in time $T(k)$. Then there is an algorithm A' that given a k-dimensional vector x computes $\hat{x}$ in time $O(T(k))$.*

Proof. Given a k-dimensional vector x, we define $y_i = x_{i \bmod k}$, for $i = 0 \ldots n-1$. Whenever A requests a sample y_i, we compute it from x in constant time. Moreover, we have that $\hat{y}_i = \hat{x}_{i/(n/k)}$ if i is a multiple of (n/k), and $\hat{y}_i = 0$ otherwise. Thus $\hat{y}$ is k-sparse. Since $\hat{x}$ can be immediately recovered from $\hat{y}$, the lemma follows. □

Corollary B.0.2. *Assume that the n-dimensional DFT cannot be computed in $o(n \log n)$ time. Then any algorithm for the k-sparse DFT (for vectors of arbitrary dimension) must run in $\Omega(k \log k)$ time.*

Appendix C

Lower Bound of the Sparse Fourier Transform in the General Case

In this appendix, we show any algorithm satisfying Equation (4.1) must access $\Omega(k \log(n/k)/\log\log n)$ samples of x.

We translate this problem into the language of compressive sensing:

Theorem C.0.3. *Let $F \in \mathbb{C}^{n\times n}$ be orthonormal and satisfy $|F_{i,j}| = 1/\sqrt{n}$ for all i, j. Suppose an algorithm takes m adaptive samples of Fx and computes x' with*

$$\|x - x'\|_2 \leq 2 \min_{k\text{-sparse } x^*} \|x - x^*\|_2$$

for any x, with probability at least $3/4$. Then it must have $m = \Omega(k \log(n/k)/\log\log n)$.

Corollary C.0.4. *Any algorithm computing the approximate Fourier transform must access $\Omega(k \log(n/k)/\log\log$ samples from the time domain.*

If the samples were chosen non-adaptively, we would immediately have $m = \Omega(k \log(n/k))$ by [146]. However, an algorithm could choose samples based on the values of previous samples. In the sparse recovery framework allowing general linear measurements, this adaptivity can decrease the number of measurements to $O(k \log\log(n/k))$ [86]; in this section, we show that adaptivity is much less effective in our setting where adaptivity only allows the choice of Fourier coefficients.

We follow the framework of Section 4 of [146]. In this section we use standard notation from information theory, including $I(x; y)$ for mutual information, $H(x)$ for discrete entropy, and $h(x)$ for continuous entropy. Consult a reference such as [36] for details.

Let $\mathcal{F} \subset \{S \subset [n] : |S| = k\}$ be a family of k-sparse supports such that:

- $|S \oplus S'| \geq k$ for $S \neq S' \in \mathcal{F}$, where $\oplus$ denotes the exclusive difference between two sets, and
- $\log|\mathcal{F}| = \Omega(k \log(n/k))$.

This is possible; for example, a random code on $[n/k]^k$ with relative distance $1/2$ has these properties.

For each $S \in \mathcal{F}$, let $X^S = \{x \in \{0, \pm 1\}^n \mid \text{supp}(x^S) = S\}$. Let $x \in X^S$ uniformly at random. The variables x_i, $i \in S$, are i.i.d. subgaussian random variables with parameter $\sigma^2 = 1$, so for any row F_j of F, $F_j x$ is subgaussian with parameter $\sigma^2 = k/n$. Therefore

$$\Pr_{x \in X^S}[|F_j x| > t\sqrt{k/n}] < 2e^{-t^2/2}$$

hence for each S, we can choose an $x^S \in X^S$ with

$$\|Fx^S\|_\infty < O(\sqrt{\frac{k \log n}{n}}). \tag{C.1}$$

Let $X = \{x^S \mid S \in \mathcal{F}\}$ be the set of such x^S.

Let $w \sim N(0, \alpha \frac{k}{n} I_n)$ be i.i.d. normal with variance $\alpha k/n$ in each coordinate.

Consider the following process:

Procedure. First, Alice chooses $S \in \mathcal{F}$ uniformly at random, then selects the $x \in X$ with $\text{supp}(x) = S$. Alice independently chooses $w \sim N(0, \alpha \frac{k}{n} I_n)$ for a parameter $\alpha = \Theta(1)$ sufficiently small. For $j \in [m]$, Bob chooses $i_j \in [n]$ and observes $y_j = F_{i_j}(x + w)$. He then computes the result $x' \approx x$ of sparse recovery, rounds to X by $\hat{x} = \arg\min_{x^* \in X} \|x^* - x'\|_2$, and sets $S' = \text{supp}(\hat{x})$. This gives a Markov chain $S \to x \to y \to x' \to \hat{x} \to S'$.

We will show that deterministic sparse recovery algorithms require large m to succeed on this input distribution $x + w$ with $3/4$ probability. By Yao's minimax principle, this means randomized sparse recovery algorithms also require large m to succeed with $3/4$ probability.

Our strategy is to give upper and lower bounds on $I(S; S')$, the mutual information between S and S'.

Lemma C.0.5 (Analog of Lemma 4.3 of [146] for $\epsilon = O(1)$)**.** *There exists a constant $\alpha' > 0$ such that if $\alpha < \alpha'$, then $I(S; S') = \Omega(k \log(n/k))$.*

Proof. Assuming the sparse recovery succeeds (as happens with 3/4 probability), we have $\|x' - (x + w)\|_2 \leq 2\|w\|_2$, which implies $\|x' - x\|_2 \leq 3\|w\|_2$. Therefore

$$\begin{aligned} \|\hat{x} - x\|_2 &\leq \|\hat{x} - x'\|_2 + \|x' - x\|_2 \\ &\leq 2\|x' - x\|_2 \\ &\leq 6\|w\|_2. \end{aligned}$$

We also know $\|x' - x''\|_2 \geq \sqrt{k}$ for all distinct $x', x'' \in X$ by construction. Because $\mathbb{E}[\|w\|_2^2] = \alpha k$, with probability at least $3/4$ we have $\|w\|_2 \leq \sqrt{4\alpha k} < \sqrt{k}/6$ for sufficiently small α. But then $\|\hat{x} - x\|_2 < \sqrt{k}$, so $\hat{x} = x$ and $S = S'$. Thus $\Pr[S \neq S'] \leq 1/2$.

Fano's inequality states $H(S \mid S') \leq 1 + \Pr[S \neq S'] \log |\mathcal{F}|$. Thus

$$I(S; S') = H(S) - H(S \mid S') \geq -1 + \frac{1}{2} \log |\mathcal{F}| = \Omega(k \log(n/k))$$

as desired. □

We next show an analog of their upper bound (Lemma 4.1 of [146]) on $I(S; S')$ for adaptive measurements of bounded ℓ_∞ norm. The proof follows the lines of [146], but is more careful about dependencies and needs the ℓ_∞ bound on Fx.

Lemma C.0.6.

$$I(S; S') \leq O(m \log(1 + \frac{1}{\alpha} \log n)).$$

Proof. Let $A_j = F_{i_j}$ for $j \in [m]$, and let $w'_j = A_j w$. The w'_j are independent normal variables with variance $\alpha\frac{k}{n}$. Because the A_j are orthonormal and w is drawn from a rotationally invariant distribution, the w' are also independent of x.

Let $y_j = A_j x + w'_j$. We know $I(S; S') \leq I(x; y)$ because $S \to x \to y \to S'$ is a Markov chain. Because the variables A_j are deterministic given $y_1, \ldots, y_{j-1}$,

$$\begin{aligned} I(x; y_j \mid y_1, \ldots, y_{j-1}) &= I(x; A_j x + w'_j \mid y_1, \ldots, y_{j-1}) \\ &= h(A_j x + w'_j \mid y_1, \ldots, y_{j-1}) - h(A_j x + w'_j \mid x, y_1, \ldots, y_{j-1}) \\ &= h(A_j x + w'_j \mid y_1, \ldots, y_{j-1}) - h(w'_j). \end{aligned}$$

By the chain rule for information,

$$\begin{aligned} I(S; S') &\leq I(x; y) \\ &= \sum_{j=1}^{m} I(x; y_j \mid y_1, \ldots, y_{j-1}) \\ &= \sum_{j=1}^{m} h(A_j x + w'_j \mid y_1, \ldots, y_{j-1}) - h(w'_j) \\ &\leq \sum_{j=1}^{m} h(A_j x + w'_j) - h(w'_j). \end{aligned}$$

Thus it suffices to show $h(A_j x + w'_j) - h(w'_j) = O(\log(1 + \frac{1}{\alpha} \log n))$ for all j.

Note that A_j depends only on $y_1, \ldots, y_{j-1}$, so it is independent of w'_j. Thus

$$\mathbb{E}[(A_j x + w'_j)^2] = \mathbb{E}[(A_j x)^2] + \mathbb{E}[(w'_j)^2] \leq O(\frac{k \log n}{n}) + \alpha\frac{k}{n}$$

by Equation (C.1). Because the maximum entropy distribution under an ℓ_2 constraint is a Gaussian, we have

$$\begin{aligned} h(A_j x + w'_j) - h(w'_j) &\leq h(N(0, O(\frac{k \log n}{n}) + \alpha\frac{k}{n})) - h(N(0, \alpha\frac{k}{n})) \\ &= \frac{1}{2} \log(1 + \frac{O(\log n)}{\alpha}) \\ &= O(\log(1 + \frac{1}{\alpha} \log n)). \end{aligned}$$

as desired. □

Theorem C.0.3 follows from Lemma C.0.5 and Lemma C.0.6, with $\alpha = \Theta(1)$.

Appendix D

Efficient Constructions of Window Functions

In this appendix, we show how to efficiently implement our flat window functions which we use in Chapters 3 and 4.

Claim D.0.7. *Let* cdf *denote the standard Gaussian cumulative distribution function. Then:*

1. $\mathrm{cdf}(t) = 1 - \mathrm{cdf}(-t)$.
2. $\mathrm{cdf}(t) \leq e^{-t^2/2}$ *for* $t < 0$.
3. $\mathrm{cdf}(t) < \delta$ *for* $t < -\sqrt{2\log(1/\delta)}$.
4. $\int_{x=-\infty}^{t} \mathrm{cdf}(x)dx < \delta$ *for* $t < -\sqrt{2\log(3/\delta)}$.
5. *For any* δ*, there exists a function* $\widetilde{\mathrm{cdf}}_\delta(t)$ *computable in* $O(\log(1/\delta))$ *time such that* $\|\mathrm{cdf} - \widetilde{\mathrm{cdf}}_\delta\|_\infty < \delta$.

Proof.

1. Follows from the symmetry of Gaussian distribution.
2. Follows from standard moment generating function bound on Gaussian random variables.
3. Follows from (2).
4. Property (2) implies that $\mathrm{cdf}(t)$ is at most $\sqrt{2\pi} < 3$ times larger than the Gaussian pdf. Then apply (3).
5. By (1) and (3), $\mathrm{cdf}(t)$ can be computed as $\pm\delta$ or $1 \pm \delta$ unless $|t| < \sqrt{2(\log(1/\delta))}$. But then an efficient expansion around 0 only requires $O(\log(1/\delta))$ terms to achieve precision $\pm\delta$. For example, we can truncate the representation [117]

$$\mathrm{cdf}(t) = \frac{1}{2} + \frac{e^{-t^2/2}}{\sqrt{2\pi}}\left(t + \frac{t^3}{3} + \frac{t^5}{3\cdot 5} + \frac{t^7}{3\cdot 5\cdot 7} + \cdots\right)$$

at $O(\log(1/\delta))$ terms.

□

Claim D.0.8. *Define the continuous Fourier transform of $f(t)$ by*

$$\widehat{f}(s) = \int_{-\infty}^{\infty} e^{-2\pi \mathbf{i} st} f(t)\, dt.$$

For $t \in [n]$, define

$$g_t = \sqrt{n} \sum_{j=-\infty}^{\infty} f(t + nj)$$

and

$$g'_t = \sum_{j=-\infty}^{\infty} \widehat{f}(t/n + j).$$

Then $\widehat{g} = g'$, where $\widehat{g}$ is the n-dimensional DFT of g.

Proof. Let $\Delta_1(t)$ denote the Dirac comb of period 1: $\Delta_1(t)$ is a Dirac delta function when t is an integer and zero elsewhere. Then $\widehat{\Delta_1} = \Delta_1$. For any $t \in [n]$, we have

$$\begin{aligned}
\widehat{g}_t &= \sum_{s=1}^{n} \sum_{j=-\infty}^{\infty} f(s + nj) e^{-2\pi \mathbf{i} ts/n} \\
&= \sum_{s=1}^{n} \sum_{j=-\infty}^{\infty} f(s + nj) e^{-2\pi \mathbf{i} t(s+nj)/n} \\
&= \sum_{s=-\infty}^{\infty} f(s) e^{-2\pi \mathbf{i} ts/n} \\
&= \int_{-\infty}^{\infty} f(s) \Delta_1(s) e^{-2\pi \mathbf{i} ts/n} ds \\
&= \widehat{(f \cdot \Delta_1)}(t/n) \\
&= (\widehat{f} * \Delta_1)(t/n) \\
&= \sum_{j=-\infty}^{\infty} \widehat{f}(t/n + j) \\
&= g'_t.
\end{aligned}$$

□

Lemma D.0.9. *For any parameters $B \geq 1, \delta > 0$, and $\alpha > 0$, there exist flat window functions G and $\widehat{G'}$ such that G can be computed in $O(\frac{B}{\alpha} \log(n/\delta))$ time, and for each i $\widehat{G'}_i$ can be evaluated in $O(\log(n/\delta))$ time.*

Proof. We will show this for a function $\widehat{G'}$ that is a Gaussian convolved with a box-car filter. First we construct analogous window functions for the continuous Fourier transform. We then show that discretizing these functions gives the desired result.

Let D be the pdf of a Gaussian with standard deviation $\sigma > 1$ to be determined later, so $\widehat{D}$ is the pdf of a Gaussian with standard deviation $1/\sigma$. Let $\widehat{F}$ be a box-car filter of length $2C$ for some parameter $C < 1$; that is, let $\widehat{F}(t) = 1$ for $|t| < C$ and $F(t) = 0$ otherwise, so $F(t) = 2C\text{sinc}(t/(2C))$. Let $G^* = D \cdot F$, so $\widehat{G^*} = \widehat{D} * \widehat{F}$.

Then $|G^*(t)| \leq 2C\,|D(t)| < 2C\delta$ for $|t| > \sigma\sqrt{2\log(1/\delta)}$. Furthermore, G^* is computable in $O(1)$ time.

Its Fourier transform is $\widehat{G^*}(t) = \text{cdf}(\sigma(t+C)) - \text{cdf}(\sigma(t-C))$. By Claim D.0.7 we have for $|t| > C + \sqrt{2\log(1/\delta)}/\sigma$ that $\widehat{G^*}(t) = \pm\delta$. We also have, for $|t| < C - \sqrt{2\log(1/\delta)}/\sigma$, that $\widehat{G^*}(t) = 1 \pm 2\delta$.

Now, for $i \in [n]$ let $H_i = \sqrt{n}\sum_{j=\infty}^{\infty} G^*(i+nj)$. By Claim D.0.8 it has DFT $\widehat{H_i} = \sum_{j=\infty}^{\infty} \widehat{G^*}(i/n + j)$. Furthermore,

$$\begin{aligned}
\sum_{|i|>\sigma\sqrt{2\log(1/\delta)}} |G^*(i)| &\leq 4C \sum_{i<-\sigma\sqrt{2\log(1/\delta)}} |D(i)| \\
&\leq 4C\left(\int_{-\infty}^{-\sigma\sqrt{2\log(1/\delta)}} |D(x)|\,dx + D(-\sigma\sqrt{2\log(1/\delta)})\right) \\
&\leq 4C(\text{cdf}(-\sqrt{2\log(1/\delta)}) + D(-\sigma\sqrt{2\log(1/\delta)})) \\
&\leq 8C\delta \leq 8\delta.
\end{aligned}$$

Thus if we let

$$G_i = \sqrt{n} \sum_{\substack{|j|<\sigma\sqrt{2\log(1/\delta)} \\ j\equiv i \pmod n}} G^*(j)$$

for $|i| < \sigma\sqrt{2\log(1/\delta)}$ and $G_i = 0$ otherwise, then $\|G - H\|_1 \leq 8\delta\sqrt{n}$.

Now, note that for integer i with $|i| \leq n/2$,

$$\begin{aligned}
\widehat{H_i} - \widehat{G^*}(i/n) &= \sum_{\substack{j\in\mathbb{Z} \\ j\neq 0}} \widehat{G^*}(i/n + j) \\
\left|\widehat{H_i} - \widehat{G^*}(i/n)\right| &\leq 2\sum_{j=0}^{\infty} \widehat{G^*}(-1/2 - j) \\
&\leq 2\sum_{j=0}^{\infty} \text{cdf}(\sigma(-1/2 - j + C)) \\
&\leq 2\int_{-\infty}^{-1/2} \text{cdf}(\sigma(x + C))\,dx + 2\,\text{cdf}(\sigma(-1/2 + C)) \\
&\leq 2\delta/\sigma + 2\delta \leq 4\delta
\end{aligned}$$

by Claim D.0.7, as long as

$$\sigma(1/2 - C) > \sqrt{2\log(3/\delta)}. \tag{D.1}$$

Let

$$\widehat{G'}_i = \begin{cases} 1 & |i| \le n(C - \sqrt{2\log(1/\delta)}/\sigma) \\ 0 & |i| \ge n(C + \sqrt{2\log(1/\delta)}/\sigma) \\ \widetilde{\mathrm{cdf}}_\delta(\sigma(i+C)/n) - \widetilde{\mathrm{cdf}}_\delta(\sigma(i-C)/n) & \text{otherwise} \end{cases}$$

where $\widetilde{\mathrm{cdf}}_\delta(t)$ computes $\mathrm{cdf}(t)$ to precision $\pm\delta$ in $O(\log(1/\delta))$ time, as per Claim D.0.7. Then $\widehat{G'}_i = \widehat{G^*}(i/n) \pm 2\delta = \widehat{H}_i \pm 6\delta$. Hence

$$\begin{aligned} \|\widehat{G} - \widehat{G'}\|_\infty &\le \|\widehat{G'} - \widehat{H}\|_\infty + \|\widehat{G} - \widehat{H}\|_\infty \\ &\le \|\widehat{G'} - \widehat{H}\|_\infty + \|\widehat{G} - \widehat{H}\|_2 \\ &= \|\widehat{G'} - \widehat{H}\|_\infty + \|G - H\|_2 \\ &\le \|\widehat{G'} - \widehat{H}\|_\infty + \|G - H\|_1 \\ &\le (8\sqrt{n} + 6)\delta. \end{aligned}$$

Replacing δ by δ/n and plugging in $\sigma = \frac{4B}{\alpha}\sqrt{2\log(n/\delta)} > 1$ and $C = (1 - \alpha/2)/(2B) < 1$, we have the required properties of flat window functions:

- $|G_i| = 0$ for $|i| \ge \Omega(\frac{B}{\alpha}\log(n/\delta))$
- $\widehat{G'}_i = 1$ for $|i| \le (1-\alpha)n/(2B)$
- $\widehat{G'}_i = 0$ for $|i| \ge n/(2B)$
- $\widehat{G'}_i \in [0, 1]$ for all i.
- $\|\widehat{G'} - \widehat{G}\|_\infty < \delta$.
- We can compute G over its entire support in $O(\frac{B}{\alpha}\log(n/\delta))$ total time.
- For any i, $\widehat{G'}_i$ can be computed in $O(\log(n/\delta))$ time for $|i| \in [(1-\alpha)n/(2B), n/(2B)]$ and $O(1)$ time otherwise.

The only requirement was Equation (D.1), which is that

$$\frac{4B}{\alpha}\sqrt{2\log(n/\delta)}(1/2 - \frac{1-\alpha/2}{2B}) > \sqrt{2\log(3n/\delta)}.$$

This holds if $B \ge 2$. The $B = 1$ case is trivial using the constant function $\widehat{G'}_i = 1$. □

Appendix E

Sample Lower Bound for The Bernoulli Distribution

We will show that the lower bound of $\Omega(k \log (n/k))$ on ℓ_2/ℓ_2 recovery from [146] applies to our Bernoulli distribution from Section 5.1.4. First, we state their bound:

Lemma E.0.10 ([146] section 4)**.** *For any $k < n/\log n$ and constant $\epsilon > 0$, there exists a distribution D_k over k-sparse vectors in $\{0, 1, -1\}^n$ such that, for every distribution of matrices $A \in \mathbb{R}^{m \times n}$ with $m = o(k \log(n/k))$ and recovery algorithms $\mathcal{A}$,*

$$\Pr[\|\mathcal{A}(A(x + w)) - x\|_2 < \sqrt{k}/5] < 1/2$$

as a distribution over $x \sim D_k$ and $w \sim N(0, \sigma^2 I_n)$ with $\sigma^2 = \epsilon k/n$, as well as over A and $\mathcal{A}$.

First, we note that we can replace D_k with U_k, the uniform distribution over k-sparse vectors in $\{0, 1, -1\}^n$ in Lemma E.0.10. To see this, suppose we have an $(A, \mathcal{A})$ that works with $1/2$ probability over U_k. Then for any k-sparse $x \in \{0, 1, -1\}^n$, if we choose a random permutation matrix P and sign flip matrix S, $PSx \sim U_k$. Hence, the distribution of matrices APS and algorithm $\mathcal{A}'(x) = \mathcal{A}((PS)^{-1}x)$ works with $1/2$ probability for any x, and therefore on average over D_k. This implies that A has $\Omega(k \log(n/k))$ rows by Lemma E.0.10. Hence, we can set $D_k = U_k$ in Lemma E.0.10.

Our algorithm works with $3/4$ probability over vectors x that are not necessarily k-sparse, but have a binomial number $B(n, k/n)$ of nonzeros. That is, it works over the distribution U that is $U_{k'} : k' \sim B(n, k/n)$. With $1 - e^{-\Omega(k)} > 3/4$ probability, $k' \in [k/2, 2k]$. Hence, our algorithm works with at least $1/2$ probability over $(U_{k'} : k' \sim B(n, k/n) \cap k' \in [k/2, 2k])$. By an averaging argument, there must exist a $k' \in [k/2, 2k]$ where our algorithm works with at least $1/2$ probability over $U_{k'}$; but the lemma implies that it must therefore take $\Omega(k' \log(n/k')) = \Omega(k \log(n/k))$ samples.

Appendix F

Analysis of QuickSync System

F.1 Analysis of the baseline algorithm

The baseline algorithm computes $a_k = \mathbf{c}^{(k)} \cdot \mathbf{x}$ for all shifts $k = 0 \dots n-1$. The probability that the algorithm reports an incorrect output is equal to

$$P(\sigma) = \Pr[a_t \leq \max_{k \neq t} a_k]$$

To estimate the probability of success as a function of σ, we derive the distribution of the coordinates a_k. From our assumptions we have that $a_k = \mathbf{c}^{(k)} \cdot (\mathbf{c}^{(t)} + \mathbf{g}) = v_k + u_k$, where $v_k = \mathbf{c}^{(k)} \cdot \mathbf{c}^{(t)}$ and $u_k = \mathbf{c}^{(k)} \cdot \mathbf{g}$. Note that u_k has normal distribution with zero mean and variance

$$\text{Var}[u_k] = \text{Var}[\sum_{i=0}^{n-1} c_i^{(k)} g_i] = n\text{Var}[c_1^{(k)}]\text{Var}[g_1] = n\sigma$$

Regarding v_k, we have the following two cases:

- If $k = t$, i.e., for the correct value of the shift, we have $v_t = \mathbf{c}^{(t)} \cdot \mathbf{c}^{(t)} = n$.
- If $k \neq t$, i.e., for an incorrect value of the shift the expectation of v_k is 0.

We need to bound $P(\sigma) = \Pr[n + u_t \leq \max_{k \neq t} v_k + u_k]$. The following theorem establishes the sufficient (and as we show later, necessary) condition for the baseline algorithm to be correct with probability $1 - o(1)$, i.e., $P(\sigma) \to 0$ as $n \to \infty$.

Lemma F.1.1. *Assume that $\sigma \leq c(n)n/\ln n$ for $c(n) = o(1)$. Then $P(\sigma) = o(1)$.*

Proof. We will bound the probabilities of the following events: E_1: $\exists_{k \neq t} u_k \geq n/3$; E_2: $u_t \leq -n/3$; E_3: $\exists_{k \neq t} v_k \geq n/3$. If none of the events hold then the algorithm output is correct. We will show that $\Pr[E_1] + \Pr[E_2] + \Pr[E_3] = o(1)$.

To analyze E_1 and E_2 recall the following fact:

Fact F.1.2. *Let $\Phi(s)$ be the c.d.f. of the normal distribution with zero mean and unit variance. Then for $s > 0$*

$$1 - \Phi(s) \leq e^{-s^2/2}/s$$

We can now bound

$$\Pr[E_1] \leq n \Pr[u_k \geq n/3] = n(1 - \Phi(\frac{n/3}{\sqrt{\mathrm{Var}[u_k]}}))$$

where k is any index distinct from t. Since $\mathrm{Var}[u_k] = n\sigma \leq c(n)n^2/\ln n$, we have

$$\Pr[E_1] \leq n(1 - \Phi(\sqrt{\ln n/(9c(n))})) \leq e^{\ln n} \cdot e^{-\frac{\ln(n)}{18c(n)}} = o(1)$$

The probability $\Pr[E_2]$ can be bounded in the same way.

To bound $\Pr[E_3]$, assume without loss of generality that $t = 0$. In this case

$$\begin{aligned} v_k &= \mathbf{c}^{(k)} \cdot \mathbf{c} \\ &= (c_k c_0 + c_{k+1} c_1 + \ldots + c_{n-1} c_{n-k-1}) + (c_0 c_{n-k} + \ldots + c_{k-1} c_{n-1}) \\ &= S_k + S'_k \end{aligned}$$

The terms in the sum $S_k + S'_k$ are in general *not* independent. In particular, if $k = n/2$, then $S_k = S'_k$. However, we observe the following.

Claim F.1.3. *Each of S_k and S'_k is a sum of independent random variables taking values in $\{-1, 1\}$ with probability $1/2$.*

The claim enables us to bound each sum separately. We will bound $\Pr[S'_k \geq n/6]$ first. If $k < n/6$ then the probability is zero. Otherwise, by applying the Chernoff bound we have

$$\Pr[S'_k \geq n/6] \leq e^{-(n/6)^2/(2k)} \leq e^{-n/72}$$

The probability $\Pr[S_k \geq n/6]$ can be bounded in the same way. Hence $\Pr[E_3] \leq ne^{-n/72} = o(1)$. □

Theorem 8.4.1 follows from Lemma F.1.1.

F.2 Tightness of the variance bound

In this section we show that the assumption on the noise variance used in Theorem 8.4.1 is asymptotically tight. Specifically, we show that if $\sigma \geq cn/\ln n$ for some large enough constant $c > 0$, then there with probability $1 - o(1)$ the output of the baseline algorithm is incorrect. This will prove Theorem 8.4.2.

Recalling the notation in Section F.1, we have $a_t = n + u_t$, $a_k = v_k + u_k$ for $k \neq t$. We need to show that $P(\sigma) = \Pr[a_t \leq \max_{k \neq t} a_k]$ approaches 1 for c large enough. To this end, we first

observe that (i) $v_k \geq -n$ holds always and (ii) $\Pr[u_t \geq n] = o(1)$ since u_t is a normal variable with variance $n\sigma = O(n^2/\ln n)$, so the desired bound holds e.g., by Chebyshev inequality. Hence it suffices to show that

$$\Pr[\sup_{k \neq t} u_k \leq 3n] = o(1) \tag{F.1}$$

The main difficulty in proving Equation F.1 is the fact that the random variables u_k are *not* independent. If they were, a simple calculation would show that the expected value of the maximum of n independent normal variables with variance $n\sigma$ is at least $\sqrt{n\sigma \ln n} = cn$, which is larger than $a_t = n$ for a large enough c. This would then ensure that the reported shift is distinct from t with constant probability. Unfortunately, the independence could be guaranteed only if the shifted codes $c^{(k)}$ were orthogonal, which is not the case.

Instead, our argument utilizes the fact that the shifted codes $\mathbf{c}^{(k)}$ are "almost" orthogonal. Specifically, let $C = \{\mathbf{c}^{(k)} : k \neq t\}$. Since (as shown in the earlier section in the context of the event E_3) the probability that for any pair $\mathbf{c} \neq \mathbf{c}' \in C$ we have $\mathbf{c} \cdot \mathbf{c}' \leq n/3$ is $o(1)$, it follows that $\|\mathbf{c} - \mathbf{c}'\|_2^2 \geq n$ for all such $\mathbf{c}, \mathbf{c}' \in C$ with probability $1 - o(1)$.

We can now use a powerful inequality due to Sudakov [143] to show that the random variables u_k are "almost" independent, and thus the expected value of the maximum is still $\sqrt{n\sigma \ln n}$. In our context, the inequality states the following.

Fact F.2.1. *There exists a constant $c_2 > 0$ such that if D is the Euclidean distance between the closest pair of vectors in C, then:*

$$E = E[\max_{\mathbf{c} \in C} \mathbf{c} \cdot \mathbf{g}] \geq c_2 D \sqrt{\sigma \ln n}$$

Since $D = \sqrt{n}$, we obtain that

$$E \geq c_2 \sqrt{n} \sqrt{cn/\ln n \cdot \ln n} = c_2 n$$

The lower bound on the expected value of the maximum can be then converted into an upper bound on probability that the maximum is much lower than its expectation (this follows from simple but somewhat tedious calculations). This leads to Equation F.1 and completes the proof of Theorem 8.4.2.

F.3 Analysis of the QuickSync algorithm

In this section we show that the probability of correctness for the QuickSync algorithm that aliases into n/p buckets exhibits a similar behavior to the baseline algorithm, albeit with the bucket variance larger by a factor of $O(p)$. At the same time, the running time of our algorithm is equal to $O(pn + (n/p)\log(n/p))$. This improves over the baseline algorithm which has $O(n \log n)$ runtime as long as the term pn is smaller than $(n/p)\log(n/p)$.

Recall that the algorithm first computes the aliased spreading code $\mathbf{c}(p)$ and signal $\mathbf{x}(p)$, de-

fined as

$$c(p)_i = \sum_{q=0}^{p-1} c_{i+qn/p} \text{ and } x(p)_i = \sum_{q=0}^{p-1} x_{i+qn/p}$$

for $i = 0 \dots n/p - 1$. The aliased noise vector $\mathbf{g}(p)$ is defined in an analogous way.

The application of FFT, coordinate-wise multiplication and inverse FFT computes

$$a(p)_k = c(p)^{(k)} \cdot x(p)$$

for all shifts $k = 0 \dots n/p - 1$. The algorithm then selects $a(p)_k$ with the largest value. The last step of the algorithm fails if for $t' = t \mod n/p$ we have $a(p)_{t'} \leq \max_{k \neq t'} a(p)_k$. Let $P'(\sigma)$ be the probability of this event. We will show that as long as $\sigma = o(pn/\ln n)$ we have $P'(\sigma) = o(1)$.

Lemma F.3.1. *Assume that $\sigma \leq c(n)\frac{n}{p \ln n}$ for $c(n) = o(1)$, and that $p = o(n^{1/6})$. Then $P'(\sigma) = o(1)$.*

Proof. We start by decomposing each term $a(p)_k$:

$$\begin{aligned} a(p)_k &= \sum_{i=0}^{n/p-1} \left(\sum_{q=0}^{p-1} c_{i+qn/p}^{(k)} \right) \left(\sum_{q=0}^{p-1} c_{i+qn/p}^{(t')} + \sum_{q=0}^{p-1} g_{i+qn/p} \right) \\ &= \sum_{i=0}^{n/p-1} \left(\sum_{q=0}^{p-1} c_{i+qn/p}^{(k)} \right) \left(\sum_{q=0}^{p-1} c_{i+qn/p}^{(t')} \right) + \sum_{i=0}^{n/p-1} \left(\sum_{q=0}^{p-1} c_{i+qn/p}^{(k)} \right) \left(\sum_{q=0}^{p-1} g_{i+qn/p} \right) \\ &= v_k + u_k \end{aligned}$$

Consider the following events:

- E_0: $v_{t'} \leq n - n/4$;
- E_1: $\exists_{k \neq t'} u_k \geq n/4$;
- E_2: $u_{t'} \leq -n/4$;
- E_3: $\exists_{k \neq t'} v_k \geq n/4$.

If none of the events hold, then the algorithm output is correct. We need to show that $\Pr[E_0] + \Pr[E_1] + \Pr[E_2] + \Pr[E_3] = o(1)$.

Events E_1 and E_2 Let $\hat{c}_i = \sum_{q=0}^{p-1} c_{i+qn/p}$, $\hat{g}_i = \sum_{q=0}^{p-1} g_{i+qn/p}$ and $m = n/p$. Observe that $|\hat{c}| \leq p$ and $\hat{g}_i$'s are i.i.d. random variables chosen from the normal distribution with mean zero and variance $p\sigma$. Conditioned on the choice of $\hat{c}_i$s, the random variable $u_k = \sum_{i=0}^{m-1} \hat{c}_{i+k}\hat{g}_i$ has normal distribution with variance $\mu p\sigma$, where $\mu = \sum_{i=0}^{m-1}(\hat{c}_{i+k})^2$. We first show that $\mu \leq 4pm$ with very high probability, and then bound the tail of a normal random variable with variance $4pm$.

The following fact is adapted from [119], Theorem 3.1.

Fact F.3.2. *Let R_{ij}, $i = 0 \dots m-1$ and $j = 0 \dots p-1$, be i.i.d. random variable taking values uniformly at random from $\{-1, 1\}$. Let $T_i = 1/\sqrt{m} \sum_{j=0}^{p-1} R_{ij}$. There is an absolute constant C*

such that

$$\Pr[\sum_{i=0}^{m-1} T_i^2 \leq 4p] \geq 1 - 2e^{-m/C}$$

Applying the fact to $R_{ij} = c^{(k)}_{i+jn/p}$, we conclude that with probability $1 - o(1)$ we have $\mu \leq m\sum_{i=0}^{m-1} T_i^2 \leq 4pm = 4n$. In that case $\text{Var}[u_k] \leq 4np\sigma \leq 4npc(n)\frac{n}{p\ln n} = 4n^2 c(n)/\ln(n)$. We then follow the proof of Lemma F.1.1.

Event E_0 Observe that $v_{t'}$ is a sum of terms $q_i = (\sum_{q=0}^{p-1} c^{(t')}_{i+qn/p})^2$, where each q_i is a square of a sum of p independent random variables taking values in $\{-1, 1\}$ with probability $1/2$. It follows that $E[q_i] = p$, and therefore $E[v_{t'}] = n/p \cdot p = n$. To bound the deviation of $v_{t'}$ from its mean n, we first compute its variance.

$$\begin{aligned}
\text{Var}[v_k] &= \text{Var}[\sum_{i=0}^{n/p-1} q_i] \\
&\leq n/p \cdot E[q_1^2] \\
&= n/p \cdot E[(\sum_{q=0}^{p-1} c^{(t')}_{1+qn/p})^4] \\
&= n/p \cdot (\binom{4}{2} \sum_{q\neq q'} E[(c^{(t')}_{1+qn/p})^2 (c^{(t')}_{1+q'n/p})^2] + \sum_q E[(c^{(t')}_{1+qn/p})^4]) \\
&= n/p \cdot (6p(p-1) + p) \leq 7pn
\end{aligned}$$

We can now use Chebyshev's inequality:

$$\Pr[|v_{t'} - n| \geq n/4] \leq \text{Var}[v_{t'}]/(n/4)^2 \leq 7pn/(n/4)^2 = o(1)$$

Event E_3 It suffices to bound $\Pr[E_3]$. To this end we bound $\Pr[v_k \geq n/4]$. Without loss of generality we can assume $t = 0$. Then $v_k = \sum_{i=0}^{n/p-1} \hat{c}_i \hat{c}_{i+k}$, where $i + k$ is taken modulo n/p.

We first observe that in each term $\hat{c}_i \hat{c}_{i+k}$, the random variables $\hat{c}_i$ and $\hat{c}_{i+k}$ are independent (since $k \neq 0$). This implies $E[\hat{c}_i \hat{c}_{i+k}] = E[\hat{c}_i]E[\hat{c}_{i+k}] = 0$, and therefore $E[v_k] = 0$.

To bound $\Pr[v_k \geq n/4]$, we compute the fourth moment of $v_{t'}$ (using the second moment does not give strong enough probability bound). We have

$$\begin{aligned}
E[v_k^4] &= E[(\sum_{i=0}^{n/p-1} \hat{c}_i \hat{c}_{i+k})^4] \\
&= \sum_{i_1,i_2,i_3,i_4=0}^{n/p-1} E[\hat{c}_{i_1}\hat{c}_{i_1+k} \cdot \hat{c}_{i_2}\hat{c}_{i_2+k} \cdot \hat{c}_{i_3}\hat{c}_{i_3+k} \cdot \hat{c}_{i_4}\hat{c}_{i_1+k}]
\end{aligned}$$

Observe that the expectation of any term in the above sum that contains an odd power of $\hat{c}_i$ is zero. Hence the only remaining terms have the form $E[\hat{c}_{j_1}^2 \hat{c}_{j_2}^2 \hat{c}_{j_3}^2 \hat{c}_{j_4}^2]$, where $j_1 \ldots j_4$ are not necessarily

distinct. Let I be a set of such four-tuples (j_1, j_2, j_3, j_4). We observe that for (j_1, j_2, j_3, j_4) to belong in I, at least two disjoint pairs of indices in the sequence $i_1, i_1 + k, i_2, i_2 + k, i_3, i_3 + k, i_4, i_4 + k$ must be equal. This means that $|I| = C(n/p)^2$ for some constant C. Since $|\hat{c}_i| \leq p$, we have $E[v_k^4] \leq C(n/p)^2 p^8 \leq Cn^2 p^6$. Thus

$$\Pr[v_k \geq n/4] \leq E[v_k^4]/(n/4)^4 \leq \frac{Cn^2 p^6}{(n/4)^4} = C \cdot 4^4 \cdot p^6/n^2$$

This implies $\Pr[E_3] \leq n \Pr[v_k \geq n/4] \leq C \cdot 4^4 \cdot p^6/n$ which is $o(1)$ if $p = o(n^{1/6})$. □

We now show that if the noise variance σ is "small" then one can check each shift using few time domain samples.

Lemma F.3.3. *Assume that $\sigma \leq c(n)\frac{n}{p \ln n}$. Consider an algorithm that, given a set K of p shifts, computes*

$$a'_k = \mathbf{c}^{(k)}[0 \ldots T-1] \cdot \mathbf{x}[0 \ldots T-1]$$

for $T = n/p$, and selects the largest a_k over $k \in K$. Then

$$P''(\sigma) = \Pr[a_t \leq \max_{k \neq t} a_k] = o(1)$$

Proof. The argument is similar to the proof of Lemma F.1.1. We verify it for an analog of the event E_1; the proofs for E_2 and E_3 are straightforward syntactic modifications of the original arguments.

First, observe that $\mathbf{c}^{(t)}[0 \ldots T-1] \cdot \mathbf{c}^{(t)}[0 \ldots T-1] = T$. Let $u'_k = \mathbf{c}^{(k)}[0 \ldots T-1] \cdot \mathbf{g}[0 \ldots T-1]$. Consider the event $E'_1 : \exists_{k \in K} u'_k \geq T/3$. We can bound

$$\Pr[E'_1] \leq p \Pr[u'_k \geq T/3] = n(1 - \Phi(\frac{T/3}{\sqrt{\text{Var}[u'_k]}}))$$

Since $\text{Var}[u'_k] = T\sigma \leq c(n)\frac{n^2}{p^2 \ln n}$, we have

$$\Pr[E'_1] \leq p(1 - \Phi(\sqrt{\ln n/(9c(n))})) \leq e^{\ln n} e^{-\frac{\ln(n)}{18c(n)}} = o(1)$$

□

Proofs of Theorem 8.4.3 and Theorem 8.4.4 To prove Theorem 8.4.3, recall that given a signal $\mathbf{x}$ consisting of p blocks, each of length n and with noise variance σ, QuickSync starts by aliasing the p blocks into one. This creates one block of length n, with noise variance σ/p (after normalization). We then apply Lemmas F.3.1 and F.3.3.

Theorem 8.4.4 follows from the assumption that $\sigma = c(n)\frac{n}{p \ln n}$ and Lemma F.3.1.

Appendix G

A 0.75 Million Point Sparse Fourier Transform Chip

In this appendix, we present the first VLSI implementation of the Sparse Fourier Transform algorithm. The chip implements a 746,496-point Sparse Fourier Transform, in 0.6mm^2 of silicon area. At 0.66V, it consumes 0.4pJ/sample and has an effective throughput of 36GS/s. The effective throughput is computed over all frequencies but frequencies with negligible magnitudes are not produced. The chip works for signals that occupy up to 0.1% of the transform frequency range (0.1% sparse). It can be used to detect a signal that is frequency hopping in a wideband, to perform pattern matching against a long code, or to detect a blocker location with very high frequency resolution. For example, it can detect and recover a signal that occupies 18 MHz randomly scattered anywhere in an 18 GHz band with a frequency resolution of $\approx$ 24 kHz.

G.1 The Algorithm

We start by describing the Sparse Fourier Transform algorithm implemented on this chip. Below are bucketization, estimation, and collision resolution techniques we used for our implementation.

Bucketization: The algorithm starts by mapping the spectrum into buckets. This is done by sub-sampling the signal and then performing an FFT. Sub-sampling in time causes aliasing in frequency. Since the spectrum is sparsely occupied, most buckets will be either empty or have a single active frequency, and only few buckets will have a collision of multiple active frequencies. Empty buckets are discarded and non-empty buckets are passed to the estimation step.

Estimation: This step estimates the value and frequency number (i.e. location in the spectrum) of each active frequency. In the absence of a collision, the value of an active frequency is the value of its bucket. To find the frequency number, the algorithm repeats the bucketization on the original signal after shifting it by 1 sample. A shift in time causes a phase change in the frequency domain of $2\pi f\tau/N$, where f is the frequency number, τ is the time shift, and N is the Sparse Fourier Transform size. Thus, the phase change can be used to compute the frequency number.

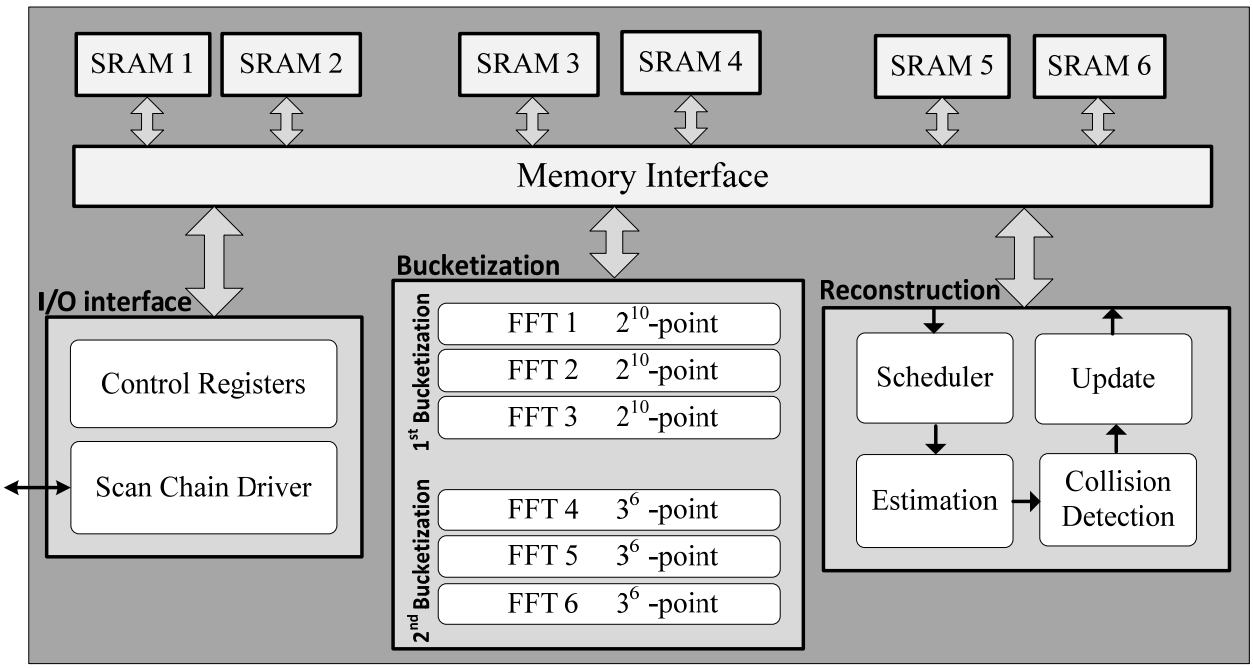


Figure G-1: **A Block Diagram of the $2^{10} \times 3^6$-point Sparse Fourier Transform:** The I/O interface, Bucketization and Reconstruction blocks operate in parallel on three different Sparse Fourier Transform frames. The figure also shows the input samples to the six FFTs.

Collision Resolution: The algorithm detects collisions as follows: If a bucket contains a collision then repeating the bucketization with a time shift causes the bucket's magnitude to change since the colliding frequencies rotate by different phases. In contrast, the magnitude does not change if the bucket has a single active frequency. After detecting collisions, the algorithm resolves them by using bucketization multiple times with co-prime sampling rates (FFTs with co-prime sizes). The use of co-prime sampling rates guarantees that any two frequencies that collide in one bucketization do not collide in other bucketizations.

G.2 The Architecture

The block diagram of the Sparse Fourier Transform chip is shown in Figure G-1. A 12-bit 746,496-point ($2^{10} \times 3^6$-point) Sparse Fourier Transform is implemented. Two types of FFTs (2^{10} and 3^6-point) are used for bucketization. The input to the 2^{10}-point FFT is the signal sub-sampled by 3^6, while the input to the 3^6-point FFT is the signal sub-sampled by 2^{10}. FFTs of sizes 2^{10} and 3^6 were chosen since they are co-prime and can be implemented with simple low-radix FFTs. Three FFTs of each size are used with inputs shifted by 0, 1 or 32 time samples, as shown in Figure G-1. In principle, shifts of 0 and 1 are sufficient. However, the third shift is used to increase the estimation accuracy. One 1024-word and one 729-word SRAMs are used for three 2^{10}-point and three 3^6-point FFTs, respectively. SRAMs are triplicated to enable pipelined operation of the I/O interface, bucketization and reconstruction blocks. Thus, 3 Sparse Fourier Transform frames exist in the

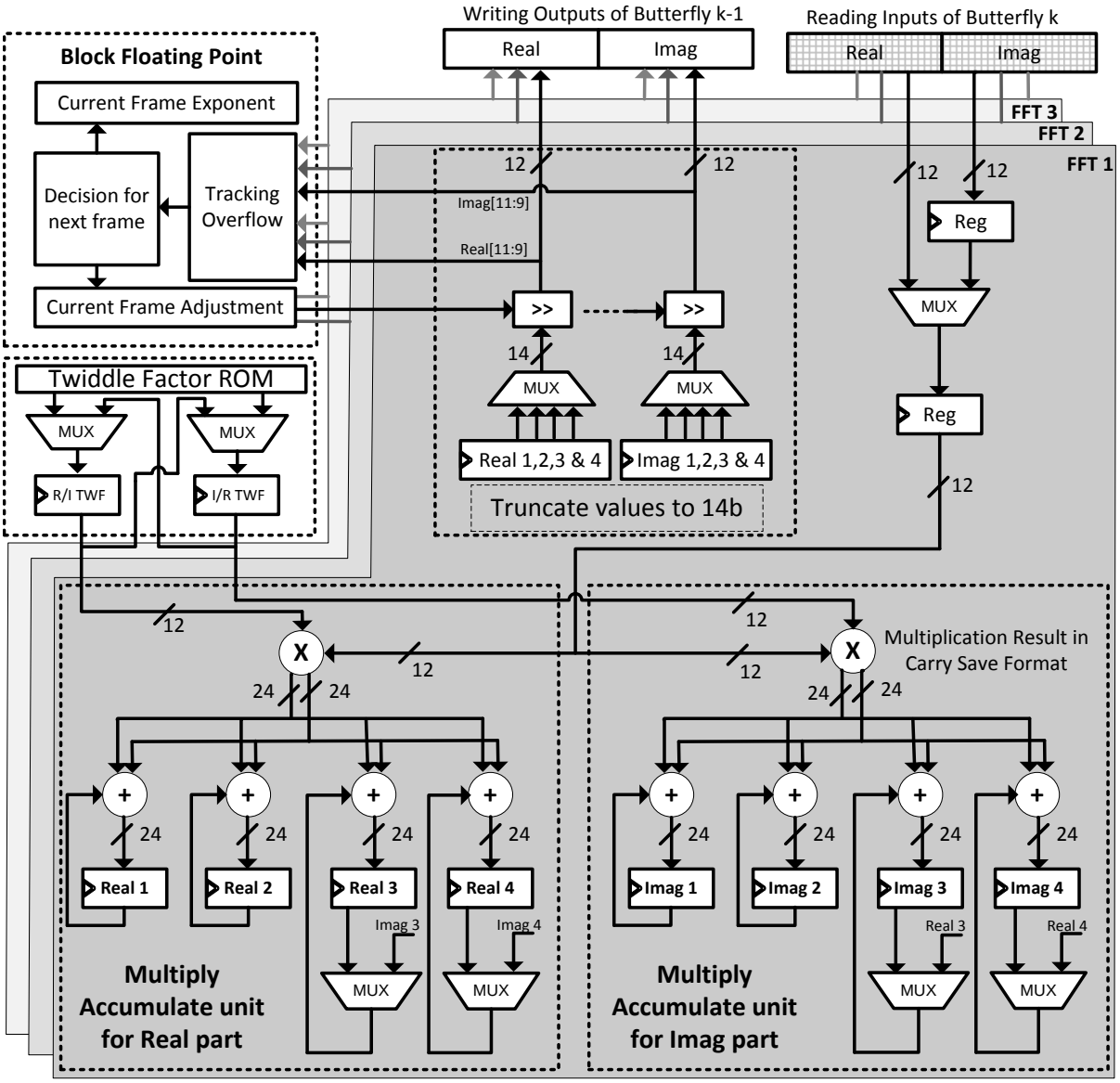


Figure G-2: **The Micro-Architecture of the 2^{10}-point FFT's.** Block floating point (BFP) is used to reduce the bit-width requirement during the computation of the FFT, while maintaining the required resolution at the output.

pipeline.

The micro-architecture of the 2^{10}-point FFT is shown in Figure G-2. Each 2^{10}-point FFT uses one radix-4 butterfly to perform an in-place FFT, which is optimized to reduce area and power consumption as follows: First, the FFT block performs read and write operations at even and odd clock cycles, respectively, which enables the use of single port SRAMs. A single read operation provides three complex values, one for each radix-4 butterfly. The complex multiplication is computed over two clock cycles using two multipliers for each butterfly. Second, a twiddle factor (TWF) control

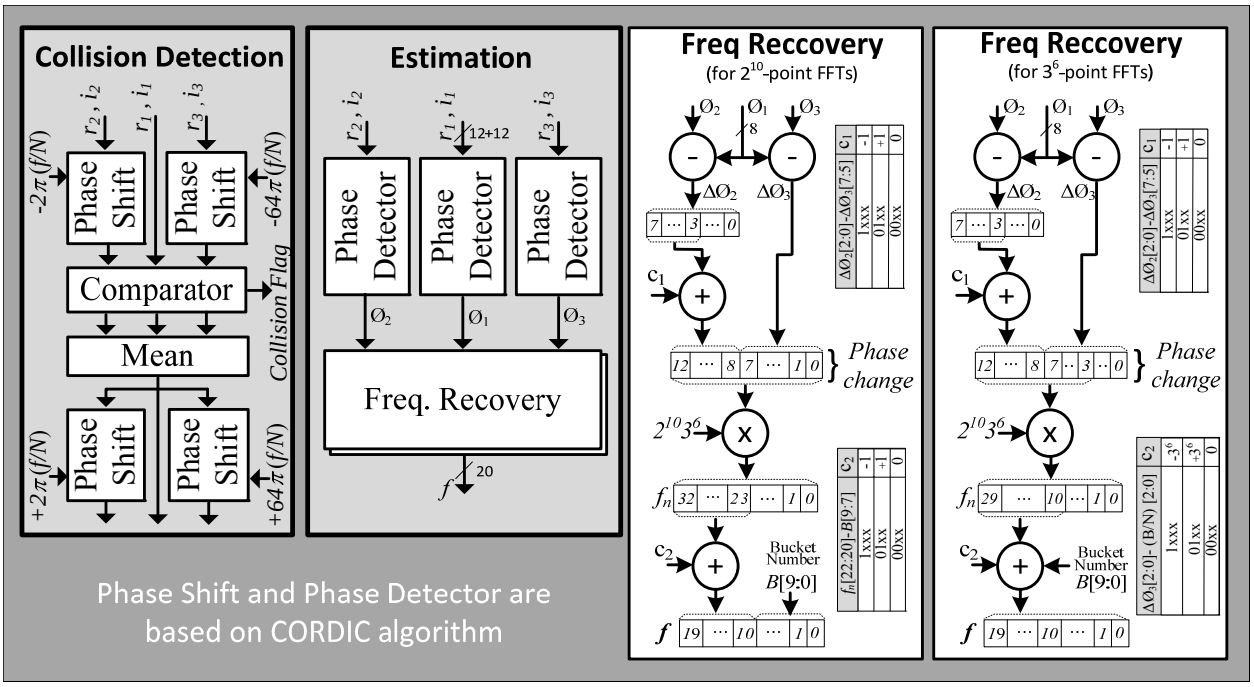


Figure G-3: **The Micro-Architecture of Collision Detection, Estimation, and Frequency Recovery.** The complex values (r_1, i_1), (r_2, i_2) and (r_3, i_3) are the output of Bucketization for time-shifts 0, 1 and 32 samples. In Frequency Recovery, 3 bits of overlap are used to fix errors due to concatenation (c_1, c_2).

unit is shared between the three butterflies. Third, the block floating point (BFP) technique is used to minimize the quantization error [189]. BFP is implemented using a single exponent shared between FFTs, and scaling is done by shifting in case of overflow. Round-half-away-from-zero is implemented by initializing the accumulator registers with 0.5LSB and truncating the results. The 3^6-point FFTs are similar, but use radix-3 butterflies.

The micro-architecture of estimation and collision detection is shown in Figure G-3. Phase shift and phase detector units use the CORDIC algorithm. The estimation block operates in two steps. First, time shifts of 1 and 32 samples are used to compute the MSBs and LSBs of the phase change, respectively. A 3-bit overlap is used to fix errors due to concatenation. Since the 5 MSBs of phase change are taken directly from the output of phase detectors, active frequencies have to be $\approx$ 30dB above the quantization noise to be detected correctly. Frequencies below this level are considered negligible. The frequency number is estimated from the phase change. This frequency number may have errors in the LSBs due to quantization noise. The second step corrects any such errors by using the bucket number to recover the LSBs of the frequency number. This is possible because all frequencies in a bucket share the same remainder B ($B = f \mod M$, where f is the frequency number and M is the FFT size), which is also the bucket number. Thus, in the frequency recovery block associated with the 2^{10}-point FFTs, the bucket number gives the 10 LSBs of the frequency number. However, in the frequency recovery for the 3^6-point FFTs, the LSBs cannot be directly replaced by the bucket number since $M = 3^6$ is not a power of 2. Instead, the remainder of

Technology	45 nm SOI CMOS
Core Area	0.6 mm $\times$ 1.0 mm
SRAM	3$\times$75 kbits and 3$\times$54 kbits
Core Supply Voltage	0.66-1.18V
Clock Frequency	0.5-1.5 GHz
Core Power	14.6-174.8 mW

Table G.1: **Sparse Fourier Transform Chip Features**

dividing the frequency number by 3^6 is calculated and subtracted from the frequency number. The bucket number is then added to the result of the subtraction. In our implementation, calculating and subtracting the remainder is done indirectly by truncating the LSBs of the phase change.

The collision detection block in Figure G-3 compares the values of the buckets with and without time-shifts. It uses the estimated frequency to remove the phase change in the time-shifted bucketizations and compares the three complex values to detect collisions. In the case of no collision, the three values are averaged to reduce noise. The result is used to update the output of the Sparse Fourier Transform in SRAMs.

G.3 The Chip

The testchip is fabricated in IBM's 45nm SOI technology. Table G.1 shows the features of the Sparse Fourier Transform chip and Figure G-4 shows the die photo of the testchip. The Sparse Fourier Transform core occupies 0.6mm^2 including SRAMs. At 1.18V supply, the chip operates at a maximum frequency of 1.5 GHz, resulting in an effective throughput of 109 GS/s. At this frequency, the measured energy efficiency is 1.2μJ per 746,49-point Fourier transform. Reducing the clock frequency to 500 MHz enables an energy efficiency of 298nJ per Fourier transform at 0.66V supply. Energy and operating frequency for a range of supply voltages are shown in Figure G-5.

Since no prior ASIC implementations of the Sparse Fourier Transform exist, we compare with recent low power implementations of the traditional FFT [29, 155, 182]. The measured energy is normalized by the Fourier transform size to obtain the energy per sample (the Sparse Fourier Transform chip, however, outputs only active frequencies). Table G.2 shows that the implementations in [29, 155, 182] work for sparse and non-sparse signals while the Sparse Fourier Transform chip works for signal sparsity up to 0.1%. However, for such sparse signals, the chip delivers $\approx 40\times$ lower energy per sample for a 36$\times$ larger FFT size. Finally, the 746,496-point Sparse Fourier Transform chip runs in 6.8μs when operated at 1.5 GHz which corresponds to an 88$\times$ reduction in runtime compared to the C++ implementation that takes 600μs on an Intel-Core i7 CPU operating at 3.4 GHz.

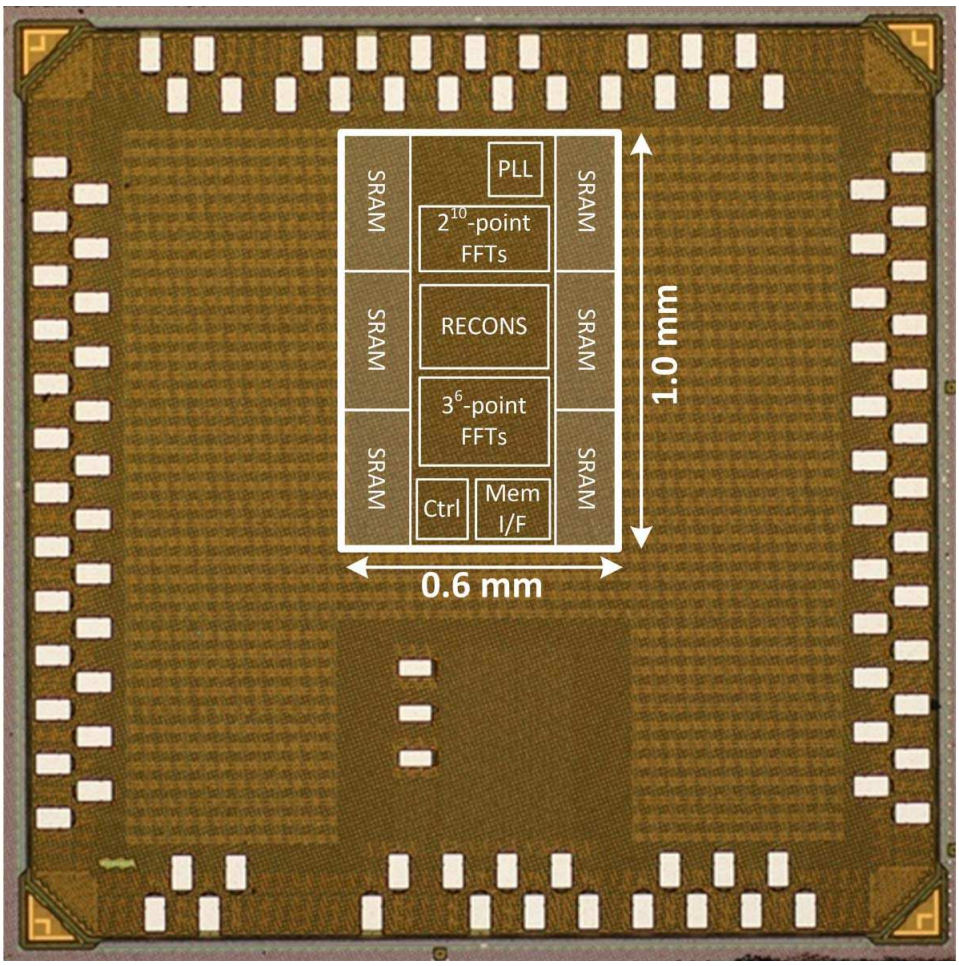


Figure G-4: **Die Photo of the Sparse Fourier Transform Chip**

	[155]	[29]	[182]	**This Chip**
Technology	65 nm	90 nm	65 nm	45 nm
Signal Type	Any Signal	Any Signal	Any Signal	Sparse Signal
Size	2^{10}	2^8	2^7 to 2^{11}	$3^6 \times 2^{10}$
Word Width	16 bits	10 bits	12 bits	12 bits
Area	8.29 mm^2	5.1 mm^2	1.37 mm^2	0.6 mm^2
Throughput	240 MS/s	2.4 GS/s	1.25-20 MS/s	36.4-109.2 GS/s
Energy/Sample	17.2 pJ	50 pJ	19.5-50.6 pJ	0.4-1.6 pJ

Table G.2: **Comparison of Sparse Fourier Transform Chip with FFT Chips** The measured energy efficiency and performance of the Sparse Fourier Transform chip compared to published FFTs. For applications with frequency-sparse signals, the Sparse Fourier Transform enables 43$\times$ lower energy per sample.

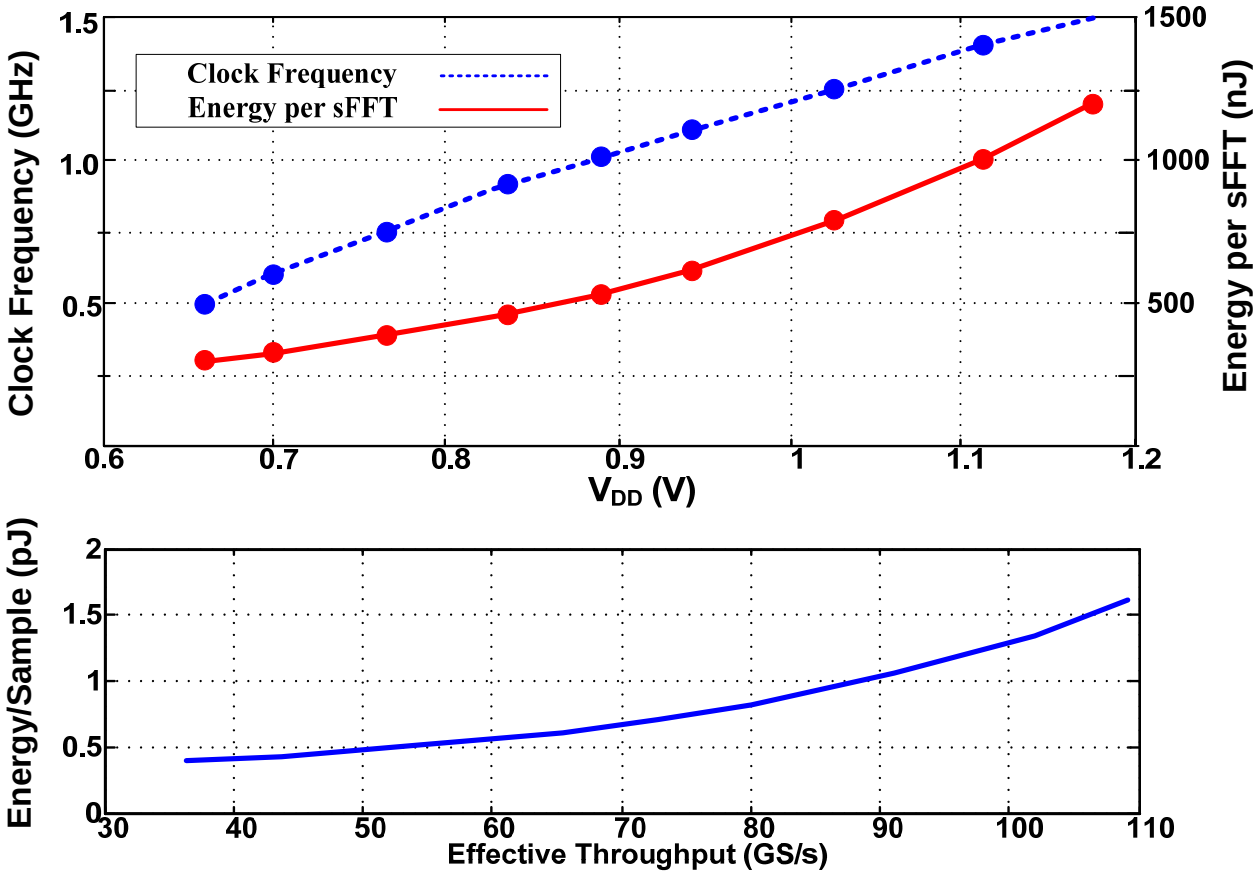


Figure G-5: **Chip Measurement Results.** Energy and frequency of operation for a range of voltage, and throughput versus Energy per sample for computing a $2^{10} \times 3^6$-point Sparse Fourier Transform.

Bibliography

[1] O. Abari, F. Chen, F. Lim, and V. Stojanovic. Performance trade-offs and design limitations of analog-to-information converter front-ends. In *ICASSP*, 2012.

[2] O. Abari, F. Lim, F. Chen, and V. Stojanovic. Why analog-to-information converters suffer in high-bandwidth sparse signal applications. *IEEE Transactions on Circuits and Systems I*, 2013.

[3] Omid Abari, Ezzeldin Hamed, Haitham Hassanieh, Abhinav Agarwal, Dina Katabi, Anantha Chandrakasan, and Vladimir Stojanovic. 27.4 a 0.75-million-point fourier-transform chip for frequency-sparse signals. In *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers*, ISSCC'14, pages 458–459, Feb 2014.

[4] E. H. Adelson and J. Y. A. Wang. Single lens stereo with a plenoptic camera. *IEEE PAMI*, 14(2):99–106, February 1992.

[5] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. *Int. Conf. on Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[6] A. Akavia. Deterministic sparse fourier approximation via fooling arithmetic progressions. *COLT*, pages 381–393, 2010.

[7] A. Akavia, S. Goldwasser, and S. Safra. Proving hard-core predicates using list decoding. *FOCS*, pages 146–, 2003.

[8] M. Akcakaya and V. Tarokh. A frame construction and a universal distortion bound for sparse representations. *Signal Processing, IEEE Transactions on*, 56(6):2443 –2450, june 2008.

[9] O. C. Andronesi, G. S. Kim, E. Gerstner, T. Batchelor, A. A. Tzika, V. R. Fantin, M. G. Vander Heiden, and A. G. Sorensen. Detection of 2-Hydroxyglutarate in IDH-mutated Glioma Patients by Spectral-editing and 2D Correlation Magnetic Resonance Spectroscopy. *Science Translational Medicine*, 4(116), 2012.

[10] O. C. Andronesi, S. Ramadan, C. E. Mountford, and A. G. Sorensen. Low-Power Adiabatic Sequences for In Vivo Localized Two-Dimensional Chemical Shift Correlated MR Spectroscopy. *Magnetic Resonance in Medicine*, 64:1542–1556, 2010.

[11] Ovidiu C. Andronesi, Borjan A. Gagoski, Elfar Adalsteinsson, and A. Gregory Sorensen. Correlation Chemical Shift Imaging with Low-Power Adiabatic Pulses and Constant-Density Spiral Trajectories. *NMR Biomedicine*, 25(2):195–209, 2012.

[12] P.C. Aoto, R.B. Fenwick, G.J.A. Kroon, and P.E. Wright. Accurate Scoring of Non-Uniform Sampling Schemes for Quantitative NMR. *Journal of Magnetic Resonance*, 246:31–35, 2014.

[13] Paramvir Bahl, Ranveer Chandra, Thomas Moscibroda, Rohan Murty, and Matt Welsh. White space networking with wi-fi like connectivity. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, 2009.

[14] V. Bahskarna and K. Konstantinides. *Image and Video Compression Standards : Algorithms and Architectures*. Kluwer Academic Publishers, 1995.

[15] J.C.J. Barna, E.D. Laue, M.R. Mayger, J. Skilling, and S.J.P. Worrall. Exponential Sampling, an Alternative Method for Sampling in Two-Dimensional NMR Experiments. *Journal of Magnetic Resonance*, 73:69–77, 1987.

[16] T. Baykas, M. Kasslin, M. Cummings, Hyunduk Kang, J. Kwak, R. Paine, A. Reznik, R. Saeed, and S.J. Shellhammer. Developing a standard for TV white space coexistence. *IEEE Wireless Communications*, 19(1), 2012.

[17] M. Billeteri and V.Y. Orekhov. Preface: Fast NMR Methods Are Here to Stay. In:Novel Sampling Approaches in Higher Dimensional NMR,. *Springer*, 316:ix–xiv, 2012.

[18] Tom E. Bishop, Sara Zanetti, and Paolo Favaro. Light field superresolution. In *In IEEE ICCP*, 2009.

[19] G. Bodenhausen and R.R. Ernst. The Accordion Experiment, A Simple Approach to 3-Dimensional NMR Spectroscopy. *Journal of Magnetic Resonance*, 45:367–373, 1981.

[20] R.N. Bracewell. Strip Integration in Radio Astronomy. *Aust. Journal of Physics*, 9:198–217, 1956.

[21] G.L. Bretthorst. Bayesian-Analysis. 1. Parameter-Estimation Using Quadrature NMR Models. *Journal of Magnetic Resonance*, 88:533–551, 1990.

[22] Bernhard Buchli, Felix Sutton, and Jan Beutel. GPS-equipped wireless sensor network node for high-accuracy positioning applications. In *EWSN 2012*, Trento, Italy.

[23] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *SIGGRAPH '01*, pages 425–432. ACM, 2001.

[24] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low-cost outdoor localization for very small devices. *IEEE Personal Communications*, Oct. 2000.

[25] E. Candes and T. Tao. Near optimal signal recovery from random projections: Universal encoding strategies. *IEEE Trans. on Info.Theory*, 2006.

[26] E.J. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory,*, 52(2):489–509, 2006.

[27] Y. Chan and V. Koo. An Introduction to Synthetic Aperture Radar (SAR). *Progress In Electromagnetics Research B*, 2008.

[28] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos. Data driven signal processing: An approach for energy efficient computing. *International Symposium on Low Power Electronics and Design*, 1996.

[29] Yuan Chen, Yu-Wei Lin, Yu-Chi Tsao, and Chen-Yi Lee. A 2.4-Gsample/s DVFS FFT Processor for MIMO OFDM Communication Systems. *IEEE Journal of Solid-State Circuits,*, 43(5):1260–1273, May 2008.

[30] C. Cheng and K. Parhi. Low-cost fast VLSI algorithm for discrete fourier transform. *IEEE Transactions on Circuits and Systems*, April 2007.

[31] R.A. Chylla and J.L. Markley. Theory And Application of The Maximum-Likelihood Principle To NMR Parameter-Estimation of Multidimensional NMR Data. *Journal of Biomolecular NMR*, 5:245–258, 1995.

[32] Barry A. Cipra. The Best of the 20th Century: Editors Name Top 10 Algorithms. *SIAM News*, 33(4), 2000.

[33] B.E. Coggins, R.A. Venters, and P. Zhou. Radial Sampling for Fast NMR: Concepts and Practices Over Three Decades. *Progress in Nuclear Magnetic Resonance*, 57:381–419, 2010.

[34] J.W. Cooley and J.W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Comp.*, 19:297–301, 1965.

[35] G. Cormode and S. Muthukrishnan. Combinatorial algorithms for compressed sensing. *SIROCCO*, 2006.

[36] Thomas Cover and Joy Thomas. *Elements of Information Theory*. Wiley Interscience, 1991.

[37] Abe Davis, Marc Levoy, and Fredo Durand. Unstructured light fields. *Comp. Graph. Forum*, 31(21):305–314, May 2012.

[38] F. Delaglio, S. Grzesiek, G.W. Vuister, G. Zhu, J. Pfeifer, and A. Bax. NMRPipe: a Multidimensional Spectral Processing System Based on UNIX Pipes. *Journal of Biomolecular NMR*, 6:277–293, 1995.

[39] Dexter Industries. dGPS for LEGO MINDSTORMS NXT. http://dexterindustries.com.

[40] DigiKey, ADCs. http://www.digikey.com/.

[41] G.M. Djuknic and R.E. Richton. Geolocation and assisted GPS. *Computer*, 34(2):123 –125, feb 2001.

[42] Frédo Durand, Nicolas Holzschuch, Cyril Soler, Eric Chan, and François X. Sillion. A frequency analysis of light transport. *ACM Transactions on Graphics*, 24(3):1115–1126, August 2005.

[43] Kevin Egan, Yu-Ting Tseng, Nicolas Holzschuch, Frédo Durand, and Ravi Ramamoorthi. Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Transactions on Graphics*, 28(3):93:1–93:13, July 2009.

[44] H.R. Eghbalnia, A. Bahrami, M. Tonelli, K. Hallenga, and J.L. Markley. High-Resolution Iterative Frequency Identification for NMR as a General Strategy for Multidimensional Data Collection. *Journal of the American Chemical Society*, 127:12528–12536, 2005.

[45] Sparkfun Electronics. Sige gn3s sampler v3. http://www.sparkfun.com.

[46] A. Elkhaled, L. E. Jalbert, J. J. Phillips, H. A. I. Yoshihara, R. Parvataneni, R. Srinivasan, G. Bourne, M. S. Berger, S. M. Chang, S. Cha, and S. J. Nelson. Magnetic Resonance of 2-Hydroxyglutarate in IDH1-Mutated Low-Grade Gliomas. *Science Translational Medicine*, 4(116), 2012.

[47] Ettus Inc. USRP. http://ettus.com.

[48] FCC, Second Memorandum Opinion & Order 10-174. http://www.fcc.gov/encyclopedia/white-space-database-administration.

[49] C. Fernandez-Prades, J. Arribas, P. Closas, C. Aviles, and L. Esteve. GNSS-SDR: an open source tool for researchers and developers. In *ION GNSS Conference*, 2011.

[50] Glenn Fleishman. How the iphone knows where you are. Macworld, Aug. 2011.

[51] Glenn Fleishman. Inside assisted GPS: helping GPS help you. Arstechnica, Jan. 2009.

[52] R. Freeman and E. Kupce. New Methods for Fast Multidimensional NMR. *Journal of Biomolecular NMR*, 27:101–113, 2003.

[53] M.A. Frey, Z.M. Sethna, G.A. Manley, S. Sengupta, K.W. Zilm, J.P. Loria, and S.E. Barrettm. Accelerating Multidimensional NMR and MRI Experiments Using Iterated Maps. *Journal of Magnetic Resonance*, 237:100–109, 2013.

[54] M. Frigo and S. G. Johnson. FFTW 3.2.3. http://www.fftw.org/.

[55] Todor Georgeiv, Ke Colin Zheng, Brian Curless, David Salesin, Shree Nayar, and Chintan Intwala. Spatio-angular resolution tradeoff in integral photography. In *In Eurographics Symposium on Rendering*, pages 263–272, 2006.

[56] Badih Ghazi, Haitham Hassanieh, Piotr Indyk, Dina Katabi, Eric Price, and Lixin Shi. Sample-Optimal Average-Case Sparse Fourier Transform in Two Dimensions. In *Proceedings of the 51st Annual Allerton Conference on Communication, Control, and Computing*, Allerton'13, pages 1258–1265, Oct 2013.

[57] A. Gilbert, S. Guha, P. Indyk, M. Muthukrishnan, and M. Strauss. Near-optimal sparse fourier representations via sampling. *STOC*, 2002.

[58] A. Gilbert and P. Indyk. Sparse recovery using sparse matrices. *Proceedings of IEEE*, 2010.

[59] A. Gilbert, M. Muthukrishnan, and M. Strauss. Improved time bounds for near-optimal space fourier representations. *SPIE Conference, Wavelets*, 2005.

[60] A. Gilbert, M. Muthukrishnan, and M. Strauss. Improved time bounds for near-optimal space fourier representations. In *SPIE*, 2005.

[61] A.C. Gilbert, M.J. Strauss, and J. A. Tropp. A tutorial on fast fourier sampling. *Signal Processing Magazine*, 2008.

[62] Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. Approximate sparse recovery: optimizing time and measurements. In *STOC*, pages 475–484, 2010.

[63] O. Goldreich and L. Levin. A hard-corepredicate for allone-way functions. pages 25–32, 1989.

[64] Andrea Goldsmith. *Wireless Communications*. Cambridge University Press, 2005.

[65] Joseph W. Goodman. *Introduction To Fourier Optics*. McGraw-Hill, 1996.

[66] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *SIGGRAPH '96*, pages 43–54, New York, NY, USA, 1996. ACM.

[67] Sherrie Gossett. GPS implant makes debut. WND, May 2003.

[68] Y. M. Greshishchev, J. Aguirre, M. Besson, R. Gibbins, C. Falt, P. Flemke, N. Ben-Hamida, D. Pollex, P. Schvan, and S. C. Wang. A 40 gs/s 6b adc in 65 nm cmos. In *IEEE Solid-State Circuits Conference (ISSCC)*, 2010.

[69] Haitham Hassanieh, Fadel Adib, Dina Katabi, and Piotr Indyk. Faster GPS via the Sparse Fourier Transform. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, MOBICOM '12, pages 353–364, New York, NY, USA, 2012. ACM.

[70] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly Optimal Sparse Fourier Transform. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 563–578, New York, NY, USA, 2012. ACM.

[71] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. SFFT: Sparse Fast Fourier Transform. 2012. http://www.sparsefft.com.

[72] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Simple and Practical Algorithm for Sparse Fourier Transform. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1183–1194. SIAM, 2012.

[73] Haitham Hassanieh, Maxim Mayzel, Lixin Shi, Dina Katabi, and VladislavYu Orekhov. Fast multi-dimensional NMR acquisition and processing using the sparse FFT. *Journal of Biomolecular NMR*, 63(1):9–19, 2015.

[74] Haitham Hassanieh, Lixin Shi, Omid Abari, Ezzeldin Hamed, and Dina Katabi. GHz-Wide Sensing and Decoding Using the Sparse Fourier Transform. In *Proceedings of the IEEE International Conference on Computer Communications*, INFOCOM'14, pages 2256–2264, April 2014.

[75] Paul Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, University of California at Berkeley, Computer Science Division, June 1989.

[76] J. Heiskala and J. Terry. *OFDM Wireless LANs: A Theoretical and Practical Guide*. Sams Publishing, 2001.

[77] Juha Heiskala and John Terry, Ph.D. *OFDM Wireless LANs: A Theoretical and Practical Guide*. Sams, Indianapolis, IN, USA, 2001.

[78] Aden Hepburn. Infographic: Mobile stats & facts 2011. Digital Buzz, April 2011.

[79] C. Herley and P.W. Wong. Minimum rate sampling and reconstruction of signals with arbitrary frequency support. *IEEE Transactions on Information Theory*, 45(5):1555–1564, 1999.

[80] S. Hiller, F. Fiorito, K. Wuthrich, and G. Wider. Automated Projection Spectroscopy (APSY). *Proceedings of the National Academy of Science USA*, 102:10876–10881, 2005.

[81] S. Hiller, C. Wasmer, G. Wider, and K. Wuthrich. Sequence-Specific Resonance Assignment of SolubleNnonglobular Proteins by 7D APSY-NMR Spectroscopy. *Journal of the American Chemical Society*, 129:10823–108–28, 2007.

[82] J.C. Hoch, M.W. Maciejewski, M. Mobli, A.D. Schuyler, and A.S. Stern. Nonuniform Sampling and Maximum Entropy Reconstruction in Multidimensional NMR. *Accounts of Chemical Research*, 47:708–717, 2014.

[83] D.J. Holland, M.J. Bostock, L.F. Gladden, and D. Nietlispach. Fast Multidimensional NMR Spectroscopy Using Compressed Sensing. *Angewandte Chemie International Edition*, 50:6548–6551, 2011.

[84] Steven Siying Hong and Sachin Rajsekhar Katti. DOF: a local wireless information plane. In *ACM SIGCOMM*, 2011.

[85] S.G. Hyberts, H. Arthanari, S.A. Robson, and G. Wagner. Perspectives in Magnetic Resonance: NMR in the Post-FFT Era. *Journal of Magnetic Resonance*, 241:60–73, 2014.

[86] P. Indyk, E. Price, and D. P. Woodruff. On the power of adaptivity in sparse recovery. *FOCS*, 2011.

[87] M. Iwen. AAFFT (Ann Arbor Fast Fourier Transform). 2008. http://sourceforge.net/projects/aafftannarborfa/.

[88] M. A. Iwen. Combinatorial sublinear-time fourier algorithms. *Foundations of Computational Mathematics*, 10:303 – 338, 2010.

[89] M. A. Iwen, A. Gilbert, and M. Strauss. Empirical evaluation of a sub-linear time sparse dft algorithm. *Communications in Mathematical Sciences*, 5, 2007.

[90] M.A. Iwen. Improved approximation guarantees for sublinear-time fourier algorithms. *Arxiv preprint arXiv:1010.0014*, 2010.

[91] Jackson Labs, Fury GPSDO. http://jackson-labs.com/.

[92] Jimmy LaMance Jani Jarvinen, Javier DeSalas. Assisted GPS: A low-infrastructure approach. GPSWorld, March 2002.

[93] V. Jaravine, I. Ibraghimov, and V.Y. Orekhov. Removal of a Time Barrier for High-Resolution Multidimensional NMR Spectroscopy. *Nature Methods*, 3:605–607, 2006.

[94] V.A. Jaravine and V.Y. Orekhov. Targeted Acquisition for Real-Time NMR Spectroscopy. *Journal of the American Chemical Society*, 128:13421–13426, 2006.

[95] J. Kahn, G. Kalai, and N. Linial. The influence of variables on boolean functions. *FOCS*, 1988.

[96] Elliott D. Kaplan. *Understanding GPS Principles and Applications*. Artech House Publishers, February 1996.

[97] M. Karim and M. Sarraf. *W-CDMA and CDMA2000 for 3G mobile networks*. McGraw-Hill, 2002.

[98] K. Kazimierczuk, W. Kozminski, and I. Zhukov. Two-dimensional Fourier Rransform of Arbitrarily Sampled NMR Data Sets. *Journal of Magnetic Resonance*, 179:323–328, 2006.

[99] K. Kazimierczuk and V.Y. Orekhov. Accelerated NMR Spectroscopy by Using Compressed Sensing. *Angewandte Chemie International Edition*, 50:5556–5559, 2011.

[100] K. Kazimierczuk, A. Zawadzka-Kazimierczuk, and W. Kozminski. Non-Uniform Frequency Domain for Optimal Exploitation of Non-Uniform Sampling. *Journal of Magnetic Resonance*, 205:286–292, 2010.

[101] E. Kupce and R. Freeman. Projection-Reconstruction Technique for Speeding Up Multidimensional NMR Spectroscopy. *Journal of the American Checmial Society*, 126:6429–6440, 2004.

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

[103] J.N. Laska, W.F. Bradley, T.W. Rondeau, K.E. Nolan, and B. Vigoda. Compressive sensing for dynamic spectrum access networks: Techniques and tradeoffs. In *IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), 2011*, 2011.

[104] D. Lawlor, Y. Wang, and A. Christlieb. Adaptive sub-linear time fourier algorithms. *arXiv:1207.6368*, 2012.

[105] E. Lescop, P. Schanda, and B. Brutscher. A Set of BEST Triple-Resonance Experiments for Time-Optimized Protein Resonance Assignment. *Journal of Magnetic Resonance*, 187:163–169, 2007.

[106] A. Levin and F. Durand. Linear view synthesis using a dimensionality gap light field prior. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1831–1838, 2010.

[107] Anat Levin, Rob Fergus, Frédo Durand, and William T. Freeman. Image and depth from a conventional camera with a coded aperture. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[108] Anat Levin, William T. Freeman, and FrÂŐdo Durand. Understanding camera trade-offs through a bayesian analysis of light field projections. *MIT CSAIL TR*, 2008.

[109] Anat Levin, Samuel W Hasinoff, Paul Green, Frédo Durand, and William T Freeman. 4d frequency analysis of computational cameras for depth of field extension. In *ACM Transactions on Graphics (TOG)*, volume 28, page 97. ACM, 2009.

[110] Anat Levin, Peter Sand, Taeg Sang Cho, Frédo Durand, and William T Freeman. Motion-invariant photography. In *ACM Transactions on Graphics (TOG)*, volume 27, page 71. ACM, 2008.

[111] Marc Levoy and Pat Hanrahan. Light field rendering. In *SIGGRAPH '96*, pages 31–42, New York, NY, USA, 1996. ACM.

[112] Mengda Lin, A. P. Vinod, and Chong Meng Samson See. A new flexible filter bank for low complexity spectrum sensing in cognitive radios. *Journal of Signal Processing Systems*, 62(2):205–215, 2011.

[113] Jie Liu, Bodhi Priyantha, Ted Hart, Heitor Ramos, Antonio Loureiro, and Qiang Wang. Energy Efficient GPS Sensing with Cloud Offloading. In *SenSys*, 2014.

[114] M. Lustig, D.L. Donoho, J.M. Santos, and J.M. Pauly. Compressed sensing mri. *Signal Processing Magazine, IEEE*, 25(2):72–82, 2008.

[115] Richard Lyons. *Understanding Digital Signal Processing*. 1996.

[116] Y. Mansour. Randomized interpolation and approximation of sparse polynomials. *ICALP*, 1992.

[117] G. Marsaglia. Evaluating the normal distribution. *Journal of Statistical Software*, 11(4):1–7, 2004.

[118] K. Marwah, G. Wetzstein, Y. Bando, and R. Raskar. Compressive Light Field Photography using Overcomplete Dictionaries and Optimized Projections. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 32(4):1–11, 2013.

[119] J. Matousek. On variants of the johnson-lindenstrauss lemma. *Random Structures & Algorithms*, 33(2), 2008.

[120] Y. Matsuki, M.T. Eddy, and J. Herzfeld. Spectroscopy by Integration of Frequency and Time Domain Information for Fast Acquisition of High-Resolution Dark Spectra. *Journal of the American Checmial Society*, 131:4648–4656, 2009.

[121] Maxim IC. MAX2745 single-chip global positioning system front-end downconverter. http://www.maxim-ic.com.

[122] M. Mayzel, K. Kazimierczuk, and V.Y. Orekhov. The Causality Principle in the Reconstruction of Sparse NMR Spectra. *Chemical Communications*, 50:8947–8950, 2014.

[123] Microsoft Spectrum Observatory. http://spectrum-observatory.cloudapp.net.

[124] M. Mishali and Y.C. Eldar. From Theory to Practice: Sub-Nyquist Sampling of Sparse Wideband Analog Signals. *IEEE Journal of Selected Topics in Signal Processing*, 4(2):375–391, 2010.

[125] M. Mishali and Y.C. Eldar. Wideband Spectrum Sensing at Sub-Nyquist Rates. *IEEE Signal Processing Magazine*, 28(4):102–135, 2011.

[126] Don P. Mitchell. Spectrally optimal sampling for distributed ray tracing. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, volume 25, pages 157–164, July 1991.

[127] M. Mobli, A.S. Stern, and J.C. Hoch. Spectral Reconstruction Methods in Fast NMR: Reduced Dimensionality, Random Sampling and Maximum Entropy. *Journal of Magnetic Resonance*, 182:96–105, 2006.

[128] Rene Jr. Landry Mohamed Sahmoudi, Moeness G. Amin. Acquisition of weak gnss signals using a new block averaging pre-processing. In *IEEE/ION Position, Location and Navigation Symposium 2008*.

[129] V. Motackova, J. Novacek, A. Zawadzka-Kazimierczuk, K. Kazimierczuk, L. Zidek, H. Sanderova, L. Krasny, W. Kozminski, and V. Sklenar. Strategy for Complete NMR Assignment of Disordered Proteins with Highly Repetitive Sequences Based on Resolution-Enhanced 5D Experiments. *Journal of Biomolecular NMR*, 48:169–177, 2010.

[130] Abdullah Mueen, Suman Nath, and Jie Liu. Fast approximate correlation for massive time-series data. In *SIGMOD Conference*, pages 171–182, 2010.

[131] B. Murmann. A/d converter trends: Power dissipation, scaling and digitally assisted architectures. In *IEEE Custom Integrated Circuits Conference (CICC)*, 2008.

[132] Ren Ng. Fourier slice photography. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 735–744. ACM, 2005.

[133] Ren Ng, Marc Levoy, Mathieu Bredif, Gene Duval, Mark Horowitz, and Pat Hanrahan. Light field photography with a hand-held plenoptic camera. Technical Report CSTR 2005-02, Stanford University Computer Science, 2005.

[134] D. Niculescu and B. Nath. Ad hoc positioning system (APS). In *IEEE GLOBECOM*, 2001.

[135] Nikon USA. Coolpix p6000. http://www.nikonusa.com.

[136] Shahriar Nirjon, Jie Liu, Gerald DeJean, Bodhi Priyantha, Yuzhe Jin, and Ted Hart. COIN-GPS: Indoor Localization from Direct GPS Receiving. In *MobiSys*, 2014.

[137] D. Nishimura. *Principles of Magnetic Resonance Imaging*. Society for Industrial and, 2010.

[138] A. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-time signal processing*. Upper Saddle River, N.J.: Prentice Hall. ISBN 0-13-754920-2., 1999.

[139] OProfile. Linux profiler. http://oprofile.sourceforge.net.

[140] V.Y. Orekhov and V.A. Jaravine. Analysis of Non-Uniformly Sampled Spectra with Multi-Dimensional Decomposition. *Progress in Nuclear Magnetic Spectroscopy*, 59:271–292, 2011.

[141] OriginGPS. ORG447X series datasheet. http://www.acaltechnology.com.

[142] Perthold Engineering LLC. SkyTraq Venus 6 GPS Module. http://www.perthold.de.

[143] Gilles Pisier. *The Volume of Convex Bodies and Banach Space Geometry*. Cambridge University Press, May 1999.

[144] Darius Plausinaitis. GPS receiver technology mm8. Danish GPS Center, http://kom.aau.dk.

[145] E. Porat and M. Strauss. Sublinear time, measurement-optimal, sparse recovery for all. *Manuscript*, 2010.

[146] E. Price and D. P. Woodruff. $(1+\epsilon)$-approximate sparse recovery. *FOCS*, 2011.

[147] X. Qu, M. Mayzel, J.F. Cai, Z. Chen, and V. Orekhov. Accelerated NMR Spectroscopy with Low-Rank Reconstruction. *Angewandte Chemie International Edition*, 2014.

[148] J.M. Rabaey, A.P. Chandrakasan, and B. Nikolic. *Digital integrated circuits*. Prentice-Hall, 1996.

[149] Hariharan Rahul, Nate Kushman, Dina Katabi, Charles Sodini, and Farinaz Edalat. Learning to Share: Narrowband-Friendly Wideband Networks. In *ACM SIGCOMM*, 2008.

[150] Heitor S. Ramos, Tao Zhang, Jie Liu, Nissanka B. Priyantha, and Aman Kansal. Leap: a low energy assisted GPS for trajectory-based services. In *ACM Ubicomp 2011*, pages 335–344, Beijing, China.

[151] M. Rashidi, K. Haghighi, A. Panahi, and M. Viberg. A NLLS based sub-nyquist rate spectrum sensing for wideband cognitive radio. In *IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), 2011*, 2011.

[152] D. Raskovic and D. Giessel. Battery-Aware embedded GPS receiver node. In *IEEE MobiQuitous*, 2007.

[153] RFeye Node. http://media.crfs.com/uploads/files/2/crfs-md00011-c00-rfeye-node.pdf.

[154] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan. Computational solutions to large-scale data management and analysis. 2011.

[155] Mingoo Seok, Dongsuk Jeon, C. Chakrabarti, D. Blaauw, and D. Sylvester. A 0.27V 30MHz 17.7nJ/Transform 1024-pt Complex FFT Core with Super-Pipelining. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, ISSCC'11, pages 342–344, Feb 2011.

[156] Lixin Shi, Ovidiu Andronesi, Haitham Hassanieh, Badih Ghazi, Dina Katabi, and Elfar Adalsteinsson. MRS Sparse-FFT: Reducing Acquisition Time and Artifacts for In Vivo 2D Correlation Spectroscopy. In *Proceedings of the International Society for Magnetic Resonance in Medicine Annual Meeting & Exhibition*, ISMRM'13, Salt Lake City, USA, April 2013.

[157] Lixin Shi, Haitham Hassanieh, Abe Davis, Dina Katabi, and Fredo Durand. Light Field Reconstruction Using Sparsity in the Continuous Fourier Domain. *ACM Transactions on Graphics*, 34(1):12:1–12:13, December 2014.

[158] Emil Sidky. What does compressive sensing mean for X-ray CT and comparisons with its MRI application. In *Conference on Mathematics of Medical Imaging*, 2011.

[159] Anthony Silva. Reconstruction of Undersampled Periodic Signals, 1986. MIT Technical Report.

[160] Julius O. Smith. *Spectral Audio Signal Processing, October 2008 Draft*. accessed 2011-07-11. http://ccrma.stanford.edu/ jos/sasp/.

[161] Cyril Soler and François X. Sillion. Fast calculation of soft shadow textures using convolution. In *Computer Graphics* Proceedings, Annual Conference Series, pages 321–332. ACM SIGGRAPH, 1998.

[162] J. Stanek, R. Augustyniak, and W. Kozminski. Suppression of Sampling Artefacts in High-Resolution Four-Dimensional NMR Spectra Using Signal Separation Algorithm. *Journal of Magnetic Resonance*, 214:91–102, 2012.

[163] Stanford. Stanford Light Field Archive. http://lightfield.stanford.edu/, 2008. [Online; accessed 2013].

[164] T. Szyperski, G. Wider, J.H. Bushweller, and K. Wuthrich. Reduced Dimensionality in Triple-Resonance NMR Experiments. *Journal of the American Chemical Society*, 115:9307–9308, 1993.

[165] T.C Tan, Guoan Bi, Yonghong Zeng, and H.N Tan. Dct hardware structure for sequentially presented data. *Signal Processing*, 81(11), 2001.

[166] Y. Tanet, J. Duster, C-T. Fu, E. Alpman, A. Balankutty, C. Lee, A. Ravi, S. Pellerano, K. Chandrashekar, H. Kim, B. Carlton, S. Suz uki, M. Shafi, Y. Palaskas, and H. Lakdawala. A 2.4 ghz wlan transceiver with fully-integrated highly-linear 1.8 v 28.4 dbm pa, 34 dbm t/r switch, 240 ms/s dac, 320 ms/s adc, and dpll in 32 nm soc cmos. In *IEEE Symposium on VLSI Technolagy and Circuits*, 2012.

[167] AGI Creative Team. *Fundamentals of Global Positioning System Receivers: A Software Approach*. Wiley-Interscience, 2000.

[168] Ted Schadler. GPS: Personal Navigation Device. Texas Instruments. http://www.ti.com.

[169] Tektronix Spectrum Analyzer. http://tek.com.

[170] Arvind Thiagarajan, Lenin Ravindranath, Hari Balakrishnan, Samuel Madden, and Lewis Girod. Accurate, low-energy trajectory mapping for mobile devices. In *NSDI 2011*.

[171] M. Albert Thomas, Kenneth Yue, Nader Binesh, Pablo Davanzo, Anand Kumar, Benjamin Siegel, Mark Frye, John Curran, Robert Lufkin, Paul Martin, and Barry Guze. Localized Two-Dimensional Shift Correlated MR Spectroscopy of Human Brain. *Magnetic Resonance in Medicine*, 46:58–67, 2001.

[172] J.A. Tropp, J.N. Laska, M.F. Duarte, J.K. Romberg, and R.G. Baraniuk. Beyond Nyquist: Efficient Sampling of Sparse Bandlimited Signals. *IEEE Transactions on Information Theory*, 56(1):520–544, 2010.

[173] Nicholas Tzou, Debesh Bhatta, Sen-Wen Hsiao, Hyun Woo Choi, and Abhijit Chatterjee. Low-Cost Wideband Periodic Signal Reconstruction Using Incoherent Undersampling and Back-end Cost Optimization. In *IEEE Inter. Test Conference*, 2012.

[174] PCAST: Realizing the full potential of government held spectrum to spur economic growth, 2012. http://www.whitehouse.gov/sites/default/files/microsites/ostp/pcast_spectrum_report_final_july_20_2012.pdf.

[175] P.P. Vaidyanathan and P. Pal. Sparse Sensing With Co-Prime Samplers and Arrays. *IEEE Transactions on Signal Processing*, 59(2):573–586, 2011.

[176] D.J.R. Van Nee and A.J.R.M. Coenen. New fast GPS code-acquisition technique using FFT. *Electronics Letters*, 27(2), Jan 1991.

[177] Ashok Veeraraghavan, Ramesh Raskar, Amit Agrawal, Ankit Mohan, and Jack Tumblin. Dappled photography: Mask enhanced cameras for heterodyned light fields and coded aperture refocusing. *ACM Transactions on Graphics*, 26(3):69, 2007.

[178] R. Venkataramani and Y. Bresler. Perfect reconstruction formulas and bounds on aliasing error in sub-nyquist nonuniform sampling of multiband signals. *IEEE Transactions on Information Theory*, 46(6):2173–2183, 2000.

[179] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville Talvala, Emilio Antunez, Adam Barth, Andrew Adams, Mark Horowitz, and Marc Levoy. High performance imaging using large camera arrays. *ACM Trans. Graph.*, 24(3):765–776, 2005.

[180] X-G Xia. An efficient frequency-determination algorithm from multiple undersampled waveforms. *Signal Processing Letters, IEEE*, 7(2):34–37, 2000.

[181] Xiang-Gen Xia. On estimation of multiple frequencies in undersampled complex valued waveforms. *Signal Processing, IEEE Transactions on*, 47(12):3417–3419, 1999.

[182] Chia-Hsiang Yang, Tsung-Han Yu, and D. Markovic. Power and Area Minimization of Reconfigurable FFT Processors: A 3GPP-LTE Example. *IEEE Journal of Solid-State Circuits*, 47(3):757–768, March 2012.

[183] O. Yilmaz. *Seismic Data Analysis: Processing, Inversion, and Interpretation of Seismic Data*. Society of Exploration Geophysicists, 2008.

[184] Juhwan Yoo, S. Becker, M. Loh, M. Monge, E. Candès, and A. E-Neyestanak. A 100MHz–2GHz 12.5x subNyquist rate receiver in 90nm CMOS. In *IEEE RFIC*, 2012.

[185] Juhwan Yoo, Christopher Turnes, Eric Nakamura, Chi Le, Stephen Becker, Emilio Sovero, Michael Wakin, Michael Grant, Justin Romberg, Azita Emami-Neyestanak, and Emmanuel Candès. A compressed sensing parameter extraction platform for radar pulse signal acquisition. *IEE Journal on Emerging and Selected Topics in Circuits and Systems,*, 2012.

[186] Sungro Yoon, Li Erran Li, Soung Liew, Romit Roy Choudhury, Kun Tan, and Injong Rhee. Quicksense: Fast and energy-efficient channel sensing for dynamic spectrum access wireless networks. In *IEEE INFOCOM*, 2013.

[187] T. Yucek and H. Arslan. A survey of spectrum sensing algorithms for cognitive radio applications. *Communications Surveys Tutorials, IEEE*, 11(1), 2009.

[188] Zhengyun Zhang and Marc Levoy. Wigner distributions and how they relate to the light field. In *Computational Photography (ICCP), 2009 IEEE International Conference on*, pages 1–10. IEEE, 2009.

[189] Guichang Zhong, Fan Xu, and Jr. Willson, A.N. A Power-Scalable Reconfigurable FFT/IFFT IC Based on a Multi-Processor Ring. *Solid-State Circuits, IEEE Journal of*, 41(2):483–495, Feb 2006.