Combinatorial Compressive Sampling with Applications

by

Mark A. Iwen

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Applied and Interdisciplinary Mathematics)
in The University of Michigan
2008

Doctoral Committee:

Assistant Professor Martin J. Strauss, Co-Chair
Associate Professor Jignesh M. Patel, Co-Chair
Professor John P. Boyd
Associate Professor Anna Catherine Gilbert
Professor Robert Krasny

Acknowledgements

I would like to start by thanking Martin Strauss, Jignesh Patel, and Anna Gilbert for giving me loads of advice and guidance over the last several years, and for tolerating my sometimes overly willful behavior. In addition, I would like to thank them (along with Graham Cormode, Piotr Indyk, Willis Lang, Michael Lieberman, G. S. Mandair, Michael Morris, Muthu Muthukrishnan, Holger Rauhut, Craig Spencer, and Lisa Zhang) for deciphering my often confusing and too hastily composed e-mails, commenting on preprints, and answering questions. I would also like to thank all the other faculty members who talked with me and answered questions over the past 5 years. In particular, I would like to thank John Boyd, Andrew Christlieb, Charles Doering, Robert Krasny, Peter Miller, Karen Rhea, and Peter Smereka.

Thank you helpful and friendly office staff! — Tara McQueen in particular. Without her help I'm not sure I would have made it through the process.

My life in Ann Arbor would have impoverished without Dom's Bakery, dodge ball, ultimate frisbee, late night departmental ping pong, snowball fights, summer cookouts, and walks in the arboretum: I thank everyone who ever collaborated with me in any of these. In particular, I thank Hualong Feng, Joel Lepak, and Craig Spencer for letting me crash at their houses more or less at will throughout my time here. I'm likewise still in Sourya Shrestha's debt for feeding me more times than I can count first year. I have many fond memories of biking all over Ann Arbor and the surrounding country side with Dave Allen, Joel Lepak, Mike Lieberman,

Diane Vavrichek, and Craig Spencer. My running partners - Dave Anderson & Mike Lieberman - were very kind in not ditching me over the years, and for being gentle with me the (many) times I got even more out of shape than usual. Katka Bodova gets a thousand thanks for teaching me how to throw a frisbee properly second year. Finally, special thanks to my roommates Heather Adams, Hualong Feng, Jen Kostrzewski, and Maneesh Sharma for their companionship over the years.

Wendy Grus: Expressing my appreciation for her in such a small space is difficult. Her practical advice, generosity, comforting sense of humor, and irrepressible cuteness earn her a gold medal! Her presence in my life has improved its quality in every respect.

Slight variants of some thesis chapters have been published previously. Chapter II appeared in "Communications in Mathematical Sciences" in 2007 [66] and was joint work with Anna Gilbert and Martin Strauss. Chapter III appeared in the proceedings of the ACM-SIAM Symposium of Discrete Algorithms (SODA'08) [65]. Chapter IV appeared in the proceedings of the Conference on Information Sciences and Systems (CISS'08) [69] and was joint work with Craig V. Spencer. Chapters V and VI have been submitted. Chapter VI is joint work with Craig V. Spencer. Appendix A appeared in the proceedings of the The 24th International Conference on Data Engineering (ICDE'08) [67] and was joint work with Jignesh Patel and Willis Lang. Appendix B appeared in the proceedings of the 2007 International Conference on Acoustics, Speech, and Signal Processing (ICASSP'07) [68] and was joint work with G. S. Mandair, M. D. Morris, and M. Strauss.

Table of Contents

List of Algorithms

List of Figures

List of Tables

List of Appendices

Chapter I

Introduction

This thesis is concerned with quickly finding sparse representations of signals (e.g., vectors of data, periodic functions, etc). Traditional applications of sparse signal representations include image, video, and music compression (see [85]). Compactly representing such signals is generally a good idea for the sake of minimizing both communication and storage costs. Our additional interest in *quickly* determining compact representations is (hopefully) self-explanatory: we seek to reduce the computational costs associated with obtaining sparse signal representations as much as possible. To better understand the types of problems we are interested in here we will next consider a simple application-based example in which the FFT can be replaced by faster sparse Fourier methods.

1.1 Example: Sub-Nyquist Single Frequency Acquisition

Let $f : [0, 2\pi] \to \mathbb{C}$ be a non-identically zero function of the form

$$f(x) = C \cdot e^{\mathrm{i}\omega x}$$

consisting of a single unknown frequency $\omega \in (-N, N]$ (e.g., consider a windowed sinusoidal portion of a wideband frequency-hopping signal [77]). Sampling at the Nyquist-rate would dictate the need for at least $2N$ equally spaced samples from f

in order to discover ω via the FFT without aliasing. Thus, we would have to compute the FFT of the $2N$-length vector

$$\mathbf{A}(j) = f\left(\frac{\pi j}{N}\right),\ 0 \leq j < 2N.$$

However, if we use aliasing to our advantage, we can correctly determine ω with significantly fewer f-samples as follows:

Let $\mathbf{A}_2$ be a 2-element array of f-samples with

$$\mathbf{A}_2(0) = f(0) = C, \text{ and } \mathbf{A}_2(1) = f(\pi) = C \cdot (-1)^{\omega}.$$

Calculating $\widehat{\mathbf{A}_2}$ we get that

$$\widehat{\mathbf{A}_2}(0) = C \cdot \frac{1 + (-1)^{\omega}}{\sqrt{2}}, \text{ and } \widehat{\mathbf{A}_2}(1) = C \cdot \frac{1 + (-1)^{\omega+1}}{\sqrt{2}}.$$

Note that since ω is an integer, exactly one element of $\widehat{\mathbf{A}_2}$ will be non-zero. If $\widehat{\mathbf{A}_2}(0) \neq 0$ then we know that $\omega \equiv 0$ modulo 2. On the other hand, $\widehat{\mathbf{A}_2}(1) \neq 0$ implies that $\omega \equiv 1$ modulo 2. In this same fashion we may use several potentially aliased Fast Fourier Transforms in parallel to discover ω modulo $3, 5, \ldots,$ the $O(\log N)^{\text{th}}$ prime. Once we have collected these moduli we can reconstruct ω via the famous **Chinese Remainder Theorem (CRT)**.

Theorem I.1. Chinese Remainder Theorem (CRT): *Any integer x is uniquely specified mod N by its remainders modulo m relatively prime integers $p_1, \ldots, p_m$ as long as $\prod_{l=1}^{m} p_l \geq N$.*

To finish our example, suppose that $N = 500,000$ and that we have used three FFT's with 100, 101, and 103 samples to determine that $\omega \equiv 34 \mod 100$, $\omega \equiv 3 \mod 101$, and $\omega \equiv 1 \mod 103$, respectively. Using that $\omega \equiv 1 \mod 103$ we can see that $\omega = 103 \cdot a + 1$ for some integer a. Using this new expression for ω in our second

modulus we get

$$(103 \cdot a + 1) \equiv 3 \bmod 101 \Rightarrow a \equiv 1 \bmod 101.$$

Therefore, $a = 101 \cdot b + 1$ for some integer b. Substituting for a we get that $\omega = 10403 \cdot b + 104$. By similar work we can see that $b \equiv 10 \bmod 100$ after considering ω modulo 100. Hence, $\omega = 104,134$ by the CRT. As an added bonus we note that our three FFTs will have also provided us with three different estimates of ω's coefficient C.

The end result is that we have used significantly less than $2N$ samples to determine ω. Using the CRT we required only $100 + 101 + 103 = 304$ samples from f to determine ω since $100 \cdot 101 \cdot 103 > 1,000,000$. In contrast, a million f-samples would be gathered during Nyquist-rate sampling. Besides needing significantly less samples than the FFT, this CRT-based single frequency method dramatically reduces required computational effort. Of course, a single frequency signal is incredibly simple. Signals involving more than 1 non-zero frequency are much more difficult to handle since frequency moduli may begin to collide modulo various numbers. However, the take-home lesson is clear: knowledge of inherent signal sparsity can be taken advantage of to reduce both the sampling and runtime requirements involved in signal recovery.

1.2 General Problem Setup

More formally, we are interested in the following type of problem. Let X be a Hilbert space with a countable orthonormal basis $\Psi = \{\psi_j\}_{j \in \mathbb{Z}}$. Furthermore, suppose we are given a signal $f \in X$ which is **compressible** with respect to Ψ. That is, suppose there exists an ordering

$$|\langle f, \psi_{j_1} \rangle| \geq |\langle f, \psi_{j_2} \rangle| \geq \cdots \geq |\langle f, \psi_{j_l} \rangle| \geq \ldots$$

such that for some $p \in \mathbb{R}^+$ we have

$$\sum_{l=k+1}^{\infty} |\langle f, \psi_{j_l} \rangle| = O\left(k^{-p}\right).$$

We seek a k-**sparse** $\tilde{f} \in X$ of the form

$$\tilde{f} = \sum_{l=1}^{k} C_l \cdot \psi_{m_l}$$

which is close to f in an induced norm. For example, in subsequent chapters we will look for a sparse approximation $\tilde{f}$ to f with

$$\|f - \tilde{f}\|_2^2 = O\left(k^{-1-2p}\right). \tag{1.1}$$

Note that there is potential computational difficulty here. Parseval's equality tells us that in order for $\tilde{f}$ to be a good approximation to f as per Equation 1.1 we need to identify a substantial portion of f's most important basis elements (i.e., determine most of $\{j_1, j_2, \ldots, j_k\}$). If $j_{\max} = \max\{|j_1|, \cdots, |j_k|\}$ is large, a straightforward calculation of $\tilde{f}$ by computing $O(j_{\max})$ inner products may be computationally taxing. This is especially true when the cost of obtaining an inner product is high.

For example, consider medical imaging. Certain imaging procedures (e.g., some types of MR-imaging [83, 84]) yield compressible patient images (in space). However, they collect image information in the Fourier domain. Typically each patient scan yields a small subset of the Fourier transform of the patient image. Thus, the sparser the patient image, the more patient scans (i.e., Fourier inner products) generally required by straightforward scanning techniques to identify and properly render important image pixels. In addition, every patient scan is both time- and energy-intensive. In such cases it is highly desirable to be able to generate a high fidelity image of the patient using only a small number of scans (i.e., Fourier inner products).

A natural question arises: Is there a method of determining an $\tilde{f}$ using a number of inner products determined primarily by f's inherent compressibility? For example, can we determine a valid $\tilde{f}$ using at most

$$k^{O\left(1+\frac{1}{p}\right)} \cdot \log^{O(1)}\left(\max\left\{j_1, \cdots, j_{k^{O\left(1+\frac{1}{p}\right)}}\right\}\right)$$

samples (e.g., inner products) from f? The answer (to both questions) is 'yes'. Methods concerned with answering this question are collectively referred to as Compressed Sensing (CS) methods.

1.3 Compressed Sensing

For the remainder of this section we'll assume our Hilbert space X has a finite orthonormal basis $\Psi = \{\psi_j\}_{j\in\mathbb{Z}_N}$ (i.e., when concerned with the approximation of a compressible signal in a separable Hilbert space we can always project onto a large finite dimensional subspace). As before, $f \in X$ will be a p-compressible signal that we would like to approximate with a k-sparse $\tilde{f}$. Note that any optimal sparse approximation, $\tilde{f}^{\text{opt}}$, will have

$$\|f - \tilde{f}^{\text{opt}}\|_q = \inf_{k-\text{sparse}\ v\in X} \|f - v\|_q.$$

There are generally two components to a Compressed Sensing (CS) method for approximating $f \in X$.

1. **Measurement Operator:** A bounded linear operator $\mathcal{M} : X \to \mathbb{C}^d$ where $d = o(N^\epsilon) \cdot (\frac{k}{\delta})^{O\left(1+\frac{1}{p}\right)}$, and
2. **Recovery Algorithm:** an algorithm $\mathcal{A}$ which, when given $\mathcal{M}(f)$ and δ as input, outputs an $\tilde{f}$ with

$$\|f - \tilde{f}\|_q = (1+\delta)\|f - \tilde{f}^{\text{opt}}\|_q$$

in $N^{O(1)}$ time.

Note that the size, d, of $\mathcal{M}$'s target dimension is typically more important than the recovery algorithm $\mathcal{A}$'s runtime. We generally want to gather as little information as possible about f. For example, in the MR-imaging example above we are more concerned with reducing the number of patient scans than we are with the computational time required to recover the patient's image from the collected scans.

Of course, CS methods' operator properties, recovery algorithms, and error guarantees vary widely. Most notably there are three general types of recovery algorithms employed by current CS methods: linear programming, greedy pursuit, and combinatorial. In what follows we will briefly survey CS methods subdivided by recovery algorithm type. In the process we will restrict our treatment to CS methods which are tolerant to noise (i.e., are capable of approximating compressible signals as opposed to only recovering *exact* sparse signals).

1.3.1 Linear Programming

Linear Programming (LP)-based compressed sensing methods were the first to be developed and refined [41, 40, 39, 19, 18, 13] (see [3] for a more comprehensive bibliography). These LP methods generally utilize measurement operators with the property that for a given $\delta \in \mathbb{R}^+$ all k-sparse $f' \in X$ have

$$(1-\delta)\|f'\|_q \leq \|\mathcal{M}(f')\|_q \leq (1+\delta)\|f'\|_q. \tag{1.2}$$

This property is generally referred to as the *Restricted Isometry Property (RIP)*. If $\mathcal{M}$ has the RIP (for either $q = 2$ [19], or $q = 1 + \frac{O(1)}{\log N}$ [13]) a linear program can recover an accurate approximation to a compressible $f \in X$ by solving

$$\min \|f'\|_1 \text{ subject to } \mathcal{M}(f') = \mathcal{M}(f).$$

Construction Type	Random/Deterministic	Norm Type q	Target Dimension d
Gaussian	R	2	$O(k \cdot \log(N/k))$ [99, 42]
Fourier	R	2	$O(k \cdot \log^4 N)$ [99]
Algebraic	D	2	$O(k^2 \cdot \log^{O(1)}(N))$ [36]
Expander	R	1	$O(k \cdot \log(N/k))$ [13]
Expander	D	1	$O(k \cdot N^{\epsilon})$ [13]

Table 1.1: RIP Measurement Operator Constructions

Given that linear programs require $N^{O(1)}$-time to solve, these methods are of most interest when great measurement compression is sought. Hence, most LP based CS work focuses on the construction of RIP operators with small target dimension (i.e., d minimized).

Initial constructions of RIP matrices were motivated by randomized embedding results due to Johnson and Lindenstrauss [70]. Hence, the first measurement operators $\mathcal{M} : X \to \mathbb{C}^d$ consisted of taking an input f's inner product with d randomly constructed $m \in X$. For example, if each m is determined by independently choosing $\langle m, \psi_j \rangle$ from a properly normalized Gaussian distribution for each $j \in \mathbb{Z}_N$, $\mathcal{M}$ can be shown to have the RIP with $q = 2$ with high probability [10, 99]. Other measurement operator constructions use d elements $m \in X$ whose inner products with the N basis elements match d randomly selected $N \times N$ discrete Fourier matrix rows. For a summary of standard RIP measurement operator constructions see Table 1.1. Please note that Equation 1.2's δ is considered to be a fixed constant with respect to Table 1.1.

In Table 1.1 the first column lists the type of measurement operator construction, the second column lists whether the construction is randomized or deterministic, the third column lists the type of RIP property the operator satisfies (see Equation 1.2), and the fourth lists the dimension of the target space. It should be noted that the randomized constructions are near optimal with respect to the operator target di-

Algorithm 1.1 GREEDY PURSUIT

1: **Input: Signal** $f \in X$**, Measurements** $\mathcal{M}(f)$**, Measurement Operator** $\mathcal{M}$
2: **Output:** $\tilde{f} \in X$
3: Set $r = f$, $\tilde{f} = 0$.
4: **while** $\|\mathcal{M}(r)\|$ is too large **do**
5: Use $\mathcal{M}(r)$ to get a decent sparse approximation, $\tilde{r} \in X$, to r
6: Set $r = r - \tilde{r}$, and $\tilde{f} = \tilde{f} + \tilde{r}$
7: **end while**
8: Return $\tilde{f}$

mension d (within $\log N$ factors). The deterministic algebraic operator construction is also near optimal for its class (i.e., $q = 2$ with binary entries) [21]. Similarly, improving the deterministic expander construction is probably difficult [13].

1.3.2 Greedy Pursuit

Greedy pursuit compressed sensing methods were motivated by Orthogonal Matching Pursuit (OMP) and its successful application to best basis selection problems and their variants [85]. Hence, OMP was the first greedy pursuit method to be applied in the CS context [104]. OMP and related CS greedy pursuit recovery algorithms all work along the lines of Algorithm 1.1. The analysis of these methods typically consists of verifying that line 4's residual energy will shrink quickly given that line 5's fast approximation method maintains required iterative invariants. Although the analysis can be difficult, the algorithms themselves are typically simple to implement and faster than LP solution methods [74].

Recent developments in compressed sensing have led to several greedy pursuit methods which use RIP measurement operators first developed for LP-based methods to reconstruct compressible signals using a small number of measurements [95, 94, 93, 63]. Hence, these new greedy pursuit methods can simultaneously take advantage of both the fast runtimes of greedy pursuit methods and the impressive measurement properties (i.e., the (near)-optimal target dimensions) of the RIP con-

structions listed in Table 1.1. Driven by these two simultaneous advantages greedy pursuit CS methods appear poised to replace LP-based CS methods in most applications.

Perhaps the most interesting aspect of CS greedy pursuit methods is that they may be combined with group testing ideas [44, 55] to yield reconstruction algorithms which have $(k \cdot \log(N))^{O(1)}$ time complexity. This is generally done by using structured measurement operators to collect information identifying high-energy basis elements, thereby eliminating the need for the reconstruction algorithm to consider the vast majority of Ψ. Having effectively pruned the $N \gg k$ basis down to a subset of size $K = (k \cdot \log(N))^{O(1)}$ using what amounts to a high-energy subspace projection operator, a $N^{O(1)}$-time greedy pursuit method maybe employed at reduced cost (i.e., N is replaced with K). Examples of such algorithms include [56] and [53, 54] (developed in the Fourier context).

1.3.3 Combinatorial

Combinatorial compressed sensing methods [32, 33, 92, 61] were first developed using ideas related to streaming algorithms [91, 51]. A combinatorial CS measurement operator, $\mathcal{M}$, is structured so that it separates the influence of f's k-largest magnitude Ψ-basis elements from one another in some k-dimensional subspace, S, of $\mathcal{M}$'s target space. Hence, S is guaranteed to contain a high fidelity projection of f's best-basis coefficients. A combinatorial recovery algorithm then utilizes knowledge of $\mathcal{M}$'s structure to both locate S and to determine which subspace of Ψ must have produced it. The majority of this thesis is concerned with combinatorial CS methods. Thus, we postpone a more detailed discussion until later chapters.

For now, we simply note that combinatorial CS methods are also easily combined with group testing ideas to yield incredibly fast reconstruction algorithms. Further-

more, some combinatorial CS methods exhibit a useful sampling structure which can be modified to be highly beneficial in the Fourier compressed sensing case. As a result, we are able to modify combinatorial CS methods to create fast Fourier transform algorithms for frequency-sparse signals/functions. These new combinatorial Fourier methods can be viewed as a beneficial translation of earlier sparse Fourier methods [53, 54] into a different context. As a result of this translation, we not only achieve the first known deterministic sublinear-time Fourier methods, but also explicitly link these sparse Fourier results to a general compressed sensing methodology.

1.4 Thesis Outline

The majority of this Thesis is concerned with compressed sensing in the Fourier context. More specifically, suppose we are given a periodic function $f : [0, 2\pi] \to \mathbb{C}$ which is well approximated by a k-sparse trigonometric polynomial

$$\tilde{f}(x) = \sum_{j=1}^{k} C_j e^{\mathrm{i}\omega_j \cdot x}, \ \{\omega_1, \ldots, \omega_k\} \subset \left[-\frac{N}{2}, \frac{N}{2}\right], \tag{1.3}$$

where the smallest such N is much larger than k. We seek methods for recovering a high-fidelity approximation to $\tilde{f}$ using both $(k \cdot \log(N))^{O(1)}$ time and f-samples.

Table 1.2 compares the Fourier CS algorithms developed in this thesis to other existing Fourier methods. All the methods listed are robust with respect to noise. The runtime and sampling requirements are for recovering exact k-sparse trigonometric polynomials (see Equation 1.3). The second column indicates whether the result recovers (an approximation to) the input signal with high probability (W.H.P.) or deterministically (D). "With high probability" indicates a nonuniform $O\left(\frac{1}{N^{O(1)}}\right)$ failure probability per signal. In some cases, for simplicity, a factor of "$\log(k)$" or "$\log(N/k)$" was weakened to "$\log(N)$".

Looking at Table 1.2 we can see that CoSaMP [93] achieves the best theoret-

Fourier Algorithm	W.H.P./D	Runtime	Function Samples
LP [19] or ROMP [95]	W.H.P.	$N^{O(1)}$	$O(k \log(N))$ [18]
CoSaMP [93]	W.H.P.	$O(N \cdot \log^2(N))$	$O(k \cdot \log(N))$ [18]
Chapter V	W.H.P.	$O(N \cdot \log^3(N))$	$O(k \cdot \log^2(N))$
Chapter V	D	$O(N \cdot k \cdot \log^2(N))$	$O(k^2 \cdot \log N)$
Sparse Fourier [54]	W.H.P.	$O(k \cdot \log^{O(1)}(N))$	$O(k \cdot \log^{O(1)}(N))$
Chapter V	W.H.P.	$O(k \cdot \log^5(N))$	$O(k \cdot \log^4(N))$
Chapter V	D	$O(k^2 \cdot \log^4(N))$	$O(k^2 \cdot \log^3(N))$

Table 1.2: Fourier CS Algorithms

ical superlinear Fourier runtimes (outperforming LP and ROMP). In comparison, our W.H.P. Chapter V results require an additional $\log(N)$ factor in terms of both runtime and sampling complexity. However, we should note that the Chapter V algorithms are simpler to implement and optimize than CoSaMP. The Chapter V algorithms are also capable of *exactly* reconstructing k-sparse signals in an exact arithmetic setting. More interestingly, we note that our Chapter V Monte-Carlo sublinear-time result matches the previous sparse Fourier method [54]. In addition, our Monte-Carlo result can be modified to yield the first known *deterministic* sublinear-time sparse Fourier algorithm.

The remainder of this thesis proceeds as follows: In Chapter II we empirically evaluate implementations of existing Monte-Carlo Fourier algorithms [53, 54] for solving the Fourier CS problem. Next, in Chapter III, we present a combinatorial CS method for solving the general compressed sensing problem and quickly sketch its application to the Fourier CS problem. In Chapter IV tight sampling and runtime bounds are worked out for the previous chapter's combinatorial CS method. Finally, an improved deterministic solution of the Fourier CS problem is presented in Chapter V (along with a new Monte-Carlo solution method). An interesting implication of compressed sensing for the complexity of matrix multiplication is noted in Chapter VI.

1.4.1 The Appendices

It should be noted that the Fourier results herein can be considered as sparse interpolation results. Traditional (trigonometric) polynomial interpolation methods require $O(N)$ function samples in order to recover an N^{th}-degree polynomial [52, 73]. On the other hand, sparse interpolation results for recovering k-term polynomials of maximum degree N only require $O(k)$ function samples [87, 12, 71]. Similarly, randomized sparse trigonometric polynomial interpolation results (similar to [53, 54]) exist for recovering k-term trigonometric polynomials using $(k \cdot \log(N))^{O(1)}$ function evaluations [86, 23]. Chapter V presents the first known fast deterministic interpolation result for trigonometric polynomials.

Given existing Fourier CS methods' relationships to trigonometric interpolation it isn't surprising that they have been applied to both numerical methods [35] (via spectral techniques [16, 103]) and medical imaging [83, 84]. Likewise, sparse interpolation methods can be considered as learning methods along the lines of [75] and thereafter applied to classification problems. Due to these connections, two related appendices have been added to the end of this thesis. Appendix A discusses a heuristic method for classifying gene expression data. Appendix B outlines a method for reducing the total imaging time of test specimens under a given cost model.

1.5 The Fourier Case

Since the majority of the remaining chapters are concerned with computing the Fourier transform of a frequency-sparse periodic function, we will conclude this chapter with a brief review of the Discrete Fourier Transform (DFT) and its standard related results. In the process, we will establish notation used throughout subsequent chapters.

1.5.1 The Discrete Fourier Transform

We will refer to a vector in $\mathbb{C}^N$ as an **array** or **signal**. Furthermore, we'll denote the j^{th} component of any array $\mathbf{A}$ by $\mathbf{A}[j]$. The **inner product** of two arrays, $\mathbf{A}$ and $\mathbf{B}$, is defined as

$$\langle \mathbf{A}, \mathbf{B} \rangle = \sum_{j=0}^{N-1} \mathbf{A}[j] \cdot \overline{\mathbf{B}[j]}.$$

Using the inner product we define the L^2**-norm** of an array, $\mathbf{A}$, as

$$\|\mathbf{A}\|_2 = \sqrt{\langle \mathbf{A}, \mathbf{A} \rangle} = \sqrt{\sum_{j=0}^{N-1} |\mathbf{A}[j]|^2}.$$

Finally, let

$$g_{_N} = e^{\frac{-2\pi \mathbb{i}}{N}}.$$

We define the **discrete delta function** $\delta_{_N} : [0, N) \times [0, N) \to \{0, 1\}$ to be

$$\delta_{_N}(j,k) = \frac{1}{N} \sum_{\omega=0}^{N-1} g_{_N}^{\omega \cdot (k-j)} = \begin{cases} \frac{\sum_{\omega=0}^{N-1} 1}{N} = 1 & \text{if } k = j \\ \frac{1 - g_{_N}^{N \cdot (k-j)}}{N(1 - g_{_N}^{k-j})} = 0 & \text{if } k \neq j \end{cases}. \tag{1.4}$$

Let $G_{_N}$ be the $N \times N$ matrix $(G_{_N})_{\omega,j} = \frac{g_{_N}^{\omega \cdot j}}{\sqrt{N}}$. In effect, we note that the set of vectors

$$(\mathbf{G}_{_N})_\omega[j] = \frac{g_{_N}^{\omega \cdot j}}{N},\ \omega \in [0, N)$$

form an orthonormal basis.

The **Discrete Fourier Transform (DFT)** of an array $\mathbf{A}$ is $\hat{\mathbf{A}} = G_{_N}\mathbf{A}$. Thus, we have

$$\hat{\mathbf{A}}[\omega] = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \mathbf{A}[j]\, g_{_N}^{\omega \cdot j},\ \omega \in [0, N). \tag{1.5}$$

Similarly, the **Inverse Discrete Fourier Transform (IDFT)** of any array $\mathbf{A}$ is defined as

$$\hat{\mathbf{A}}^{-1}[j] = \frac{1}{\sqrt{N}} \sum_{\omega=0}^{N-1} \mathbf{A}[\omega]\, g_{_N}^{-\omega \cdot j},\ j \in [0, N). \tag{1.6}$$

Not surprisingly, the IDFT allows us to recover our original signal $\mathbf{A}$ from $\hat{\mathbf{A}}$. For any given $j \in [0, N)$ we can see that

$$\begin{aligned}\widehat{\hat{\mathbf{A}}^{-1}}[j] &= \frac{1}{\sqrt{N}} \sum_{\omega=0}^{N-1} \hat{\mathbf{A}}[\omega]\, g_{N}^{-\omega \cdot j} = \sum_{\omega=0}^{N-1} \left(\frac{1}{N} \sum_{k=0}^{N-1} \mathbf{A}[k]\, g_{N}^{\omega \cdot k} \right) g_{N}^{-\omega \cdot j} \\ &= \sum_{k=0}^{N-1} \mathbf{A}[k] \left(\frac{1}{N} \sum_{\omega=0}^{N-1} g_{N}^{\omega \cdot (k-j)} \right) = \mathbf{A}[j]\end{aligned}$$

using Equation 1.4. Finally, **Parseval's equality** states that the DFT and IDFT don't change the L^2-norm of an array: For any array $\mathbf{A}$ we have $\|\hat{\mathbf{A}}\|_2 = \|\mathbf{A}\|_2 = \|\hat{\mathbf{A}}^{-1}\|_2$. This is proven by noting that

$$\begin{aligned}\langle \hat{\mathbf{A}}, \hat{\mathbf{A}} \rangle &= \sum_{\omega=0}^{N-1} \left(\frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \mathbf{A}[j]\, g_{N}^{\omega \cdot j} \right) \left(\frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \overline{\mathbf{A}[k]}\, g_{N}^{-\omega \cdot k} \right) \\ &= \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} \mathbf{A}[j]\, \overline{\mathbf{A}[k]} \left(\frac{1}{N} \sum_{\omega=0}^{N-1} g_{N}^{\omega \cdot (j-k)} \right).\end{aligned}$$

Using Equation 1.4 one more time we get

$$\|\hat{\mathbf{A}}\|_2^2 = \langle \hat{\mathbf{A}}, \hat{\mathbf{A}} \rangle = \sum_{j=0}^{N-1} \mathbf{A}[j]\, \overline{\mathbf{A}[j]} = \langle \mathbf{A}, \mathbf{A} \rangle = \|\mathbf{A}\|_2^2.$$

We conclude this section with one final definition. The **discrete convolution** of two arrays, $\mathbf{A}$ and $\mathbf{B}$, is defined as

$$(\mathbf{A} \star \mathbf{B})[k] = \sum_{j=0}^{N-1} \mathbf{A}[j] \cdot \mathbf{B}\left[(k-j) \bmod N\right], \ k \in [0, N).$$

The discrete convolution of two arrays has the following useful relationship to the two arrays' Discrete Fourier Transforms: $\widehat{(\mathbf{A} \star \mathbf{B})}[\omega] = \sqrt{N} \cdot \hat{\mathbf{A}}[\omega] \cdot \hat{\mathbf{B}}[\omega]$ for all $\omega \in [0, N)$. To see this we note that

$$\widehat{(\mathbf{A} \star \mathbf{B})}[\omega] = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} (\mathbf{A} \star \mathbf{B})[j]\, g_{N}^{\omega \cdot j} = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} \mathbf{A}[k] \cdot \mathbf{B}\left[(j-k) \bmod N\right] g_{N}^{\omega \cdot j}.$$

Rearranging the final double sum we have

$$\widehat{(\mathbf{A} \star \mathbf{B})}[\omega] = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \mathbf{A}[k]\, g_{N}^{\omega \cdot k} \sum_{j=0}^{N-1} \mathbf{B}\left[(j-k) \bmod N\right] \, g_{N}^{\omega \cdot (j-k)} = \sqrt{N} \cdot \hat{\mathbf{A}}[\omega] \cdot \hat{\mathbf{B}}[\omega].$$

Using this relationship we can compute the discrete convolution of arrays $\mathbf{A}$ and $\mathbf{B}$ using their DFTs. Specifically, we have

$$\sqrt{N} \cdot \widehat{\hat{\mathbf{A}} \cdot \hat{\mathbf{B}}}^{-1} = (\mathbf{A} \star \mathbf{B}) \tag{1.7}$$

where $(\hat{\mathbf{A}} \cdot \hat{\mathbf{B}})[\omega] = \hat{\mathbf{A}}[\omega] \cdot \hat{\mathbf{B}}[\omega]$ for all $\omega \in [0, N)$ as expected.

1.5.2 The Fast Fourier Transform

Computing the DFT/IDFT of an N-length signal, $\mathbf{A}$, via Equation 1.5/1.6 requires $O(N^2)$-time. The Fast Fourier Transform (FFT) [27] allows us to reduce the computational expense considerably. In this section we will outline how the FFT can be used to reduce the cost of calculating a signal's DFT from $O(N^2)$-time to $O(N \log_2 N)$-time for any length N. In particular, we will later apply the FFT to signals with lengths containing large prime factors. Most FFT treatments only consider signals whose sizes consist solely of small prime factors (e.g., N a power of 2). However, even for N itself a prime, we will later require an $O(N \log_2 N)$-time DFT.

Suppose our signal $\mathbf{A}$ has length N with prime factorization

$$N = p_1 \cdot p_2 \cdots p_m, \text{ where } p_1 \leq p_2 \leq \cdots \leq p_m.$$

Choose an $\omega \in [0, N)$. By splitting $\hat{\mathbf{A}}[\omega]$'s sum (i.e., Equation 1.5) into p_1 smaller sums, one for each possible residue modulo p_1, we can see that

$$\hat{\mathbf{A}}[\omega] = \frac{1}{\sqrt{N}} \sum_{k=0}^{p_1-1} g_N^{k\omega} \cdot \sum_{j=0}^{\frac{N}{p_1}-1} \mathbf{A}[p_1 j + k] \cdot \left(g_N^{p_1}\right)^{\omega \cdot j}.$$

If we define $\mathbf{A}_{k,p_1}$ to be the entries of $\mathbf{A}$ for indexes congruent to $k \in [0, p_1)$ modulo p_1 we have

$$\mathbf{A}_{k,p_1} = \mathbf{A}[j \cdot p_1 + k], \; j \in \left[0, \frac{N}{p_1}\right).$$

Algorithm 1.2 FAST FOURIER TRANSFORM (FFT)

1: **Input: Signal A, length** N**, prime factorization** $p_1 \leq \cdots \leq p_m$
2: **Output:** $\hat{\mathbf{A}}$
3: **if** $N == 1$ **then**
4: Return $\mathbf{A}$
5: **end if**
6: **for** k from 0 to $p_1 - 1$ **do**
7: $\hat{\mathbf{A}}_{k,p_1} \leftarrow \text{FFT}\left(\mathbf{A}_{k,p_1}, \frac{N}{p_1}, p_2 \leq p_3 \leq \cdots \leq p_m\right)$
8: **end for**
9: **for** ω from 0 to N **do**
10: $\hat{\mathbf{A}}[\omega] \leftarrow \frac{1}{\sqrt{p_1}}\left(\sum_{k=0}^{p_1-1} g_N^{k\omega} \cdot \hat{\mathbf{A}}_{k,p_1}\left[\omega \bmod \frac{N}{p_1}\right]\right)$
11: **end for**
12: Return $\hat{\mathbf{A}}$

Our equation for $\hat{\mathbf{A}}[\omega]$ becomes

$$\hat{\mathbf{A}}[\omega] = \frac{1}{\sqrt{p_1}}\left(\sum_{k=0}^{p_1-1} g_N^{k\omega} \cdot \hat{\mathbf{A}}_{k,p_1}\left[\omega \bmod \frac{N}{p_1}\right]\right). \tag{1.8}$$

We can now recursively continue this sum-splitting procedure. In order to compute each of the p_1 discrete Fourier transforms, $\hat{\mathbf{A}}_{k,p_1}$ with $k \in [0, p_1)$, we may split each of their p_1 sums into p_2 additional sums, etc.. Repeatedly sum-splitting in this fashion leads to the **Fast Fourier Transform (FFT)** shown in Algorithm 1.2. Analogous sum-splitting leads to the **Inverse Fast Fourier Transform (IFFT)** which can be obtained from Algorithm 1.2 by replacing line 10's $g_N^{k\omega}$ by $g_N^{-k\omega}$ and replacing each '$\hat{\mathbf{A}}$' by a '$\hat{\mathbf{A}}^{-1}$'.

Let T_N be the time required to compute $\hat{\mathbf{A}}$ from an N-length signal $\mathbf{A}$ via Algorithm 1.2. In order to determine T_N we note that lines 6 – 8 require time $p_1 \cdot T_{\frac{N}{p_1}}$ while lines 9 – 11 take $O(p_1 N)$-time. Therefore we have

$$T_N = O(p_1 N) + p_1 \cdot T_{\frac{N}{p_1}}.$$

However, Algorithm 1.2 is recursively invoked again to solve $\hat{\mathbf{A}}_{1,p_1}, \ldots, \hat{\mathbf{A}}_{p_1-1,p_1}$ by sum-splitting in line 7. Taking this into account we can see that

$$T_{\frac{N}{p_1}} = O\left(\frac{p_2 N}{p_1}\right) + p_2 \cdot T_{\frac{N}{p_1 p_2}}.$$

We now have

$$T_N = O(p_1 N) + p_1 \cdot \left(O\left(\frac{p_2 N}{p_1}\right) + p_2 \cdot T_{\frac{N}{p_1 p_2}} \right) = O\left(N(p_1 + p_2)\right) + p_1 p_2 \cdot T_{\frac{N}{p_1 p_2}}.$$

Repeating this recursive sum-splitting $n \leq m$ times shows us that

$$T_N = O\left(N \cdot \sum_{l=1}^{n} p_1\right) + \prod_{l=1}^{n} p_l \cdot T_{\frac{N}{p_1 \cdots p_n}}.$$

Using that $T_1 = O(1)$ (see Algorithm 1.2's lines 3 – 5) we have

$$T_N = O\left(N \cdot \sum_{l=1}^{m} p_1\right) + O(N) = O(m \cdot p_m \cdot N). \tag{1.9}$$

Note that $m \leq \log_2 N$ while p_m is N's largest prime factor.

Equation 1.9 tells us that the FFT can significantly speed up computation of the DFT. For example, if N is a power of 2 we'll have $m = \log_2 N$ and $p_m = 2$ leaving Algorithm 1.2 with an $O(N \log_2 N)$ runtime. This is clearly an improvement over the $O(N^2)$-time required to use Equation 1.5 directly. However, if N has large prime factors the speed up is less impressive. In the worst case, when N is prime, we have $m = 1$ and $p_1 = N$. This leaves Algorithm 1.2 with a $O(N^2)$ runtime which, in practice, is slower than the direct method. The FFT's inability to handle signal's with sizes containing large prime factors isn't a setback in most applications because the end-user may demand, with little or no repercussions, that signal sizes containing only small prime factors are used. However, in later chapters (i.e., Chapters III, IV, and V) we will need to take many DFT's of signal's with sizes containing large prime factors. Thus, we conclude this subsection with a reduction (along the lines of [14, 97]) of such DFTs to a convolution of slightly larger size.

For any $\omega \in [0, N)$ we may rewrite $\hat{\mathbf{A}}[\omega]$ as

$$\hat{\mathbf{A}}[\omega] = g_N^{-\frac{\omega^2}{2}} g_N^{\frac{\omega^2}{2}} \hat{\mathbf{A}}[\omega] = \frac{g_N^{\frac{\omega^2}{2}}}{\sqrt{N}} \cdot \sum_{j=0}^{N-1} \mathbf{A}[j]\, g_N^{\omega \cdot j - \frac{\omega^2}{2}} = \frac{g_N^{\frac{\omega^2}{2}}}{\sqrt{N}} \cdot \sum_{j=0}^{N-1} \mathbf{A}[j]\, g_N^{\frac{-(\omega-j)^2}{2}} g_N^{\frac{j^2}{2}}. \tag{1.10}$$

The last sum in Equation 1.10 resembles a convolution. In order to make the resemblance more concrete we define two new signals. Let

$$\tilde{\mathbf{A}}[j] = \begin{cases} \mathbf{A}[j] \cdot g_N^{\frac{j^2}{2}} & \text{if } 0 \leq j < N \\ 0 & \text{if } N \leq j < 2^{\lceil \log_2 N \rceil + 1} \end{cases}$$

and let

$$\mathbf{B}[j] = \begin{cases} g_N^{\frac{-j^2}{2}} & \text{if } 0 \leq j < N \\ 0 & \text{if } N \leq j \leq (2^{\lceil \log_2 N \rceil + 1} - N) \\ g_N^{\frac{-\left(j - 2^{\lceil \log_2 N \rceil + 1}\right)^2}{2}} & \text{if } (2^{\lceil \log_2 N \rceil + 1} - N) < j < 2^{\lceil \log_2 N \rceil + 1} \end{cases}.$$

Equation 1.10 now becomes

$$\hat{\mathbf{A}}[\omega] = \frac{g_N^{\frac{\omega^2}{2}}}{\sqrt{N}} \cdot \sum_{j=0}^{2^{\lceil \log_2 N \rceil + 1} - 1} \tilde{\mathbf{A}}[j]\, B\left[(\omega - j) \bmod 2^{\lceil \log_2 N \rceil + 1}\right] = \frac{g_N^{\frac{\omega^2}{2}}}{\sqrt{N}} \cdot (\tilde{\mathbf{A}} \star \mathbf{B})[\omega].$$

This final convolution can be computed by the FFT and IFFT using Equation 1.7 in time $O(N \log_2 N)$. We have now established the following theorem:

Theorem I.2. *Let $\boldsymbol{A}$ be a complex valued signal of length N. $\boldsymbol{A}$'s Discrete Fourier Transform, $\hat{\boldsymbol{A}}$, can be calculated using $O(N \log_2 N)$-time.*

We are now in the position to consider sparse Fourier transforms in the next chapter.

Chapter II

Empirical Evaluation of a Sublinear-Time Sparse DFT Algorithm

In this chapter we empirically evaluate a recently-proposed Fast Approximate Discrete Fourier Transform (FADFT) algorithm, FADFT-2 [54], for the first time. FADFT-2 returns approximate Fourier representations for frequency-sparse signals and works by random sampling. Its implementation is benchmarked against two competing methods. The first is the popular exact FFT implementation FFTW version 3.1. The second is an implementation of FADFT-2's ancestor, FADFT-1 [53]. Experiments verify the theoretical runtimes of both FADFT-1 and FADFT-2. In doing so it is shown that FADFT-2 not only generally outperforms FADFT-1 on all but the sparsest signals, but is also significantly faster than FFTW 3.1 on large sparse signals. Furthermore, it is demonstrated that FADFT-2 is indistinguishable from FADFT-1 in terms of noise tolerance despite FADFT-2's better execution time.

2.1 Introduction

The Discrete Fourier Transform (DFT) for real/complex-valued signals is utilized in myriad applications as is the Fast Fourier Transform (FFT) [27], a model divide-and-conquer algorithm used to quickly compute a signal's DFT. The FFT reduces the time required to compute a length N signal's DFT from $O(N^2)$ to $O(N \log(N))$.

Although an impressive achievement, for huge signals (i.e., N large) the FFT can still be computationally infeasible. This is especially true when the FFT is repeatedly utilized as a subroutine by more complex algorithms for large signals.

In some signal processing applications [77, 72] and numerical methods for multiscale problems [35] only the top few most energetic terms of a very large signal/solution's DFT may be of interest. In such applications the FFT, which computes all DFT terms, is computationally wasteful. This was the motivation behind the development of FADFT-2 [54] and its predecessor FADFT-1 [53]. Given a length N signal and a user provided number m, both of the FADFT algorithms output high fidelity estimates of the signal's m most energetic DFT terms. Furthermore, both FADFT algorithms have a runtime which is primarily dependent on m (largely independent of the signal size N). FADFT-1 and 2 allow any large frequency-sparse (e.g. smooth, or C^∞) signal's DFT to be approximated with little dependence on the signal's mode distribution and relative frequency sizes.

Related work to FADFT-1/2 includes sparse signal (including Fourier) reconstruction methods via Basis Pursuit and Orthogonal Matching Pursuit [18, 104]. These methods, referred to as "compressive sensing" methods, require a small number of measurements (i.e., $O(m \text{ polylog } N)$ samples [99, 42]) from an N-length m-frequency sparse signal in order to calculate its DFT with high probability. Hence, compressive sensing is potentially useful in applications such as MRI imaging where sampling costs are high [83, 84]. However, despite the small number of required samples, current compressive sensing DFTs are more computationally expensive than FFTs such as FFTW 3.1 [50] for all signal sizes and nontrivial sparsity levels. To the best of our knowledge FADFT-1 and 2 are alone in being competitive with FFT algorithms in terms of frequency-sparse DFT run times.

Algorithm Name	Implementation Name	Output for length N signal	Run Time
FFT [27]	FFTW 3.1 [50]	Full DFT of length N signal	$O(N \log(N))$
FADFT-1* [66]	RAℓSFA [66]	m most energetic DFT terms	$O(m^2 \cdot \text{polylog}(N))$
FADFT-1 [53]	AAFFT 0.5	m most energetic DFT terms	$O(m^2 \cdot \text{polylog}(N))$
FADFT-2 [54]	AAFFT 0.9	m most energetic DFT terms	$O(m \cdot \text{polylog}(N))$

Table 2.1: Algorithms and Implementations

A variant of the FADFT-1 algorithm, FADFT-1*, has been implemented and empirically evaluated [66]. However, no such evaluation has yet been performed for FADFT-2. In this chapter FADFT-2 is empirically evaluated against both FADFT-1 and FFTW 3.1 [50]. During the course of the evaluation it is demonstrated that FADFT-2 is faster than FADFT-1 while otherwise maintaining essentially identical behavior in terms of noise tolerance and approximation error. Furthermore, it is shown that both FADFT-1 and 2 can outperform FFTW 3.1 at finding a small number of a large signal's top magnitude DFT terms. See Table 2.1 for descriptions/comparisons of all the algorithms mentioned in this chapter.

The main contributions of this chapter are:

1. We introduce the first publicly available implementation of FADFT-2, the Ann Arbor Fast Fourier Transform (AAFFT) 0.9, as well as AAFFT 0.5, the first publicly available implementation of FADFT-1.

2. Using AAFFT 0.9 we perform the first empirical evaluation of FADFT-2. The evaluation demonstrates that FADFT-2 is generally superior to FADFT-1 in terms of runtime while maintaining similar noise tolerance and approximation error characteristics. Furthermore, we see that both FADFT algorithms outperform FFTW 3.1 on large sparse signals.

3. In the course of benchmarking FADFT-2 we perform a more thorough evaluation of the one dimensional FADFT-1 algorithm than previously completed.

The remainder of this chapter is organized as follows: First, in Section 2.2, we introduce relevant background material and present a short introduction to both FADFT-1 and FADFT-2. Then, in Section 2.3, we present an empirical evaluation of our new FADFT implementations, AAFFT 0.5/0.9. During the course of our Section 2.3.1 evaluation we investigate how AAFFT's runtime varies with signal size and degree of sparsity. Furthermore, we present results on AAFFT's accuracy vs. runtime trade off. Next, in Section 2.3.2, we study AAFFT's noise tolerance and its dependence on signal size, the signal to noise ratio, and the number of signal samples used. Finally, we conclude with a short discussion in Section 2.4.

2.2 Preliminaries

Throughout the remainder of this paper we will be interested in complex-valued signals (or arrays) of length N. We shall denote such signals by $\mathbf{A}$, where $\mathbf{A}(j) \in \mathbb{C}$ is the signal's j^{th} complex value for all $j \in [0, N-1] \subset \mathbb{N}$. Hereafter we will refer to the process of either calculating, measuring, or retrieving any $\mathbf{A}(j) \in \mathbb{C}$ from machine memory as *sampling* from $\mathbf{A}$. Given a signal $\mathbf{A}$ we define its discrete L^2-norm, or Euclidean norm, to be

$$\|\mathbf{A}\|_2 = \sqrt{\sum_{j=0}^{N-1} |\mathbf{A}(j)|^2}.$$

We will also refer to $\|\mathbf{A}\|_2^2$ as $\mathbf{A}$'s energy.

For any signal, $\mathbf{A}$, its Discrete Fourier Transform (DFT), denoted $\widehat{\mathbf{A}}$, is another signal of length N defined as follows:

$$\widehat{\mathbf{A}}(\omega) = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{\frac{-2\pi i \omega j}{N}} \mathbf{A}(j), \ \forall \omega \in [0, N-1].$$

Furthermore, we may recover $\mathbf{A}$ from its DFT via the Inverse Discrete Fourier Trans-

form (IDFT) as follows:

$$\mathbf{A}(j) = \widehat{\widehat{\mathbf{A}}}^{-1}(j) = \frac{1}{\sqrt{N}} \sum_{\omega=0}^{N-1} e^{\frac{2\pi i \omega j}{N}} \widehat{\mathbf{A}}(\omega), \ \forall j \in [0, N-1].$$

We will refer to any index, ω, of $\widehat{\mathbf{A}}$ as a frequency. Furthermore, we will refer to $\widehat{\mathbf{A}}(\omega)$ as frequency ω's coefficient for each $\omega \in [0, N-1]$. Parseval's equality tells us that $\|\widehat{\mathbf{A}}\|_2 = \|\mathbf{A}\|_2$ for any signal. In other words, the DFT preserves Euclidean norm and energy. Note that any non-zero coefficient frequency will contribute to $\widehat{\mathbf{A}}$'s energy. Hence, we will also refer to $|\widehat{\mathbf{A}}(\omega)|^2$ as frequency ω's energy. If $|\widehat{\mathbf{A}}(\omega)|$ is relatively large we'll say that ω is energetic.

We will also refer to three other common discrete signal quantities besides the Euclidean norm throughout the remainder of this paper. The first is the L^1, or taxi-cab, norm. The L^1-norm of a signal $\mathbf{A}$ is defined to be

$$\|\mathbf{A}\|_1 = \sum_{j=0}^{N-1} |\mathbf{A}(j)|.$$

The second discrete quantity is the L^∞ value of a signal. The L^∞ value of a signal $\mathbf{A}$ is defined to be

$$\|\mathbf{A}\|_\infty = \max\{|\mathbf{A}(j)|, j \in [0, N-1]\}.$$

Finally, the third common discrete signal quantity is the signal-to-noise ratio, or SNR, of a signal. In some situations it is beneficial to view a signal, $\mathbf{A}$, as consisting of two parts: a meaningful signal, $\tilde{\mathbf{A}}$, with added noise, $\mathbf{G}$. In these situations, when we have $\mathbf{A} = \tilde{\mathbf{A}} + \mathbf{G}$, we define the $\mathbf{A}$'s signal-to-noise ratio, or SNR, to be

$$\text{SNR}(\mathbf{A}) = 20 \cdot \log_{10}\left(\frac{\|\tilde{\mathbf{A}}\|_2}{\|\mathbf{G}\|_2}\right).$$

Both FADFT algorithms produce output of the form $(\omega_1, C_1), \ldots, (\omega_m, C_m)$ where each $(\omega_j, C_j) \in [0, N-1] \times \mathbb{C}$. We will refer to any such set of $m < N$ tuples

$$\{(\omega_j, C_j) \in [0, N-1] \times \mathbb{C} \text{ s.t. } 1 \leq j \leq m\}$$

as a **sparse Fourier representation** and denote it with a superscript 's'. Note that if we are given a sparse Fourier representation, $\widehat{\mathbf{R}}^{\text{s}}$, we may consider $\widehat{\mathbf{R}}^{\text{s}}$ to be a length-N signal. We simply view $\widehat{\mathbf{R}}^{s}$ as the N length signal

$$\widehat{\mathbf{R}}(j) = \begin{cases} C_j & \text{if } (j, C_j) \in \widehat{\mathbf{R}}^{\text{s}} \\ 0 & \text{otherwise} \end{cases}$$

for all $j \in [0, N-1]$. Using this idea we may, for example, compute $\mathbf{R}$ from $\widehat{\mathbf{R}}^{\text{s}}$ via the IDFT.

We continue with one final definition: An m-term/tuple sparse Fourier representation is m-optimal for a signal $\mathbf{A}$ if it contains the m most energetic frequencies of $\widehat{\mathbf{A}}$ along with their coefficients. More precisely, we'll say that a sparse Fourier representation

$$\widehat{\mathbf{R}}^{\text{s}} = \{(\omega_j, C_j) \in [0, N-1] \times \mathbb{C} \text{ s.t. } 1 \leq j \leq m\}$$

is m**-optimal** for $\mathbf{A}$ if there exists a valid ordering of $\widehat{\mathbf{A}}$'s coefficients by magnitude

$$|\widehat{\mathbf{A}}(k_1)| \geq |\widehat{\mathbf{A}}(k_2)| \geq \cdots \geq |\widehat{\mathbf{A}}(k_j)| \geq \cdots \geq |\widehat{\mathbf{A}}(k_N)|$$

so that $(k_l, \widehat{\mathbf{A}}(k_l)) \in \widehat{\mathbf{R}}^{\text{s}}$ for all $l \in [1, m]$. Note that a signal may have several m-optimal Fourier representations if its frequency coefficient magnitudes are non-unique. For example, there are two 1-optimal sparse Fourier representations for the signal

$$\mathbf{A}(j) = 2e^{\frac{2\pi ij}{N}} - 2ie^{\frac{4\pi ij}{N}}, \ N > 2.$$

However, all m-optimal $\widehat{\mathbf{R}}^{\text{s}}$ for any signal $\mathbf{A}$ will always have both the same unique $\|\mathbf{R}\|_2$ and $\|\mathbf{A} - \mathbf{R}\|_2$ values.

Given an input signal, $\mathbf{A}$, the purpose of both FADFT-1 and FADFT-2 is to identify the m most energetic frequencies, $\omega_1 \leq \cdots \leq \omega_m$, from $\widehat{\mathbf{A}}$ and approximate

Algorithm 2.1 FADFT-1/2 Algorithm

Input: Signal $\mathbf{A}$, Number of most energetic frequencies m, Approximation error ϵ, Failure Probability δ
Output: An approximate m-optimal sparse Fourier representation for $\mathbf{A}$
Set sparse Fourier representation, $\widehat{\mathbf{R}}^{\mathrm{s}}$, to $\emptyset$.
Set energetic frequencies, I, to $\emptyset$.
for all $i \leftarrow 0$ **to** $O\left(\frac{\log(\frac{M}{\epsilon})}{\epsilon^2}\right)$ **do**
    Find a list, L, of energetic frequencies ω with $|(\widehat{\mathbf{A}} - \widehat{\mathbf{R}})(\omega)|^2 \geq O\left(\frac{\epsilon^2}{m}\right) \cdot \| \mathbf{A} - \mathbf{R} \|_2^2$.
    Set $I = I \cup L$.
    Update $\widehat{\mathbf{R}}^{\mathrm{s}}$ by estimating coefficients $\forall \omega \in I$ so that $|(\widehat{\mathbf{A}} - \widehat{\mathbf{R}})(\omega)|^2 \leq O\left(\frac{\epsilon^2}{|I|+m}\right) \cdot \| \mathbf{A} - \mathbf{A}_I \|_2^2$.
end for
Output top m terms of $\widehat{\mathbf{R}}^{\mathrm{s}}$.

their coefficients. Put another way, the goals of both FADFT-1 and FADFT-2 are as follows: Given an input signal, $\mathbf{A}$, both FADFT-1 and FADFT-2 are designed to output an approximate m-optimal sparse Fourier representation for $\mathbf{A}$.

2.2.1 FADFT-1 Algorithm

The main result of [53] is an algorithm, FADFT-1, with the following properties: Denote an m-optimal Fourier representation of a one dimensional signal $\mathbf{A}$ of length N by $\widehat{\mathbf{R}}^{\mathrm{s}}_{\mathrm{opt}}$ and assume that, for some M, we have

$$\frac{1}{M} \leq \| \mathbf{A} - \mathbf{R}_{\mathrm{opt}} \|_2 \leq \| \mathbf{A} \|_2 \leq M.$$

Then, the FADFT-1 algorithm uses time and space $m^2 \cdot \mathrm{poly}(\log(\frac{1}{\delta}), \log(N), \log(M), \frac{1}{\epsilon})$ to return a sparse Fourier representation $\widehat{\mathbf{R}}^{\mathrm{s}}$ such that

$$\| \mathbf{A} - \mathbf{R} \|_2^2 \leq (1+\epsilon) \| \mathbf{A} - \mathbf{R}_{\mathrm{opt}} \|_2^2$$

with probability at least $1 - \delta$.

Note that for $m \ll N$ the FADFT-1 algorithm is sub-linear time. Also note that the ϵ and δ parameters allow the user to manage approximation error and failure probability, respectively. For a pseudo-code outline of FADFT-1/2 see Algorithm 2.1.

FADFT-1 is a randomized greedy pursuit algorithm which, in this case, means that it iteratively produces approximations to an $\widehat{\mathbf{R}}^{\mathrm{s}}_{\mathrm{opt}}$ which get better with high

probability as time goes on (i.e. i increases in Algorithm 1). Intuitively, Algorithm 1's step 6 will discover frequencies in $\mathbf{A}-\mathbf{R}$ with large magnitudes relative to $\| \mathbf{A}-\mathbf{R} \|_2^2$ with high probability (the larger the frequency's magnitude, the better chance it will be found). Hence, as long as step 8 continues to estimate frequency coefficients to high enough precision, important frequencies which haven not yet been detected will become increasingly overwhelming in $\mathbf{A} - \mathbf{R}$ as $\| \mathbf{A} - \mathbf{R} \|_2^2$ shrinks (i.e., as $\widehat{\mathbf{R}}^{\mathrm{s}} \rightarrow$ an $\widehat{\mathbf{R}}^{\mathrm{s}}_{\mathrm{opt}}$). The end result is that it becomes increasingly difficult for the top m frequencies in $\mathbf{A}$ to evade detection as time goes on. If the search continues long enough they will all be found with high probability.

2.2.2 FADFT-2 Algorithm

The FADFT-2 algorithm [54] is identical to the FADFT-1 algorithm with two main exceptions. First, FADFT-2 utilizes a faster method of coefficient estimation (Algorithm 1's step 8) than FADFT-1 does. Second, FADFT-2 also samples from intermediate sparse representations via a faster algorithm than the naive method used by FADFT-1. In order to better understand the differences between FADFT-1 and FADFT-2, we next compare how both algorithms perform coefficient estimation. We refer the reader to [53, 54] for more detailed descriptions of each algorithm's energetic frequency isolation and identification (i.e., Algorithm 2.1's step 6) methods.

FADFT-1 Coefficient Estimation

As before, let $\mathbf{A}$ be a given input signal of length N. Furthermore, suppose that we've identified an energetic frequency, ω_{big}, whose value we wish to estimate. Independently choose two uniformly random integers $c, l \in [0, N-1]$ making sure that l is invertible $mod\ N$. We can now estimate ω_{big}'s coefficient by computing the

following sum:

$$\widehat{\mathbf{A}}'(\omega_{\text{big}}) = \frac{\sqrt{N}}{K} \sum_{k=0}^{K-1} e^{\frac{-2\pi i \omega_{\text{big}}(c+l\cdot k)}{N}} A(c + l \cdot k) \tag{2.1}$$

where $K \ll N$ will be specified later. Here we have $\mathbf{E}[\widehat{\mathbf{A}}'(\omega_{\text{big}})] = \widehat{\mathbf{A}}(\omega_{\text{big}})$ and $\text{Var}[\widehat{\mathbf{A}}'(\omega_{\text{big}})]$ is $O\left(\frac{\|\mathbf{A}\|_2^2}{K}\right)$. Hence, if we let K be $O(\frac{1}{\nu})$ we'll have $|\widehat{\mathbf{A}}'(\omega_{\text{big}}) - \widehat{\mathbf{A}}(\omega_{\text{big}})|^2 < \nu \parallel \mathbf{A} \parallel_2^2$ with constant probability by the Markov inequality. If we next approximate $\widehat{\mathbf{A}}(\omega_{\text{big}})$ by taking the median of $E = O(\log(\frac{1}{\delta}))$ copies of i.i.d. $\widehat{\mathbf{A}}'(\omega_{\text{big}})$'s the Chernoff inequality tells us we'll achieve precision $\nu \parallel \mathbf{A} \parallel_2^2$ with probability $\geq 1 - \delta$. See [53] for details.

Note that K is proportional to the desired number, m, of most energetic frequencies (FADFT-1 step 8 requires that frequency coefficients are estimated with accuracy $O\left(\frac{\epsilon^2}{|I|+m}\right) \cdot \parallel \mathbf{A} - \mathbf{A}_I \parallel_2^2$). Furthermore, we'll need to estimate coefficients for at least m frequencies. Hence, the time to find an m-term Fourier representation using this coefficient estimation method as stated will be proportional to m^2. Fortunately there are also $O(m \cdot \text{polylog}(m))$-time methods for calculating m coefficients to within the same precision.

FADFT-2 Coefficient Estimation

The main difference between FADFT-1 and FADFT-2 is that FADFT-2 utilizes Unequally Spaced Fast Fourier Transform (USFFT) techniques [9, 45, 47, 78] both to sample from sparse representations and to perform coefficient estimation (i.e., compute Equation 2.1 for m frequencies) in $O(m \cdot \text{polylog}(m))$-time. In this way FADFT-2 is able to avoid all FADFT-1's $O(m^2)$-time Fourier matrix multiplications. A brief explanation of how FADFT-2 utilizes an USFFT along the lines of [9] to perform coefficient estimation follows. An analogous method allows FADFT-2 to sample m values from the inverse transform of a m-term Fourier representation in

time proportional to $m \cdot \text{polylog}(m)$.

Suppose we want to estimate the coefficients of m frequencies $\omega_1, \ldots, \omega_m$ in an input signal $\mathbf{A}$ of length N. Independently choose two uniformly random integers $c, l \in [0, N-1]$ making sure that l is invertible $mod\ N$. In order to estimate the m frequencies' coefficients we need to calculate Equation 2.1 for $j = 1, \ldots, m$:

$$\widehat{\mathbf{A}}'(\omega_j) = \frac{\sqrt{N}}{K} \sum_{k=0}^{K-1} e^{\frac{-2\pi i \omega_j (c+l\cdot k)}{N}} A(c + l \cdot k) = \frac{\sqrt{N}}{K} e^{\frac{-2\pi i \omega_j c}{N}} f(\omega'_j) \text{ for } j = 1, \ldots, m,$$

where we let $f(\omega) = \sum_{k=0}^{K-1} e^{\frac{-2\pi i \omega k}{N}} A(c + l \cdot k)$ and $\omega'_j = \omega_j \cdot l$. Now, let $R = 8 \cdot K$ and define r_ω to be the integer $r \in [0, R]$ that minimizes $|\omega - \frac{r \cdot N}{R}|$. Expanding f in a Taylor series we see that

$$f(\omega_j) = f(r_{\omega_j} N/R) + f'(r_{\omega_j} N/R) \cdot \Delta_{\omega_j} + f''(r_{\omega_j} N/R) \cdot (\Delta^2_{\omega_j}/2) + \ldots$$

where

$$\Delta_{\omega_j} = \omega_j - r_{\omega_j} \frac{N}{R} \text{ for each j.}$$

Calculating the derivatives and setting $a_k = A(c + l \cdot k)$ for $k \in [0, K-1]$ (0 otherwise) we get that

$$f(\omega_j) = \left(\sum_{k=0}^{R-1} a_k e^{-2\pi i r_{\omega_j} k/R} \right) + \frac{-2\pi i \Delta_{\omega_j}}{N} \cdot \left(\sum_{k=0}^{R-1} a_k k e^{-2\pi i r_{\omega_j} k/R} \right) + \frac{1}{2} \left(\frac{-2\pi i \Delta_{\omega_j}}{N} \right)^2 \cdot \left(\sum_{k=0}^{R-1} a_k k^2 e^{-2\pi i r_{\omega_j} k/R} \right) + \ldots . \tag{2.2}$$

Each sum in the expression above may be calculated for all r_{ω_j} simultaneously in time $O(K \log(K))$ via the FFT. And, since $|\frac{-2\pi i \Delta_{\omega_j}}{N}| \cdot K < \frac{1}{2}$, we only need $O(\log(\frac{1}{\nu}))$ such sums to get $\pm\nu \parallel \mathbf{A} \parallel_2^2$ precision. The upshot is that we only need time $O\left(m \cdot \text{polylog}(m) \log(\frac{1}{\nu})\right)$ in order to estimate the coefficients of ω_1 through ω_m.

It is important to note that in Equation 2.1 $\mathbf{A}$ is sampled along an arithmetic progression. The k^{th} sample is at location $c + l \cdot k$. It is exactly sampling $\mathbf{A}$ in this

fashion that allows USFFT techniques to be utilized. To the best of our knowledge all known USFFT methods require either frequencies or sample positions to be represented as an arithmetic progression. Depending on the user's ability to dictate what samples are used, this may or may not be a weakness of FADFT-2.

FADFT-2 Result

The main result of [54] is that FADFT-2 has the following properties: Denote an m-optimal Fourier representation of a one dimensional signal $\mathbf{A}$ of length N by $\widehat{\mathbf{R}}^{\mathrm{s}}_{\mathrm{opt}}$ and assume that, for some M, we have

$$\frac{1}{M} \leq \| \mathbf{A} - \mathbf{R}_{\mathrm{opt}} \|_2 \leq \| \mathbf{A} \|_2 \leq M.$$

Then, the FADFT-2 algorithm uses time and space $m \cdot \mathrm{poly}(\log(\frac{1}{\delta}), \log(N), \log(M), \frac{1}{\epsilon})$ to return a Fourier representation $\widehat{\mathbf{R}}^{\mathrm{s}}$ such that

$$\| \mathbf{A} - \mathbf{R} \|_2^2 \leq (1+\epsilon) \| \mathbf{A} - \mathbf{R}_{\mathrm{opt}} \|_2^2$$

with probability at least $1 - \delta$. When working to double (i.e., 64-bit) precision it should be safe to assume

$$M \approx \max(10^{16}, \| \mathbf{A} \|_2).$$

In other words, even if $\mathbf{A}$ is an exact superposition (e.g., a sinusoid), machine noise (i.e., roundoff errors) will generally limit the accuracy of our m-optimal Fourier representation $\widehat{\mathbf{R}}^{\mathrm{s}}_{\mathrm{opt}}$.

Note that this result indicates FADFT-2 is essentially *linear* in m as opposed to FADFT-1 which is *quadratic*. Second, it is important to note that FADFT-2 is designed to quickly output a high fidelity approximation to FADFT-1's output for any given input signal without having to utilize any extra information (e.g., signal samples). Hence, if given good parameter settings and a frequency-sparse input signal, both versions of FADFT should yield approximately the same output.

2.3 FADFT Implementation and Evaluation

Both FADFT-1 and FADFT-2 were implemented in C++ utilizing the Standard Template Library (for readability). Hereafter these implementations will be referred to as different versions of the Ann Arbor Fast Fourier Transform (AAFFT). Version 0.5 of AAFFT is the straightforward quadratic time in m, the desired number of largest Fourier terms, implementation the FADFT-1 algorithm. Version 0.9 of AAFFT is an implementation of FADFT-2. All AAFFT source code and documentation is available at [64].

Calculating the optimal m-term Fourier representation for a length-N signal may be done naively by computing the entire DFT and then reporting its largest m Fourier terms. This naive approach requires time $O(N \log(N))$ using an FFT implementation. Given the absence of other fast competitors, below we benchmark AAFFT 0.5 and AAFFT 0.9 against this naive approach with FFTW version 3.1 [50] serving as the FFT implementation. All experiments were carried out on a dual 3.6 GHz processor multi-threaded Dell desktop with 3G of memory. Below FFTW will always refer to FFTW version 3.1 using an FFTW_ESTIMATE In_Place_Transform_Data plan. In order to help us remain as unbiased as possible we don't include any sorting or non-zero coefficient search time in FFTW's reported run times below. All reported signal sizes are powers of 2.

It is important to note that both AAFFT implementations rely on 20 different user-provided parameter settings that influence approximation error, runtime, the number of signal samples utilized, memory usage, etc.. For the sake of readability we only mention individual parameters in subsequent sections when absolutely necessary. Instead, we will report observable consequences of various parameter settings

(e.g. runtime, approximation error, etc.) without providing detailed descriptions of what parameter settings produced them. For a detailed discussion of all AAFFT parameters along with example parameter settings used for various experiments below we invite the reader to visit http://aafftannarborfa.sourceforge.net/. Besides the AAFFT source code, this site contains a file called README.pdf which contains detailed parameter information.

2.3.1 Empirical Evaluation: Run Time and Accuracy

Run Time: In Figure 2.1 we report how AAFFT's run time changes with input signal size. The 10 reported signal sizes for each implementation are $2^{17}, 2^{18}, \ldots, 2^{26}$. The run time reported at each signal size for each implementation is the average of 1000 test signal DFT times. It is important to remember that AAFFT is randomized and approximate so the run time depends on how much error the user is willing to tolerate. Parameters for both AAFFT implementations were chosen so that the average L^1 (taxi-cab) error between AAFFT and FFTW's returned representations was between 10^{-5} and 10^{-7} at each signal size.

The test signals were randomly generated 60-frequency exact superpositions. Hence, m was fixed to 60 for all the AAFFT runs used to create Figure 2.1. The magnitude of each non-zero frequency was 1 so that all frequencies were of the same importance. This is the most difficult type of sparse signal for AAFFT since the energetic frequency isolation and identification portion of the FADFT algorithm works best at finding single frequencies larger than all others. For each of the 1000 test superpositions we generated 60 integers $\omega_1, \ldots, \omega_{60} \in [0, N-1]$ and 60 phases $p_1, \ldots, p_{60} \in [0, 2\pi]$ uniformly at random. We then set the test signal, $\mathbf{A}$, to be

$$\mathbf{A}(x) = \frac{1}{\sqrt{N}} \sum_{j=1}^{60} e^{2\pi i p_j} e^{\frac{2\pi i \omega_j x}{N}} \ \forall x \in [0, N-1].$$

In Figure 2.1 below we graph the maximum, minimum, and mean run times for FFTW 3.1, AAFFT 0.5, and AAFFT 0.9 over the 1000 test signals at each signal size. At each data point the top and bottom of the point's vertical line gives the associated implementation's maximum and minimum run times, respectively. The data point itself is located at the associated implementation's mean run time. Note in Figure 2.1 below that both AAFFT 0.9 and AAFFT 0.5 have relatively constant run times despite being randomized.

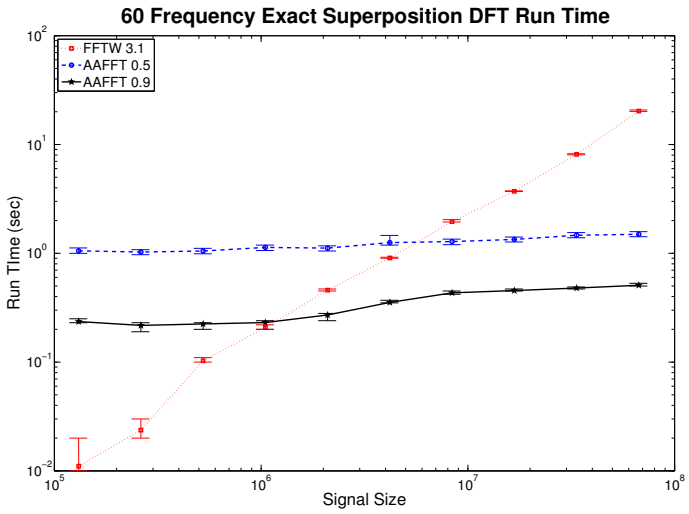

Figure 2.1: AAFFT Run Time Vs. Signal Size

Recall that AAFFT 0.9's theoretical run time is $m \cdot \text{poly}(\log(\frac{1}{\delta}), \log(N), \log(M), \frac{1}{\epsilon})$ where m is the number of desired output representation terms, $1 - \delta$ is the probability of achieving multiplicative error bound ϵ, M is a bound for the signal's energy, and N is the signal size. Similarly, AAFFT 0.5's theoretical run time is $m^2 \cdot \text{poly}(\log(\frac{1}{\delta}), \log(N), \log(M), \frac{1}{\epsilon})$. Figure 2.1's run times were generated from sparse exact 60 superpositions with all terms magnitude 1 so that m and M remained fixed for all experiments. Furthermore, requiring that the average L^1 (taxi-cab) error between AAFFT and FFTW's returned representations be between 10^{-5} and 10^{-7} at

each signal size kept δ and ϵ fairly stable. Hence, we expect the run times of AAFFT 0.5 and AAFFT 0.9 to increase with signal size like polylog(N). Our expectation does appear to be realized in Figure 2.1 where we see both AAFFT 0.9 and AAFFT 0.5's run times gently increase with N. Note that AAFFT 0.9 is faster than AAFFT 0.5 for all signal sizes when $m = 60$. Figure 2.1 also contains a graph of FFTW 3.1's run times which appear to increase something like the expected $O(N \log N)$. Note that for signal sizes greater than $2^{20}(i.e.\ 1,048,576)$ AAFFT 0.9 is faster at recovering an exact 60 frequency superposition than FFTW 3.1. Similarly, AAFFT 0.5 begins to beat FFTW 3.1 at signal sizes greater than $2^{23}(i.e.\ 8,388,608)$.

In the group of tests used to produce Figure 2.2 below we held the signal size N constant at $2^{22} = 4,194,304$ and varied m. As before, at each reported number of superposition frequencies we graph the maximum, minimum, and mean run times for FFTW 3.1, AAFFT 0.5, and AAFFT 0.9 over the 1000 tests. Each test run was performed on a randomly-generated test m-superposition similar to above. For a fixed m we create each test signal by generating m integers $\omega_1, \ldots, \omega_m \in [0, N-1]$ and m random phases $p_1, \ldots, p_m \in [0, 2\pi]$. We then set the the test signal, $\mathbf{A}$, to be $\mathbf{A}(x) = \frac{1}{2048} \sum_{j=1}^{m} e^{2\pi i p_j} e^{\frac{2\pi i \omega_j x}{N}}\ \forall x \in [0, 4194303]$. Again, as above, we required that the average L^1 (taxi-cab) error between AAFFT and FFTW's returned representations was between 10^{-5} and 10^{-7} at each superposition size m. We expect little dependence on M, N, ϵ, and δ in our AAFFT runtime results.

As expected, AAFFT 0.5 displays quadratic run time in m while AAFFT 0.9's run time looks linear. Also, not surprisingly, FFTW 3.1's run time is essentially constant. Note that AAFFT 0.9 can recover superpositions with ≤ 135 frequencies more quickly than FFTW at signal size 2^{22}. Meanwhile, AAFFT 0.5 is only capable of computing ≤ 45-sparse signals more quickly then FFTW. Also notice that

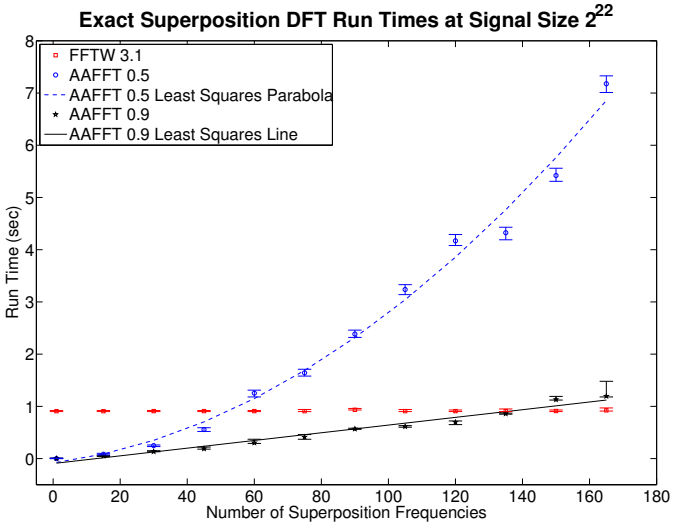

Figure 2.2: AAFFT Run Time Vs. Superposition Size

AAFFT 0.9 is competitive with AAFFT 0.5 for all values of m. AAFFT 0.5 is, on average, slightly faster than AAFFT 0.9 for small frequency (e.g., $m = 1$) superpositions. This is due to AAFFT 0.5's naive $\mathrm{O}(m^2)$-time coefficient estimation and sparse Fourier representation sampling (I)DFT matrix/vector multiplications having a smaller constant runtime factor than the USFFT techniques AAFFT 0.9 employs. However, for all $m \geq 15$ AAFFT 0.9's $O(m \cdot \mathrm{polylog}(m))$-time USFFT techniques outperform AAFFT 0.5's straightforward (I)DFT methods. Hence, AAFFT 0.9 is generally faster than AAFFT 0.5 for all values of $m \geq 15$.

Approximation Error: When using AAFFT for numerical analysis applications one may desire greater average accuracy than the 5 or 6 digits per term guaranteed above. Hence, we next present some results concerning AAFFT's accuracy vs. run time trade-offs. As before, every Figure 2.3 data point results from 1000 runs on randomly generated 60-superpositions whose frequencies each have magnitude 1 with random phase. Furthermore, the signal sizes, N, are once again fixed to 2^{22} for every trial run.

Recall that AAFFT 0.9 frequency coefficient estimation (as well as representation sampling) is carried out by using truncated Taylor series with T terms in order to calculate multiple frequencies' coefficient estimates at once (see Section 2.2.2). Also recall that for each identified frequency, ω_{big}, the median of E such coefficient estimates becomes ω_{big}'s coefficient update for each round of the program (see Section 2.2.2). In general, the larger T and E are the more accurate and reliable the final frequency coefficient estimates should be. Note that AAFFT 0.5 works in the same way except that Taylor series are not used. Thus, AAFFT 0.5 does not depend on T.

In Figure 2.3 below we investigated the effect of varying E and T on AAFFT 0.5/0.9's accuracy. All other parameters were held fixed. To create Figure 2.3 we varied E for AAFFT 0.5 and three different T-valued AAFFT 0.9 variants (with $T = 5$, 10, and 15). The mean, mean + 1 standard deviation, and maximum L^∞ approximation error values over each of five 1000 run trials (with $E = 1$, 3, 5, 7, and 9) were graphed for all 4 AAFFT versions. In order to give a better idea of AAFFT's approximation error vs. run time trade offs, the L^∞ values were graphed against their associated trial's maximum run time for each data point.

As expected, the runtime (and, generally, accuracy) of all 4 AAFFT variants increased monotonically with E. Hence, for each of the 4 curves in Figure 2.3 the uppermost-left data point corresponds to $E = 1$, the second highest-left data point to $E = 3$, etc.. Also as expected, we can see that both AAFFT 0.9's accuracy and runtime tend to increase with T. The 5 Taylor term variant of AAFFT 0.9 is only accurate to $\approx 10^{-5}$ despite the number of medians used. On the other hand, the 10 Taylor term AAFFT 0.9 variant is comparable in accuracy to both AAFFT 0.5 and the 15 Taylor term AAFFT 0.9 variant for each E value. Furthermore, we can see

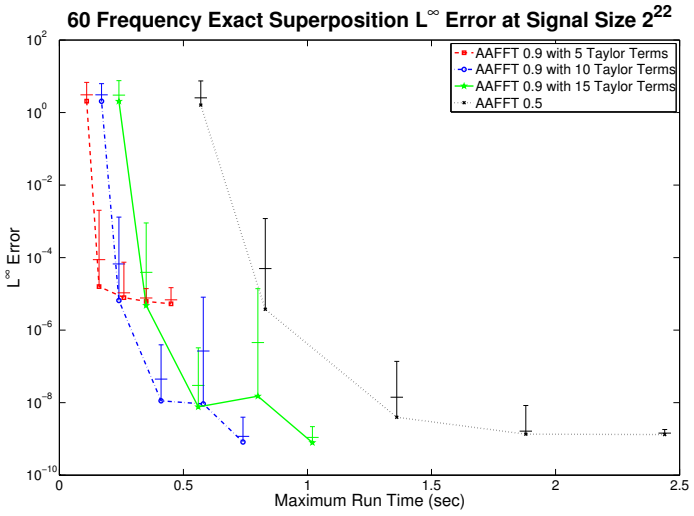

Figure 2.3: AAFFT Error Vs. Parameters

that AAFFT 0.9 with 10 Taylor terms appears to be faster than both AAFFT 0.5 and AAFFT 0.9 with 15 Taylor terms.

Both AAFFT 0.5/0.9 and FFTW 3.1 utilize double precision (i.e., 64-bit) arithmetic/variables. Hence, for Figure 2.3's experiments FFTW 3.1 always reported frequency coefficients that were accurate to within 10^{-15}. Looking at Figure 2.3 above it appears as if AAFFT 0.5/0.9's average worst-case frequency coefficient estimates are only accurate to within $\approx 10^{-9}$ at best. However, we expect to get better accuracy by increasing AAFFT's K parameter (see Section 2.2.2's Equation 2.1) which was fixed at 128 during these experiments [64]. For example, in the extreme case where K is increased to N, we can expect that AAFFT 0.5/0.9 will calculate each energetic frequency's coefficient to within $\approx 10^{-12}$ or better. More generally, as K is increased toward N we expect AAFFT's accuracy (and run time) to also increase. However, testing the limits of AAFFT's accuracy is left as future work.

2.3.2 Empirical Evaluation: Noise Tolerance and Sampling Complexity

Noise Tolerance: Our next series of experiments report on the noise tolerance of both the AAFFT 0.9 and AAFFT 0.5 implementations. In order to determine each implementation's level of noise tolerance we will work with signals consisting of a single non-zero coefficient frequency buried in complex Gaussian noise. Given such signals we will try to determine how the noise level influences AAFFT's ability to recover the hidden frequency. In essence, we wish to investigate AAFFT's utility as a denoiseing tool.

Below we work exclusively with signals consisting of a single non-zero frequency signal, $\tilde{\mathbf{A}}$, in Gaussian noise. Let N be our signal size. Then,

$$\tilde{\mathbf{A}}(x) = Ce^{2\pi ip} \cdot e^{\frac{2\pi i\omega x}{N}} \ \forall x \in [0, N-1],$$

where $C \in \mathcal{R}^+$ is chosen to control the signal to noise ratio, p is a uniformly random phase $\in [0, 2\pi]$, and ω is a uniformly random frequency $\in [0, N-1]$. As above, we generate a new $\tilde{\mathbf{A}}$ for every AAFFT trial run.

Furthermore, in all subsequent experiments each trial run's Gaussian noise is (re)generated each run by adding standard (i.e., mean 0, variance 1) normally distributed values independently to both the real and imaginary components of every element of the complex hidden signal $\tilde{\mathbf{A}}$. All normally distributed values are generated by the Polar Box-Muller method [15]. For the remainder of this chapter we'll denote the noise added to $\tilde{\mathbf{A}}(x)$ by $\mathbf{G}(x)$ $\forall x \in [0, N-1]$. Hence, every trial run's input signal, $\mathbf{A}$, is of the form $\mathbf{A} = \tilde{\mathbf{A}} + \mathbf{G}$. The signal to noise ratio, or SNR, of $\mathbf{A}$ is $20 \cdot \log_{10}\left(\frac{\|\tilde{\mathbf{A}}\|_2}{\|\mathbf{G}\|_2}\right)$. Furthermore, for fixed $\tilde{\mathbf{A}}$, note that

$$\min\left\{k \in [0, N-1] \text{ s.t. } |\widehat{\mathbf{A}}(k)| = \|\widehat{\mathbf{A}}\|_\infty\right\}$$

is $\tilde{\mathbf{A}}$'s single nonzero frequency with high probability (depending on the SNR).

For a fixed m, decreasing $\mathbf{A}$'s SNR will tend to increase $\| \mathbf{A} - \mathbf{R}_{\text{opt}} \|_2^2$. Hence, looking back at Sections 2.2.1/2.2.2, we will have weaker accuracy guarantees for the m-term Fourier representations returned by AAFFT 0.5/0.9 as SNR decreases. If the accuracy guarantees become weak enough we will not even be able to expect AAFFT to correctly discover which $\widehat{\mathbf{A}}$ frequencies are most energetic. Thus, decreasing $\mathbf{A}$'s SNR generally requires us to both increase m and/or decrease ϵ in order to properly determine, and then estimate the coefficients of, $\mathbf{A}$'s most energetic DFT modes. Therefore, the lower the SNR, the more samples and run time AAFFT will need in order to recover our $\widehat{\widetilde{\mathbf{A}}}$ frequency with high probability.

Figure 2.4 investigates the probability of AAFFT 0.9 and 0.5 successfully recovering an input signal $\mathbf{A}$'s smallest DFT frequency of largest coefficient magnitude. Each Figure 2.4 graph was generated using 200 three-dimensional (i.e., # AAFFT sample points $\times$ average $\mathbf{A}$ SNR $\times$ success probability) data points. Each data point was generated via 1000 AAFFT trail runs. The signal size, N, of all data points' trial signals was fixed at 2^{22}.

Every Figure 2.4 data point had its 1000 runs' input signals' (i.e., $\mathbf{A}$s') SNR values controlled through the use of a uniform magnitude value, C, over its 1000 randomly generated single frequency $\tilde{\mathbf{A}}$'s. Though new Gaussian noise was generated every run, each data point's 1000 input signal SNRs were tightly grouped around the mean SNR (standard deviation from each of the 200 data point's reported mean SNRs was < 0.0025). Each data point plots the mean SNR value of its 1000 associated runs against each Figure 2.4 plot's vertical axis.

Note that the sub-linear run times of AAFFT 0.9 and AAFFT 0.5 necessitate that neither method can read the entire input signal $\mathbf{A}$. During Figure 2.4's experiments the number of samples used by both AAFFT 0.9 and 0.5 depended deterministically

on a subset of parameter settings common to both implementations. For each of the 200 data points making up Figure 2.4 a uniform set of parameters were used across each point's 1000 trial runs. The number of signal samples resulting from each data point's parameters (listed as a percentage of N) is plotted against Figure 2.4's horizontal axis.

Each Figure 2.4 plot's color/shade at any (percent sampled x, average SNR y) pair indicates the probability of AAFFT 0.9/0.5 successfully determining the smallest frequency, k, so that $|\widehat{\mathbf{A}}(k)| = \|\widehat{\mathbf{A}}\|_\infty$ for a trail signal $\mathbf{A}$ with SNR y if AAFFT 0.9/0.5 is only allowed to use $\frac{x \cdot N}{100}$ samples from $\mathbf{A}$. For each data point the probability of success was calculated from its 1000 trail runs by counting the number of times AAFFT 0.9/0.5 returned the same minimum largest-magnitude frequency as FFTW 3.1, divided by 1000. Figure 2.4's color bars indicate how the gray scale values in each graph correspond to success probabilities. Lighter values indicate high success probabilities while darker values indicated lower success probabilities.

Looking at Figure 2.4 we can see that there is no significant difference between the performance of AAFFT 0.5 and 0.9 on noisy signals. This is unsurprising given that AAFFT 0.9 was, in essence, designed to quickly return a high fidelity approximation to AAFFT 0.5's output for any given input signal without using additional samples. Thus, we'll concentrate on AAFFT 0.9's noise tolerance results for the remainder of this section.

Figure 2.4's AAFFT 0.9 graph (bottom graph) behaves as expected. If we fix any SNR value we can see that increasing the number of samples AAFFT 0.9 uses allows an increase in success probability. In effect, the shading lightens from left to right along any SNR line. Similarly, if we fix the number of AAFFT 0.9 samples and increase the SNR the shading lightens (i.e. success probability increases). In general,

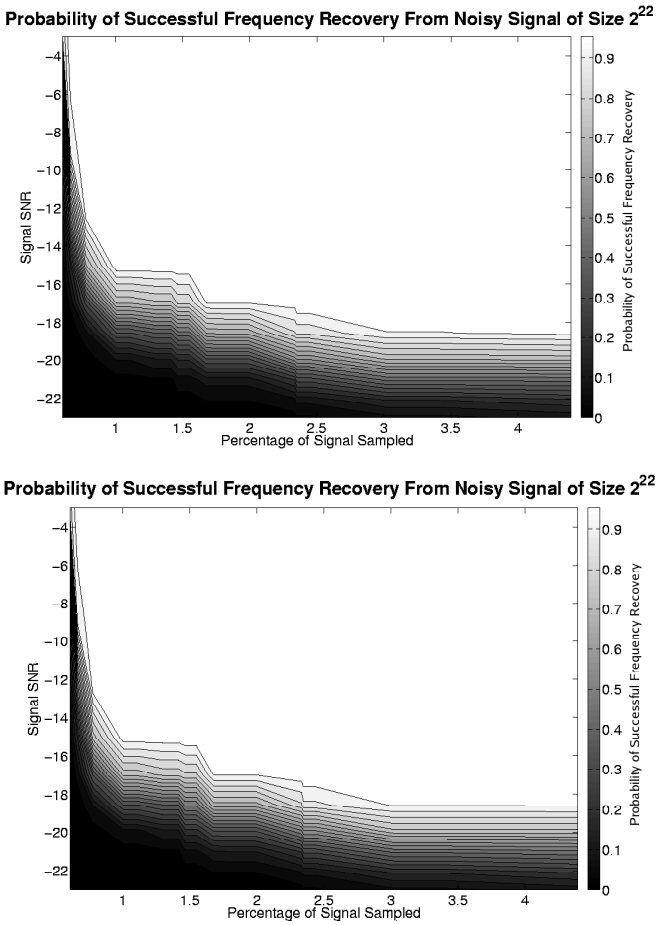

Figure 2.4: Probability of Hidden Signal Recovery for AAFFT 0.5 (top) and AAFFT 0.9 (bottom)

Figure 2.4 indicates that AAFFT tolerates small amounts of noise (SNR > -15) well as long as it's allowed to use $> 42,000$ samples ($> 1\%$ of 2^{22}).

In order to more clearly see AAFFT 0.9's noise tolerance results for lower SNR values we present Figure 2.5. Figure 2.5 shows four fixed SNR success probability curves from Figure 2.4's AAFFT 0.9 graph. Again, as expected, Figure 2.5 demonstrates that AAFFT 0.9 is more tolerant of smaller levels of noise than larger levels (i.e. larger SNR value curves are higher than lower SNR value curves). Furthermore, each SNR curve increases with increasing AAFFT sample usage. Looking at the -17 SNR curve it appears as if AAFFT 0.9 will always successfully locate the smallest

signal of largest coefficient magnitude when it's allowed to use $> 100,000$ samples.

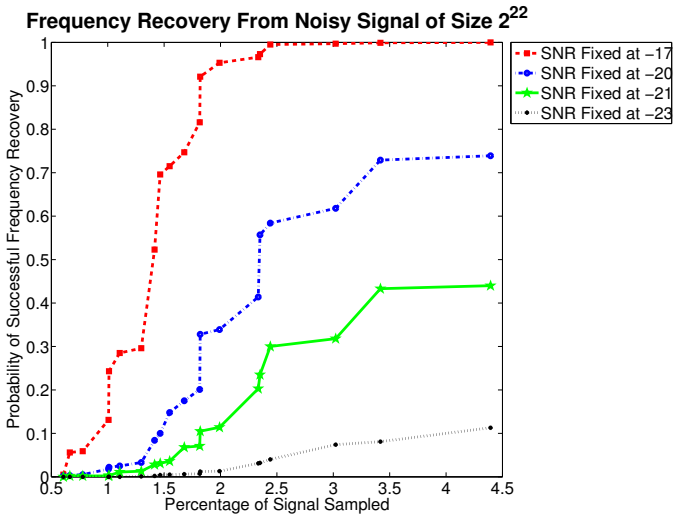

Figure 2.5: AAFFT 0.9's Probability of Hidden Signal Recovery from Signals with Various Noise Levels

Figure 2.6 illustrates how signal size influences success probability. Every data point in Figure 2.6 is generated by 1000 trial runs on randomly generated input signals $\mathbf{A}$. The 1000 signals used for each data point vary in size from 2^{17} through 2^{26}. All sizes are powers of two. Otherwise each trial signal $\mathbf{A}$ is created just as before (i.e. consists of a randomly generated single frequency signal, $\tilde{\mathbf{A}}$, with added Gaussian noise, $\mathbf{G}$). The standard complex Gaussian noise is regenerated for each trial run via the Polar Box-Muller method. For every Figure 2.6 data point the magnitude of $\tilde{\mathbf{A}}$ is chosen so that mean SNR of all the data point's trail signals is tightly grouped around -17 (SNR standard deviations for all data points are < 0.013). Probabilities of successfully calculating the minimum frequency of maximum energy are also calculated just as before.

Figure 2.6 presents the variation of success probability with signal size for AAFFT 0.9 with three different numbers of sample cutoffs. Again, the number of samples was determined by AAFFT's parameter settings. The data points to use for each cutoff

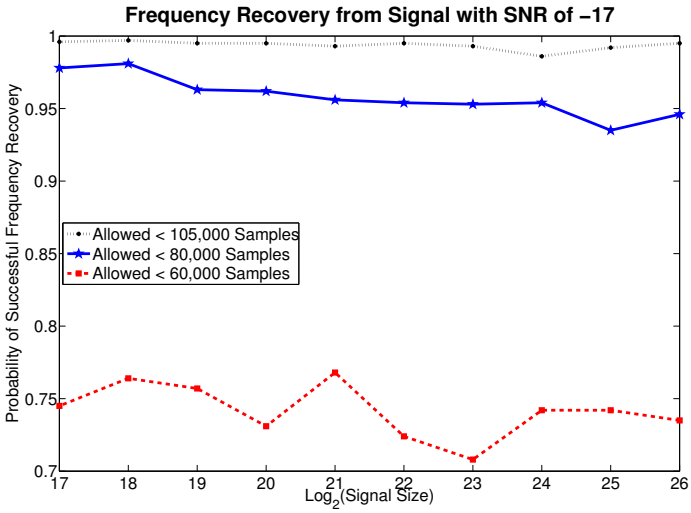

Figure 2.6: Probability of Hidden Signal Recovery Vs Signal Size for AAFFT 0.9

curve were selected as follows: For each signal size, 6 data points were created each using a different number of samples $< 105,000$. The data point for the y-sample cutoff curve at signal size x is the x-size data point using $< y$ samples with the highest success probability.

Looking at Figure 2.6 we can see that the achievable success probability appears to vary little with signal size. Each cutoff curve is essentially constant. Also, we see the expected increase of success probability with the number of allowed samples. Based on these results it seems safe to conclude that $\sim 10^5$ samples should be sufficient to achieve near perfect hidden frequency identification for any signal with SNR ≥ -17 that is storable in current computer memory.

Sparse Recovery: In our final experiment we investigated the number of signal positions we must read in order to recover all the frequencies of a sparse superposition. Figure 2.7 contains the results. As before, a sparse superposition was created for each individual trial run by selecting m frequencies uniformly at random from $[0, N)$ and then giving each selected frequency a magnitude 1 coefficient with uniformly random

phase. Also, as before, each Figure 2.7 data point is the result of 1000 such trial runs. The probability of successful superposition frequency recovery was calculated by counting the number of trial runs for which AAFFT 0.9's L^∞ error was $< \frac{1}{2}$, divided by 1000. However, for each Figure 2.7 data point, AAFFT 0.9's mean L^∞ error was < 0.02 (i.e., better than $\frac{1}{2}$).

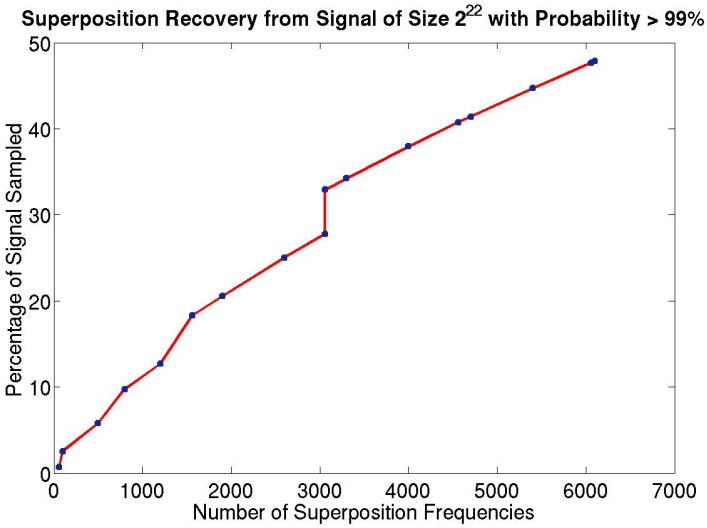

Figure 2.7: Signal Samples for Sparse Superposition Recovery via AAFFT 0.9

We know from Section 2.2.2 that AAFFT 0.9's runtime should (given a fixed signal size N, failure probability δ, and desired accuracy ϵ) scale linearly in the input signal's sparsity level m. Therefore, assuming good parameter settings, the worst case number of samples AAFFT 0.9 requires to recover a signal must also scale linearly in the sparsity level. Looking at Figure 2.7 we can see that the number of samples required to recover a sparse superposition with high probability does indeed appear to scale linearly with superposition sparsity level (the number of non-zero coefficient frequencies m). Figure 2.7 also indicates that, with high probability, AAFFT 0.9 can approximate the DFT of any ≈ 6000-term superposition of length $N = 2^{22}$ using less than half of the superposition's samples.

To date, L^1-minimization based sparse Fourier methods [18] have not been shown to allow exact reconstruction of an m-term/N-length superposition's DFT with high uniform probability unless at least $O(m \log^4 N)$ signal samples are used [99]. Hence, we can see that the number of samples AAFFT 0.9 requires to approximate a superposition's Fourier transform with high probability is at worst a polylog(N) multiple of the number of samples required to calculate (to machine precision) a superposition's Fourier transform with high uniform probability via L^1-minimization. This is a potentially promising result given that L^1-minimization based methods have higher theoretical run time complexity than AAFFT 0.9.

2.4 Conclusion

In this chapter we empirically demonstrated that FADFT-2 [54] retains all the advantages of FADFT-1 [53, 66] while also being more computationally efficient. To accomplish this task a C++ implementation, AAFFT 0.9, of FADFT-2 was compared against a C++ implementation, AAFFT 0.5, of FADFT-1. Both implementations were bench-marked against FFTW 3.1 [50].

In Section 2.3.1 the runtime and approximation error of AAFFT 0.9 and 0.5 were compared for sparse superpositions (i.e. signals with a small number of non-zero frequencies). Section 2.3.1's comparisons demonstrated that AAFFT 0.9 is generally faster than AAFFT 0.5 while retaining similar accuracy. Furthermore, it was demonstrated that both AAFFT 0.9 and AAFFT 0.5 outperform FFTW 3.1 for large sparse superpositions.

In Section 2.3.2 we saw that AAFFT 0.9 and AAFFT 0.5 are essentially indistinguishable in terms of noise tolerance. Furthermore, we saw that AAFFT 0.9's noise tolerance is relatively independent of signal size. Based on Section 2.3.2's results

we may safely conclude that both AAFFT 0.9 and 0.5 are highly tolerant to small amounts of noise (e.g. SNR > -10) as long as AAFFT 0.9/0.5 may use a few tens of thousands of samples from signals of size $\sim 10^6$. Finally, we saw that AAFFT 0.9 is capable of approximating the output of higher time complexity L^1-minimization methods using, at worst, polylog(N) times L^1-minimization's required number of samples. As future work we plan to perform a more careful empirical comparison between AAFFT and L^1-minimization based sparse Fourier methods in order to more accurately determine their runtime/sampling complexity tradeoffs.

Chapter III

A Deterministic Sparse Fourier Algorithm via Non-adaptive Compressed Sensing Methods

We consider the problem of estimating the best B term Fourier representation for a given frequency-sparse signal (i.e., vector) $\mathbf{A}$ of length $N \gg B$. More precisely, we investigate how to deterministically identify B of the largest magnitude frequencies of $\hat{\mathbf{A}}$, and estimate their coefficients, in polynomial$(B, \log N)$ time. Randomized sub-linear time Monte Carlo algorithms exist for solving this problem. However, for failure intolerant applications such as those involving mission-critical hardware designed to process many signals over a long lifetime, deterministic algorithms with no probability of failure are highly desirable. In this chapter we build on the deterministic Compressed Sensing results of Cormode and Muthukrishnan (CM) [92, 32, 33] in order to develop the first known deterministic sub-linear time sparse Fourier Transform algorithm suitable for failure intolerant applications. Furthermore, in the process of developing our new Fourier algorithm, we present a simplified deterministic Compressed Sensing algorithm which improves on CM's algebraic compressibility results while simultaneously maintaining their results concerning exponential compressibility.

3.1 Compressed Sensing and Related Work

Compressed Sensing (CS) methods [18, 104, 92, 32, 33] provide a robust framework for reducing the number of measurements required to summarize sparse signals. For this reason CS methods are useful in areas such as MR imaging [83, 84] and analog-to-digital conversion [77, 72] where measurement costs are high. The general CS setup is as follows: Let $\mathbf{A}$ be an N-length signal/vector with complex valued entries, and Ψ be a full rank $N \times N$ change of basis matrix. Furthermore, suppose that $\Psi \cdot \mathbf{A}$ is sparse (i.e., only $k \ll N$ entries of $\Psi \cdot \mathbf{A}$ are significant/large in magnitude). CS methods deal with generating a $K \times N$ measurement matrix, $\mathcal{M}$, with the smallest number of rows possible (i.e., K minimized) so that the k significant entries of $\Psi \cdot \mathbf{A}$ can be approximately recovered from the K-element result of

$$\mathcal{M} \cdot \Psi \cdot \mathbf{A}. \tag{3.1}$$

Note that CS is inherently algorithmic since a procedure for recovering $\Psi \cdot \mathbf{A}$'s largest k-entries from the result of Equation 3.1 must be specified.

For the remainder of this chapter we will consider the special CS case where Ψ is the $N \times N$ Discrete Fourier Transform matrix. Hence, we have

$$\Psi_{i,j} = \frac{e^{\frac{-2\pi \mathbf{i} \cdot i \cdot j}{N}}}{\sqrt{N}}. \tag{3.2}$$

Our problem of interest is to find, and estimate the coefficients of, the k significant entries (i.e., frequencies) of $\hat{\mathbf{A}}$ given a frequency-sparse (i.e., smooth) signal $\mathbf{A}$. In this case the deterministic Fourier CS measurement matrixes, $\mathcal{M} \cdot \Psi$, produced by [104, 92, 32, 33] require super-linear $O(KN)$-time to multiply by $\mathbf{A}$ in Equation 3.1. Similarly, the energetic frequency recovery procedure of [18, 36] requires super-linear time in N. Hence, none of [18, 104, 36, 92, 32, 33] have both sub-linear measurement and reconstruction time.

Existing randomized sub-linear time Fourier algorithms [53, 66, 54] not only show great promise for decreasing measurement costs, but also for speeding up the numerical solution of computationally challenging multi-scale problems [35]. However, these algorithms are not deterministic, and so can produce incorrect results with some small probability on each input signal. Thus, they aren't appropriate for long-lived failure intolerant applications.

In this chapter we build on the deterministic Compressed Sensing methods of Cormode and Muthukrishnan (CM) [92, 32, 33] in order to construct the first known deterministic sub-linear time sparse Fourier algorithm. In order to produce our new Fourier algorithm we must modify CM's work in two ways: First, we alter CM's measurement construction in order to allow sub-linear time computation of Fourier measurements via aliasing. Thus, our algorithm can deterministically approximate the result of Equation 3.1 in time $K \cdot \text{polylog}(N)$. Second, CM use a k-strongly selective collection of sets [60] to construct their measurements for algebraically compressible signals. We introduce the notion of a K-majority k-strongly selective collection of sets which leads us to a new reconstruction algorithm with better algebraic compressibility results than CM's algorithm. As a result, our deterministic sub-linear time Fourier algorithm has better then previously known algebraic compressibility behavior.

The main contributions of this chapter are:

1. We present a new deterministic compressed sensing algorithm that both (i) improves on CM's algebraically compressible signal results, and (ii) has comparable measurement/run time requirements to CM's algorithm for exponentially decaying signals.

2. We present the first known deterministic sub-linear time sparse Fourier method.

In the process, we explicitly demonstrate the connection between compressed sensing and sub-linear time Fourier transform methods.

3. We introduce K-majority k-strongly selective collections of sets which have potential applications to streaming algorithms along the lines of [91, 51].

The remainder of this chapter is organized as follows: In section 3.2 we introduce relevant definitions and terminology. Then, in section 3.3, we define K-majority k-strongly selective collections of sets and use them to construct our compressed sensing measurements. Section 3.4 contains our new deterministic compressed sensing algorithm along with analysis of its accuracy and run time. Finally, we present our deterministic sub-linear time Fourier algorithm in sections 3.5. Section 3.6 contains a short conclusion.

3.2 Preliminaries

Throughout the remainder of this chapter we will be interested in complex-valued functions $f : [0, 2\pi] \rightarrow \mathbb{C}$ and signals (or arrays) of length N containing f values at various $t \in [0, 2\pi]$. We shall denote such signals by $\mathbf{A}$, where $\mathbf{A}(j) \in \mathbb{C}$ is the signal's j^{th} complex value for all $j \in [0, N-1] \subset \mathbb{N}$. As in Chapter II our algorithm produces output (i.e., a sparse Fourier representation $\hat{\mathbf{R}}^{\text{s}}$) of the form $(\omega_0, C_0), \ldots, (\omega_{B-1}, C_{B-1})$ where each $(\omega_j, C_j) \in [0, N-1] \times \mathbb{C}$. Recall that a B-term/tuple sparse Fourier representation is B-optimal for a signal $\mathbf{A}$ if there exists a valid ordering of $\hat{\mathbf{A}}$'s coefficients by magnitude

$$|\hat{\mathbf{A}}(\omega_0)| \geq \cdots \geq |\hat{\mathbf{A}}(\omega_j)| \geq \cdots \geq |\hat{\mathbf{A}}(\omega_{N-1})| \tag{3.3}$$

so that $\left\{(\omega_l, \hat{\mathbf{A}}(\omega_l)) \;\middle|\; l \in [0, B)\right\} = \hat{\mathbf{R}}^{\text{s}}$. Furthermore, all B-optimal Fourier representations, $\hat{\mathbf{R}}^{\text{s}}_{\text{opt}}$, for any signal $\mathbf{A}$ will always have both the same unique $\|\mathbf{R}_{\text{opt}}\|_2$ and $\|\mathbf{A} - \mathbf{R}_{\text{opt}}\|_2$ values.

We continue with two final definitions: Let ω_b be a b^{th} most energetic frequency as per Equation 3.3. We will say that a signal $\hat{\mathbf{A}}$ is *(algebraically) p-compressible* for some $p > 1$ if $|\hat{\mathbf{A}}(\omega_b)| = O(b^{-p})$ for all $b \in [1, N)$. If $\mathbf{R}^{\mathrm{s}}_{\mathrm{opt}}$ is a B-optimal Fourier representation we can see that

$$\|\mathbf{A} - \mathbf{R}_{\mathrm{opt}}\|_2^2 = \sum_{b=B}^{N-1} |\hat{\mathbf{A}}(\omega_b)|_2^2 = O\left(\int_B^{\infty} b^{-2p} db\right). \tag{3.4}$$

Hence, any p-compressible signal $\mathbf{A}$ (i.e., any signal with a fixed $c \in \mathbb{R}$ so that $|\mathbf{A}(\omega_b)|_2 \leq c \cdot b^{-p}$ for all $b \in [1, N)$) will have $\|\mathbf{A} - \mathbf{R}^{\mathrm{opt}}_B\|_2^2 \leq \tilde{c}_p \cdot B^{1-2p}$ for some $\tilde{c}_p \in \mathbb{R}$. For any p-compressible signal class (i.e., for any choice of p and c) we will refer to the related optimal $O(B^{1-2p})$-size worst case error value (i.e., Equation 3.4 above) as $\|C^{\mathrm{opt}}_B\|_2^2$. Similarly, we define an *exponentially compressible* (or *exponentially decaying*) signal for a fixed $\alpha \in \mathbb{R}^+$ to be one for which $|\hat{\mathbf{A}}(\omega_b)| = O(2^{-\alpha b})$. The optimal worst-case error is then

$$\|C^{\mathrm{opt}}_B\|_2^2 = O\left(\int_B^{\infty} 4^{-\alpha b} db\right) = O(4^{-\alpha B}). \tag{3.5}$$

Fix δ small (e.g., $\delta = 0.1$). Given a compressible input signal, $\mathbf{A}$, our deterministic Fourier algorithm will identify B of the most energetic frequencies from $\hat{\mathbf{A}}$ and approximate their coefficients to produce a Fourier representation $\hat{\mathbf{R}}^{\mathrm{s}}$ with $\|\mathbf{A} - \mathbf{R}\|_2^2 \leq \|\mathbf{A} - \mathbf{R}_{\mathrm{opt}}\|_2^2 + \delta\|C^{\mathrm{opt}}_B\|_2^2$. These are the same types of compressible signal results proven by CM [32, 33].

3.3 Construction of Measurements

We will use the following types of subset collections to form our measurements:

Definition III.1. A collection, $\mathcal{S}$, of s subsets of $[0, N)$ is called **K-majority k-strongly selective** if for any $X \subset [0, N)$ with $|X| \leq k$, and for all $x \in X$, the following are true: (i) x belongs to at least K subsets in $\mathcal{S}$ and, (ii) more than

two-thirds of $S_j \in \mathcal{S}$ containing x are such that $S_j \cap X = \{x\}$ (i.e., every member of X occurs separated from all other members of X in more than two-thirds of the $\mathcal{S}$-subsets it belongs to).

A K-majority k-strongly selective collection of sets is a more structured version of a **k-strongly selective collection of sets** [60, 92]. Every K-majority k-strongly selective collection of sets not only isolates each $x \in X$, but does so a $\frac{2}{3}^{\text{rd}}$'s majority of the time. Thus, our newly defined K-majority k-strongly selective collections will help us count how many times each significant signal entry is isolated. This added structure allows a new reconstruction algorithm (Algorithm 3.1) with better algebraic compressibility properties than previous methods.

Next, we will build $O(\log N)$ K-majority k-strongly selective collections of subsets. Each of these $O(\log N)$ collections will ultimately be used to determine energetic frequencies modulo a small prime less than N. These moduli will then be used along with the Chinese Remainder Theorem to reconstruct each energetic frequency in a manner akin to the introduction's simple example. Our technique is motivated by the method of prime groupings first employed in [91]. To begin, we will denote each of the $O(\log N)$ collections of subsets by $\mathcal{S}_l$ where $0 \leq l \leq O(\log N)$. We construct each of these K-majority k-strongly selective collections as follows:

Define $p_0 = 1$ and let

$$p_1, p_2, \ldots, p_l, \ldots, p_m$$

be the first m primes where m is such that

$$\prod_{l=1}^{m-1} p_l \leq \frac{N}{k} \leq \prod_{l=1}^{m} p_l.$$

Hence, p_l is the l^{th} prime natural number and we have

$$p_0 = 1, p_1 = 2, p_2 = 3, p_3 = 5, \ldots, p_m = O(m \log m).$$

Note that we know $p_m = O(m \log m)$ via the Prime Number Theorem, and so $p_m = O(\log N \log\log N)$. Each p_l will correspond to a different K-majority k-strongly selective collection of subsets of $[0, N) = \{0, \ldots, N-1\}$.

Along these same lines we let q_1 through q_K be the first K (to be specified later) consecutive primes such that

$$\max(p_m, k) \leq q_1 \leq q_2 \leq \cdots \leq q_K.$$

We are now ready to build $\mathcal{S}_0$, our first K-majority k-strongly selective collection of sets. We begin by letting $S_{0,j,h}$ for all $1 \leq j \leq K$ and $0 \leq h \leq q_j - 1$ be

$$S_{0,j,h} = \{n \in [0, N) \mid n \equiv h \bmod q_j\}.$$

Next, we progressively define $S_{0,j}$ to be all integer residues mod q_j, i.e.,

$$S_{0,j} = \{S_{0,j,h} \mid h \in [0, q_j)\},$$

and conclude by setting $\mathcal{S}_0$ equal to all K such q_j-residue groups:

$$\mathcal{S}_0 = \bigcup_{j=1}^{K} S_{0,j}.$$

More generally, for $0 \leq l \leq m$ we define $\mathcal{S}_l$ by

$$\mathcal{S}_l = \bigcup_{j=1}^{K} \left\{ \{n \in [0, N) \mid n \equiv h \bmod p_l q_j\} \;\middle|\; h \in [0, p_l q_j) \right\}.$$

Lemma III.2. *Fix k. If we set $K \geq 3(k-1)\lfloor \log_k N \rfloor + 1$ then $\mathcal{S}_0$ will be a K-majority k-strongly selective collection of sets. Furthermore, if $K = O(k \log_k N)$ then $|\mathcal{S}_0| = O\left(k^2 \log_k^2 N \cdot \max(\log k, \log\log_k N)\right)$.*

Proof. Let $X \subset [0, N)$ be such that $|X| \leq k$. Furthermore, let $x, y \in X$ be such that $x \neq y$. By the Chinese Remainder Theorem we know that x and y may only collide modulo at most $\lfloor \log_k N \rfloor$ of the K q-primes $q_K \geq \cdots \geq q_1 \geq k$. Hence, x may collide

with all the other elements of X (i.e., with $X - \{x\}$) modulo at most $(k-1)\lfloor \log_k N \rfloor$ q-primes. We can now see that x will be isolated from all other elements of X modulo at least $K - (k-1)\lfloor \log_k N \rfloor \geq 2(k-1)\lfloor \log_k N \rfloor + 1 > \frac{2K}{3}$ q-primes. This leads us to the conclusion that $\mathcal{S}_0$ is indeed K-majority k-strongly selective.

Finally, we have that

$$|\mathcal{S}_0| \leq \sum_{j=1}^{K} q_j \leq K \cdot q_K.$$

Furthermore, given that $K > \max(k, m)$, the Prime Number Theorem tells us that $q_K = O(K \log K)$. Thus, we can see that $\mathcal{S}_0$ will indeed contain

$$O\left(k^2 \log_k^2 N \cdot \max(\log k, \log \log_k N)\right)$$

sets. □

Note that at least $O(k \log_k N)$ primes are required in order to create a (K-majority) k-strongly separating collection of subsets using primes in this fashion. Given any $x \in [0, N)$ a $k-1$ element subset X can be created via the Chinese Remainder Theorem and x moduli so that every element of X collides with x in any desired $O(\log_k N)$ q-primes.

We next consider the properties of the other m collections we have defined: $\mathcal{S}_1, \ldots, \mathcal{S}_m$.

Lemma III.3. *Let $S_{l,j,h} = \{n \in [0, N) \mid n \equiv h \bmod p_l q_j\}$, $X \subset [0, N)$ have $\leq k$ elements, and $x \in X$. Furthermore, suppose that $S_{0,j,h} \cap X = \{x\}$. Then, for all $l \in [1, m]$, there exists a unique $b \in [0, p_l)$ so that $S_{l,j,h+b \cdot q_j} \cap X = \{x\}$.*

Proof. Fix any $l \in [1, m]$. $S_{0,j,h} \cap X = \{x\}$ implies that $x = h + a \cdot q_j$ for some unique integer a. Using a's unique representation modulo p_l (i.e., $a = b + c \cdot p_l$) we get that $x = h + b \cdot q_j + c \cdot q_j p_l$. Hence, we can see that $x \in S_{l,j,h+bq_j}$. Furthermore, no other

element of X is in $S_{l,j,h+t\cdot q_j}$ for any $t \in [0, p_l)$ since its inclusion therein would imply that it was also an element of $S_{0,j,h}$. □

Note that Lemma III.3 and Lemma III.2 together imply that each $\mathcal{S}_1, \ldots, \mathcal{S}_m$ is also a K-majority k-strongly separating collection of subsets. Also, we can see that if $x \in S_{l,j,h+b\cdot q_j}$ we can find $x \bmod p_l$ by simply computing $h + bq_j \bmod p_l$. Finally, we form our measurement matrix:

Set $\mathcal{S} = \cup_{l=0}^{m} \mathcal{S}_l$. To form our measurement matrix, $\mathcal{M}$, we simply create one row for each $S_{l,j,h} \in \mathcal{S}$ by computing the N-length characteristic function vector of $S_{l,j,h}$, denoted $\chi_{S_{l,j,h}}$. This leads to $\mathcal{M}$ being a $\tilde{O}(k^2)$ x N measurement matrix. Here we bound the number of rows in $\mathcal{M}$ by noting that: (i) $|\mathcal{S}| < m \cdot K \cdot p_m q_K$, (ii) $m = O(\log N)$, (iii) $p_m = O(\log N \cdot \log\log N)$, (iv) $K = O(k \log N)$, and (v) $q_K = O(K \log K)$.

3.4 Signal Reconstruction from Measurements

Let $\hat{\mathbf{A}}$ be an N-length signal of complex numbers with its N entries numbered 0 through $N-1$. Our goal is to identify B of the largest magnitude entries of $\hat{\mathbf{A}}$ (i.e., the first B entries in a valid ordering of $\hat{\mathbf{A}}$ as in Equation 3.3) and then estimate their signal values. Toward this end, set

$$\epsilon = \frac{|\hat{\mathbf{A}}(\omega_B)|}{\sqrt{2}C} \tag{3.6}$$

where $C > 1$ is a constant to be specified later, and let B' be the smallest integer such that

$$\sum_{b=B'}^{N-1} |\hat{\mathbf{A}}(\omega_b)| < \frac{\epsilon}{2}. \tag{3.7}$$

Note that B' identifies the most energetic *insignificant* frequency (i.e., with energy $<$ a fraction of $|\hat{\mathbf{A}}(\omega_B)|$). We expect to work with sparse/compressible signals so

Algorithm 3.1 SPARSE APPROXIMATE

Input: Signal $\hat{\mathbf{A}}$, integers B, B'
Output: $\hat{\mathbf{R}}^{\text{s}}$, a sparse representation for $\hat{\mathbf{A}}$
Initialize $\hat{\mathbf{R}}^{\text{s}} \leftarrow \emptyset$
Set $K = 3B'\lfloor \log_{B'} N \rfloor$
Form measurement matrix, $\mathcal{M}$, via K-majority B'-strongly selective collections (Section 3.3)
Compute $\mathcal{M} \cdot \hat{\mathbf{A}}$

IDENTIFICATION

for j from 0 to K **do**
  Sort $\langle \chi_{S_{0,j,0}}, \hat{\mathbf{A}} \rangle, \ldots, \langle \chi_{S_{0,j,q_j-1}}, \hat{\mathbf{A}} \rangle$ by magnitude
  for b from 0 to B' **do**
    $k_{j,b} \leftarrow b^{\text{th}}$ largest magnitude $\langle \chi_{S_{0,j,\cdot}}, \hat{\mathbf{A}} \rangle$
    $r_{0,b} \leftarrow k_{j,b}$'s associated residue mod q_j
    for l from 1 to m **do**
      $t_{\min} \leftarrow \min_{t \in [0,p_l)} |k_{j,b} - \langle \chi_{S_{l,j,t \cdot q_j + r_{0,b}}}, \hat{\mathbf{A}} \rangle|$
      $r_{l,b} \leftarrow r_{0,b} + t_{\min} \cdot q_j \bmod p_l$
    end for
    Construct $\omega_{j,b}$ from $r_{0,b}, \ldots, r_{m,b}$ via the CRT
  end for
end for
Sort $\omega_{j,b}$'s maintaining duplicates and set $C(\omega_{j,b}) =$ the number of times $\omega_{j,b}$ was constructed via line 16

ESTIMATION

for j from 0 to K **do**
  for b from 0 to B' **do**
    if $C(\omega_{j,b}) > \frac{2K}{3}$ **then**
      $C(\omega_{j,b}) \leftarrow 0$
      $x = \text{median}\{\text{real}(k_{j',b'}) | \omega_{j',b'} = \omega_{j,b}\}$
      $y = \text{median}\{\text{imag}(k_{j',b'}) | \omega_{j',b'} = \omega_{j,b}\}$
      $\hat{\mathbf{R}}^{\text{s}} \leftarrow \hat{\mathbf{R}}^{\text{s}} \cup \{(\omega_{j,b}, x + \mathbb{i}y)\}$
    end if
  end for
end for
Output B largest magnitude entries in $\hat{\mathbf{R}}^{\text{s}}$

that $B \leq B' \ll N$. Later we will give specific values for C and B' depending on B, the desired approximation error, and $\hat{\mathbf{A}}$'s compressibility characteristics. For now we show that we can identify/approximate B of $\hat{\mathbf{A}}$'s largest magnitude entries each to within ϵ-precision via Algorithm 3.1.

Algorithm 3.1 works by using $\mathcal{S}_0$ measurements to separate $\hat{\mathbf{A}}$'s significantly energetic frequencies $\Omega = \{\omega_0, \ldots, \omega_{B'-1}\} \subset [0, N)$. Every measurement which successfully separates an energetic frequency ω_j from all other members of Ω will both (i) provide a good (i.e., within $\frac{\epsilon}{2} \leq \frac{|\hat{\mathbf{A}}(\omega_B)|}{2\sqrt{2}}$) coefficient estimate for ω_j, and (ii) yield

information about ω_j's identity. Frequency separation occurs because our $\mathcal{S}_0$ measurements can not collide any fixed $\omega_j \in \Omega$ with any other member of Ω modulo more than $(B'-1)\log_{B'} N$ q-primes (see Lemma III.2). Therefore, more than $\frac{2}{3}^{\text{rds}}$ of $\mathcal{S}_0$'s $3B'\log_{B'} N + 1$ q-primes will isolate any fixed $\omega_j \in \Omega$. This means that our reconstruction algorithm will identify all frequencies at least as energetic as ω_B at least $2B'\log_{B'} N+1$ times. We can ignore any frequencies that are not recovered this often. On the other hand, for any frequency that is identified more than $2B'\log_{B'} N$ times, at most $B'\log_{B'} N$ of the measurements which lead to this identification can be significantly contaminated via collisions with valid Ω members. Therefore, we can take a median of the more than $2B'\log_{B'} N$ measurements leading to the recovery of each frequency as that frequency's coefficient estimate. Since more than half of these measurements must be accurate, the median will be accurate.

Theorem III.4. *Let $\hat{\boldsymbol{R}}_{\text{opt}}$ be a B-optimal Fourier representation for our input signal $\boldsymbol{A}$. Then, the B term representation, $\hat{\boldsymbol{R}}^{\text{s}}$, returned from Algorithm 3.1 is such that $\|\boldsymbol{A}-\boldsymbol{R}\|_2^2 \leq \|\boldsymbol{A}-\boldsymbol{R}_{\text{opt}}\|_2^2 + \frac{6B\cdot|\hat{\boldsymbol{A}}(\omega_B)|^2}{C}$. Furthermore, Algorithm 3.1's Identification and Estimation (lines 7 - 30) run time is $O(B'^2\log^4 N)$. The number of measurements used is $O(B'^2\log^6 N)$.*

Theorem III.4 immediately indicates that Algorithm 3.1 gives us a deterministic $O(m^2\log^6 N)$-measurement, $O(m^2\log^4 N)$-reconstruction time method for exactly recovering vectors with m non-zero entries. If $\hat{\mathbf{A}}$ has exactly m non-zero entries then setting $B' = B = m$ and $C = 1$ will be sufficient to guarantee that both $|\hat{\mathbf{A}}(\omega_B)|^2 = 0$ and $\sum_{b=B'}^{N-1}|\hat{\mathbf{A}}(\omega_b)| = 0$ are true. Hence, we may apply Theorem III.4 with $B' = B = m$ and $C = 1$ to obtain a perfect reconstruction via Algorithm 3.1. However, we are mainly interested in the more realistic cases where $\hat{\mathbf{A}}$ is either algebraically or exponentially compressible. The following theorem presents itself.

Algorithm 3.2 FOURIER MEASURE

1: **Input:** f**-samples, integers** m, K
2: **Output:** $< \chi_{S_{l,j,h}}, \hat{f} >$**-measurements**
3: Zero a $O(q_K p_m)$-element array, $\mathbf{A}$
4: **for** j from 1 to K **do**
5: **for** l from 1 to m **do**
6: $\mathbf{A} \leftarrow f(0), f\left(\frac{2\pi}{q_j p_l}\right), \ldots, f\left(\frac{2\pi(q_j p_l - 1)}{q_j p_l}\right)$
7: Calculate $\hat{\mathbf{A}}$ via Chirp z-Transform [97, 14]
8: $< \chi_{S_{l,j,h}}, \hat{f} > \leftarrow \hat{\mathbf{A}}(h)$ for each $h \in [0, q_j p_l)$
9: **end for**
10: **end for**
11: Output $< \chi_{S_{l,j,h}}, \hat{f} >$-measurements

Theorem III.5. *Let $\hat{\boldsymbol{A}}$ be p-compressible. Then, Algorithm 3.1 can return a B term sparse representation, $\hat{\boldsymbol{R}}^{\mathrm{s}}$, with $\|\boldsymbol{A} - \boldsymbol{R}\|_2^2 \leq \|\boldsymbol{A} - \boldsymbol{R}_{\mathrm{opt}}\|_2^2 + \delta \|C_B^{\mathrm{opt}}\|_2^2$ using $O\left(B^{\frac{2p}{p-1}} \delta^{\frac{2}{1-p}} \log^4 N\right)$ total identification/estimation time and $O\left(B^{\frac{2p}{p-1}} \delta^{\frac{2}{1-p}} \log^6 N\right)$ measurements. If $\hat{\boldsymbol{A}}$ decays exponentially, Algorithm 3.1 can return a B term sparse representation, $\hat{\boldsymbol{R}}^{\mathrm{s}}$, with $\|\boldsymbol{A} - \boldsymbol{R}\|_2^2 \leq \|\boldsymbol{A} - \boldsymbol{R}_{\mathrm{opt}}\|_2^2 + \delta \|C_B^{\mathrm{opt}}\|_2^2$ using both $\left(B^2 + \log^2 \delta^{\frac{-1}{\alpha}}\right) \cdot$ polylog(N) measurements and identification/estimation time.*

For p-compressible signals, $p > 2$, CM's algorithm [32, 33] takes $O\left(B^{\frac{6p}{p-2}} \delta^{\frac{6}{2-p}} \log^6 N\right)$-identification/estimation time and $O\left(B^{\frac{4p}{p-2}} \delta^{\frac{4}{2-p}} \log^4 N\right)$-measurements to achieve the same error bound. As a concrete comparison, CM's algorithm requires $O(B^{18} \delta^{-6} \log^6 N)$- identification/estimation time and $O(B^{12} \delta^{-4} \log^4 N)$-measurements for 3-compressible signals. Algorithm 3.1, on the other hand, requires only $O(B^3 \delta^{-1} \log^4 N)$- identification/estimation time and $O(B^3 \delta^{-1} \log^6 N)$-measurements. Hence, we have improved on CM's algebraic compressibility results. All that is left to do in order to develop a deterministic sub-linear time Fourier algorithm is to compute our CS Fourier measurements (Algorithm 3.1 lines 1 - 6) in sub-linear time.

3.5 Fast Fourier Measurement Acquisition

Our goal in this section is to demonstrate how to use Algorithm 3.1 as means to ap-

proximate the Fourier transform of a signal/function $f : [0, 2\pi] \to \mathbb{C}$, where (i) f has an integrable p^{th} derivative, and (ii) $f(0) = f(2\pi), f'(0) = f'(2\pi), \ldots, f^{(p-2)}(0) = f^{(p-2)}(2\pi)$. In this case we know the Fourier coefficients for f to be p-compressible [16, 49]. Hence, for $N = q_1 \cdot p_1 \cdots p_m$ sufficiently large, if we can collect the necessary Algorithm 3.1 (lines 5 and 6) measurements in sub-linear time we will indeed be able to use Algorithm 3.1 as a sub-linear time Fourier algorithm for f.

Choose any Section 3.3 q-prime q_j, $j \in [1, K]$, and any p-prime p_l with $l \in [0, m]$. Furthermore, pick $h \in [0, q_j p_l)$. Throughout the rest of this discussion we will consider f to be accessible to sampling at any desired predetermined positions $t \in [0, 2\pi]$. Given this assumption, we may sample f at $t = 0, \frac{2\pi}{q_j p_l}, \ldots, \frac{2\pi(q_j p_l - 1)}{q_j p_l}$ in order to perform the following DFT computation:

$$\langle \chi_{S_{l,j,h}}, \hat{f} \rangle = \frac{1}{q_j p_l} \sum_{k=0}^{q_j p_l - 1} f\left(\frac{2\pi k}{q_j p_l}\right) e^{\frac{-2\pi \mathbb{i} h k}{q_j p_l}}.$$

Using the Fourier expansion for f yields

$$\langle \chi_{S_{l,j,h}}, \hat{f} \rangle = \frac{1}{q_j p_l} \sum_{k=0}^{q_j p_l - 1} \left(\sum_{\omega=-\infty}^{\infty} \hat{f}(\omega) e^{\frac{2\pi \mathbb{i} \omega k}{q_j p_l}} \right) e^{\frac{-2\pi \mathbb{i} h k}{q_j p_l}}.$$

Finally, exchanging the order of summation above, we see that $\langle \chi_{S_{l,j,h}}, \hat{f} \rangle$ reduces to

$$\frac{1}{q_j p_l} \sum_{\omega=-\infty}^{\infty} \hat{f}(\omega) \sum_{k=0}^{q_j p_l - 1} e^{\frac{2\pi \mathbb{i}(\omega - h)k}{q_j p_l}} = \sum_{\omega \equiv h \bmod q_j p_l} \hat{f}(\omega)$$

via aliasing [16].

Using Sections 3.3 and 3.4 we can see that these measurements are exactly what we need in order to determine B of the most energetic frequencies of $\hat{f}$ modulo $N = q_1 \cdot p_1 \cdots p_m$ (i.e., B of the most energetic frequencies of f's band-limited interpolant's DFT). We are now in the position to modify Algorithm 3.1 in order to find a sparse Fourier representation for $\hat{f}$. To do so we proceed as follows: First, remove lines 5 and 6 and replace them with Algorithm 3.2 for computing all the

necessary $< \chi_{S_{l,j,h}}, \hat{f} >$-measurements. Second, replace each "$< \chi_{S_{l,j,h}}, \hat{\mathbf{A}} >$" by "$< \chi_{S_{l,j,h}}, \hat{f} >$" in Algorithm 3.1's IDENTIFICATION section. It remains to show that these Algorithm 3.1 modifications indeed yield a sub-linear time approximate Fourier transform. The following theorem presents itself:

Theorem III.6. *Let $f : [0, 2\pi] \to \mathbb{C}$ have (i) an integrable p^{th} derivative, and (ii) $f(0) = f(2\pi), f'(0) = f'(2\pi), \ldots, f^{(p-2)}(0) = f^{(p-2)}(2\pi)$ for some $p > 1$. Furthermore, assume that $\hat{f}$'s $B' = O\left(B^{\frac{p}{p-1}}\delta^{\frac{1}{1-p}}\right)$ largest magnitude frequencies all belong to $\left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$. Then, we may use Algorithm 3.1 to return a B term sparse Fourier representation, $\hat{\boldsymbol{R}}^{\text{s}}$, for $\hat{f}$ such that $\|\hat{f} - \hat{\boldsymbol{R}}\|_2^2 \leq \|\hat{f} - \hat{\boldsymbol{R}}_{\text{opt}}\|_2^2 + \delta\|C_B^{\text{opt}}\|_2^2$ using $O\left(B^{\frac{2p}{p-1}}\delta^{\frac{2}{1-p}}\log^7 N\right)$-time and $O\left(B^{\frac{2p}{p-1}}\delta^{\frac{2}{1-p}}\log^6 N\right)$-measurements from f.*

If $f : [0, 2\pi] \to \mathbb{C}$ is smooth (i.e., has infinitely many continuous derivatives on the unit circle where 0 is identified with 2π) it follows from Theorem III.6 that Algorithm 3.1 can be used to find an $O\left(\frac{1}{N}\right)$-accurate sparse B-term Fourier representation for $\hat{f}$ using $\tilde{O}(B^2)$-time/measurements. This result differs from previous sub-linear time Fourier algorithms [53, 54] in that both the algorithm and the measurements/samples it requires are deterministic. Recall that the deterministic nature of the algorithm's required samples is potentially beneficial for failure intolerant hardware. In signal processing applications the sub-Nyquist sampling required to compute Algorithm 3.1's $< \chi_{S_{l,j,h}}, \hat{f} >$-measurements could be accomplished via $\tilde{O}(B)$ parallel low-rate analog-to-digital converters.

3.6 Conclusion

Compressed Sensing (CS) methods provide algorithms for approximating the result of any large matrix multiplication as long as it is known in advance that the result will be sparse/compressible. Hence, CS is potentially valuable for many numerical

applications such as those involving multiscale aspects [35]. In this chapter we used CS methods to develop the first known deterministic sub-linear time sparse Fourier transform algorithm. In the process, we introduced a new deterministic Compressed Sensing algorithm along the lines of Cormode and Muthukrishnan (CM) [32, 33]. Our new deterministic CS algorithm improves on CM's algebraic compressibility results while simultaneously maintaining their results concerning exponential compressibility.

Compressed Sensing is closely related to hashing methods, combinatorial group testing, and many other algorithmic problems [91, 51]. Thus, K-majority k-strongly selective collections of sets and Algorithm 3.1 may help improve deterministic results concerning stream hashing/heavy-hitter identification. Further development of these/other algorithmic applications is left as future work.

Note that in order to validate the use of Algorithm 3.1 (or any other sparse approximate Fourier Transform method [53, 54]) we must assume that f exhibits some multiscale behavior. If $\hat{f}$ contains no unpredictably energetic large (relative to the number of desired Fourier coefficients) frequencies then it is more computationally efficient to simply use standard FFT/USFFT methods [27, 78, 9, 45, 47]. The responsible user, therefore, is not entirely released from the obligation to consider $\hat{f}$'s likely characteristics before proceeding with computations.

Chapter IV

Improved Bounds for a Deterministic Sublinear-Time Sparse Fourier Algorithm

This chapter improves on the best-known runtime and measurement bounds for a recently proposed **D**eterministic sublinear-time **S**parse **F**ourier **T**ransform algorithm (hereafter called **DSFT**). In Chapter III, it is shown that DSFT can exactly reconstruct the Fourier transform (FT) of an N-bandwidth signal f, consisting of $B \ll N$ non-zero frequencies, using $O(B^2 \cdot \text{polylog}(N))$ time and $O(B^2 \cdot \text{polylog}(N))$ f-samples. DSFT works by taking advantage of natural aliasing phenomena to hash a frequency-sparse signal's FT information modulo $O(B \cdot \text{polylog}(N))$ pairwise coprime numbers via $O(B \cdot \text{polylog}(N))$ small Discrete Fourier Transforms. Number theoretic arguments then guarantee the original DFT frequencies/coefficients can be recovered via the Chinese Remainder Theorem. DSFT's usage of primes makes its runtime and signal sample requirements highly dependent on the sizes of sums and products of small primes. Our new bounds utilize analytic number theoretic techniques to generate improved (asymptotic) bounds for DSFT. As a result, we provide better bounds for the sampling complexity/number of low-rate analog-to-digital converters (ADCs) required to deterministically recover frequency-sparse wideband signals via DSFT in signal processing applications [77, 72].

4.1 Introduction

Compressed Sensing (CS) is an exciting new signal acquisition and recovery paradigm in which highly compressible signals can be (approximately) recovered from a few linear measurements, considerably fewer measurements than previously assumed [39, 18]. This chapter will focus on a particular type of compressible signal, namely signals consisting of a small number of significant Fourier modes. Thus, we sample a frequency-sparse signal f on a small deterministic sample set and then reconstruct the signal by returning a list of the predominant frequencies in the spectrum of f. This sensing paradigm is useful in many areas, including MR imaging [83, 84], numerical methods for multiscale problems [35], and ADC design [77, 72].

Existing CS (and related) Fourier reconstruction algorithms [39, 18, 36, 104, 53] are all either (i) super-linear time *in the signal's bandwidth*, making them computationally intensive for wideband signals, or (ii) capable of producing incorrect results with some small probability, making them inappropriate for failure intolerant applications. DSFT [65] is both sublinear-time and deterministic. Hence, it is an improvement over previous CS (and related) Fourier reconstruction algorithms for N-bandwidth signals containing $B \ll N$ significant (e.g., non-zero) frequencies, although it does require a $O(B \cdot \text{polylog}(N))$-factor increase in the number of signal samples over previous randomized approaches [54]. Furthermore, DSFT is consistent with recently proposed ADC designs [77, 72] that suggest a radical new approach to analog-to-digital conversion. These ADC designs, which are based on random sampling, currently require the implementation of random clocks, pseudo-random switches, etc. Due to its deterministic nature, DSFT would allow one to build similar circuits with fixed sample sets in the hardware, thus simplifying the circuit design.

In this chapter, we employ analytic number theory to give the first asymptotic runtime/sample complexity bounds for DSFT on B-support wideband signals (i.e., wideband signals consisting of exactly B non-zero frequencies). Furthermore, we present experiments which both validate our theoretical sample bounds and investigate the number of significant frequencies DSFT may recover from signals of various sizes while maintaining sub-linear sample usage. Finally, we briefly discuss algorithmic improvements which significantly decrease DSFT's sampling and runtime requirements in practice. Our new bounds, besides advancing our knowledge of DSFT's computational properties, also allow us to better bound the number of low-rate parallel ADCs required to deterministically recover wideband frequency-sparse signals along the lines of [77, 72].

4.2 Preliminaries

Throughout the remainder of this chapter, we will be interested in complex-valued signals, $f : [0, 2\pi] \to \mathbb{C}$, which are band-limited and frequency-sparse. Hence, we will assume there exists an $N \in \mathbb{N}$ such that for all our signals f,

$$\Omega_f = \left\{ \omega \in \mathbb{Z} \mid \widehat{f}(\omega) \not\approx 0 \right\} \subsetneq \left(-\left\lceil \frac{N}{2} \right\rceil, \left\lfloor \frac{N}{2} \right\rfloor \right].$$

Also, we assume that for all f, we have $B = |\Omega_f| \ll N$. For any signal f, we will refer to the B non-zero elements of Ω_f as $\omega_1, \omega_2, \ldots, \omega_B$. Furthermore, we will refer to the process of either calculating or measuring f at any $t \in [0, 2\pi]$ as *sampling* from f. Finally, we will say that N is f's *bandwidth*.

Recently, a Deterministic sublinear-time Sparse Fourier Transform algorithm (DSFT) [65] was developed by building upon the number theoretic hashing techniques first proposed in [91, 32]. For a given input signal f, DSFT produces output of the form $(\omega_1, \widehat{f}(\omega_1)), \ldots, (\omega_B, \widehat{f}(\omega_B))$ using $O(B^2 \log^7 N)$ time and $O(B^2 \log^6 N)$ samples [65].

In effect, DSFT works by hashing f's B frequencies modulo $O(B \cdot \text{polylog}(N))$ relatively prime numbers. We next define the numbers which DSFT uses to hash signal frequencies via aliasing. See Chapter III (or [65]) for details.

Let $p_0 = 1$ and p_l be the l^{th} prime. Next, choose m such that

$$\prod_{l=1}^{m} p_l \geq \frac{N}{B} > \prod_{l=1}^{m-1} p_l.$$

Finally, let $q_K > \cdots > q_1 > \max(B, p_m)$ be the smallest $K = 3B\lfloor \log_B N \rfloor + 1$ primes $> \max(B, p_m)$. DSFT requires the computation of the Discrete Fourier Transform (DFT) of

$$f(0), f\left(\frac{2\pi}{p_l q_j}\right), \ldots, f\left(\frac{2\pi(p_l q_j - 1)}{p_l q_j}\right)$$

for every $0 \leq l \leq m$ and $1 \leq j \leq K$. Note that each of these $(m+1) \cdot K$ DFT's will take $O(p_m q_K \log q_K)$ time independent of the factorizations of the array lengths [14, 97].

Fact IV.1. *The number of ADC f-samples required by DSFT [65] is*

$$\sum_{l=0}^{m} \sum_{j=1}^{K} p_l q_j = \left(\sum_{l=0}^{m} p_l\right) \cdot \left(\sum_{j=1}^{K} q_j\right). \tag{4.1}$$

Fact IV.2. *The runtime of DSFT [65] is*

$$\Theta\left(\sum_{l=0}^{m} \sum_{j=1}^{K} p_l q_j \log q_j\right). \tag{4.2}$$

The remainder of this chapter is dedicated to studying (4.1) and (4.2). In the following section, we establish necessary lemmas so that in Section 4.4, we may analyze DSFT's runtime and number of required samples. Then, in Section 4.5, we both empirically validate our Section 4.4 sample bounds for (4.1) and investigate DSFT's

ability to reconstruct superpositions of various sizes using a sub-linear number of samples. Finally, we discuss methods of improving DSFT's sampling performance in Section 4.6 before concluding with a brief discussion in Section 4.7.

4.3 Required Lemmas

In this section, we will assume that $R \geq 3$, $B \geq 2$, and $N/B \geq 3$. Furthermore, p will always stand for a prime number, and $\pi(x)$ will denote the number of primes less than or equal to x. The following lemma recalls three forms of the Prime Number Theorem.

Lemma IV.3. *One has that*

$$\pi(x) = \frac{x}{\ln x} + O\left(\frac{x}{\ln^2 x}\right) \tag{4.3}$$

and that

$$p_l = l \ln l + O(l \ln \ln l). \tag{4.4}$$

Also, there exists a positive constant c such that

$$\sum_{p \leq x} \ln p = x + O\left(\frac{x}{\exp(c\sqrt{\ln x})}\right). \tag{4.5}$$

We now use the Prime Number Theorem to establish asymptotics required in the analysis of DSFT.

Lemma IV.4. *Choose m such that*

$$\prod_{l=1}^{m} p_l \geq \frac{N}{B} > \prod_{l=1}^{m-1} p_l.$$

There exists a positive constant c such that

$$p_m = \ln \frac{N}{B} + O\left(\frac{\ln \frac{N}{B}}{\exp\left(c\sqrt{\ln \ln \frac{N}{B}}\right)}\right).$$

Proof. Note that

$$\prod_{p\leq R} p \geq \frac{N}{B} \text{ if and only if } \sum_{p\leq R} \ln p \geq \ln \frac{N}{B}.$$

It now follows from formula (4.5) that there exists a positive constant c such that

$$p_m = \ln \frac{N}{B} + O\left(\frac{\ln \frac{N}{B}}{\exp\left(c\sqrt{\ln \ln \frac{N}{B}}\right)}\right). \qquad \square$$

Lemma IV.5. *One has*

$$\sum_{p\leq R} p \ln p = \frac{R^2}{2} + O\left(\frac{R^2}{\ln R}\right).$$

Proof. By Riemann-Stieltjes integration (see Chapter 10 of [58]), integration by parts, and formula (4.3), we obtain

$$\begin{aligned}
\sum_{p\leq R} p \ln p &= \int_{2^-}^{R^+} (x \ln x) d\pi(x) \\
&= (R \ln R)\pi(R) - \int_2^R \pi(x) d(x \ln x) \\
&= R^2 + O\left(\frac{R^2}{\ln R}\right) - \int_2^R \left(\frac{x}{\ln x} + O\Big(\frac{x}{\ln^2 x}\Big)\right)(\ln x + 1) dx \\
&= \frac{R^2}{2} + O\left(\frac{R^2}{\ln R}\right). \qquad \square
\end{aligned}$$

Lemma IV.6. *One has*

$$\sum_{p\leq R} p = \frac{R^2}{2\ln R} + O\left(\frac{R^2}{\ln^2 R}\right).$$

Proof. Note that

$$\int_2^{\frac{R}{\ln R}} \frac{x}{\ln x} dx = O\left(\frac{R^2}{\ln^3 R}\right)$$

and that

$$\begin{aligned}\int_{\frac{R}{\ln R}}^{R} \frac{x}{\ln x}dx &= \frac{R^2}{\ln R}\int_{\frac{1}{\ln R}}^{1} \frac{y}{1+\frac{\ln y}{\ln R}}dy \\ &= \frac{R^2}{\ln R}\int_{\frac{1}{\ln R}}^{1} \left(y - \frac{y\ln y}{\ln R + \ln y}\right)dy \\ &= \frac{R^2}{\ln R}\int_{\frac{1}{\ln R}}^{1} \left(y + O\left(\left|\frac{y\ln y}{\ln R}\right|\right)\right)dy \\ &= \frac{R^2}{2\ln R} + O\left(\frac{R^2}{\ln^2 R}\right).\end{aligned}$$

Therefore, we may conclude that

$$\int_{2}^{R} \frac{x}{\ln x}dx = \frac{R^2}{2\ln R} + O\left(\frac{R^2}{\ln^2 R}\right). \tag{4.6}$$

By Riemann-Stieltjes integration, integration by parts, and formulas (4.3) and (4.6), it follows that

$$\begin{aligned}\sum_{p\leq R} p &= \int_{2^-}^{R^+} x d\pi(x) = R\pi(R) - \int_{2}^{R} \pi(x)dx \\ &= \frac{R^2}{\ln R} + O\left(\frac{R^2}{\ln^2 R}\right) \\ &\quad - \int_{2}^{R}\left(\frac{x}{\ln x} + O\left(\frac{x}{\ln^2 x}\right)\right)dx \\ &= \frac{R^2}{2\ln R} + O\left(\frac{R^2}{\ln^2 R}\right).\end{aligned}$$

□

4.4 Runtime and Measurement Bounds

We now are prepared to analyze the performance of DSFT. By formula (4.3), we have

$$\pi(\max(B, p_m)) = O\left(\frac{B}{\ln B} + \frac{\ln \frac{N}{B}}{\ln\ln \frac{N}{B}}\right).$$

When

$$\begin{aligned}S &= \pi(\max(B, p_m)) + K \\ &= 3B\lfloor \log_B N \rfloor + O\left(\frac{B}{\ln B} + \frac{\ln \frac{N}{B}}{\ln\ln \frac{N}{B}}\right),\end{aligned}$$

then by formula (4.4), we have

$$\begin{aligned} q_K = p_S &= S\ln S + O(S\ln\ln S) \\ &= 3B\lfloor \log_B N\rfloor \ln(B\ln N)\left(1+O\left(\frac{\ln\ln(B\ln N)}{\ln(B\ln N)}\right)\right). \end{aligned}$$

Using Lemma IV.6, we see that

$$\begin{aligned} \sum_{j=1}^{K} q_j &= \sum_{\max(B,p_m)<p\le q_K} p \\ &= \frac{9}{2}B^2\lfloor \log_B N\rfloor^2 \ln(B\ln N)\left(1+O\left(\frac{\ln\ln(B\ln N)}{\ln(B\ln N)}\right)\right) \end{aligned}$$

and

$$\begin{aligned} \sum_{l=0}^{m} p_l &= \frac{p_m^2}{2\ln p_m} + O\left(\frac{p_m^2}{\ln^2 p_m}\right) \\ &= \frac{\ln^2 \frac{N}{B}}{2\ln\ln\frac{N}{B}}\left(1+O\left(\frac{1}{\ln\ln\frac{N}{B}}\right)\right). \end{aligned} \tag{4.7}$$

Combining the above two estimates with Fact IV.1, we obtain the following theorem.

Theorem IV.7. *The number of f-samples required by DSFT is*

$$\frac{9B^2\lfloor \log_B N\rfloor^2 \ln(B\ln N)\ln^2\frac{N}{B}}{4\ln\ln\frac{N}{B}}\cdot \left(1+O\left(\frac{1}{\ln\ln\frac{N}{B}}+\frac{\ln\ln(B\ln N)}{\ln(B\ln N)}\right)\right).$$

By Lemma IV.5, we have

$$\begin{aligned} \sum_{j=1}^{K} q_j\ln q_j &= \sum_{\max(B,p_m)<p\le q_K} p\ln p \\ &= \frac{q_K^2}{2} + O\left(\frac{q_K^2}{\ln q_K} + B^2 + p_m^2\right) \\ &= \frac{(3B\lfloor \log_B N\rfloor \ln(B\ln N))^2}{2}\left(1+O\left(\frac{\ln\ln(B\ln N)}{\ln(B\ln N)}\right)\right). \end{aligned} \tag{4.8}$$

By combining Fact IV.2 with formulas (4.7) and (4.8), we obtain the following theorem.

Theorem IV.8. *The runtime of DSFT is*

$$\Theta\left(B^2 \cdot \frac{\ln^2 N \cdot \ln^2 \frac{N}{B} \cdot \ln^2(B \ln N)}{\ln^2 B \cdot \ln \ln \frac{N}{B}}\right).$$

Let $\alpha \in (0, \frac{1}{2})$ be a constant, and suppose that $B = \Theta(N^\alpha)$. In this case, we have improved the previous best sample bound for DSFT from $O(B^2 \log^6 N)$ to $\Theta\left(B^2 \cdot \frac{\log^3 N}{\log \log N}\right)$. Furthermore, we have improved the previous best bound for DSFT's runtime from $O(B^2 \log^7 N)$ to $\Theta\left(B^2 \cdot \frac{\log^4 N}{\log \log N}\right)$. Finally, in signal processing applications along the lines of [77, 72], we can see that the sub-Nyquist sampling required to compute DSFT's $(m+1) \cdot K$ DFTs can be accomplished via $(m+1) \cdot K = O\left(B \cdot \frac{\log N}{\log \log N}\right)$ parallel analog-to-digital converters, each with rate $O(p_m q_K) = O\left(B \log^2 N\right)$ Hz. Hence, DSFT provides a promising deterministic method for quickly reconstructing frequency-sparse wideband signals. We next empirically investigate the signal sizes for which DSFT (as formulated in [65]) can reconstruct sparse superpositions with sub-linear sampling requirements.

4.5 Sampling: Empirical Evaluation

In order to test DSFT's sample asymptotic (Theorem IV.7), we compare the number of f-samples DSFT requires to perfectly recover an N-bandwidth signal f containing exactly $B = 512$ non-zero frequencies against the sample asymptotic's main term in Theorem IV.7. The number of DSFT samples required to recover a 512-frequency superposition, divided by the associated asymptotic value in Theorem IV.7, is plotted in Figure 4.1 for various bandwidth values N. Figure 4.1 demonstrates that our asymptotic is within a constant multiple of 2 of the true number of samples required by DSFT for all tested bandwidth values. Thus, despite the fact that our asymptotic converges to DSFT's number of utilized samples at an exceedingly

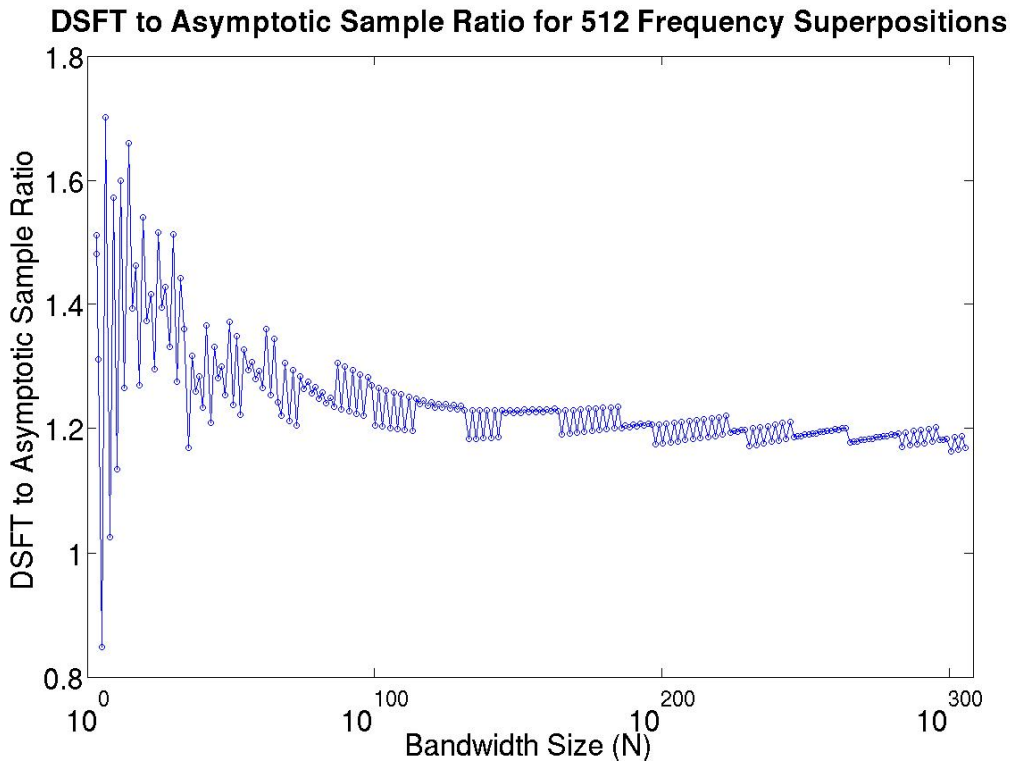

Figure 4.1: Empirical Test of Theorem IV.7.

slow pace, it appears as if the asymptotic generally gives us a reliable estimate of DSFT's sampling requirements. Experiments performed for smaller B reinforce this observation.

Given that a standard FFT can determine the Fourier transform of an N-bandwidth signal f by taking N samples from f, it is important for us to determine when DSFT enables us to utilize less than N samples to recover $\widehat{f}$. Figure 4.2 addresses this issue by plotting, for each bandwidth value N, the maximum number of non-zero frequencies f may contain while still allowing DSFT to determine $\widehat{f}$ using less than N f-samples. Figure 4.2 demonstrates that DSFT, as formulated in [65], does not exhibit sub-linear sampling for $B > 1$ until the bandwidth is $\geq 2^{22}$ (about 4 million). In Section 4.6, we will discuss improvements/modifications for DSFT which allow sub-linear sampling for significantly smaller signals.

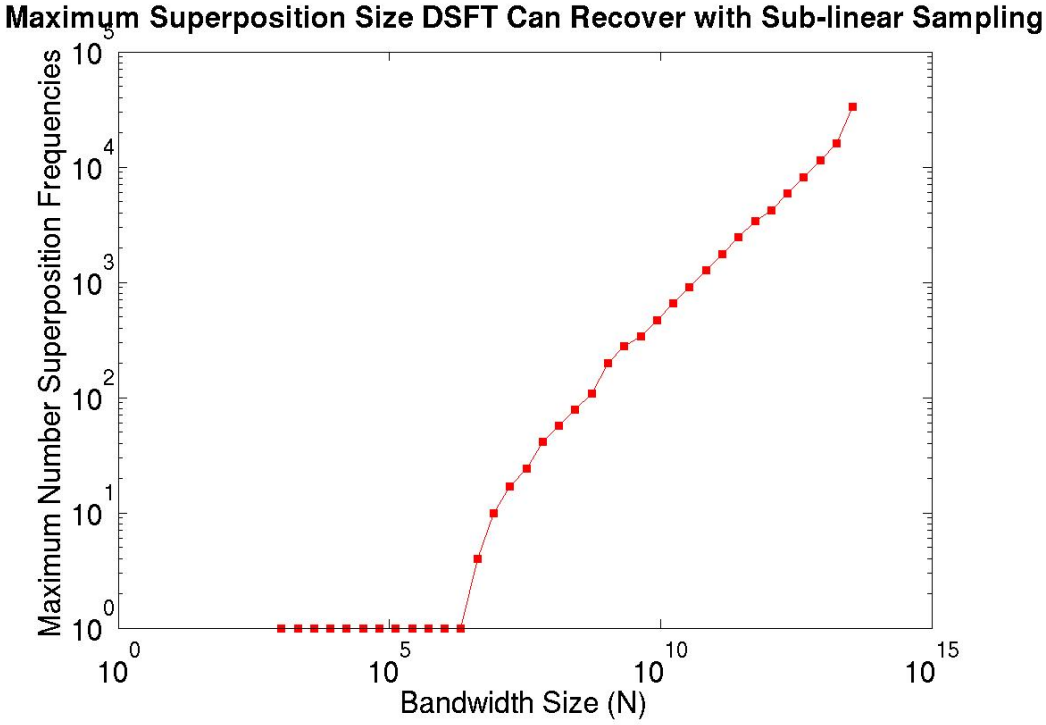

Figure 4.2: Maximum B Value Yielding Less Than N Samples.

Finally, Figure 4.3 plots the number of f-samples DSFT requires to recover $\widehat{f}$ divided by f's bandwidth, N, for three different bandwidth values. It is interesting to note that DSFT's number of required samples occasionally decreases as B increases (for small B). This is due to $K = 3B\lfloor \log_B N \rfloor + 1$ decreasing in size, implying that DSFT requires fewer q_j-primes (see Section 4.2). We will use this phenomenon to our advantage in the following section.

4.6 Sampling: Improving DSFT's Performance

DSFT's sample usage can be mildly decreased (i.e., by a constant factor) through a more careful choice of which primes p_l and q_j from Section 4.2 are used for sampling. In this section, we will discuss three such DSFT improvements. We will also briefly mention two more radical changes to DSFT which dramatically reduce the sample

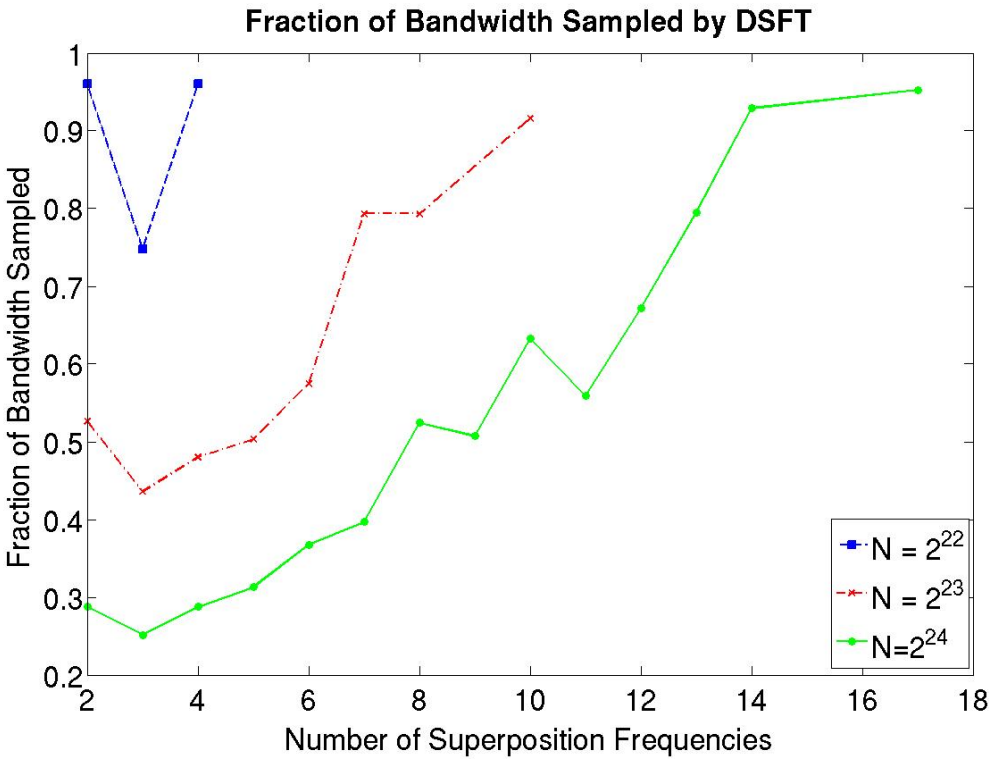

Figure 4.3: Fraction of Bandwidth Sampled for Various B Values.

usage, but at the expense of either losing DSFT's sub-linear reconstruction time or deterministic nature.

First, it should be noted that using powers of p_l-primes can decrease DSFT's sample usage. Instead of performing DFTs of size $p_0 \cdot q_j, \ldots, p_m \cdot q_j$ for each q_j-prime, one can implement DSFT using DFTs of size

$$p_0 \cdot q_j, p_1^{\alpha_{j,1}} \cdot q_j, \ldots, p_v^{\alpha_{j,v}} \cdot q_j, \ldots, p_{m_j}^{\alpha_{j,m_j}} \cdot q_j$$

for each q_j-prime (see Section 4.2). This would require that

$$\prod_{v=1}^{m_j} p_v^{\alpha_{j,v}} \geq \frac{N}{q_j} > \prod_{v=1}^{m_j-1} p_v^{\alpha_{j,v}}$$

for each q_j. We would then replace each

$$\left(\sum_{l=0}^{m} p_l\right) \cdot q_j \quad \text{term with a} \quad \left(\sum_{v=0}^{m_j} p_v^{\alpha_{j,v}}\right) \cdot q_j \quad \text{term}$$

in our bounds for the number of f-samples and the runtime. Finally, the condition that $q_1 > \max(B, p_m)$ would be replaced with the requirement that $q_j > \max(B, p_{m_j})$ for each $1 \leq j \leq K$.

Second, as pointed out at the end of the last section, using a larger B for DSFT sometimes decreases sampling requirements (see Figure 4.3). We may use this phenomenon to our advantage by increasing the size of q_1 and redefining our required number of q_j-primes to be

$$K = 3B\lfloor \log_{q_1} N \rfloor + 1. \tag{4.9}$$

Here $q_1 > \max(B, p_{m_1})$, as in the previous paragraph, but q_1 need not be the smallest prime $> \max(B, p_{m_1})$. By altering q_1 and K's definitions in this fashion, it becomes clear that slightly increasing q_1 can be beneficial.

Third, by a careful analysis of the arguments in [65], DSFT can be modified to require only $K = 2B\lfloor \log_{q_1} N \rfloor + 1$ q_j-primes (i.e., K's factor of 3 can be reduced to a 2) while still maintaining its sub-linear $\tilde{\Theta}(B^2)$-runtime in Theorem IV.8. No modification of the ${p_v}^{\alpha_{j,v}}$-values are required. We will next consider an example that demonstrates the utility of these three modifications with respect to DSFT's sample usage.

Consider an $N = 50,000$ bandwidth signal f containing exactly $B = 5$ non-zero frequencies. DSFT, as formulated in Section 4.2 and [65] would require almost $950,000$ samples to recover $\widehat{f}$. DSFT would use $2, 3, 5, 7, 11$, and 13 as its p_l-primes, leading to a total of

$$41 \cdot \left(\sum_{j=1}^{K} q_j \right)$$

DSFT samples. However, if we require q_1 to be greater than 40 and use $4, 9, 5$, and 7 as the ${p_v}^{\alpha_{j,v}}$-values for all q_j-primes, we can modestly reduce DSFT's sample usage

to

$$25 \cdot \left(\sum_{j=1}^{K} q_j \right).$$

In addition to our savings from using different ${p_v}^{\alpha_{j,v}}$-values, using $q_1 = 41$ instead of $q_1 = 11$ also allows the use of fewer q_j-primes (see (4.9)). This, in combination with the aforementioned algorithmic reduction of K's constant factor from 3 to 2, will allow a well optimized DSFT implementation to recover our $N = 50,000$ bandwidth, 5-frequency superposition f using roughly $43,000$ samples (about 22 times fewer samples than previously required). Thus, optimizing DSFT can dramatically improve its performance.

Further reductions in DSFT's sampling requirements can be obtained if the user is willing to tolerate a super-linear $\tilde{O}(B \cdot N)$ Fourier reconstruction time for N-bandwidth signals with B non-zero frequencies. After performing DFT's of length $q_1, q_2, \ldots, q_K$, as in Section 4.2, one can determine the Fourier coefficient for any of the signal's N, possibly non-zero, frequencies as follows: For each

$$\omega \in \left(-\left\lceil \frac{N}{2} \right\rceil, \left\lfloor \frac{N}{2} \right\rfloor \right],$$

we first determine ω's residue modulo q_j for each q_j. Proofs analogous to those in [65] then guarantee that ω's Fourier coefficient's real/imaginary part will equal the median of the real/imaginary parts of the K DFT entries associated with ω's residues modulo each q_j. Thus, we no longer need any ${p_v}^{\alpha_{j,v}}$-values if we are willing to inspect all N frequencies in this fashion. The number of required samples is reduced to

$$\sum_{j=1}^{K} q_j.$$

Returning to the last paragraph's example, this modified DSFT method only needs $1,791$ samples to correctly recover an $N = 50,000$ bandwidth, $B = 5$ superposition's

Fourier transform. This represents roughly an additional 24-fold decrease in DSFT's sampling needs. However, we are forced to abandon DSFT's sub-linear runtime.

Finally, it is also worthwhile to note that Monte Carlo Fourier results similar to those of [54] may be obtained by limiting our q_j-prime usage in Section 4.2. If we only use a small subset of randomly chosen q_j, we will still be able to isolate all non-zero superposition frequencies with high probability. The frequency's coefficients can then either (i) be approximated by USFFT techniques [54, 45, 47, 78] or (ii) be recovered exactly (assuming non-zero frequency isolation occurs more often then not) using a procedure similar to the one outlined in the previous paragraph. This allows one to use only $\tilde{O}(B)$-samples/runtime for B-frequency superposition reconstruction, which is within a polylogarithmic factor of the current best sample bounds for sparse signal reconstruction via Linear Programming [99, 42]. However, modifying our DSFT techniques in this fashion only allows one to reconstruct sparse superpositions with high probability, and the deterministic nature of our algorithm is lost.

4.7 Conclusion

In this chapter, we utilized analytic number theory to develop the first known asymptotic runtime/sample complexity bounds for DSFT on B-support wideband signals. We then empirically evaluated our new DSFT sampling bounds in Section 4.5. Let $\alpha \in (0, \frac{1}{2})$ be a constant, and suppose that $B = \Theta(N^{\alpha})$. In this case, we have improved the previous best sample bound for DSFT from $O(B^2 \log^6 N)$ to $\Theta\left(B^2 \cdot \frac{\log^3 N}{\log \log N}\right)$. Furthermore, we have improved the previous best bound for DSFT's runtime from $O(B^2 \log^7 N)$ to $\Theta\left(B^2 \cdot \frac{\log^4 N}{\log \log N}\right)$. In Section 4.6, we demonstrated that if one is willing to tolerate a super-linear $O\left(B\lfloor \log_{q_1} N \rfloor \log(B\lfloor \log_{q_1} N \rfloor) \cdot N\right)$ reconstruction runtime (after all DFTs have been taken), then DSFT's sampling

bound can be reduced to $\Theta(B^2 \cdot \log N)$. Furthermore, if one is willing to exchange determinism for success with high probability, we can reduce both DSFT's runtime and sampling needs to $O(B \cdot \text{polylog}(N))$.

In signal processing applications [77, 72], we have shown that the sub-Nyquist sampling required to compute DSFT's $(m+1) \cdot K$ DFTs can be accomplished via $(m+1) \cdot K = O\left(B \cdot \frac{\log N}{\log \log N}\right)$ parallel ADCs, each with rate $O\left(B \log^2 N\right)$ Hz. Hence, DSFT provides a promising deterministic method for quickly reconstructing frequency-sparse wideband signals. Finally, it is worth noting that our DSFT methods are closely related to combinatorial group testing and many other algorithmic problems involving hashing by consecutive primes [91, 51]. Our new DSFT bounds should also provide asymptotic bounds for these related methods.

Chapter V

Combinatorial Sublinear-Time Fourier Algorithms

In this chapter we improve and simplify the deterministic sublinear-time sparse Fourier Transform algorithm methods outlined in Chapter III. A simple relaxation of our improved deterministic Fourier result leads to a new Monte Carlo Fourier algorithm with similar runtime/sampling bounds to the current best randomized Fourier method [54]. Finally, the Fourier algorithm we develop here implies a simpler optimized version of the deterministic compressed sensing method previously developed in Chapter III (and [65]).

5.1 Introduction

In many applications only the top few most energetic terms of a signal's Fourier Transform (FT) are of interest. In such applications the Fast Fourier Transform (FFT), which computes all FT terms, is computationally wasteful. Compressed Sensing (CS) methods [39, 18, 104, 95, 74, 36] provide a robust framework for reducing the number of signal samples required to estimate a signal's FT. For this reason CS methods are useful in areas such as MR imaging [83, 84] and analog-to-digital conversion [77, 72] when sampling costs are high. However, despite small sampling requirements, standard CS Fourier methods utilizing Basis Pursuit (BP) [39, 18, 36] and Orthogonal Matching Pursuit (OMP) [104, 95] have runtime requirements which

are superlinear in the signal's size/bandwidth. Hence, these methods are inappropriate for applications involving large signal sizes/bandwidths where runtime is of primary importance (e.g., numerical methods for multiscale problems [35]).

A second body of work on algorithmic compressed sensing includes methods which have achieved near-optimal reconstruction runtime bounds [53, 54, 32, 33, 92, 56, 61]. However, with the notable exception of [53, 54], these CS algorithms don't permit sublinear sampling in the Fourier case. Hence, despite their efficient reconstruction algorithms, their total Fourier measurement and reconstruction runtime costs are still superlinear in the signal size/bandwidth. In the Fourier case they generally require more operations than a regular FFT for all nontrivial sparsity levels while utilizing approximately the same number of signal samples.

To date only the randomized Fourier methods [53, 54] have been shown to outperform the FFT in terms of runtime on frequency-sparse broadband superpositions while utilizing only a fraction of the FFT's required samples [66]. However, these algorithms are not deterministic and so can produce incorrect results with some small probability on each input signal. Thus, they aren't appropriate for long-lived failure intolerant applications.

In this chapter we construct the first known deterministic sublinear time sparse Fourier algorithm. In order to produce our new Fourier algorithm we introduce a combinatorial object called a k-majority separating collection of sets which can be constructed using number theoretic methods along the lines of [32, 46]. This new combinatorial object yields a simple new CS reconstruction algorithm with better algebraic compressibility results than previous fast deterministic CS methods [32, 33, 92, 61]. Furthermore, the number-theoretic nature of our construction allows the sublinear-time computation of Fourier measurements via aliasing. As a result, we

are able to obtain a deterministic sublinear-time Fourier algorithm which behaves well on both algebraically and exponentially compressible signals. Finally, a simple relaxation of our deterministic Fourier method provides a new randomized Fourier algorithm with runtime/sampling bounds similar to [54].

Work related to our results here include all of the aforementioned CS methods (see [3] for many more). Most closely related of these are the deterministic CS methods [32, 33, 92, 61, 36]. The deterministic constructions in [36] require BP- or OMP-based reconstruction methods [5, 95] with runtimes that are superlinear in the input signal size/bandwidth. On the other hand, our deterministic CS based methods utilize faster recovery procedures along the lines of those first introduced by Cormode and Muthukrishnan (CM) [32, 33, 92]. Indyk's recent work [61] also utilizes similar recovery procedures and achieves theoretically faster reconstruction times on exact superpositions. However, his iterative reconstruction methods don't appear to generalize to algebraically compressible signals. Furthermore, as previously stated, neither Indyk's nor CM's compressed sensing algorithms permit sublinear sampling in the Fourier setting.

Previous randomized Fourier algorithms [53, 54] are similar to our deterministic results in that they obtain both sublinear reconstruction time and sampling (as opposed to other CS Fourier methods). However, they employ random sampling techniques and thus fail to output good approximate answers with non-zero probability. Other related work includes earlier methods for the reconstruction of sparse trigonometric polynomials via random sampling [86, 23]. In turn, these methods can be traced back further to algorithms for learning sparse multivariate polynomials over fields of characteristic zero [71, 87].

Finally, our CS recovery techniques are related group testing methods [44]. In

particular, our k-majority separating collection of sets construction is closely related to the number theoretic group testing construction utilized in [46]. This relationship to group testing, in combination with the Fourier transform's natural aliasing behavior, is essentially what allows our sublinear Fourier methods to be constructed. For more on group testing in statistical signal recovery see [55].

The main contributions of this chapter are:

1. We simplify and optimize Chapter III's deterministic sublinear-time sparse DFT methods. In the process, we improve Chapter III's compressed sensing methods.

2. We present a simple randomized Fourier algorithm with runtime superlinear in the input signal's size/bandwidth which exactly recovers k-frequency superpositions with high probability using a near-optimal number of samples. When modified to run in sublinear time, we obtain a Fourier algorithm with runtime/sampling requirements similar to [54].

3. We introduce k-majority strongly selective collections of sets which have potential applications to streaming algorithms along the lines of [91, 51].

The remainder of this chapter is organized as follows: In section 5.2 we introduce relevant definitions, terminology, and background. Then, in Section 5.3 we define k-majority selective collections of sets and present number theoretic constructions. Section 5.4 contains simple superlinear-time Fourier algorithms along with analysis of their runtime and sampling requirements. In section 5.5 we modify Section 5.4's algorithms to produce sublinear-time Fourier algorithms. Finally, the discrete versions of our algorithms are presented in Section 5.6. Section 5.7 contains a short conclusion.

5.2 Preliminaries

As in previous chapters we will be interested in complex valued functions $f : [0, 2\pi] \mapsto \mathbb{C}$ and signals (or arrays) of length N containing f values at various $x \in [0, 2\pi]$. We denote such signals by $\mathbf{A}$, where $\mathbf{A}(j) \in \mathbb{C}$ is the signal's j^{th} complex value for all $j \in [0, N)$. Hereafter we will refer to the process of either calculating, measuring, or retrieving the f value associated any $\mathbf{A}(j) \in \mathbb{C}$ from machine memory as *sampling* from f and/or $\mathbf{A}$.

Given a signal $\mathbf{A}$ we define its discrete L^q-norm to be

$$\|\mathbf{A}\|_q = \left(\sum_{j=0}^{N-1} |\mathbf{A}(j)|^q \right)^{\frac{1}{q}}. \tag{5.1}$$

More specifically, we will refer to $\|\mathbf{A}\|_2^2$ as $\mathbf{A}$'s *energy*. We will say that $\mathbf{A} \in L^q$ if $\|\mathbf{A}\|_q^q$ converges (i.e., we allow $N = \infty$). Finally, j and ω will always denote integers below.

5.2.1 Compressed Sensing and Compressibility

Given a signal $\mathbf{A}$, let Ψ be any $N \times N$ change of basis matrix/transform under which $\mathbf{A}$ is sparse (i.e., only $k \ll N$ entries of $\Psi \cdot \mathbf{A}$ are significant/large in magnitude). Algorithmic compressed sensing (CS) methods [53, 54, 32, 33, 92, 56, 61, 65] deal with generating a $K \times N$ measurement matrix, $\mathcal{M}$, with the smallest number of rows possible (i.e., K minimized) so that the k significant entries of $\Psi \cdot \mathbf{A}$ can be well approximated using the K-element vector result of

$$(\mathcal{M} \cdot \Psi) \cdot \mathbf{A}. \tag{5.2}$$

Recall that CS a procedure for recovering $\Psi \cdot \mathbf{A}$'s largest k-entries from the result of Equation 5.2 must be specified.

As in previous chapters our recovery algorithm produces output of the form $(\omega_1, C_1), \ldots, (\omega_k, C_k)$ where each $(\omega_j, C_j) \in [0, N) \times \mathbb{C}$. We will refer to any such set of $k < N$ tuples

$$\{(\omega_j, C_j) \in [0, N) \times \mathbb{C} \text{ s.t. } j \in [1, k]\}$$

as a **sparse Ψ representation**. Note that we may reconstruct $\mathbf{R}$ in any desired basis using $\mathbf{R}^{\mathrm{s}}_{\Psi}$. Finally, a sparse Ψ representation $\mathbf{R}^{\mathrm{s}}_{\Psi}$ is **k-optimal** for $\mathbf{A}$ if there exists a valid ordering of $\Psi \cdot \mathbf{A}$ by magnitude

$$\left|(\Psi \cdot \mathbf{A})(\omega_1)\right| \geq \left|(\Psi \cdot \mathbf{A})(\omega_2)\right| \geq \cdots \geq \left|(\Psi \cdot \mathbf{A})(\omega_j)\right| \geq \cdots \geq \left|(\Psi \cdot \mathbf{A})(\omega_N)\right| \tag{5.3}$$

so that $\left\{ (\omega_l, (\Psi \cdot \mathbf{A})(\omega_l)) \;\middle|\; l \in [1, k] \right\} = \mathbf{R}^{\mathrm{s}}_{\Psi}$.

We conclude this subsection by recalling compressibility: Let ω_b be a b^{th} largest magnitude entry of $\Psi \cdot \mathbf{A}$ as per Equation 5.3. A signal $\Psi \cdot \mathbf{A}$ is **(algebraically) p-compressible** for some $p > 1$ if $|(\Psi \cdot \mathbf{A})(\omega_b)| = O(b^{-p})$ for all $b \in [1, N]$. For any p-compressible signal class (i.e., for any choice of p) we will refer to the related optimal $O(k^{1-2p})$-size worst case error value as $\|C^{\mathrm{opt}}_k\|^2_2$. Similarly, an **exponentially compressible** (or **exponentially decaying**) signal for a fixed α to be one for which $\left|(\Psi \cdot \mathbf{A})(\omega_b)\right| = O(2^{-\alpha b})$. The optimal worst case error is then

$$\|C^{\mathrm{opt}}_k\|^2_2 = O\left(\int_k^{\infty} 4^{-\alpha b} db\right) = O(4^{-\alpha k}). \tag{5.4}$$

5.2.2 The Fourier Case

We are primarily interested in the special CS case where Ψ is the $N \times N$ Discrete Fourier Transform (DFT) matrix

$$\Psi_{i,j} = \frac{2\pi}{N} \cdot e^{\frac{2\pi \mathbb{i} \cdot i \cdot j}{N}}. \tag{5.5}$$

Thus, in this chapter we define $\widehat{\mathbf{A}}$ as follows:

$$\widehat{\mathbf{A}}(\omega) = \frac{2\pi}{N} \cdot \sum_{j=0}^{N-1} e^{\frac{-2\pi \mathrm{i}\omega j}{N}} \mathbf{A}(j), \ \forall \omega \in \left(-\left\lceil \frac{N}{2} \right\rceil, \left\lfloor \frac{N}{2} \right\rfloor \right]. \tag{5.6}$$

The Inverse Discrete Fourier Transform (IDFT) of $\widehat{\mathbf{A}}$ is defined as:

$$\mathbf{A}(j) = \widehat{\widehat{\mathbf{A}}}^{-1}(j) = \frac{1}{2\pi} \cdot \sum_{\omega=1-\lceil \frac{N}{2} \rceil}^{\lfloor \frac{N}{2} \rfloor} e^{\frac{2\pi i \omega j}{N}} \widehat{\mathbf{A}}(\omega), \ \forall j \in [0, N). \tag{5.7}$$

Parseval's equality tells us that $\|\widehat{\mathbf{A}}\|_2 = \sqrt{\frac{2\pi}{N}} \cdot \|\mathbf{A}\|_2$.

Fix δ small (e.g., $\delta = 0.1$). Given an input signal, $\mathbf{A}$, with a compressible Fourier transform, our deterministic Fourier algorithm will identify k of the most energetic frequencies from $\widehat{\mathbf{A}}$ and approximate their coefficients to produce a sparse Fourier representation $\hat{\mathbf{R}}^{\mathrm{s}}$ with $\|\widehat{\mathbf{A}} - \hat{\mathbf{R}}\|_2^2 \leq \|\widehat{\mathbf{A}} - \hat{\mathbf{R}}_{\mathrm{opt}}\|_2^2 + \delta \|C_k^{\mathrm{opt}}\|_2^2$. It should be noted that the Fourier reconstruction algorithms below all extend naturally to the general compressed sensing case presented in Section 5.2.1 above via work analogous to that presented in [65].

5.3 Combinatorial Constructions

The following combinatorial structures are motivated by k-strongly separating sets [60, 32]. There properties directly motivate our Fourier reconstruction procedures in Sections 5.4 and 5.5.

Definition V.1. A collection, $\mathcal{S}$, of subsets of $\left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$ is called k-**majority selective** if for all $X \subset \left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$ with $|X| \leq k$ and all $n \in \left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$, more than half of the subsets $S \in \mathcal{S}$ containing n are such that $S \cap X = \{n\} \cap X$ (i.e., every $n \in \left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$ occurs separated from all (other) members of X in more than half of the $\mathcal{S}$ elements containing n).

Definition V.2. Fix an unknown $X \subset \left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$ with $|X| \leq k$. A randomly assembled collection of $\left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$ subsets, $\mathcal{S}$, is called (k, σ)**-majority selective** if the following is true with probability at least σ: For all $n \in \left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$, more than half of the subsets $S \in \mathcal{S}$ containing n have $S \cap X = \{n\} \cap X$ (i.e., with probability $\geq \sigma$ every $n \in \left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$ occurs separated from all (other) members of X in more than half of the $\mathcal{S}$ elements containing n).

The existence of such sets is easy to see. For example, the collection of subsets

$$\mathcal{S} = \left\{ \{n\} \middle| n \in \left(-\left\lceil \frac{N}{2} \right\rceil, \left\lfloor \frac{N}{2} \right\rfloor \right] \right\}$$

consisting of all the singleton subsets of $\left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$ is k-majority selective for all $k \leq N$. Generally, however, we are interested in creating k-majority selective collections which contain as few subsets as possible (i.e., much fewer than N subsets). We next give a construction for a k-majority selective collection of subsets for any $k, N \in \mathbb{N}$ with $k \leq N$. Our construction is motivated by the prime groupings techniques first employed in [91]. We begin as follows:

Define $p_0 = 1$ and let p_l be the l^{th} prime natural number. Thus, we have

$$p_0 = 1, p_1 = 2, p_2 = 3, p_3 = 5, p_4 = 7, \ldots$$

Choose $q, K \in \mathbb{N}$ (to be specified later). We are now ready to build a collection of subsets, $\mathcal{S}$. We begin by letting $S_{j,h}$ for all $0 \leq j \leq K$ and $0 \leq h \leq p_j - 1$ be

$$S_{j,h} = \left\{ n \in \left(-\left\lceil \frac{N}{2} \right\rceil, \left\lfloor \frac{N}{2} \right\rfloor \right] \;\middle|\; n \equiv h \bmod p_{q+j} \right\}. \tag{5.8}$$

Next, we progressively define S_j to be all integer residues mod p_{q+j}, i.e.,

$$S_j = \{S_{j,h} \mid h \in [0, p_{q+j})\}, \tag{5.9}$$

and conclude by setting $\mathcal{S}$ equal to all K such p_{q+j} residue groups:

$$\mathcal{S} = \bigcup_{j=0}^{K} S_j. \tag{5.10}$$

We now prove that $\mathcal{S}$ is indeed k-majority selective if K is chosen appropriately.

Lemma V.3. *Fix k. If we set $K \geq 2k\lfloor \log_{p_q} N \rfloor$ then $\mathcal{S}$ as constructed above will be a k-majority selective collection of sets.*

Proof: Let $X \subset \left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$ be such that $|X| \leq k$. Furthermore, choose $n \in \left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$ and let $x \in X$ be such that $x \neq n$. By the Chinese Remainder Theorem we know that x and n may only collide modulo at most $\lfloor \log_{p_q} N \rfloor$ of the $K+1$ primes $p_{q+K} \geq \cdots \geq p_q$. Hence, n may collide with all the (other) elements of X (i.e., with $X - \{n\}$) modulo at most $k\lfloor \log_{p_q} N \rfloor$ $\mathcal{S}_j$-primes. We can now see that n will be isolated from all the (other) elements of X modulo at least $K + 1 - k\lfloor \log_{p_q} N \rfloor \geq k\lfloor \log_{p_q} N \rfloor + 1 > \frac{K+1}{2}$ $\mathcal{S}_j$-primes. Furthermore, n will appear in at most $K+1$ of $\mathcal{S}$'s subsets. This leads us to the conclusion that $\mathcal{S}$ is indeed k-majority selective. $\square$

Note that at least $\Omega(k)$ coprime integers are required in order to create a k-majority separating collection of subsets in this fashion. Given any $n \in \left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$ a k element subset X can be created via the Chinese Remainder Theorem and n moduli so that every element of X collides with n in any desired $\Omega(1)$ $\mathcal{S}_j$-coprime numbers $\leq \frac{N}{2}$. Thus, it is not possible to significantly decrease the number of relatively prime values required to construct k-majority separating collections using these arguments.

The number of coprime integers required to construct each k-majority separating collection is directly related to the $\Omega(k^2)$ signal samples required by our subsequent Fourier algorithms. Given that we depend on the number theoretic nature of our constructions in order to take advantage of aliasing phenomena, it is unclear how to reduce the sampling complexity for our deterministic Fourier methods below. However, this does not stop us from appealing to randomized number theoretic constructions in order to decrease the number of required coprime values (and, therefore,

samples). We next present a construction for (k, σ)-majority selective collections which motives our subsequent Monte Carlo Fourier algorithms.

Lemma V.4. *Fix k and an unknown $X \subset \left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$ with $|X| \leq k$. We may form a (k, σ)-majority selective collection of subsets, $\mathcal{S}$, as follows: Set $K \geq 3k\lfloor \log_{p_q} N \rfloor$ and create $J \subset [q, q+K]$ by choosing $O\left(\log\left(\frac{N}{1-\sigma}\right)\right)$ elements from $[q, q+K]$ uniformly at random. Set $\mathcal{S} = \cup_{j \in J} S_j$ (see Equation 5.9).*

Proof: Choose any $n \in \left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$. A prime chosen uniformly at random from $\{p_q, \ldots, p_{q+K}\}$ will separate n from all (other) elements of X with probability at least $\frac{2}{3}$ (see proof of Lemma V.3). Using the Chernoff bound we can see that choosing $O\left(\log\left(\frac{N}{1-\sigma}\right)\right)$ primes for J is sufficient to guarantee that the probability of n being congruent to any element of X modulo more than half of J's primes is less than $\frac{1-\sigma}{N}$. The union bound can now be employed to show that J's primes separate every element of $\left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]$ from the (other) elements of X with probability at least σ. $\square$

We conclude this section by bounding the number of subsets contained in our k-majority and (k, σ)-majority selective collections. These subset bounds will ultimately provide us with sampling and runtime bounds for our Fourier algorithms. The following lemma is easily proved using results from Chapter IV (and [69]).

Lemma V.5. *Choose q so that p_q is the smallest prime $\geq k$. If $\mathcal{S}$ is a k-majority selective collection of subsets created as per Lemma V.3, then $|\mathcal{S}|$ is $\Theta\left(k^2 \cdot \log_k^2 N \cdot \log(k \log N)\right)$. If $\mathcal{S}$ is a $\left(k, 1 - \frac{1}{N^{O(1)}}\right)$-majority selective collection of subsets created as per Lemma V.4, then $|\mathcal{S}|$ is $O\left(k \cdot \log_k N \cdot \log(k \log N) \cdot \log N\right)$.*

Let $\alpha \in (0, 1)$ be a constant, and suppose that $k = \Theta(N^\alpha)$. In this case, we have a construction for k-majority selective collections, $\mathcal{S}$, with $|\mathcal{S}| = \Theta\left(k^2 \cdot \log N\right)$.

Furthermore, we have a construction for $\left(k, 1 - \frac{1}{N^{O(1)}}\right)$-majority selective collections, $\mathcal{S}$, with $|\mathcal{S}| = O\left(k \cdot \log^2 N\right)$.

5.4 Superlinear-Time Fourier Algorithms

For the remainder of the chapter we will assume that $f : [0, 2\pi] \mapsto \mathbb{C}$ has the property that $\hat{f} \in L^1$. Our goal is to identify k of the most energetic frequencies in $\hat{f}$ (i.e., the first k entries in a valid ordering of $\hat{f}$ as in Equation 5.3) and then estimate their Fourier coefficients. Intuitively, we want f to be a continuous multiscale function. In this scenario our algorithms will allow us to ignore f's separation of scales and sample at a rate primarily dependent on the number of energetic frequencies present in f's Fourier spectrum.

Let $C \geq 1$ be a constant (to be specified later) and set

$$\epsilon = \frac{|\hat{f}(\omega_k)|}{C}. \tag{5.11}$$

Furthermore, let B be the smallest integer such that

$$\sum_{b=B+1}^{\infty} |\hat{f}(\omega_b)| \leq \frac{\epsilon}{2}. \tag{5.12}$$

Note that B is defined to be the last possible significant frequency (i.e., with energy greater than a fraction of $|\hat{f}(\omega_k)|$). We will assume below that N is chosen large enough so that

$$\Omega = \{\omega_1, \ldots, \omega_B\} \subset \left(-\left\lceil \frac{N}{2} \right\rceil, \left\lfloor \frac{N}{2} \right\rfloor\right]. \tag{5.13}$$

We expect to work with multiscale signals so that $k \leq B \ll N$. Later we will give specific values for C and B depending on k, the desired approximation error, and $\hat{f}$'s compressibility characteristics. For now we show that we can identify/approximate k of $\hat{f}$'s largest magnitude entries each to within ϵ-precision via Algorithm 5.1.

Algorithm 5.1 SUPERLINEAR APPROXIMATE

1: **Input: Signal pointer f, integers $k \leq B \leq N$**
2: **Output: $\hat{\mathbf{R}}^{\mathrm{s}}$, a sparse representation for $\hat{f}$**
3: Initialize $\hat{\mathbf{R}}^{\mathrm{s}} \leftarrow \emptyset$
4: Set $K = 2B\lfloor \log_B N \rfloor$, q so that $p_{q-1} < B \leq p_q$
5: **for** j from 0 to K **do**
6: $\quad \mathbf{A}_{p_{q+j}} \leftarrow f(0), f\left(\frac{2\pi}{p_{q+j}}\right), \ldots, f\left(\frac{2\pi(p_{q+j}-1)}{p_{q+j}}\right)$
7: $\quad \widehat{\mathbf{A}_{p_{q+j}}} \leftarrow \mathbf{DFT}[\mathbf{A}_{p_{q+j}}]$
8: **end for**
9: **for** ω from $1 - \left\lceil \frac{N}{2} \right\rceil$ to $\left\lfloor \frac{N}{2} \right\rfloor$ **do**
10: $\quad \Re\{C_\omega\} \leftarrow$ median of multiset $\left\{ \Re\left\{ \widehat{\mathbf{A}_{\mathrm{p_{q+j}}}}(\omega \bmod \mathrm{p_{q+j}}) \right\} \mid 0 \leq \mathrm{j} \leq \mathrm{K} \right\}$
11: $\quad \Im\{C_\omega\} \leftarrow$ median of multiset $\left\{ \Im\left\{ \widehat{\mathbf{A}_{\mathrm{p_{q+j}}}}(\omega \bmod \mathrm{p_{q+j}}) \right\} \mid 0 \leq \mathrm{j} \leq \mathrm{K} \right\}$
12: **end for**
13: $\hat{\mathbf{R}}^{\mathrm{s}} \leftarrow (\omega, C_\omega)$ entries for k largest magnitude C_ω's

Algorithm 5.1 works by using the k-majority separating structure created by the aliased DFTs in line 7 to isolate $\hat{f}$'s significantly energetic frequencies. Every DFT which successfully separates a frequency ω_j from all the (other) members of Ω will provide a good $\left(\text{i.e., within } \frac{\epsilon}{2} \leq \frac{|\hat{\mathbf{A}}(\omega_k)|}{2} \right)$ coefficient estimate for ω_j. Frequency separation occurs because more than $\frac{1}{2}$ of our aliased DFT's will not collide any $n \in \left(-\left\lceil \frac{N}{2} \right\rceil, \left\lfloor \frac{N}{2} \right\rfloor\right]$ with any (other) member of Ω (see Lemma V.3). At most $B \log_B N$ of the DFT calculations for any particular frequency can be significantly contaminated via collisions with Ω members. Therefore, we can take medians of each frequency's associated $2B \log_B N + 1$ DFT residue's real/imaginary parts as a good estimate of that frequency coefficient's real/imaginary parts. Since more than half of these measurements must be accurate, the medians will be accurate. In order to formalize this argument we need the following lemma.

Lemma V.6. *Every C_ω calculated in lines 10 and 11 is such that $|\hat{f}(\omega) - C_\omega| \leq \epsilon$.*

Proof: Suppose that C_ω is calculated by lines 10 and 11. Then, its real/imaginary part is given the median of K estimates of $\hat{f}(\omega)$'s real/imaginary parts. Each of

these estimates is calculated by

$$\widehat{\mathbf{A}_{p_{q+j}}}(h) = \frac{2\pi}{p_{q+j}} \sum_{k=0}^{p_{q+j}-1} f\left(\frac{2\pi k}{p_{q+j}}\right) e^{\frac{-2\pi \mathbb{i} hk}{p_{q+j}}} \tag{5.14}$$

for some $0 \leq j \leq K$, $0 \leq h < p_{q+j}$. Via aliasing each estimate reduces to

$$\begin{aligned}
\widehat{\mathbf{A}_{p_{q+j}}}(h) &= \frac{2\pi}{p_{q+j}} \sum_{k=0}^{p_{q+j}-1} f\left(\frac{2\pi k}{p_{q+j}}\right) e^{\frac{-2\pi \mathbb{i} hk}{p_{q+j}}} \\
&= \frac{2\pi}{p_{q+j}} \sum_{k=0}^{p_{q+j}-1} \left(\frac{1}{2\pi} \sum_{\rho=-\infty}^{\infty} \hat{f}(\rho) e^{\frac{2\pi \mathbb{i} \rho k}{p_{q+j}}}\right) e^{\frac{-2\pi \mathbb{i} hk}{p_{q+j}}} \\
&= \sum_{\rho=-\infty}^{\infty} \hat{f}(\rho) \left(\frac{1}{p_{q+j}} \sum_{k=0}^{p_{q+j}-1} e^{\frac{2\pi \mathbb{i}(\rho-h)k}{p_{q+j}}}\right) = \sum_{\rho \equiv h \bmod p_{q+j}} \hat{f}(\rho) \\
&= \left\langle \chi_{S_{j,h}}, \hat{f} \cdot \chi_{\left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]} \right\rangle + \sum_{\rho \equiv h \bmod p_{q+j}, \rho \notin \left(-\lceil \frac{N}{2} \rceil, \lfloor \frac{N}{2} \rfloor\right]} \hat{f}(\rho).
\end{aligned} \tag{5.15}$$

Thus, by Lemma V.3 and Equations 5.12 and 5.13, more than half of our $\hat{f}(\omega)$ estimates will have

$$\left|\hat{f}(\omega) - \widehat{\mathbf{A}_{p_{q+j}}}(\omega \bmod p_{q+j})\right| \leq \sum_{\rho \notin \Omega} \left|\hat{f}(\rho)\right| \leq \frac{\epsilon}{2}.$$

It follows that taking medians as per lines 10 and 11 will result in the desired ϵ-accurate estimate for $\hat{f}(\omega)$. $\square$

The following Theorem presents itself.

Theorem V.7. *Let $\hat{\boldsymbol{R}}_{\text{opt}}$ be a k-optimal Fourier representation for our input function f's Fourier transform. Then, the k-term representation $\hat{\boldsymbol{R}}^{\text{s}}$ returned from Algorithm 5.1 is such that $\|\hat{f} - \hat{\boldsymbol{R}}\|_2^2 \leq \|\hat{f} - \hat{\boldsymbol{R}}_{\text{opt}}\|_2^2 + \frac{9k \cdot |\hat{f}(\omega_k)|^2}{C}$. Furthermore, Algorithm 5.1's runtime is $O\left(N \cdot B \cdot \frac{\log^2 N \cdot \log^2(B \log N)}{\log^2 B}\right)$. The number of f samples used is $\Theta\left(B^2 \cdot \log_B^2 N \cdot \log(B \log N)\right)$.*

Proof: Choose any $b \in (0, k]$. Using Lemma V.6 we can see that only way some $\omega_b \notin \hat{\mathbf{R}}_B^{\text{s}}$ is if there exists some associated $b' \in (k, N]$ so that $\omega_{b'} \in \hat{\mathbf{R}}^{\text{s}}$ and

$$|\hat{f}(\omega_k)| + \epsilon \geq |\hat{f}(\omega_{b'})| + \epsilon \geq |C_{\omega_{b'}}| \geq |C_{\omega_b}| \geq |\hat{f}(\omega_b)| - \epsilon \geq |\hat{f}(\omega_k)| - \epsilon.$$

In this case we'll have $2\epsilon > |\hat{f}(\omega_b)| - |\hat{f}(\omega_{b'})| \geq 0$ so that

$$|\hat{f}(\omega_{b'})|^2 + 4\epsilon\left(\epsilon + |\hat{f}(\omega_k)|\right) \geq |\hat{f}(\omega_{b'})|^2 + 4\epsilon\left(\epsilon + |\hat{f}(\omega_{b'})|\right) \geq |\hat{f}(\omega_b)|^2. \tag{5.16}$$

Now using Lemma V.6 we can see that

$$\|\hat{f} - \hat{\mathbf{R}}\|^2 = \sum_{(\omega,\cdot)\notin\hat{\mathbf{R}}^s} |\hat{f}(\omega)|^2 + \sum_{(\omega,C_\omega)\in\hat{\mathbf{R}}^s} |\hat{f}(\omega) - C_\omega|^2 \leq \sum_{(\omega,\cdot)\notin\hat{\mathbf{R}}^s} |\hat{\mathbf{A}}(\omega)|^2 + k\cdot\epsilon^2.$$

Furthermore, we have

$$k\cdot\epsilon^2 + \sum_{(\omega,\cdot)\notin\hat{\mathbf{R}}^s} |\hat{f}(\omega)|^2 = k\cdot\epsilon^2 + \sum_{b\in(0,k],\ \omega_b\notin\hat{\mathbf{R}}^s} |\hat{f}(\omega_b)|^2 + \sum_{b'\in(k,N],\ \omega_{b'}\notin\hat{\mathbf{R}}^s} |\hat{f}(\omega_{b'})|^2.$$

Using observation 5.16 above we can see that this last expression is bounded above by

$$\begin{aligned} k\cdot(5\epsilon^2 + 4\epsilon|\hat{f}(\omega_k)|) + \sum_{b'\in(k,N],\ \omega_{b'}\in\hat{\mathbf{R}}^s} |\hat{f}(\omega_{b'})|^2 + \sum_{b'\in(k,N],\ \omega_{b'}\notin\hat{\mathbf{R}}^s} |\hat{f}(\omega_{b'})|^2 \\ \leq \|\hat{f} - \hat{\mathbf{R}}_{\text{opt}}\|_2^2 + k\cdot(5\epsilon^2 + 4\epsilon|\hat{f}(\omega_k)|). \end{aligned}$$

Substituting for ϵ (see Equation 5.11) gives us our result. Mainly,

$$k\cdot(5\epsilon^2 + 4\epsilon|\hat{f}(\omega_k)|) = \frac{k|\hat{f}(\omega_k)|^2}{C}\left(\frac{5}{C} + 4\right) \leq \frac{9k|\hat{f}(\omega_B)|^2}{C}.$$

To finish, we provide sampling/runtime bounds. Algorithm 5.1's lines 5 through 8 take $O\left(B^2\cdot\frac{\log^2 N\cdot\log^2(B\log N)}{\log^2 B}\right)$ time using the Chirp z-Transform [14, 97] (see [69] for details). Lines 9 through 13 can be accomplished in $O\left(N\cdot B\log_B N\cdot\log(B\log N)\right)$ time. Algorithm 5.1's sampling complexity follows directly from Lemma V.5. □

It's not difficult to see that the proofs of Lemma V.6 and Theorem V.7 still hold using the (k,σ)-majority selective properties of randomly chosen primes. In particular, if we run Algorithm 5.1 using randomly chosen primes along the lines of Lemma V.4 then Theorem V.7 will still hold whenever the primes behave in a

majority selective fashion. The only change required to Algorithm 5.1 is that we compute only a random subset of the DFTs in lines 5 through 8. We have the following corollary.

Corollary V.8. *Let $\hat{\boldsymbol{R}}_{\text{opt}}$ be a k-optimal Fourier representation for our input function f's Fourier transform. If we run Algorithm 5.1 using $O\left(\log\left(\frac{N}{1-\sigma}\right)\right)$ randomly selected primes along the lines of Lemma V.4, then with probability at least σ we will obtain a k-term representation $\hat{\boldsymbol{R}}^{\text{s}}$ having $\|\hat{f} - \hat{\boldsymbol{R}}\|_2^2 \leq \|\hat{f} - \hat{\boldsymbol{R}}_{\text{opt}}\|_2^2 + \frac{9k\cdot|\hat{f}(\omega_k)|^2}{C}$. The runtime will be $O\left(N \cdot \log_B N \cdot \log\left(\frac{N}{1-\sigma}\right) \cdot \log^2\left(B \log\left(\frac{N}{1-\sigma}\right)\right)\right)$. The number of f samples will be $O\left(B \cdot \log_B N \cdot \log(B \log N) \cdot \log\left(\frac{N}{1-\sigma}\right)\right)$.*

It has been popular in the compressed sensing literature to consider the recovery of k-frequency superpositions (see [74] and references therein). Suppose we have

$$f(x) = \sum_{b=1}^{k} C_b \cdot e^{\mathbb{i}\omega_b x},\ \Omega = \{\omega_1, \ldots, \omega_k\} \subset \left(-\left\lceil \frac{N}{2} \right\rceil, \left\lfloor \frac{N}{2} \right\rfloor\right]. \tag{5.17}$$

for all $x \in [0, 2\pi]$. Setting $B = k$ and $C = 1$ is then sufficient to guarantee that $\sum_{b=B+1}^{\infty} |\hat{f}(\omega_b)| = 0$. Theorem V.7 now tells us that Algorithm 5.1 will perfectly reconstruct f. We quickly obtain the final result of this section.

Corollary V.9. *Suppose f is a k-frequency superposition. Then, Algorithm 5.1 can exactly recover f in $O\left(N \cdot k \cdot \frac{\log^2 N \cdot \log^2(k \log N)}{\log^2 k}\right)$ time. The number of f samples used is $\Theta\left(k^2 \cdot \log_k^2 N \cdot \log(k \log N)\right)$. If we run Algorithm 5.1 using $O\left(\log\left(\frac{N}{1-\sigma}\right)\right)$ randomly selected primes along the lines of Lemma V.4, then we will exactly recover f with probability at least σ. In this case the runtime will be*

$$O\left(N \cdot \log_k N \cdot \log\left(\frac{N}{1-\sigma}\right) \cdot \log^2\left(k \log\left(\frac{N}{1-\sigma}\right)\right)\right).$$

The number of f samples will be

$$O\left(k \cdot \log_k N \cdot \log(k \log N) \cdot \log\left(\frac{N}{1-\sigma}\right)\right).$$

As before, let $\alpha \in (0,1)$ be a constant and suppose that $k = \Theta(N^{\alpha})$. Furthermore, let $\sigma = 1 - \frac{1}{N^{O(1)}}$. Corollary V.9 implies that our deterministic Algorithm 5.1 exactly recovers k-frequency superpositions using $O(k^2 \log N)$ samples. If randomly selected primes are used then Algorithm 5.1 can exactly reconstruct k-frequency superpositions with probability $1 - \frac{1}{N^{O(1)}}$ using $O(k \log^2 N)$ samples. In this case our randomized Algorithm 5.1's sampling complexity is within a logarithmic factor of the best known Fourier sampling bounds concerning high probability exact recovery of superpositions [18, 74]. This is encouraging given Algorithm 5.1's simplicity. Of greater interest for our purposes here, however, is that Algorithm 5.1 can be easily modified to run in sublinear-time.

5.5 Sublinear-Time Fourier Algorithms

In order to reduce Algorithm 5.1's runtime we will once again utilize the combinatorial properties of line 7's aliased DFTs. If we can correctly identify any energetic frequencies that are isolated from the other elements of Ω by any given line 7 DFT, we will be guaranteed to recover all energetic frequencies more than $\frac{K}{2}$ times. Thus, collecting all frequencies recovered from more than half of line 7's DFTs will give us the k most energetic Ω frequencies (along with some possibly 'junk frequencies'). The 'junk' can be discarded, however, by using our existing coefficient estimation method (lines 9 - 13) on the collected potentially energetic frequencies. Only truly energetic frequencies will yield large magnitude coefficient estimates by Lemma V.6. Finally, note that only $O(K \log K)$ potentially energetic frequencies may be recovered more than $\frac{K}{2}$ times via line 7's DFTs. Thus, our formally superlinear-time loop (lines 9 - 12) will be sublinearized.

Algorithm 5.2 SUBLINEAR APPROXIMATE

1: **Input: Signal pointer** f**, integers** $m, k \leq B \leq N$
2: **Output:** $\hat{\mathbf{R}}^{\mathrm{s}}$**, a sparse representation for** $\hat{f}$
3: Initialize $\hat{\mathbf{R}}^{\mathrm{s}} \leftarrow \emptyset$
4: Set $K = 2B\lfloor \log_B N \rfloor$, q so that $p_{q-1} \leq \max(B, p_m) \leq p_q$
5: **for** j from 0 to K **do**
6: **for** l from 0 to m **do**
7: $\mathbf{A}_{p_l \cdot p_{q+j}} \leftarrow f(0), f\left(\frac{2\pi}{p_l \cdot p_{q+j}}\right), \ldots, f\left(\frac{2\pi(p_l \cdot p_{q+j}-1)}{p_l \cdot p_{q+j}}\right)$
8: $\widehat{\mathbf{A}_{p_l \cdot p_{q+j}}} \leftarrow \mathbf{DFT}[\mathbf{A}_{p_l \cdot p_{q+j}}]$
9: **end for**
10: **end for**

ENERGETIC FREQUENCY IDENTIFICATION

11: **for** j from 0 to K **do**
12: $\hat{A}_{\mathrm{sort}} \leftarrow$ Sort $\widehat{\mathbf{A}_{p_0 \cdot p_{q+j}}}$ by magnitude (i.e., b^{th} largest magnitude entry in $\hat{A}_{\mathrm{sort}}(b)$)
13: **for** b from 1 to B **do**
14: $r_{0,b} \leftarrow$ index of $\widehat{\mathbf{A}_{p_0 \cdot p_{q+j}}}$'s b^{th} largest magnitude entry $\left(\text{i.e., } \hat{A}_{\mathrm{sort}}(b)\text{'s associated residue mod } p_{q+j}\right)$
15: **for** l from 1 to m **do**
16: $t_{\min} \leftarrow \min_{t \in [0, p_l)} \left|\hat{A}_{\mathrm{sort}}(b) - \widehat{\mathbf{A}_{p_l \cdot p_{q+j}}}(t \cdot p_{q+j} + r_{0,b})\right|$
17: $r_{l,b} \leftarrow (r_{0,b} + t_{\min} \cdot p_{q+j}) \bmod p_l$
18: **end for**
19: Construct $\omega_{j,b}$ from $r_{0,b}, \ldots, r_{m,b}$ via modular arithmetic
20: **end for**
21: **end for**
22: Sort $\omega_{j,b}$'s maintaining duplicates and set $\mathcal{C}(\omega_{j,b}) =$ the number of times $\omega_{j,b}$ was constructed via line 19

COEFFICIENT ESTIMATION

23: **for** j from 1 to K **do**
24: **for** b from 1 to B **do**
25: **if** $\mathcal{C}(\omega_{j,b}) > \frac{K}{2}$ **then**
26: $\Re\left\{C_{\omega_{j,b}}\right\} \leftarrow$ median of multiset $\left\{\Re\left\{\widehat{\mathbf{A}_{\mathrm{p_m \cdot p_{q+h}}}}(\omega_{\mathrm{j,b}} \bmod \mathrm{p_m \cdot p_{q+h}})\right\} \;\middle|\; 0 \leq \mathrm{h} \leq \mathrm{K}\right\}$
27: $\Im\left\{C_{\omega_{j,b}}\right\} \leftarrow$ median of multiset $\left\{\Im\left\{\widehat{\mathbf{A}_{\mathrm{p_m \cdot p_{q+h}}}}(\omega_{\mathrm{j,b}} \bmod \mathrm{p_m \cdot p_{q+h}})\right\} \;\middle|\; 0 \leq \mathrm{h} \leq \mathrm{K}\right\}$
28: **end if**
29: **end for**
30: **end for**
31: $\hat{\mathbf{R}}^{\mathrm{s}} \leftarrow (\omega_{j,b}, C_{\omega_{j,b}})$ entries for k largest magnitude $C_{\omega_{j,b}}$'s

In order to correctly identify energetic frequencies isolated by any Algorithm 5.1 DFT we will utilize a procedure along the lines of Cormode and Muthukrishnan's CS reconstruction method [92, 32, 33]. However, in order to take advantage of aliasing, we will utilize an identification procedure based on the Chinese Remainder Theorem instead of CM's Hamming code based bit testing. For a simple illustration of how our method works in the single frequency case see Chapter I (or [65, 69]). Algorithm 5.2

is the sublinear-time algorithm obtained by modifying Algorithm 5.1 as outlined above.

Let m be the smallest integer such that

$$\prod_{l=0}^{m} p_l \geq \frac{N}{B}. \tag{5.18}$$

The following lemma establishes the correctness of Algorithm 5.2's energetic frequency identification procedure.

Lemma V.10. *Lines 11 through 22 of Algorithm 5.2 are guaranteed to recover all valid $\omega_1, \ldots, \omega_k$ (i.e., all ω with $|\hat{\boldsymbol{A}}(\omega)|_2 \geq |\hat{\boldsymbol{A}}(\omega_k)|_2$ — there may be $> k$ such entries) more then $\frac{K}{2}$ times. Hence, an entry for all such $\omega_b, 1 \leq b \leq k$, will pass the test in line 25 and be added to $\hat{\boldsymbol{R}}^{\text{s}}$ in line 31.*

Proof: Fix $b \in [1, k]$. By Lemma V.3 we know that there exist more than $\frac{K}{2}$ p_{q+j}-primes that isolate ω_b from all of $\Omega - \{\omega_b\}$. Denote these primes by

$$p_{j_1}, p_{j_2}, \ldots, p_{j_{K'}}, \ \frac{K}{2} < K' \leq K.$$

We next show, for each $k' \in [1, K']$, that we get $\widehat{\mathbf{A}_{p_0 \cdot p_{j_{k'}}}}(\omega_b \bmod p_{j_{k'}})$ as one of the B largest magnitude entries found in line 12. Choose any $k' \in [1, K']$. Using Equations 5.11 and 5.12 we can see that

$$\begin{aligned} \frac{\epsilon}{2} \ &\leq \ |\hat{f}(\omega_k)| - \sum_{b'=B+1}^{\infty} |\hat{f}(\omega_{b'})| \ \leq \ |\hat{f}(\omega_b)| - \left| \sum_{b' \notin \Omega, \ \omega_{b'} \equiv \omega_b} \hat{f}(\omega_{b'}) \right| \\ &\leq \ \left| \widehat{\mathbf{A}_{p_0 \cdot p_{j_{k'}}}}(\omega_b \bmod p_{j_{k'}}) \right|. \end{aligned}$$

We also know that the $(B+1)^{\text{st}}$ largest magnitude entry of $\widehat{\mathbf{A}_{p_0 \cdot p_{j_{k'}}}}$ must be $\leq \frac{\epsilon}{2}$. Hence, we are guaranteed to execute lines 13-20 with an $r_{0,\cdot} = \omega_b \bmod p_{j_{k'}}$.

Next, choose any $l \in [1, m]$ and set

$$\bar{\Omega}' = \left\{ \omega_{b'} \ \middle| \ \omega_{b'} \notin \Omega, \ \omega_{b'} \equiv \omega_b \bmod p_{j_{k'}}, \ \omega_{b'} \neq \omega_b \bmod p_l p_{j_{k'}} \right\}.$$

Line 16 inspects all the necessary residues of $\omega_b \bmod p_l p_{j_{k'}}$ since

$$\omega_b \equiv h \bmod p_{j_{k'}} \longrightarrow \omega_b \equiv h + t \cdot p_{j_{k'}} \bmod p_l p_{j_{k'}}$$

for some $t \in [0, p_l)$. To see that t_{min} will be chosen correctly we note first that

$$\begin{aligned}\left|\widehat{\mathbf{A}_{p_0 \cdot p_{j_{k'}}}}(\omega_b \bmod p_{j_{k'}}) - \widehat{\mathbf{A}_{p_l \cdot p_{j_{k'}}}}(\omega_b \bmod p_l p_{j_{k'}})\right| &\leq \sum_{\omega_{b'} \in \bar{\Omega}'} |\hat{f}(\omega_{b'})| \leq \frac{\epsilon}{2} \\ &\leq |\hat{f}(\omega_k)| - \sum_{b'=B+1}^{\infty} |\hat{f}(\omega_{b'})|.\end{aligned}$$

Furthermore, setting $r_{0,\cdot} = \omega_b \bmod p_{j_{k'}}$ and $\tilde{\Omega}'$ to be

$$\begin{aligned}\big\{\omega_{b'} \;\big|\; &\omega_{b'} \notin \Omega,\ \omega_{b'} \equiv \omega_b \bmod p_{j_{k'}},\ \omega_{b'} \neq (r_{0,\cdot} + t p_{j_{k'}}) \bmod p_{j_{k'}} p_l \\ &\text{with } t \text{ s.t. } (r_{0,\cdot} + t p_{j_{k'}}) \neq \omega_b \bmod p_l p_{j_{k'}}\big\},\end{aligned}$$

we have

$$\begin{aligned}|\hat{f}(\omega_k)| - \sum_{b'=B+1}^{\infty} |\hat{f}(\omega_{b'})| \leq |\hat{f}(\omega_b)| - \left|\sum_{\omega_{b'} \in \tilde{\Omega}'} \hat{f}(\omega_{b'})\right| \leq \\ \left|\widehat{\mathbf{A}_{p_0 \cdot p_{j_{k'}}}}(\omega_b \bmod p_{j_{k'}}) - \widehat{\mathbf{A}_{p_l \cdot p_{j_{k'}}}}\left((r_{0,\cdot} + t q_{j_{k'}}) \neq \omega_b \bmod p_l q_{j_{k'}}\right)\right|.\end{aligned}$$

Hence, lines 16 and 17 will indeed select the correct residue for ω_b modulo p_l. And, line 19 will correctly reconstruct ω_b at least $K' > \frac{K}{2}$ times. □

Using Lemma V.10 along with Lemma V.6 and Theorem V.7 we obtain the following Theorem concerning Algorithm 5.2. The sampling and runtime bounds are computed in [65, 69].

Theorem V.11. *Let $\hat{\boldsymbol{R}}_{\text{opt}}$ be a k-optimal Fourier representation for our input function f's Fourier transform. Then, the k-term representation $\hat{\boldsymbol{R}}^{\text{s}}$ returned from Algorithm 5.2 is such that $\|\hat{f} - \hat{\boldsymbol{R}}\|_2^2 \leq \|\hat{f} - \hat{\boldsymbol{R}}_{\text{opt}}\|_2^2 + \frac{9k \cdot |\hat{f}(\omega_k)|^2}{C}$. Furthermore, Algorithm 5.2's runtime is $O\left(B^2 \cdot \frac{\log^2 N \cdot \log^2(B \log N) \cdot \log^2 \frac{N}{B}}{\log^2 B \cdot \log \log \frac{N}{B}}\right)$. The number of f samples used is $O\left(B^2 \cdot \frac{\log^2 N \cdot \log(B \log N) \cdot \log^2 \frac{N}{B}}{\log^2 B \cdot \log \log \frac{N}{B}}\right)$.*

Also, as above, if we run Algorithm 5.2 using randomly chosen p_{q+j}-primes along the lines of Lemma V.4 then Theorem V.11 will still hold whenever the p_{q+j}-primes behave in a majority selective fashion. We have the following corollary.

Corollary V.12. *Let $\hat{\boldsymbol{R}}_{\text{opt}}$ be a k-optimal Fourier representation for our input function f's Fourier transform. If we run Algorithm 5.2 using $O\left(\log\left(\frac{N}{1-\sigma}\right)\right)$ randomly selected p_{q+j}-primes for each f along the lines of Lemma V.4, then with probability at least σ we will obtain a k-term representation $\hat{\boldsymbol{R}}^{\text{s}}$ having $\|\hat{f} - \hat{\boldsymbol{R}}\|_2^2 \leq \|\hat{f} - \hat{\boldsymbol{R}}_{\text{opt}}\|_2^2 + \frac{9k\cdot|\hat{f}(\omega_k)|^2}{C}$. The runtime will be $O\left(B \cdot \frac{\log N \cdot \log\left(\frac{N}{1-\sigma}\right)\cdot\log^2\left(B\log\left(\frac{N}{1-\sigma}\right)\right)\cdot\log^2\frac{N}{B}}{\log B\cdot\log\log\frac{N}{B}}\right)$. The number of f samples will be $O\left(B \cdot \frac{\log^2\left(\frac{N}{1-\sigma}\right)\cdot\log(B\log N)\cdot\log^2\frac{N}{B}}{\log B\cdot\log\log\frac{N}{B}}\right)$.*

Let $\alpha \in (0,1)$ be a constant and suppose that $k = \Theta(N^{\alpha})$. Furthermore, suppose that $\sigma = 1 - \frac{1}{N^{O(1)}}$. Theorem V.11 tells us that our sublinear-time deterministic Algorithm 5.2 exactly recovers k-frequency superpositions in $O\left(k^2 \cdot \frac{\log^4 N}{\log\log N}\right)$ time using $O\left(k^2 \cdot \frac{\log^3 N}{\log\log N}\right)$ samples. If randomly selected p_{q+j}-primes are used then Algorithm 5.2 can exactly reconstruct k-frequency superpositions with probability $1 - \frac{1}{N^{O(1)}}$ in $O\left(k \cdot \frac{\log^5 N}{\log\log N}\right)$ time using $O\left(k \cdot \frac{\log^4 N}{\log\log N}\right)$ samples. It is worth noting here that the recent randomized sublinear-time Fourier results of [53, 54] do not yield exact reconstructions of sparse Fourier superpositions in this manner. They iteratively produce approximate solutions which converge to the true superposition in the limit.

We are now ready to give sublinear-time results concerning functions with compressible Fourier coefficients. For the remainder of this chapter we will assume that our input function $f : [0, 2\pi] \mapsto \mathbb{C}$ has both (i) an integrable p^{th} derivative, and (ii) $f(0) = f(2\pi), f'(0) = f'(2\pi), \ldots, f^{(p-2)}(0) = f^{(p-2)}(2\pi)$ for some $p > 1$. Standard Fourier coefficient bounds then imply that $\hat{f}$ is a p-compressible ∞-length signal

[49, 16]. Before applying Theorem V.11 we will determine Algorithm 5.2's B and Equation 5.11's C variables based on the desired Fourier representation's size and accuracy. Moving toward that goal, we note that since $\hat{f}$ is algebraically compressible we have

$$\frac{9k \cdot |\hat{f}(\omega_k)|^2}{C} = \frac{1}{C} O\left(k^{-2p+1}\right) = O\left(\frac{1}{C}\right) \|C_k^{\text{opt}}\|_2^2. \tag{5.19}$$

Thus, we should use $C = O\left(\frac{1}{\delta}\right)$ and a B so that

$$\sum_{b=B+1}^{\infty} |\hat{f}(\omega_b)| = O(B^{1-p}) = O(\delta \cdot |\hat{f}(\omega_k)|) = O(\delta \cdot k^{-p}). \tag{5.20}$$

Solving, we get that $B = O\left(\delta^{\frac{1}{1-p}} k^{\frac{p}{p-1}}\right)$. Applying Theorem V.11 gives us Algorithm 5.2's runtime and number of required measurements. We obtain the following Corollary.

Corollary V.13. *Let $f : [0, 2\pi] \mapsto \mathbb{C}$ have (i) an integrable p^{th} derivative, and (ii) $f(0) = f(2\pi), \ldots, f^{(p-2)}(0) = f^{(p-2)}(2\pi)$ for some $p > 1$. Furthermore, assume that $\hat{f}$'s $B = O\left(\delta^{\frac{1}{1-p}} k^{\frac{p}{p-1}}\right)$ largest magnitude frequencies all belong to $\left(-\left\lceil \frac{N}{2} \right\rceil, \left\lfloor \frac{N}{2} \right\rfloor\right]$. Then, we may use Algorithm 5.2 to return a k-term sparse Fourier representation, $\hat{\boldsymbol{R}}^{\text{s}}$, for $\hat{f}$ with $\|\hat{f} - \hat{\boldsymbol{R}}\|_2^2 \leq \|\hat{f} - \hat{\boldsymbol{R}}_{\text{opt}}\|_2^2 + \delta \|C_k^{\text{opt}}\|_2^2$ in $O\left(\delta^{\frac{2}{1-p}} k^{\frac{2p}{p-1}} \cdot \frac{\log^6 N}{\log^2 \frac{k^p}{\delta}}\right)$ time. The number of f samples used is $O\left(\delta^{\frac{2}{1-p}} k^{\frac{2p}{p-1}} \cdot \frac{\log^5 N}{\log^2 \frac{k^p}{\delta}}\right)$. If we run Algorithm 5.2 using $O\left(\log\left(\frac{N}{1-\sigma}\right)\right)$ randomly selected p_{q+j}-primes along the lines of Lemma V.4, then with probability at least σ we will obtain a k-term representation $\hat{\boldsymbol{R}}^{\text{s}}$ having $\|\hat{f} - \hat{\boldsymbol{R}}\|_2^2 \leq \|\hat{f} - \hat{\boldsymbol{R}}_{\text{opt}}\|_2^2 + \delta \|C_k^{\text{opt}}\|_2^2$ in $O\left(\delta^{\frac{1}{1-p}} k^{\frac{p}{p-1}} \cdot \frac{\log^6 N}{\log \frac{k^p}{\delta}}\right)$ time. The number of f samples used is $O\left(\delta^{\frac{1}{1-p}} k^{\frac{p}{p-1}} \cdot \frac{\log^5 N}{\log \frac{k^p}{\delta}}\right)$.*

If $f : [0, 2\pi] \to \mathbb{C}$ is smooth (i.e., has infinitely many continuous derivatives on the unit circle where 0 is identified with 2π) it follows from Corollary V.13 that Algorithm 5.2 can be used to find an δ-accurate, with $\delta = O\left(\frac{1}{N}\right)$, sparse k-term

Fourier representation for $\hat{f}$ in $O(k^2 \log^6 N)$ time using $O(k^2 \log^5 N)$ measurements. If randomly selected p_{q+j}-primes are utilized then Algorithm 5.2 can obtain a $O\left(\frac{1}{N}\right)$-accurate k-term Fourier representation for $\hat{f}$ with high probability in $O(k \log^6 N)$ time using $O(k \log^5 N)$ measurements. Similarly, standard results concerning the exponential decay of Fourier coefficients for functions with analytic extensions can be used to generate exponentially compressible Fourier results.

5.6 Discrete Fourier Results

Suppose we are provided with an array $\mathbf{A}$ containing N equally spaced samples from an unknown smooth function $f : [0, 2\pi] \to \mathbb{C}$ (i.e., $\mathbf{A}$'s band-limited interpolent). Hence,

$$\mathbf{A}(j) = f\left(\frac{2\pi j}{N}\right), \ j \in [0, N). \tag{5.21}$$

We would like to use Algorithm 5.2 to find a sparse Fourier representation for $\hat{\mathbf{A}}$. Not having access to f directly, and restricting ourselves to sublinear time approaches only, we have little recourse but to locally interpolate f around Algorithm 5.2's required samples.

For each required Algorithm 5.2 f-sample at $t = \frac{2\pi h}{p_{q+j} p_l}, h \in [0, p_{q+j} p_l)$, we may approximate $f(t)$ to within $O\left(N^{-2\kappa}\right)$ error by constructing 2 local interpolants (one real, one imaginary) around t using $\mathbf{A}$'s nearest 2κ entries [52]. These errors in f-samples can lead to errors of size $O\left(N^{-2\kappa} \cdot p_m p_{q+K} \log p_{q+K}\right)$ in each of Algorithm 5.2 line 8's DFT entries. However, as long as these errors are small enough (i.e., of size $O(\delta \cdot k^{-p})$ in the p-compressible case) Theorem V.11 and all related Section 5.5 results and will still hold. Hence, using $2\kappa = O\left(\log\left(\delta^{-1} \cdot k^p\right)\right)$ interpolation points per f-sample will be sufficient. We have the following result.

Corollary V.14. *Let $\boldsymbol{A}$ be an N-length complex valued array and suppose that*

$\hat{\boldsymbol{A}}$ is p-compressible. Then, we may use Algorithm 5.2 to return a k-term sparse Fourier representation, $\hat{\boldsymbol{R}}^{\mathrm{s}}$, for $\hat{\boldsymbol{A}}$ with $\|\hat{\boldsymbol{A}} - \hat{\boldsymbol{R}}\|_2^2 \leq \|\hat{\boldsymbol{A}} - \hat{\boldsymbol{R}}_{\mathrm{opt}}\|_2^2 + \delta\|C_k^{\mathrm{opt}}\|_2^2$ in $O\left(\delta^{\frac{2}{1-p}} k^{\frac{2p}{p-1}} \cdot \frac{\log^6 N}{\log \frac{kp}{\delta}}\right)$ time. The number of samples used is $O\left(\delta^{\frac{2}{1-p}} k^{\frac{2p}{p-1}} \cdot \frac{\log^5 N}{\log \frac{kp}{\delta}}\right)$. If we run Algorithm 5.2 using $O\left(\log\left(\frac{N}{1-\sigma}\right)\right)$ randomly selected p_{q+j}-primes along the lines of Lemma V.4, then with probability at least σ we will obtain a k-term representation $\hat{\boldsymbol{R}}^{\mathrm{s}}$ having $\|\hat{\boldsymbol{A}} - \hat{\boldsymbol{R}}\|_2^2 \leq \|\hat{\boldsymbol{A}} - \hat{\boldsymbol{R}}_{\mathrm{opt}}\|_2^2 + \delta\|C_k^{\mathrm{opt}}\|_2^2$ in $O\left(\delta^{\frac{1}{1-p}} k^{\frac{p}{p-1}} \cdot \log^6 N\right)$ time. The number of $\boldsymbol{A}$ samples used is $O\left(\delta^{\frac{1}{1-p}} k^{\frac{p}{p-1}} \cdot \log^5 N\right)$.

Notice that Corollary V.14 doesn't guarantee the exact recovery of k-frequency superpositions in the discrete setting. Generally, the sparse Fourier representations produced by Algorithm 5.2 on discrete data will always contain interpolation errors. However, for $\delta = \Theta\left(N^{-1}\right)$, we can still consider smooth data $\mathbf{A}$ to be $\Theta(\log N)$-compressible and so achieve an accurate $\tilde{O}(k^2)$-time DFT algorithm for large N.

5.7 Conclusion

In this chapter the first known deterministic Fourier algorithm with both sublinear-time sampling and runtime complexity was developed. Hence, we have established the first deterministic algorithm which can exactly reconstruct a k-frequency superposition using time polynomial in the superposition's *information content*. When viewed from this perspective the following open problem presents itself.

Open Problem 1. Construct (or show the impossibility of constructing) a deterministic Fourier algorithm guaranteed to exactly recover k-frequency superpositions in $k \cdot \log^{O(1)} N$ time.

The status of current methods with respect to Problem 1 is as follows: Gilbert, Muthukrishnan, and Strauss' randomized Fourier algorithm [54] achieves a near optimal runtime, but is neither deterministic nor exact. Similarly, our Section 5.5 Monte

Carlo algorithm achieves exact reconstruction and a near optimal runtime, but isn't deterministic. Linear programming [39, 18] and OMP-based [95] methods achieve universal sampling sets of acceptable size [99, 42], but both the verification of the sampling sets universal properties and the associated reconstruction procedures are computationally taxing. Finally, Indyk's fast deterministic CS procedure [61] obtains a promising reconstruction runtime, but doesn't allow fast Fourier measurement acquisition.

In terms of applications, there are two compelling motivations for developing fast sparse Fourier transform methods along the lines of [53, 54] and Algorithm 5.2: runtime and sample usage. In numerical applications such as [35] where runtime is the dominant concern we must assume that our input function f exhibits some multiscale behavior. If $\hat{f}$ contains no unpredictably energetic and large (relative to the number of desired Fourier coefficients) frequencies then it is more computationally efficient to simply use standard FFT/NUFFT methods [27, 78, 9, 45, 47]. In other applications [77, 72, 83, 84] where sampling costs are of greater concern than reconstruction runtime, even mild oversampling for the sake of faster reconstruction may be unacceptable. In such cases the runtime/sampling tradeoff must be carefully weighed.

Chapter VI

A Note on Compressed Sensing and the Complexity of Matrix Multiplication

We consider the conjectured $O(N^{2+\epsilon})$ time complexity of multiplying any two $N \times N$ matrices A and B. Our main result is a deterministic Compressed Sensing (CS) algorithm that both rapidly and accurately computes $A \cdot B$ provided that the resulting matrix product is sparse/compressible. As a consequence of our main result we increase the class of matrices A, for any given $N \times N$ matrix B, which allows the exact computation of $A \cdot B$ to be carried out using the conjectured $O(N^{2+\epsilon})$ operations. Additionally, in the process of developing our matrix multiplication procedure, we present a modified version of Indyk's recently proposed extractor-based CS algorithm [61] which is resilient to noise.

6.1 Introduction

Over the past several years the development and refinement of Compressed Sensing (CS) results have generated a cascade of methods exploiting inherent signal sparsity in applications ranging from numerical methods for partial differential equations [35] to summarizing streamed network data [51, 91, 92, 13]. Perhaps most fundamental of all the numerical applications to be addressed using CS-related methods is the approximation of matrix multiplication via random sampling [43, 11]. The

existence of these randomized algorithms for approximating matrix products leads us to the following two questions which we consider in this chapter:

1. Are there computationally efficient deterministic algorithms for approximating the product of two $N \times N$ matrices?

2. Does the existence of fast CS-related algorithms for approximating matrix multiplication tell us anything new about the complexity of exact matrix multiplication?

The answer to both questions is 'Yes'. In this chapter we present a deterministic algorithm for approximating the product of two $N \times N$ matrices which is guaranteed to produce accurate results as long as the matrix product is sparse/compressible. Furthermore, because we develop a noise-tolerant variant of the fastest current deterministic CS method [61] to use in our algorithm, it is fast enough to prove the conjectured $O(N^{2+\epsilon})$ runtime of $N \times N$ matrix multiplication for any two dense $N \times N$ matrices whose product is sparse in all columns (or rows).

The remainder of this chapter is organized as follows: In Section 6.2 we introduce relevant definitions, terminology, and discuss related work. In Section 6.3 we outline the development of a noise tolerant variant of Indyk's extractor-based CS method and use it to create a deterministic algorithm for approximating the product of any two $N \times N$ matrices. Finally, in Section 6.4, we conclude with a discussion our result's implications with respect to the complexity of matrix multiplication.

6.2 Preliminaries and Related Work

Throughout the remainder of this chapter we will utilize the standard Frobenius matrix norm. Let A be an $N \times N$ complex-valued matrix. A's Frobenius norm,

$\|A\|_{\mathrm{F}}$, is defined as

$$\|A\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}|A_{i,j}|^2}. \tag{6.1}$$

Here $A_{i,j}$ is A's i^{th} row's j^{th} entry. Similarly, A_i will denote A's i^{th} row and A^j will denote A's j^{th} column.

Our main result deals with compressible matrices. We say that a complex-valued vector, $\mathbf{v} \in \mathbb{C}^N$, is (C,γ)-compressible for fixed $C, \gamma \in \mathbb{R}^+$, if there exists an ordering of $\mathbf{v}$'s elements by magnitude,

$$|\mathbf{v}_{j_1}| \ \geq \ \ldots \ \geq \ |\mathbf{v}_{j_m}| \ \geq \ldots \ \geq \ |\mathbf{v}_{j_N}|, \tag{6.2}$$

such that $|\mathbf{v}_{j_l}| \leq C \cdot 2^{-\gamma \cdot l}$ for all $1 \leq l \leq N$. Furthermore, we will say that a vector containing only k nonzero-elements, $\mathbf{u}_k^{\text{opt}}$, is k-optimal with respect to vector $\mathbf{v}$ if

$$\|\mathbf{v} - \mathbf{u}_k^{\text{opt}}\|_2^2 = \sum_{l=k+1}^{N} |\mathbf{v}_{j_l}|^2 = O\left(\frac{C^2}{\gamma} \cdot 4^{-\gamma \cdot k}\right). \tag{6.3}$$

Note that the k-optimal error

$$\|\mathbf{v} - \mathbf{u}_k^{\text{opt}}\|_2^2 \tag{6.4}$$

is unique for each $v \in \mathbb{C}^N$. Finally, we will say that an $N \times N$ complex-valued matrix A is *compressible*, or (C,γ)-compressible, if all of A's column (or row) vectors are (C,γ)-compressible. For a compressible $N \times N$ matrix A, we will let U_k^{opt} denote any $N \times N$ matrix minimizer of

$$\|A - U_k\|_{\mathrm{F}}^2 \tag{6.5}$$

over the class of matrices containing $\leq k$ non-zero entries per column (or row). Without loss of generality we will assume column compressibility from now on.

Work related to our results includes existing randomized approximate matrix multiplication algorithms [43, 11]. Let A and B both be $N \times N$ real-valued matrices. In [43], Drineas et al. provide a Monte Carlo algorithm which approximates $A \cdot B$ by randomly selecting and rescaling c columns of A (and the associated c rows of B), and then computing the product, R, of the resulting $N \times c$ and $c \times N$ matrices. Provided that the columns of A/rows of B are chosen via an appropriate probability distribution they show that

$$\|AB - R\|_{\mathrm{F}} = O\left(\frac{\|A\|_{\mathrm{F}} \|B\|_{\mathrm{F}}}{\sqrt{c}}\right) \tag{6.6}$$

with high probability. However, there is no guarantee that the algorithm will return the *correct answer* (or an arbitrarily good approximation) w.h.p. for any particular class of matrices.

In [11] Belabbas et al. utilize low-rank approximation methods on a kernel related to A and B in order to generate an approximation to $A \cdot B$. They then prove that a given optimal sampling strategy nicely relates their approximate product's expected accuracy to the rank of A and B. As a result, their algorithm can compute an accurate approximation (i.e., the expected error is zero) to $A \cdot B$ w.h.p. with the conjectured $O(N^{2+\epsilon})$ operations provided (i) both A and B have rank $O(N^{\epsilon})$, and (ii) that their algorithm has access to an oracle to sample via the optimal strategy. Unfortunately, actually sampling according to the optimal distribution is computationally intractable. Thus, Belabbas et al.'s work doesn't provide complexity results of the type we seek here. Unlike [43, 11] our approximation algorithm is both (i) deterministic, and (ii) fast enough to provide new near-optimal complexity results for exactly multiplying specific types of dense matrices.

6.2.1 Compressed Sensing

Let $\mathbf{v} \in \mathbb{C}^N$ and Ψ be a complex-valued $N \times N$ matrix. Furthermore, suppose that $\Psi \cdot \mathbf{v}$ is sparse/compressible (e.g., (C, γ)-compressible). Compressed sensing methods provide a $K \times N$ measurement matrix, $\mathcal{M}$, with K minimized such that the k most significant entries of $\Psi \cdot \mathbf{v}$ can be recovered from the K-element result of

$$\mathcal{M} \cdot \Psi \cdot \mathbf{v}. \tag{6.7}$$

Standard algorithms for recovering/approximating $\Psi \cdot \mathbf{v}$'s largest k entries in magnitude from the result of (6.7) include linear programming [39, 18], orthogonal matching pursuit [104], and various faster algorithms [56, 92, 33, 65, 61] for particular types of measurement matrices $\mathcal{M}$. For the purposes of this chapter we will utilize a variant of Theorem VI.1 (proved in [61]).

Theorem VI.1. *Suppose that the vector $\Psi \cdot \boldsymbol{v} \in \mathbb{C}^N$ contains at most k non-zero elements. There exists a $k \cdot 2^{O(\log^2 \log N)} \times N$ measurement matrix, $\mathcal{M}$, which enables the exact reconstruction of $\Psi \cdot \boldsymbol{v}$ from the $k \cdot 2^{O(\log^2 \log N)}$-element result of $\mathcal{M} \cdot \Psi \cdot \boldsymbol{v}$ in $k \cdot 2^{O(\log^2 \log N)}$ time.*

We concentrate on Theorem VI.1 for two reasons. First, the reconstruction method outlined in [61] has a runtime complexity that is both sublinear in N (the vector dimension) and linear in k (the sparsity level). All deterministic variants of [39, 18, 104, 92, 33, 65] utilize reconstruction algorithms which are superlinear in either N, k, or both. Furthermore, unlike fast CS methods with uniform error guarantees (e.g., [56]), Indyk's method is both deterministic and explicit (i.e., there is no probability of failure). Although the uniformly random guarantees in [56] suffice to demonstrate the existence of deterministic matrix multiplication algorithms,

verifying any such algorithm's correctness over all sparse signals is computationally intractable.

6.2.2 Complexity of Matrix Multiplication

Clearly, multiplying two arbitrary $N \times N$ matrices requires $\Omega(N^2)$ operations (e.g., to read the input matrices). Naive multiplication of two $N \times N$ matrices uses $\Theta(N^3)$ operations. It is conjectured that for any $\epsilon > 0$, one can multiply two $N \times N$ matrices with $O(N^{2+\epsilon})$ operations, and this result would follow from various combinatorial and algebraic conjectures [24, 30].

Recent approaches to matrix multiplication include the use of tensor product constructions to produce algorithms to multiply two large matrices. The current best algorithm for multiplying two $N \times N$ matrices [30] combines tensor product constructions with a result from additive combinatorics due to Salem and D. C. Spencer [100] to derive an algorithm requiring $O(N^{2.376})$ operations. For a survey of matrix multiplication complexity and related geometry results see [76]. In this chapter, we utilize the following theorem of Coppersmith (see [29]).

Theorem VI.2. *Let $\beta = .29462...$ and $\epsilon > 0$. One can multiply matrices of size $N \times N$ and $N \times N^\beta$ with complexity $O(N^{2+\epsilon})$.*

Theorem VI.2 provides the current best result in terms of maximizing the number of rows, m, an $m \times N$ matrix may have while still being able to be multiplied by another $N \times N$ matrix with complexity $O(N^{2+\epsilon})$. In the next section we present an algorithm for computing the product of two $N \times N$ matrices using $O(N^{2+\epsilon})$ operations under the assumption that the product is sparse in each column. As a result, we generalize Theorem VI.2 with respect to the types of $N \times N$ matrices A we may multiply by any given $N \times N$ matrix B with the conjectured complexity.

6.3 Approximating Matrix Products

In this section we discuss how the combination of compressed sensing methods with Coppersmith's work (i.e., Theorem VI.2) can be used to (approximately) multiply two $N \times N$ matrices with $O(N^{2+\epsilon})$ operations when the product of the two matrices is known to be sparse/compressible. However, in order to state our simple CS based matrix multiplication method we must utilize a noise tolerant version of Theorem VI.1. By slightly modifying Indyk's recovery algorithm and measurement construction the following result can be obtained.

Theorem VI.3. *Suppose that $\boldsymbol{v} \in \mathbb{C}^N$, Ψ is a complex-valued $N \times N$ matrix, and $\Psi \cdot \boldsymbol{v}$ is (C, γ)-compressible. Then, we may construct a $\left(m + \frac{1}{\gamma}\right) \cdot 2^{O(\log^2 \log N)} \times N$ measurement matrix, $\mathcal{M}$, which allows a $\left(m + \frac{1}{\gamma}\right) \cdot 2^{O(\log^2 \log N)}$-time reconstruction algorithm to use the result of $\mathcal{M} \cdot \Psi \cdot \boldsymbol{v}$ and return a vector $\boldsymbol{u}_m$ such that*

$$\| \Psi \cdot \boldsymbol{v} - \boldsymbol{u}_m \|_2^2 \leq \| \Psi \cdot \boldsymbol{v} - \boldsymbol{u}_m^{\text{opt}} \|_2^2 + \frac{\left| (\Psi \cdot \boldsymbol{v})_{j_{m+1}} \right|^2}{N}.$$

Here, $\left| (\Psi \cdot \boldsymbol{v})_{j_{m+1}} \right|$ is the magnitude of the product's $(m+1)^{\text{st}}$-largest entry/entries.

Theorem VI.3's proof is analogous to Theorem VI.1's proof, modulo minor complications due to the presence of 'noise' (i.e., the exponentially decaying smaller magnitude entries of $\Psi \cdot \mathbf{v}$). Due to the proof's similarity to the work in [61] we will only sketch it here.

Proof Sketch:

If we want to recover the m largest magnitude entries of $\Psi \cdot \mathbf{v}$ we will substitute

$$m + O\left(\frac{\log^2 N + \log\left(\frac{C}{\gamma}\right)}{\gamma}\right) \tag{6.8}$$

for r (i.e., the sparsity level) everywhere in [61]. Furthermore, instead of using [33]'s explicit CS construction we can just as easily use the related construction/theorems

in [65]. Thus, complex values are easily handled and each non-overflowing H row can recover entries with enough accuracy to yield results along the lines of [65]'s Theorems 2 and 3 (exponential decay).

We will consider the vector we want to recover, $\Psi \cdot \mathbf{v}$, to consist of an exact r-sparse vector (containing a few more than the m largest magnitude entries we ultimately want to recover — see Equation 6.8) plus a noise vector containing all the remaining entries (i.e., the exponentially decaying 'noise'). As long as the sum of all the noise terms is small enough, Indyk's algorithm will work as before after a few modifications.

First, we must modify [61]'s REDUCE procedure by replacing the line

"IF $votes[j]$ CONTAINS $> d_A/2$ COPIES OF val THEN $y_j = val$"

with

"IF $|votes[j]| > 2d_A/3$ THEN $\mathbb{Re}(y_j) =$ MEDIAN OF $\mathbb{Re}(votes[j])$ AND $\mathbb{Im}(y_j) =$ MEDIAN OF $\mathbb{Im}(votes[j])$".

This changes the proof of [61]'s Lemma 1 only in that now $d_A/3$ vote changes are needed to make any entry y_j have a value more than the current cumulative noise level from the true value (e.g., more than $O(2^{-\gamma \cdot m} \cdot N^{-2})$ from the correct value in the final iterative call of REDUCE). Thus, if we set $\epsilon < 1/24$ more than half of the r-sparse portion of our input vector will be replaced by controllable noise after each iteration.

Second, we note that the iterative nature of Indyk's RECOVER procedure won't

degrade our final accuracy. In the worst case each iteration of the REDUCE can multiply the additive noise for every recovered entry by $O(N)$, resulting in RECOVER returning an estimate y_{j_l} for each largest magnitude entry $(\Psi \cdot \mathbf{v})_{j_l},\ 1 \leq l \leq m$, with

$$|y_{j_l} - (\Psi \cdot \mathbf{v})_{j_l}| = N^{O(\log N)} \cdot \left(\sum_{n=r+1}^{N} \left| (\Psi \cdot \mathbf{v})_{j_n} \right| \right) = N^{O(\log N)} \cdot \frac{C \cdot 2^{-\gamma \cdot r}}{\gamma}. \tag{6.9}$$

If r is replaced with Equation 6.8 we can maintain the additive error bounds needed by [65]'s recovery algorithm to maintain its required accuracy during all $O(\log N)$ iterative calls of the REDUCE procedure.

Finally, after we collect the output from the RECOVER procedure, we sort the output entries by their magnitude and return the largest m of them as our sparse representation $\mathbf{u}_m$. Because we are able to maintain the required accuracy of RECOVER's output (see preceding paragraph), an argument analogous to the proof of [65]'s Theorem 2 will give us our final result. □

With Theorem VI.3 in hand we are ready to consider matrix multiplication. Let A and B denote two $N \times N$ matrices with complex entries. Furthermore, we suppose that $A \cdot B$ is (C, γ)-compressible. To construct an approximate product matrix U_m with

$$\|A \cdot B - U_m\|_{\mathrm{F}} = O\left(\|A \cdot B - U_m^{\mathrm{opt}}\|_{\mathrm{F}} \right) \tag{6.10}$$

we proceed as follows:

1. Use a $\left(m + \frac{1}{\gamma}\right) \cdot 2^{O(\log^2 \log N)} \times N$ measurement matrix, $\mathcal{M}$, as per Theorem VI.3 to compute

   $$P = (\mathcal{M} \cdot A) \cdot B \tag{6.11}$$

   using Theorem VI.2. Provided that there exists some $\epsilon > 0$ so that both m and $\frac{1}{\gamma}$ are $O(N^{\beta - \epsilon})$ this can be accomplished in $O(N^{2+\epsilon})$ time.

2. Apply Theorem VI.3 to P^j for all $1 \leq j \leq N$ to recover U_m.

The total recovery time will be $O(N^{1+\beta})$. We quickly obtain our main theorem.

Theorem VI.4. *Let $\beta^- < .29462...$, $\epsilon > 0$, and A, B be $N \times N$ matrices. If $A \cdot B$ is (C, γ)-compressible and both m and $\frac{1}{\gamma}$ are $O(N^{\beta^-})$, then one can obtain an $N \times N$ matrix U_m such that*

$$\|(A \cdot B) - U_m\|_{\mathrm{F}}^2 \leq \|(A \cdot B) - U_m^{\mathrm{opt}}\|_{\mathrm{F}}^2 + \sum_{i=1}^{N} \frac{\left| (A \cdot B)^i_{j_{m+1}} \right|^2}{N}$$

in $O(N^{2+\epsilon})$ time.

Note that in the special case where $A \cdot B$ has $\leq m$ non-zero elements per column, we have

$$\sum_{i=1}^{N} \frac{\left| (A \cdot B)^i_{j_{m+1}} \right|^2}{N} = 0. \tag{6.12}$$

We obtain the following corollary.

Corollary VI.5. *Let $\beta^- < .29462...$ and $c, \epsilon \in \mathbb{R}^+$. Furthermore, let A and B denote $N \times N$ matrices. If the product $A \cdot B$ has at most cN^{β^-} non-zero elements in each column, then $A \cdot B$ can be computed using $O(N^{2+\epsilon})$ operations.*

Let A and B denote square $N \times N$ matrices, $\epsilon > 0$, and $c > 0$. If $A \cdot B$ is compressible in each column, we can use Theorem VI.4 to obtain a near-optimal best $cN^{.29462}$ element-per-column approximation to $A \cdot B$ using $O(N^{2+\epsilon})$ operations. More specifically, if each column of the product $A \cdot B$ has at most $cN^{.29462}$ non-zero elements, then we can use Corollary VI.5 to calculate the product $A \cdot B$ exactly using $O(N^{2+\epsilon})$ operations.

6.4 Discussion

In this chapter we discussed how compressed sensing methods can be used to (approximately) multiply two square matrices quickly if the product is known to be

sparse. In the process, we have increased the class of $N \times N$ matrices A, for any given $N \times N$ matrix B, which allow $A \cdot B$ to be calculated exactly using $O(N^{2+\epsilon})$ operations (see Corollary VI.5). Provided that $A \cdot B$ contains at most $O(N^{\beta^-})$ non-zero entries per column, it can be calculated exactly using $O(N^{2+\epsilon})$ operations. In contrast, previous results [28, 29] required that A contain $O(N^{\beta})$ non-empty (i.e., non-sparse) rows to achieve the same bound.

Furthermore, we have also provided results concerning the approximation of the product of two (dense) $N \times N$ matrices in $O(N^{2+\epsilon})$ time. Any two matrices may be approximately multiplied using our method, and the result will be accurate to the extent that the true product is compressible. The required measurement acquisition (i.e., Equation 6.11) can either be accomplished via traditional matrix multiplication or via lower complexity methods (e.g., Theorem VI.2). In the later case it is worth mentioning that any additional advances in rapid matrix multiplication similar to Theorem VI.2 will automatically strengthen our results. This is due to the reconstruction algorithm in Theorem VI.3 having $O(m \cdot N^{\epsilon})$ runtime.

We finish by noting that in practice we may not know when a matrix product is going to be column/row-sparse. Thus, although we have given a deterministic algorithm which is guaranteed to accurately approximate such products, we won't necessarily know *when* our answers are accurate. In such cases existing streaming algorithm techniques [48, 8] allow us to predict the sparsity (i.e., number of non-zero entries) of all the matrix product's columns/rows to within a small constant factor (e.g., 4) with probability $O(1 - \frac{1}{N^{O(1)}})$ in $O(N^2 \cdot \log N)$-time [62]. Thus, in the general case (where the matrix product's sparsity is unknown) a Monte-Carlo variant of Corollary VI.5 holds.

Appendices

Appendix A

Scalable Rule-Based Gene Expression Data Classification

Microarray technology allows biologists to simultaneously measure the expression of thousands of genes in a single experiment. This technology provides a unique tool to examine how a cell's gene expression pattern changes under various conditions. Microarray methods could also play a critical role in personalized medicine as they could be used to determine the unique genetic susceptibility of an individual to disease.

See Table A.1 for a sample microarray dataset shown using the common discretized relational representation. In this table, each sample row consists of (*i*) a list of discretized genes and (*ii*) a class label. A gene is present in a sample row if the sample expresses the gene. The absence of a gene in a row implies that the gene is not expressed in that sample. Thus, the sample/gene expression relationships for relational microarray data are essentially boolean.

Leading associated rule-based methods such as Top-k [25], FARMER [26], CLOSET+ [107], and CHARM [108] which have been applied to microarray datasets aim to correlate gene expression patterns with the classification labels. For these algorithms the discovered correlations take the form of **association rules** [6]. For an example association rule, consider the data shown in Table A.1. Note that only the Cancer samples s_1 and s_2 express both genes g_1 and g_3. Based on this observation we can

Sample	Expressed Genes				Class Label
s_1	g_1	g_2	g_3	g_5	Cancer
s_2	g_1	g_3	g_6		Cancer
s_3	g_2	g_4	g_6		Cancer
s_4	g_2	g_3	g_5		Healthy
s_5	g_3	g_4	g_5	g_6	Healthy

Table A.1: Running Example of Microarray Data

create the following association rule: $g_1, g_3 \Rightarrow$ Cancer. This rule means that if a query sample express both g_1 and g_3 (i.e., if g_1 and g_3's associated genes are both expressed in their associated expression intervals), then the query sample is likely to be of type Cancer. Hence, we can use this rule to *classify* query samples of unknown type as Cancer if they express both g_1 and g_3. Note that there is nothing special about the class label Cancer. After noticing that only Healthy sample s_5 expresses both g_5 and g_6, we can also create the meaningful association rule $g_5, g_6 \Rightarrow$ Healthy.

Current state-of-the-art association rule-based classifiers for gene expression data operate in two phases: (i) Association rule mining from training data followed by (ii) Classification of query data using the mined rules. In the worst case, these methods require an exponential search over the subset space of the training data set's samples and/or genes during at least one of these two phases. Hence, existing association rule-based techniques are prohibitively computationally expensive on large gene expression datasets.

Our main result is the development of a heuristic rule-based gene expression data classifier called Boolean Structure Table Classification (BSTC). BSTC is explicitly related to association rule-based methods, but is guaranteed to be polynomial space/time. Extensive cross validation studies on several real gene expression datasets demonstrate that BSTC retains the classification accuracy of current association rule-based methods while being orders of magnitude faster than the leading

classifier RCBT on large datasets. As a result, BSTC is able to finish table generation and classification on large datasets for which current association rule-based methods become computationally infeasible.

BSTC also enjoys two other advantages over association rule-based classifiers: (i) BSTC is easy to use (requires no parameter tuning), and (ii) BSTC can easily handle datasets with any number of class types. Furthermore, in the process of developing BSTC we introduce a novel class of boolean association rules which have potential applications to other data mining problems.

In this appendix we focus on association rule-based classifiers (hereafter referred to simply as rule-based classifiers) for gene expression data. We focus on rule-based classifiers for two reasons: (i) rule-based classifiers have been demonstrated to be more accurate for gene expression analysis than other methods [25, 26, 38, 79] such as SVM [34] and tree-based C4.5 family algorithms [96], and (ii) as opposed to other classifiers such as SVM, rule-based classifiers can offer concise, concrete, and biologically meaningful rules supporting their non-default classifications. However, rule-based methods are not scalable due to their high association rule mining costs. Although these rule mining costs are "one-time costs" in the sense that rules must only be mined once per training set, larger training data sets are being generated at an ever increasing rate. It is impossible for any exponential time method to keep up. Consequently, in this appendix, we focus on extending accurate association rule-based classification methods to larger data sets.

This appendix develops a scalable rule-based classifier called Boolean Structure Table Classification (BSTC) for microarray datasets. Given a labeled training set, such as the example in Table A.1, BSTC *efficiently* builds an *accurate* classifier. The emphasis on accuracy is easy to appreciate and comes from BSTC being related

to association rule-based methods. Hence, BSTC supports its classifications with intuitive rules. The emphasis on efficiency is also critical since large gene expression datasets are computationally taxing for existing association rule-based algorithms and, as successful microarray techniques fuel the growth of gene expression datasets, these methods will quickly become infeasible. In contrast, BSTC's space and runtime costs are only polynomial. Hence, BSTC is scalable to large data sets on which current association rule-based methods are challenged computationally.

In an attempt to control runtime many current association rule methods [25, 26, 82] utilize support-based rule pruning. Using a large enough support cutoff does allow rule mining to finish more quickly, but doesn't completely resolve the issue. If the user sets the support cutoff too small he/she can easily spend days waiting for rule mining to finish before giving up in frustration. A few such mistakes can result in weeks of wasted time. On the other hand, setting the support cutoff too high excludes the generation of important high-confidence lower-support rules [88]. In order to not miss too many important rules the user can't set the support cutoff too high. The end result is that in practice support cutoffs are difficult and time intensive to tune. In contrast, BSTC is fast and easy to use.

In addition, to the best of our knowledge all current association rule-based classifiers for gene expression data only handle datasets with two class labels. Although our example Table A.1 data contains just two class labels, in practice microarray data can contain an arbitrary, though small, number of class types. Unlike previous association rule-based classifiers, BSTC easily generalizes to datasets with more than two class types.

To develop an accurate, scalable, multi-class, and easy to use rule-based classifier we carefully considered the underlying primitives that power association rule-based

methods. These methods use **conjunctive association rules (CARs)**, in which the rule antecedent is restricted to being a conjunction of terms. In contrast, we approach this problem by relaxing the types of rules to an important and larger subset of the more general class of boolean association rules (BARs). We develop a novel method for compactly storing these BARs in a simple data structure called a Boolean Structure Table (BST). BSTs can then be used for BAR generation and classification. BST classification (BSTC) collectively considers many simple BARs with 100% confidence in bulk. Because the rules are simple BSTC avoids extensive rule mining. Furthermore, considering rules in bulk keeps the computational cost low.

The main contributions of this appendix are:

1. We propose a new polynomial time and space rule-based classifier for gene expression data analysis that is accurate, scalable, easy to use, and easily generalizable to multi-class classification.
2. We extensively evaluate our method against the current leading association rule-based method (RCBT [25]), and show that our method is orders of magnitude faster on large datasets while maintaining high classification accuracy.
3. We introduce a subclass of more general boolean association rules and relate them to existing CARs. This not only leads to a better appreciation of why our classification method works, but also lays the foundation for the future use of these BARs on other database problems.

The remainder of this appendix is organized as follows: First, in Section A.1 we formalize the concept of BARs. Then in Section A.2, we define a concept called Boolean Structure Tables (BSTs) which are related to an important class of BARs.

Section A.3 provides a polynomial time and parameter-free classifier based on BSTs. Section A.4 presents an extensive empirical evaluation of our classifier. Finally, Section A.5 discusses related work, and Section A.6 briefly presents our conclusions and directions for future work.

A.1 Preliminaries

We work with the following type of data: We are given a finite set G of genes and N collections of subsets from G. These N collections are disjoint and represented as $C_1 = \{s_{1,1}, \ldots, s_{1,m_1}\}$, $\ldots$, $C_N = \{s_{N,1}, \ldots, s_{N,m_N}\}$. Each C_i is called a **class type** or **class label**. Furthermore, we will refer to each set $s_{i,j} \subseteq G$ as a **sample** and every element $g \in G$ as a **gene**. We denote the total set of samples by $S = \bigcup_{i=1}^{N} C_i$. If $g \in s_{i,j}$ we will say that sample $s_{i,j}$ **expresses** gene g. Otherwise, if $g \in G$ and $g \notin s_{i,j}$ we will say that sample $s_{i,j}$ doesn't express gene g. Similarly, we say that sample s is of class type C_i if and only if C_i contains $s \subset G$. Consider the Table A.1 data. Here we have samples $S = \{s_1, s_2, s_3, s_4, s_5\}$ and genes $G = \{g_1, g_2, g_3, g_4, g_5, g_6\}$. Furthermore, we have $N = 2$ classes: $C_1 = \mathsf{Cancer} = \{s_1, s_2, s_3\}$ and $C_2 = \mathsf{Healthy} = \{s_4, s_5\}$.

Given such relational training data, a **conjunctive association rule (CAR)** is any element of $2^G \times \{1, \ldots, N\}$. A CAR $g_{j_1}, \ldots, g_{j_r} \Rightarrow n$ can be interpreted as follows: "If a query sample s contains all genes $g_{j_1}, \ldots, g_{j_r}$ then it should be grouped with class type C_n." Naturally, of the $2^{|G|} \cdot N$ possible association rules some are more useful than others. The following standard definitions were introduced in [6] to compare association rules:

Support: The support of a CAR $g_{j_1}, \ldots, g_{j_r} \Rightarrow n$, called $supp[g_{j_1}, \ldots, g_{j_r} \Rightarrow n]$, is:

$$|\{s_{n,j} \text{ s.t. } \{g_{j_1}, \ldots, g_{j_r}\} \subset s_{n,j}, 1 \leq j \leq m_n\}|.$$

Confidence: The confidence of a CAR $g_{j_1}, \ldots, g_{j_r} \Rightarrow n$ is:

$$\frac{supp[g_{j_1}, \ldots, g_{j_r} \Rightarrow n]}{|\{s_{i,j} \text{ s.t. } \{g_{j_1}, \ldots, g_{j_r}\} \subset s_{i,j} \forall i, j\}|}.$$

Consider the CAR $g_1, g_3 \Rightarrow$ Cancer for our running example in Table A.1. We can see that the example CAR has a support of 2 since only two Cancer samples, $s1$ and $s2$, contain both g_1 and g_3. Furthermore, we can see that the example CAR has confidence 1 (or 100%) since no healthy samples contain both g_1 and g_3.

A.1.1 Boolean Association Rules

For any sample s and gene g_i, $1 \leq i \leq n = |G|$, let $s[g_i] \in \{0, 1\}$ represent whether or not sample s expresses gene g_i. Furthermore, for $g_i \in G$ define $s[-g_i]$ to be $1 - s[g_i]$. Now suppose that $B(x_1, \ldots, x_n)$ is a Boolean expression whose value depends on some subset of $\{x_1, \ldots, x_n\}$. We can evaluate B to true or false given a sample s by computing $B(s[g_1], \ldots, s[g_n])$. For example, consider the boolean expression:

$$\hat{B}(x_1, x_2, x_3, x_4, x_5, x_6) = (x_1 \wedge x_3) \vee (x_2 \wedge x_4).$$

Using Table A.1 we can evaluate

$$\hat{B}(s_1[g_1], s_1[g_2], s_1[g_3], s_1[g_4], s_1[g_5], s_1[g_6]) \tag{A.1}$$

to be $(1 \wedge 1) \vee (1 \wedge 0) = 1$. Note that $\hat{B}$ will only evaluate the Table A.1 Cancer samples to $True$.

For a given class set C_i and boolean expression B we can create a **Boolean association rule (BAR)** of the form $B \Rightarrow C_i$. The interpretation of any such BAR, $B \Rightarrow C_i$, is "if $B(s[g_1], \ldots, s[g_n])$ evaluates to true for a given sample s, then s should belong to class C_i." From this point on we will work with the following generalized definitions of support and confidence:

	s1	s2	s3
g1	●	●	
g2	**(s4: g1)**		**(s4: -g3, -g5)**
g3	**(s4: g1) & (s5: -g4, -g6)**	**(s4: -g2, -g5) & (s5: -g4, -g5)**	
g4			**(s5: -g3, -g5)**
g5	**(s4: g1) & (s5: -g4, -g6)**		
g6		**(s5: -g4, -g5)**	**(s5: -g3, -g5)**

Figure A.1: Example BST for the Cancer Class

Support: The support of any BAR $B \Rightarrow C_i$, represented as $supp(B \Rightarrow C_i)$, is:

$$\{\text{samples } s \in C_i \text{ s.t. } B(s[g_1], \ldots, s[g_n]) \text{ evaluates to true}\}.$$

The corresponding numerical support value of $B \Rightarrow C_i$ is denoted as $|supp(B \Rightarrow C_i)|$.

Confidence: The confidence of a BAR $B \Rightarrow C_i$ is

$$\frac{|supp(B \Rightarrow C_i)|}{|\{\text{samples } s \text{ s.t. } B(s[g_1], \ldots, s[g_n]) \text{ evaluates to true}\}|}$$

For CARs these definitions coincide with the CAR definitions of support and confidence found in [6, 7]. Hence, they are natural generalizations of the previous definitions (see section A.2.3).

Consider our example boolean expression $\hat{B}$ in terms of Table A.1. We can see that the BAR $\hat{B} \Rightarrow \mathsf{Cancer}$ (shown in Eq. A.1) has support 3 and confidence 1.

A.2 BSTs and BARs

The discussion in this section will focus on tables for each class C_i. These tables, called Boolean Structure Tables (BSTs), will form the basis for our classification method. In order to motivate the utility of BSTs for classification, we will present their close relationship to a special category of BARs which, in turn, will be related

Algorithm A.1 Create-BST: The BST Creation Algorithm

Input: Finite set of Genes G, set of samples S, Class C_i
Output: The BST Table for Class C_i.
for all $(c, h) \in C_i \times S - C_i$ **do**
  initialize a pointer $\leftarrow$ NULL
end for
for all $(g, c) \in G \times C_i$ s.t. $g \in c$ and $g \notin \cup_{h \in S - C_i} h$ **do**
  Set $BST(g, c) \leftarrow$ Black Dot
end for
for all $(g, c, h) \in G \times C_i \times S - C_i$ s.t. $g \in c$ and $g \in h$ **do**
  if pointer $(c, h) \neq$ NULL **then**
    push a copy of $(c, h) \rightarrow BST(g, c)$
  else
    $L = \{g \in G$ s.t. $g \in h$ & $g \notin c\}$
    if $L \neq \emptyset$ **then**
      $(c, h) \leftarrow L$'s address
    else
      $L = \{g \in G$ s.t. $g \notin h$ & $g \in c\}$
      $(c, h) \leftarrow L$'s address
    end if
  end if
  Push a copy of $(c, h) \rightarrow BST(g, c)$.
end for

back to CARs. Through this discussion we will demonstrate that BSTs contain all the information of the high confidence CARs already known to be valuable for microarray data classification.

A.2.1 Boolean Structure Tables

A **Boolean Structure Table (BST)** $T(i)$ is a two dimensional table, $T(i) = G \times C_i$, where each table entry refers to a maximum of $|S| - |C_i|$ lists of up to $|G|$ genes each. For every C_i the associated BST, $T(i)$, will require $O\left((|S| - |C_i|) \cdot |G| \cdot |C_i|\right)$ space and can be constructed with proportional time complexity via Algorithm A.1.

When the Algorithm A.1 is run on the Table A.1 example input and for class Cancer, the Boolean Structure Table shown in Figure A.1 is produced. In Figure A.1 a black dot at location (g, s) indicates that no healthy samples express gene g but some cancerous sample does. A cell (g, s) is left blank only if sample s didn't express gene g. If (g, s) contains a list of the form $(h : -g_1, \ldots, -g_n)$ it means that s may be distinguished from sample h by the non-expression of any one of genes g_1 through

g_n. Similarly, if (g, s) contains a list of the form $(h : g_1, \ldots, g_n)$ it means that s may be distinguished from sample h by the expression of any one of genes g_1 through g_n. Such lists will hereafter be referred to as **exclusion lists**.

Note that there is no reason why the BST in Figure A.1 was created for the Cancer class. We can just as easily build a BST for the Healthy class using the example shown in Table A.1. In general, if a relational gene expression dataset contains N classes, we can construct N different BSTs for the data set (one for each class).

Runtime Complexity for BST Creation

We can see that the total time to construct BSTs via Algorithm A.1 for all of $C_1, \ldots, C_N$ is $O\left(\sum_{i=1}^{N}(|S| - |C_i|) \cdot |C_i| \cdot |G|\right)$. Given that the class sets C_i are all disjoint, we have $\sum_{i=1}^{N}(|S| - |C_i|) \cdot |C_i| \cdot |G| \leq \sum_{i=1}^{N} |S| \cdot |C_i| \cdot |G| = |S|^2 \cdot |G|$. Hence, BSTs can be constructed for all C_is in time $O(|S|^2 \cdot |G|)$.

A.2.2 BST Generable BARs

We view every BST cell, (g, c), as an atomic 100% confident BAR. For example, Figure A.1's $(g3, s1)$-cell corresponds to the BAR

$$g3 \text{ expressed } AND \ g1 \text{ expressed } AND \text{ (either } g4 \text{ or } g6 \text{ not expressed)} \Rightarrow \text{Cancer}.$$

We refer to this rule as the Figure A.1 BST's $(g3, s1)$-**cell rule**. Note that the cell rule is both (i) 100% confident, and (ii) supported by sample $s1$. Throughout the remainder of this section we will use such cell rules as atomic building blocks to construct more complicated BARs. Furthermore, in Section A.3, we will directly employ BST cell rules to build a new classifier called BSTC.

Mining More Complicated BST BARs

Let $T(i)$ be a BST for sample type C_i. We can view each row of $T(i)$ as a 100% confident BAR by combining the row's cell rules. To see this, choose any $g_j \in G$

Algorithm A.2 BSTRowBAR: Constructing BST Gene Row BAR

Input: Class C_i, BST for the class $T(i)$, gene g_j
Output: Row BAR for gene g_j with 100% conf.
$A \leftarrow FALSE$
for all $s \in C_i$ s.t. $T(i)$'s (g_j, s) − cell is not empty **do**
  $B \leftarrow TRUE$
  for all exclusion lists $e \in T(i)$'s (g_j, s)-cell **do**
    if $e = (s_k : -g_{l_1} \cdots - g_{l_m})$ **then**
      $B \leftarrow B\ AND\ (-g_{l_1}\ OR \ldots\ OR\ -g_{l_m})$
    else if $e = (s_k : g_{l_1} \ldots g_{l_m})$ **then**
      $B \leftarrow B\ AND\ (g_{l_1}\ OR \ldots\ OR\ g_{l_m})$
    end if
    $A \leftarrow A\ OR\ B$
  end for
end for
Return $g_j\ AND\ A \Rightarrow C_i$

and consider the CAR $g_j \Rightarrow C_i$. This CAR can be augmented with exclusion list clauses from each of $T(i)$'s g_j-row cells via Algorithm A.2. The result will be a BAR with 100% confidence which is logically equivalent to a disjunction of $T(i)$'s g_j-row cell rules. See Figure A.2 for the gene row BARs which result from applying Algorithm A.2 to the BST in Figure A.1.

For the remainder of this appendix we will restrict our attention to BARs that may be generated by taking conjunctions of BST cell rule disjuncts. Henceforth we simply refer to these as BARs. It is very important to notice that all such BARs have a special form: Their antecedents consist of a CAR antecedent *AND*ed with a disjunction of BST exclusion list clause conjunctions. Consider the BAR for gene g_6 in Figure A.2. Gene g_6's rule antecedent consists of a CAR antecedent, g_6, conjoined to a disjunction of the Figure A.1 exclusion list clauses: (either g_4 or g_5 not expressed) and (either g_3 or g_5 not expressed).

Along these same lines, BARs with more complex antecedents can be created by taking the logical AND of a BST's gene row rules. For example, consider our running example BST's gene row rules listed in Figure A.2. We can form the 100% confident CAR (g_1 expressed AND g_6 expressed) $\Rightarrow$ Cancer by ANDing Figure A.2

Gene g_1: (g_1 expressed) $\Rightarrow$ Cancer.
Gene g_2: (g_2 expressed AND [EITHER (g_1 expressed) OR (either g_5 or g_3 not expressed)]) $\Rightarrow$ Cancer.
Gene g_3: (g_3 expressed AND [EITHER {(g_1 expressed) AND (either g_4 or g_6 not expressed)} OR { (either g_2 or g_5 not expressed) AND (either g_4 or g_5 not expressed)}]) $\Rightarrow$ Cancer.
Gene g_4: (g_4 expressed AND [either g_5 or g_3 not expressed]) $\Rightarrow$ Cancer.
Gene g_5: (g_5 expressed AND [g_1 expressed AND (either g_4 or g_6 not expressed)]) $\Rightarrow$ Cancer.
Gene g_6: (g_6 expressed AND [(either g_4 or g_5 not expressed) OR (either g_3 or g_5 not expressed)]) $\Rightarrow$ Cancer.

Figure A.2: Gene Row BARs with 100% Confidence Values.

gene row rules for g_1 and g_6 as follows: While ANDing we use the BST in Figure A.1 to quickly simplify the resulting expression. First, we can tell that product will only be supported by sample s_2 because only the BST's s_2 column contains non-empty cells for both of gene rows g_1 and g_6. Thus, we only need to consider the exclusion lists in cells (g_1, s_2) and $(g_6, s2)$ while forming our product. Second, the black dot in BST entry $(g_1, s2)$ means we don't have to use the Healthy sample s_5 exclusion list information $(s_5 : -g_4, -g_5)$ from BST entry $(g_6, s2)$ in our new rule. This is because gene g_1 already excludes s_5 on its own since $g_1 \notin s_5$. By ANDing gene row rules in this manner we can create BARs with antecedents that are the conjunction of any desired CAR antecedent with a simplified exclusion list based clause (to eliminate non-C_i supporting samples). Progressive polynomial time algorithms for BAR mining via a BST can be found in an extended version of this appendix [1].

A.2.3 BARs Relationships to CARs

Let $R \Rightarrow C_i$ be any 100% confident BST created BAR containing exclusion clauses for non-C_i samples $h_1, \ldots h_m$. Removing all exclusion list clauses related to $\{\hat{h}_1, \ldots, \hat{h}_p\} \subset \{h_1, \ldots, h_m\}$, $p \leq m$, will create a new boolean association rule, $\hat{R} \Rightarrow C_i$, with support $= supp(R \Rightarrow C_i)$ and confidence $\geq \frac{|supp(R \Rightarrow C_i)|}{|(supp(R \Rightarrow C_i)|+p}$. Let's consider the g_3-row BAR from our running example:

(g_3 expressed AND [EITHER {(g_1 expressed) AND (either g_4 or g_6 not expressed)} OR { (either g_2 or g_5 not expressed) AND (either g_4 or g_5 not expressed)}]) $\Rightarrow$ Cancer.

It has 100% confidence and support $\{s1, s2\}$. Now, if we remove all exclusion list clauses related to sample row $s5$ we end up with the boolean association rule:

(g_3 expressed AND [EITHER (g_1 expressed) OR (either g_2 or g_5 not expressed)] $\Rightarrow$ Cancer.

This new rule has support $\{s1, s2\}$ and a confidence of $\frac{|\{s1,s2\}|}{|\{s1,s2,s5\}|} = \frac{2}{3}$. The preceding observation leads us to the following theorem:

Theorem A.1. *Let D be a relational data set containing s samples, no two of which are the same (i.e. no two sample rows express the exact same set of genes). Then, there exists a pure conjunction B implying a class type C (i.e., a CAR) with confidence c and support $supp$ for D if and only if there exists a 100% confident BST generated BAR $\hat{B} \Rightarrow C$ for D that: (i) has $supp(\hat{B} \Rightarrow C) = supp$, and (ii) contains exclusion list clauses actively excluding $(\frac{1}{c} - 1)|supp|$ non-C samples.*

Proof. $\Leftarrow$: From the observation directly preceding this theorem we can see that if $\hat{B} \Rightarrow C$ has $supp(\hat{B} \Rightarrow C) = supp$ then removing all the exclusion list clauses from $\hat{B}$ (by replacing them all with true) will create a new pure conjunction B with $supp(B \Rightarrow C) = supp$. Furthermore, we require that {non-C samples excluded by exclusion clauses} = {non-C samples satisfying B} (i.e., the exclusion clauses actually exclude something). Hence, $B \Rightarrow C$ will have confidence $c = \frac{|supp|}{|supp| + \#\text{ excluded samples}}$.
$\Rightarrow$: Let B be a conjunction of items/genes $g_1, \ldots, g_n$. Given that no two samples in D are the same we can build a 100% confident BST for class C of D. Furthermore, both the following are true:

1. A non-C sample h expresses all genes $g_1, \ldots, g_n \iff \forall s \in supp$ and $1 \leq i \leq n$ the BST cell (g_i, s) contains an active exclusion list for h. Thus, only non-C samples

expressing all of $g_1, \ldots, g_n$ (and therefore satisfying B) generate active exclusion lists in all relevant (g_i, s) BST cells.

2. $supp(B \Rightarrow C) = \bigcap_{1 \leq j \leq n} supp(g_j \Rightarrow C)$.

   Here we get the $\hat{B}$ by ANDing down each of the BST $supp(B \Rightarrow C)$ sample column's g_i cells and then ORing the resulting $|supp(B \Rightarrow C)|$ rules together. $\square$

Theorem 1 tells us how we can get CARs from BARs. Furthermore, it says 100% confident BARs with large support and a small number of excluded samples are equivalent to high support/confidence CARs. Hence, genes that show up in many high confidence, high support CARs will also be prevalent in many 100% confident BARS with high support and a low number excluded samples. Most importantly, we see that all high confidence CARs (which tend to be good classifiers) have closely related BAR counterparts. Furthermore, these counterparts can be mined from a BST by ANDing gene row BARs.

A.3 BST-Based Classification

In principal, 100% confident BST-generable BARs should be sufficient for classification because they contain at least as much information as all generable CARs do (see section A.2.3). Indeed, beyond what CARs with similar support are capable of, 100% confident BARs supply us with "unpolluted" ground truth. Thus, it is not too surprising that the class of BST-generable BARs we've looked at so far will be enough to enable highly accurate classification.

Let C_i be a class set of interest and $T(i)$ be the BST for class C_i constructed from the given training data. From section A.2.2 we can see that all BST generable BARs for class C_i are created by combining $T(i)$ cell rules. Thus, we expect that by restricting our attention to the $O(|G| \cdot |C_i|)$ atomic $T(i)$ cell rules we will be,

in some sense, still considering all $T(i)$ generable BARs for C_i. Our new scalable classifier, the **Boolean Structure Table Classifier (BSTC)**, capitalizes on this line of thought by ignoring BAR generation and focusing exclusively on atomic BST cell rules.

A.3.1 BSTC Overview

Let Q be a test/query gene expression data sample and $T(i)$ be a BST for class set C_i. BSTC is a heuristic rule-based classifier motivated by standard Boolean formula arithmetization techniques [90] such as those employed in fuzzy satisfiability [101]. By using these ideas we can avoid the highly costly process of support/confidence based association rule mining. Instead of explicitly generating rules, BSTC decides (heuristically), for all C_i, how well Q collectively satisfies $T(i)$'s atomic cell rules. BSTC then classifies Q as the sample class whose BST has the highest expected atomic rule satisfaction level from Q.

Intuitively, we expect BSTC to be accurate because it approximates the results of CAR-based classification: Suppose that a high support/confidence CAR exists which classifies our query sample Q as class C_j. This will only happen if all the CAR's antecedent genes, AG, appear in both (i) Q and, (ii) most of the training samples in the CAR's consequent class C_j. Let $T(j)$ be the BST for class C_j. Because of (ii) most of $T(j)$'s sample columns must contain cell entries for all the AG genes. Furthermore, all $T(j)$'s AG cell entries will have few exclusion lists in common (by Theorem 1). Hence, $T(j)$'s expected atomic rule satisfaction level from Q (i.e., Q's classification value) should be heavily influenced (increased) by the AG rows and their few shared lists.

Algorithm A.3 BST Cell rule quantized Evaluation (BSTCE)

1: **Input:** Class C_i, BST for the class $T(i)$, Samples S, Query sample Q
2: **Output:** Classification value
3: **for all** non-empty exclusion lists e in $T(i)$'s cells **do**
4: $\quad V_e \leftarrow \frac{|\{\hat{g} \in e \text{ s.t. } Q[\hat{g}]=1\}|}{|e|}$
5: **end for**
6: **for all** $(g, s) \in \{g \in G \text{ s.t. } Q[g] = 1\} \times C_i$ **do**
7: $\quad$ **if** $T(i)(g, s)$ contains a $\bullet$ **then**
8: $\quad\quad T(i)[g][s] \leftarrow 1$
9: $\quad$ **else**
10: $\quad\quad T(i)[g][s] \leftarrow$ Min $\{V_e$ s.t. e is in $T(i)(g, s)\}$
11: $\quad$ **end if**
12: **end for**
13: **for all** non-blank sample columns $s \in T(i)$ **do**
14: $\quad V_s \leftarrow$ Mean of non-blank $T(i)[\star][s]$ values
15: **end for**
16: Return the Mean of step 16's V_s values

A.3.2 BST Cell Rule Satisfaction

As above, let Q be a test/query gene expression data sample and $T(i)$ be a BST for class set C_i. Algorithm A.3, BSTCE, gives BSTC's method of calculating the level that Q satisfies a given atomic $T(i)$ cell rule. We next explain the rationale behind BSTCE.

We know that each $T(i)$ (g, s)-cell exclusion list, L, corresponds to a disjunction in $T(i)$'s (g, s)-cell rule. Hence, if Q satisfies any one negation/inclusion in L, Q will satisfy L. However, if Q expresses most of its genes in common with L's associated non-C_i sample we assume it's probably not of type C_i (i.e., Q is weakly excluded). Hence, we use BSTCE's line 4 ratio to approximate the probability that L correctly excludes Q from being of L's associated sample's class.

In order for the (g, s)-cell rule to be satisfied, all of (g, s)'s exclusion lists must be satisfied (i.e., logical *AND*). If independence of each exclusion list's correct classification is assumed it is natural to multiply all of (g, s)'s list's probabilities. We don't assume independence and use a min instead (line 10). Finally, recall that all black dots in $T(i)$ correspond to genes expressed only in class C_i samples. If Q expresses

Algorithm A.4 The BSTC Algorithm

1: **Input:** BSTs for all dataset classes $T(1), \ldots, T(N)$, Query sample Q
2: **Output:** Classification for query sample Q
3: **for all** $i \in \{1, \ldots, N\}$ **do**
4: $\quad CV(i) \leftarrow \text{BSTCE}(T(i), Q)$
5: **end for**
6: Return $min\{i | CV(i) = max\{CV(1), \ldots, CV(N)\}\}$

a black dot gene it automatically satisfies all that gene's non-empty $T(i)$ cell rules. Hence, black dots are all assigned values of 1 in BSTCE's line 8.

Once we have used BSTCE lines 1-12 to calculate Q's classification values (i.e., $T(i)$'s atomic rule satisfaction levels from Q) for each relevant simple (g, s)-cell rule, we are nearly finished. We have all the values required to judge Q's similarity to $T(i)$ via an expectation calculation. For the sake of $T(i)$'s expectation calculation, all that is left to do is imagine choosing a relevant simple $T(i)$ rule at random and then using it to classify Q. To randomly select a (g, s) rule we first imagine selecting a non-empty $T(i)$ sample column uniformly at random and then picking a cell-rule from that column uniformly at random. The expected probability of correctly classifying Q with $T(i)$ via this method (which heuristically is proportional to $T(i)$'s expected satisfaction level from Q) is then calculated by averaging the approximate cell rule satisfaction levels down each non-empty sample column (line 14) and then averaging the resulting non-empty sample averages (line 16).

A.3.3 BSTC Algorithm

Suppose we are given relational training data D containing sample rows S split up into disjoint class sets $C_1, \ldots, C_N$. BSTC uses D to construct N BSTs, $T(1), \ldots, T(N)$. Next, let G be the union of the elements contained in each sample row of D (i.e. the gene set of D) and let Q be a query sample with expression information regarding G. BSTC will use the BSTCE algorithm to classify Q as being the C_i with smallest i such that $\text{BSTCE}(T(i), Q) = max\{\text{BSTCE}(T(j), Q) | 0 \leq j \leq N\}$. See Algorithm A.4

for the BSTC algorithm.

Note that there is no reason why N must be 2. *BSTC easily generalizes to datasets containing more than two class labels.*

BSTC Runtime

As noted in section A.2.1 it takes time and space $O(|S|^2 \cdot |G|)$ to construct all the BSTs $T(1), \ldots, T(N)$. Thus, BSTC requires time and space $O(|S|^2 \cdot |G|)$ to construct. Furthermore, during classification BSTC must calculate BSTCE$(T(i), Q)$ for $1 \leq i \leq N$. BSTCE (Algorithm A.3) runs in $O\left((|S| - |C_i|) \cdot |G| \cdot |C_i|\right)$ time per query sample. Therefore the BSTC worst case evaluation time is also $O(|S|^2 \cdot |G|)$ per query sample. See Section A.6 for more on BSTC's per-query classification time.

Biological Meaning of BSTC Classification

Association rules mined from gene expression data provide an intuitive representation of biological knowledge (e.g., the expression of certain genes implies cancer). Hence, CAR-based classifiers have the desirable ability to justify each non-default consequent class query classification with the biologically meaningful CAR(s) the query satisfied. BSTC, being rule-based and related to CAR-classifiers, also has this property.

BSTC can support its query classifications with BARs of any user specified complexity. Most simply, for any given query sample Q and $c \in (0, 1]$, BSTC can justify its classification of Q as class C_i by reporting all $T(i)$ atomic cell rules with satisfaction levels $\geq c$. Note that returning this information requires no additional per-query classification time. Also note that section A.2.2 methods can be used to mine more complex highly satisfied BARs if desired.

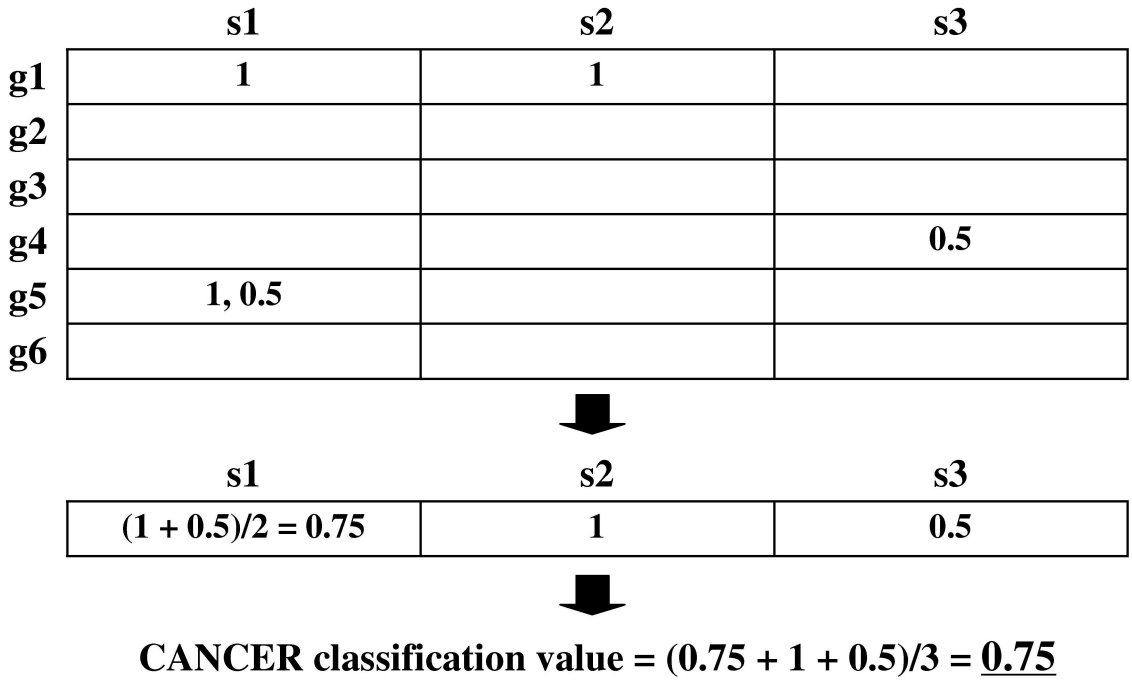

Figure A.3: BSTC cell rule Evaluation Example

A.3.4 BSTC Example

Consider our running example from Table A.1. In order to construct BSTC we must construct both $T(\mathsf{Healthy})$ and $T(\mathsf{Cancer})$ (shown in Figure A.1). Once both BSTs have been constructed we can begin to classify query samples. Suppose, for example, we are given the query sample $Q = \{g1$ expressed, $g2$ not expressed, $g3$ not expressed, $g4$ expressed, $g5$ expressed, $g6$ not expressed$\}$. To classify this query we must first calculate BSTCE$(T(\mathsf{Cancer}), Q)$ and BSTCE$(T(\mathsf{Healthy}), Q)$.

The evaluation of BSTCE$(T(\mathsf{Cancer}), Q)$ proceeds as follows: Since our query sample Q expresses gene g_5 we can see that we must, for example, determine the fraction of both of the (g_5, s_1)-cell's exclusion lists satisfied by Q. The (g_5, s_1)-cell's $(s_4 : g_1)$ exclusion list is totally satisfied since Q expresses g_1. Hence, it gets a value of 1. However, the $(s_5 : -g_4, -g_6)$ exclusion list is only half satisfied since, although Q doesn't express g_6, Q does expresses g_4. Thus, in total, we only consider half of the

simple (g_5, s_1)-cell rule to be satisfied (i.e. the s_5 exclusion list is the weakest link). Continuing to use BSTC's approximation scheme for the expected probability of Q's correct Cancer classification via the Figure A.1 BST we obtain Figure A.3. Note that only Figure A.3 gene rows corresponding to genes expressed in Q are non-empty.

If we now evaluate BST-EXPECT(T(Healthy), Q) we obtain a final value of $\frac{3}{8}$. To finish, BSTC will compare Q's Cancer classification value of $\frac{3}{4}$ to Q's Healthy classification value of $\frac{3}{8}$ and conclude that Q is most probably Cancer. Hence, Q will be classified as Cancer.

A.4 Experimental Evaluation

All experiments reported here were carried out on a 3.6 GHz Xeon machine with 3GB of memory running Red Hat Linux Enterprise 4. For our empirical evaluation we use four standard real microarray datasets [2]. Table A.2 lists the dataset names, class labels, and the number of samples of each class. All discretization was done using the entropy-minimized partition [4] as in [25].

Dataset	# Genes	Class 1 label	Class 0 label	# Class 1 samples	# Class 0 samples
ALL/AML (**ALL**)	7129	ALL	AML	47	25
Lung Cancer (**LC**)	12533	MPM	ADCA	31	150
Prostate Cancer (**PC**)	12600	tumor	normal	77	59
Ovarian Cancer (**OC**)	15154	tumor	normal	162	91

Table A.2: Gene Expression Datasets

Executables for both RCBT and Top-k were provided by the authors of [25]. In all experiments, the Top-k rule generator was used to generate rule groups for RCBT. Unless otherwise noted we ran both Top-k and RCBT with the author-suggested parameter values (i.e., support = 0.7, $k = 10$, $nl = 20$, 10 RCBT classifiers). Hence, while generating rules for RCBT we used Top-k with a minimum support value of 0.7 and found the 10 most confident covering rule groups (i.e. $k = 10$). Furthermore,

Dataset	# Class 1 Training Samples	# Class 0 Training Samples	Genes After Discr.	BSTC Accur.	RCBT Accur.	SVM Accur.	random-Forest Accuracy
ALL	27	11	866	82.35%	91.18%	91.18%	85.29%
LC	16	16	2173	100%	97.99%	93.29%	99.33%
PC	52	50	1554	100%	97.06%	73.53%	73.53%
OC	133	77	5769	100%	97.67%	100%	100 %
Avg. Accuracy				95.59%	95.98%	89.5%	89.54%

Table A.3: Results Using Given Training Data

during classification we used RCBT with the suggested 10 classifiers (1 primary and 9 standby). Finally, nl, the number of lower bound rules to use for classification per Top-k mined rule group, was set equal to 20. We coded BSTC with C++.

A.4.1 Preliminary Experiments

Each of Table A.2's four gene expression datasets comes with a clinically determined training set. The authors of [25] provided us with their discretizations of these four datasets. We ran BSTC on their discretizations and BSTC matched RCBT's reported mean accuracy (about 96%) outperforming CBA (87%), IRG (81%), Weka 3.2 (C4.5 family single tree (74%), bagging (78%), boosting(74%)), and SVM$^{\text{light}}$ 5.0 (93%) in reported mean performance [25].

To compare BSTC and RCBT with the most recent R `e1071` package SVM implementation [22] and randomForest version 4.5 [17] we rediscretized the four datasets and reran BSTC/RCBT. To keep comparisons fair we ran SVM and randomForest on the same genes selected by our entropy discretization except with their original undiscretized gene expression values. SVM was run with its default radial kernel. We ran randomForest 10 times with its default 500 trees for ALL, LC, and OC and its accuracy was constant. For PC we had to increase randomForest's number of trees to 1000 before its accuracy stabilized over the 10 runs.

Table A.3 contains the number of class 0/1 samples in the clinically determined training set, the number of genes selected by our entropy discretization, and our experimental results. As shown in this table, the overall average accuracies of BSTC and RCBT are again best at about 96% each. When compared against RCBT, SVM, and randomForest on the individual tests we can see that BSTC is alone in having 100% accuracy on the majority of datasets.

However, BSTC's performance on the preliminary AML/ALL dataset test is relatively poor. This is likely due to over fitting. Every error BSTC made mistook a class 0 (AML) test sample for a class 1 (ALL) test sample (i.e., all errors were made in this same direction). And, the ALL training data has both (*i*) about 2.5 times as many class 1 samples as class 0 samples, and (*ii*) a small number of total samples/genes. When the training set is more balanced and the number of samples/genes is larger we can expect that cancellation of errors will tend to neutralize/balance any over fitting effects in BSTC. And, BSTC is a method meant primarily for large training sets where CAR-mining is prohibitively expensive. As we will see later in Section A.4.2, BSTC's performance is much better for larger AML/ALL training set sizes.

A.4.2 Holdout Validation Studies

Holdout validation studies make comparisons less susceptible to the choice of a single training dataset and provide performance evaluations that are likely to better represent program behavior in practice. We next present results from a thorough holdout validation study completed using 100 different training/test sets from each of the ALL, LC, PC, and OC data sets. For these holdout validation tests we benchmark BSTC against Top-k/RCBT because (*i*) BSTC and RCBT perform best in our preliminary experiments, (*ii*) Top-k/RCBT is the fastest/most accurate CAR-based classifier for microarray data, and (*iii*) we are interested in BSTC's CAR-

related vs Top-k/RCBT's CAR-based scalability.

For the holdout validation study we generated training sets of sizes 40%, 60%, and 80% of the total samples. Each training set was produced by randomly selecting samples from the original combined dataset. We then used the standard R dprep package's entropy minimized partition [4] to discretize the selected training samples. Finally, the remaining dataset samples were used for testing the two classifiers after rule/BST generation on the randomly selected training data. For each training set size we produced 25 independent tests. In addition to these training sets, we created an additional 25 1-x/0-y tests. To create these tests we chose training data by randomly selecting x class 1 samples and y class 0 samples to be used as training data. As before, the remaining samples were then used to test both classifiers. For each dataset the x and y values are chosen so that the resulting 25 classification tests have the exact same training/test data proportions as the single related dataset test reported in section A.4.1. For each training set size we plot our results using a boxplot.

Boxplot Interpretation: Each boxplot that we show in this section can be interpreted as follows: The median of the measurements is shown as a diamond, and a box with boundaries is drawn at the first and the third quartile. The range between these two quartiles is called the inter-quartile range (IQR). Vertical lines (a.k.a. "whiskers") are drawn from the bottom and the edge of the box to indicate the minimum and the maximum value, unless outliers are present. If outliers are presents, the whiskers only extend to $1.5 \times IRQ$. The outliers that are near (i.e. within $3 \times IRQ$ are drawn as an empty circle, and further outliers are drawn using an asterisk.

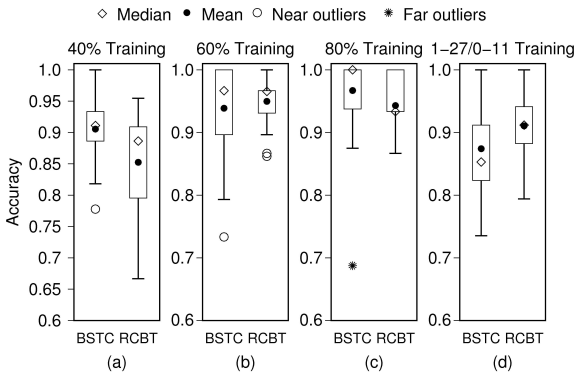

Figure A.4: ALL Holdout Validation Results

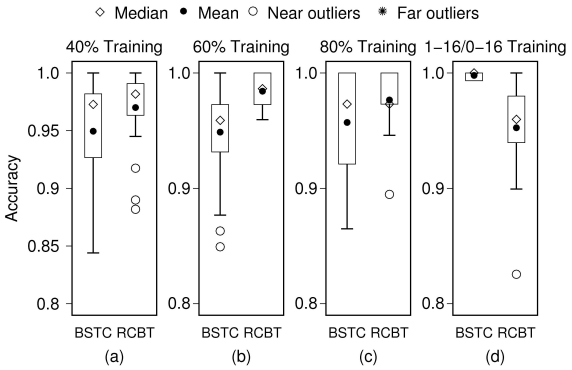

Figure A.5: LC Holdout Validation Results

ALL/AML (ALL) Experiment

Figure A.4 shows the classification accuracy for the ALL/AML dataset. As can be seen in this figure, BSTC and RCBT have similar accuracy across the ALL/AML tests as a whole. BSTC outperforms RCBT in terms of median and mean accuracy on the 40% and 80% training set sizes while RCBT has better median/mean accuracy on the 1-27/0-11 training size tests. And, both classifiers have the same median on the 60% training set size. Over the 100 ALL/AML tests we see that BSTC has a mean accuracy of 92.13% while RCBT has a mean accuracy of 91.39% (they are very close).

It's noteworthy that BSTC is 100% accurate on the majority of 80% training size tests. However, BSTC appears to have slightly higher variance than RCBT on all but the 40% training tests. Considering all the results together both BSTC and RCBT have essentially equivalent classification accuracies on the ALL/AML dataset.

Lung Cancer (LC) Experiment

The results for the Lung Cancer dataset are reported in Figure A.5. Here, again, both BSTC and RCBT have similar classification behavior. RCBT has higher mean and median accuracies on the 40% and 60% tests while BSTC outperforms RCBT

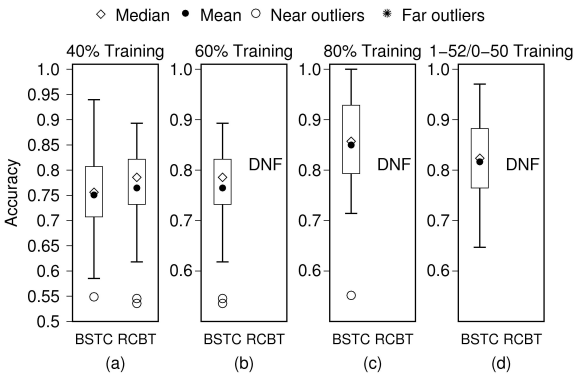

Figure A.6: PC Holdout Validation Results

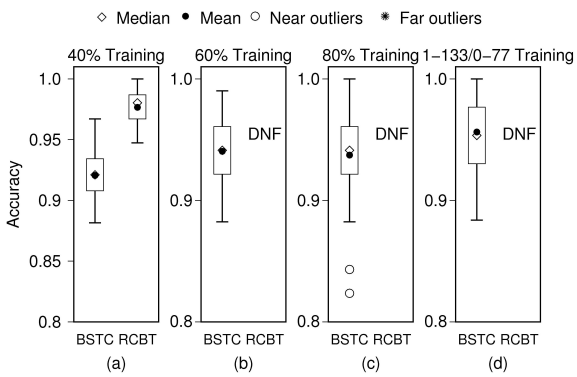

Figure A.7: OC Holdout Validation Results

on the 1-16/0-16 tests. Meanwhile, both classifier have the same median on the 80% training test. Over all 100 LC tests we find that BSTC has a mean accuracy of 96.32% while RCBT has a mean accuracy of 97.08% (again, they are very close).

As before, BSTC is alone in having 100% accuracy more then half the time for any training set size (see Figure A.5 (d)). However, RCBT has smaller variance for 3 of the 4 training set sizes. Therefore, as for the ALL/AML data set, both BSTC and RCBT have about the same classification accuracy on LC.

Prostate Cancer (PC) Experiment

RCBT begins to run into a computational difficulties on PC's larger training set sizes. This is because before using a Top-k rule group for classification RCBT must first mine *nl* lower bound rules for the rule group. RCBT accomplishes rule group lower bound mining via a pruned breadth-first search on the subset space of the rule group's upper bound antecedent genes. This breadth-first search can be quite time consuming. In the case of the Prostate Cancer (PC) dataset all 100 classification tests (25 tests for each of the 4 training set sizes) generated at least one top-10 rule group upper bound with more than 400 antecedent genes. Due to the difficulties involved with a breadth-first search over the subset space of a several hundred element set, RCBT began suffering from long run times on many PC classification tests.

Table A.4 contains four average classification test run times (in seconds) for each PC training size. The 'BSTC' column run times reflect the average time required to build both class 0 and class 1 BSTs and then use them to classify all the test samples. Each 'Top-k' column run time is the average time required for Top-k to mine the top 10 covering rule groups (with minimum support 0.7) for each training set.

Table A.4's 'RCBT' column gives average run times for RCBT using a time cutoff value of 2 hours for all the training sets. For each classification test, if RCBT was unable to complete the test in less than the cutoff time, it was terminated and its run time was reported as the cutoff time. Hence, the 'RCBT' column gives lower bounds on RCBT's average run time per training set test. Finally, the '# RCBT DNF' column gives the number of tests RCBT was unable to finish in $<$ the cutoff time, over the number of tests for which Top-K finished mining rule group upper bounds.

Training	BSTC	Top-k	RCBT	# RCBT DNF
40%	2.13	0.09	418.81	0/25
60%	4.93	5.06	$\geq$ 7110.00	24/25
80%	5.78	120.63	$\geq$ 7200 †	25/25†
1-52/0-50	5.57	21.32	$\geq$ 7200 †	25/25†

Table A.4: Average Run Times for the PC Tests (in seconds). † indicates nl was lowered to 2.

Explanation for varying nl values: Run time cutoffs were necessary to mitigate excessive holdout validation CAR-mining times. Even with a cutoff of 2 hours these 100 PC experiments required about 11 days of computation time, with most experiments not finishing. For the 80% and 1-52/0-50 training set sizes RCBT with $nl = 20$ failed to finish lower bound rule mining for all 50 tests within 2 hours. Thus, RCBT's nl parameter was lowered from the default value of 20 to 2 in an attempt to improve its chances of completing tests. Not surprisingly, decreasing nl (i.e., mining

fewer lower bound rules per Top-k rule group) decreases RCBT's runtime. However, RCBT was still unable to finish lower bound rule mining for any tests.

Training	BSTC	RCBT
40%	75.08%	79.27%
60%	78.18%	85.45%
80%	84.98%	—
1-52/0-50	81.65%	—

Table A.5: Mean Accuracies for the PC Tests that RCBT Finished.

Classification Accuracy: Figure A.6 contains accuracy results for BSTC on all four Prostate Cancer test sets. Prostate Cancer boxplots for RCBT were not constructed for training set sizes RCBT was unable to complete all 25 tests for within the time cutoffs. In contrast, BSTC was able to complete each of the 100 PC classification tests in less than 6 seconds. Table A.5 contains mean accuracies for the PC dataset with 40%, 60%, 80%, and 1-52/0-50 training. For each training set, the average accuracies were taken over the tests RCBT was able to complete within the cutoff time. Hence, the 40% row means were taken over all 25 results. Since RCBT was unable to complete any 80% or 1-52/0-50 training size tests we report these BSTC means over all 25 tests. RCBT has slightly better accuracy then BSTC on 40% training. For 60% training RCBT outperforms BSTC on the single test it could finish by more then 7%, although it should kept in mind that RCBT's results for the 24 unfinished tests could vary widely. Note that BSTC's (mean) accuracy increases monotonically with training set size as expected. At 60% training BSTC's accuracy behaves almost identically to RCBT's 40% training accuracy (see Figure A.6).

Ovarian Cancer (OC) Experiment

For the Ovarian Cancer dataset, which is the largest dataset in this collection, the Top-k mining method that is used by RCBT also runs into long computational

times. Although Top-k is an exceptionally fast CAR group upper bound miner, it still depends on performing a pruned exponential search over the training sample subset space. Thus, as the number of training samples increases Top-k quickly becomes computationally challenging to tune/use.

Table A.6 contains four average classification test run times (in seconds) for each Ovarian Cancer(OC) training size. As before, the second column run times each give the average time required to build both class 0/1 BSTs and then use them to classify all test's samples with BSTC. Note that BSTC was able to complete each OC classification test in about 1 minute. In contrast, RCBT again failed to complete processing most classification tests within 2 hours.

Training	BSTC	Top-k	RCBT	# RCBT DNF
40%	30.89	0.6186	273.37	0/25
60%	61.28	41.21	$\geq$ 5554.37	19/25
80%	71.84	$\geq$ 1421.80	$\geq$ 7205.43 †	21/22
1-133/0-77	70.38	$\geq$ 1045.65	$\geq$ 6362.86 †	20/23

Table A.6: Average Run Times for the OC Tests (in seconds). † indicates nl was lowered to 2.

Table A.6's third column gives the average times required for Top-k to mine the top 10 covering rule groups upper bounds for each training set test (with the same 2 hour cutoff procedure as used for PC testing). The fourth column gives the average run times of RCBT on the tests for which Top-k finished mining rules (also with a 2 hour cutoff). Finally, the '# RCBT DNF' column gives the number of tests that RCBT was unable to finish classifying in $<$ 2 hours each, over the number of tests for which Top-k finished. Because RCBT couldn't finish any 80% or 1-133/0-77 tests within 2 hours with $nl = 20$, we lowered nl to 2.

Classification Accuracy: Figure A.7 contains boxplots for BSTC on all four OC classification test sets. Boxplots were not generated for RCBT with 60%, 80%, or 1-133/0-77 training since it was unable to finish all 25 tests for all these training

set sizes in less than 2 hours each. Table A.7 lists the mean accuracies of BSTC and RCBT over the tests on which RCBT was able to produce results. Hence, Table A.7's 40% row consists of averages over 25 results. Meanwhile Table A.7's 60% row results are from 6 tests, 80% contains a single test's result, and 1-133/0-77 results from 3 tests. RCBT has better mean accuracy on the 40% training size, but the results are closer on the remaining sizes (less than 4% difference over RCBT's completed tests). Again, RCBT's accuracy could vary widely on its uncompleted tests.

Training	BSTC	RCBT
40%	92.05%	97.66%
60%	95.75%	96.73%
80%	94.12%	98.04%
1-133/0-77	93.80%	96.12%

Table A.7: Mean Accuracies for the OC Tests that RCBT Finished.

CAR Mining Parameter Tuning and Scalability: We attempted to run Top-k to completion on the 3 OC 80% training and 2 OC 1-133/0-77 training tests. However it could not finish mining rules within the 2 hour cutoff. Top-k finished two of the three 80% training tests in 775 min 43.64 sec (about 13 hours) and 185 min 3.29 sec. However, the third test ran for over 16,000 min (> 11 days) without finishing. Likewise, Top-k finished one of the two 1-133/0-77 tests in 126 min 45.15 sec but couldn't finish the other in 16,000 min (> 11 days). After increasing Top-k's support cutoff from 0.7 to 0.9 it was able to finish the two unfinished 80% and 1-133/0-77 training tests in 5 min 13.8 sec and 35 min 36.85 sec, respectively. However, RCBT (with $nl = 2$) then wasn't able to finish lower bound rule mining for either of these two tests within 1,500 min.(more than a day). Clearly, CAR-mining and parameter tuning on large training sets is computationally challenging. As training set sizes increase, it is likely that these difficulties will also increase.

A.5 Related Work

While operating on a microarray dataset, current CAR [25, 26, 107, 108] and other pattern/rule [81, 98] mining algorithms perform a pruned and/or compacted exponential search over either the space of gene subsets or the space of sample subsets. Hence, they are generally quite computationally expensive for datasets containing many training samples (or genes as the case may be). BSTC is explicitly related to CAR-based classifiers, but requires no expensive CAR mining.

Existing pattern/rule miners attempt to streamline the process of mining useful CARs in several ways. Part of the difficulty involved with mining CARs is that in addition to the exponentially large number of uninteresting rules that may be formed, there are usually many interesting rules as well. This means CAR miners such as CHARM [108] and CLOSET+ [107] may not only end up having to wade through a prohibitive number of low quality rules while discovering interesting CARs, but there may also be a huge number of repetitive CARs that are discovered.

The FARMER algorithm reduces the number of stored interesting rules by utilizing the notion of a **rule group**. Rule groups allow many interesting rules with similar sample support to be clustered together in a more compact form. Although rule groups provide a beneficial reduction in the number of interesting CARs which must be saved, there are typically still a large number of interesting rule groups. Hence, for large datasets it can still be prohibitively expensive for FARMER to find and store all user targeted rule groups.

More recently, the Top-k algorithm has solved the problem of generating an excessive number of interesting (i.e. high confidence) user targeted rule groups. Top-k cleverly allows the user to decide on the number of best rule groups to find and store.

Hence, a small number of non-redundant CAR rule groups may be stored and used for dataset analysis and classification. Although a significant step forward, Top-k still depends on performing a pruned exponential search of the dataset's training sample subset space. Furthermore, the RCBT [25] classifier proposed by the Top-k authors requires a potentially prohibitively expensive breadth-first search on the subset space of antecedent genes in each discovered rule group upper bound.

BSTC is also related to decision tree-based classifiers such as random forest [17] and C4.5 family [96] methods. It is possible to represent any consistent set of boolean association rules as a decision tree, and vice versa. However, it is generally unclear how the trees generated by current tree-based classifiers are related to high confidence/support CARs which are known to be particularly useful for microarray data[25, 26, 38, 79, 88]. BSTC is explicitly related to, and motivated by, CAR-based methods.

To the best of our knowledge there is no previous work on mining/classifying with BARs of the form we consider here. Perhaps the work closest to utilizing 100% BARs is the TOP-RULES [80] miner. TOP-RULES utilizes a data partitioning technique to compactly report item/gene subsets which are unique to each class set C_i. Hence, TOP-RULES discovers all 100% confident CARs in a dataset. However, the method must utilize an emerging pattern mining algorithm such as MBD-LLBORDER [37], and so generally isn't polynomial time. Also related to our BAR-based techniques are recent methods which mine gene expression training data for sets of fuzzy rules [105, 59]. Once obtained, fuzzy rules can be used for classification in a manner analogous to CARs. However, the resulting fuzzy classifiers don't appear to be as accurate as standard classification methods such as SVM [59].

A.6 Conclusions and Future Work

To address the computational difficulties involved with preclassification CAR mining (see Tables A.4 and A.6), we developed a novel method which considers a larger subset of CAR-related boolean association rules (BARs). These rules can be compactly captured in a Boolean Structure Table (BST), which can then be used to produce a BST classifier called BSTC. Comparison to the current best CAR classifier, RCBT, on several benchmark microarray datasets shows that BSTC is competitive with RCBT's accuracy while avoiding the exponential costs incurred by CAR mining (see Section A.4.2). Hence, BSTC extends generalized CAR-based methods to larger datasets then previously practical. Furthermore, unlike other association rule-based classifiers, BSTC easily generalizes to multi-class gene expression datasets.

BSTC's per-query classification time: BSTC's worst case theoretical per-query classification time is currently worse than a CAR-based method's ($O(|S|^2 \cdot |G|)$ versus $O(|S| \cdot |G|)$), after all exponential time CAR mining is completed. As future work we plan to investigate techniques to decrease BSTC's per-query classification time by carefully culling BST exclusion lists. For now we simply point out that BSTC's Section A.4 run times are reasonable and will remain so for larger problems on which CAR mining is infeasible (e.g., for OC training sets containing several hundred samples).

Generalizing BSTC: As future work we also plan to experiment with other boolean formula arithmetization procedures besides those employed to evaluate BST satisfaction levels in Algorithm A.3. Multiple BST satisfaction level arithmetization procedures could be used along with a heuristic classification confidence measure employed to select the best one. One potential confidence measure is the normalized

difference between the highest and second highest BST satisfaction level returned by each arithmetization procedure. The larger the normalized difference, the more "sure" the procedure appears to be about its classification.

Appendix B

Fast Line-based Imaging of Small Sample Features

This project aims to reduce the time required to attain more detailed scans of small interesting regions present in a quick first-pass sample image. In particular, we concentrate on high fidelity imaging of small sample features via hyperspectral Raman imaging (e.g., small scale compositional variations in bone tissue [89]). The current standard procedure for high quality hyperspectral Raman imaging of small sample features consists of four steps: First-Pass Imaging, Detail Identification, Planning, and finally Detail Imaging. Traditionally, Detail Imaging and Planning have been carried out manually by human personnel—after acquiring some quick low-quality data in First-Pass Imaging, a researcher looks for interesting features (Detail Identification) and decides how to acquire higher-quality data for the interesting features (Planning), which is done in the final Detail Imaging phase. In this appendix we will discuss automating the Detail Identification and Planning steps, resulting in a decrease of the procedure's total integration time. We fix an arbitrary way to automate Detail Identification and compare several different Planning methods. Our primary result is a method guaranteed to return a least cost (e.g., minimum integration time/number of scans) Detail Image under a general cost model. Because of their generality, the methodologies developed here may prove widely useful to basic biomedical scientists as well as to researchers in the pharmaceutical industry.

B.1 Introduction

Within the last several years many biomedical research groups have begun studying the compositional chemical properties that underlie the mechanical properties of bone. Unlike higher levels of architecture, the compositional level of bone was previously neglected due to the paucity of tools for non-destructive bone composition study. Recently the content and organization of bone at the molecular level has been successfully explored using Raman microspectroscopy and Raman imaging [20, 57, 89, 102]. These studies, as well as others in the literature, have begun to shed light on the molecular mechanisms of bone failure and response under both normal and diseased states.

An important hindrance to spectroscopic studies has been the long data acquisition time required for Raman microspectroscopy and Raman imaging. The time required to acquire a 256×256-pixel Raman image now (2008) varies between about 30 minutes and several hours. Reasons for this long imaging time include the tendency for current image acquisition protocols to be simple, manual, and non-adaptive. For example, during sample imaging a constant integration (acquisition) time is traditionally used at every data point despite the fact that there are usually several different optimal integration times for different types of regions.

Currently, small-scale sample features are imaged via Raman spectroscopy in four steps. First, during First-Pass Imaging, a low fidelity neighborhood image is quickly obtained. Then, during Detail Identification, the first-pass image is used to identify small interesting features—this stage is often done manually by a human expert. That expert then plans how to gather data during the fourth step. Finally, during Detail Imaging, the specimen is imaged again according to the plan to gather high

quality detail data. In this appendix we will propose automating Detail Identification and Planning with the following goals:

- Make Detail Identification more reliable and more repeatable than current manual processes. We expect our proposal to make this stage quicker as well, though we have not investigated this experimentally.
- Make the Planning phase provably optimal or nearly-optimal in the sense of minimizing the time for subsequent Detail Imaging.

B.2 Background and Methodology

For the remainder of the appendix we will consider each Raman image to be an $n \times m$ array of spectral data. Every image location (i, j) will correspond to a physical location in row i and column j of the sample. Each column of the image is gathered by one scan. Hence, given that each scan provides n pixels of spectral data, it takes m scans to produce an $n \times m$ image. During each scan, a sample column of data is illuminated with a laser while the induced radiation from each of the sample column's n data points is measured with an Electron Multiplying Charge Coupled Detector (EMCCD). In general we'd like to reduce the total imaging integration time not only for increased speed, but also to minimize potential sample damage due to the laser illumination. Hence, given a small collection of interesting sample positions to be imaged with a long integration time, we'd like to minimize the number of long scans required to cover the interesting sample positions.

In this appendix, our focus is the comparison of different methods for the Planning phase. To that end, we will fix a method for Detail Identification. We discuss this further in Section B.4.

The purpose of this appendix is to propose a new method for Raman imaging

and give theoretical and proof-of-concept support using a small amount of data. Ultimately, the effectiveness of our methods must be validated using many samples; that will be the subject of future work. We will avoid asking questions that can only be addressed by examining many samples.

B.3 Optimal Column/Row Scanning

In this section, we assume that Detail Identification has been performed, resulting in a set P of interesting pixels in the $[n] \times [m]$ grid. We address the Planning stage.

Traditionally, only columns are scanned. Once the sample is fixed, imaging only takes place by acquiring frames (scanning columns) from left to right. However, it is generally possible to rotate the specimen by 90°. We therefore consider the more general problem of minimizing the number of long column and/or row scans required to cover a small number of interesting sample points.

Definition B.1. Given a set $P \subseteq [n] \times [m]$ of p interesting pixel locations, a set $U = C \cup R$ is a *feasible cover of* P if $C \subseteq [m]$ is a set of columns and a set $R \subseteq [n]$ of rows such that, for every $(i, j) \in P$, either $i \in R$ or $j \in C$.

A feasible cover U of P is *optimal* if it has the minimum size of all feasible covers.

The set P is typically derived from quick First-Pass Imaging. See the 4×3 rectangular image in Figure B.1 for an example problem.

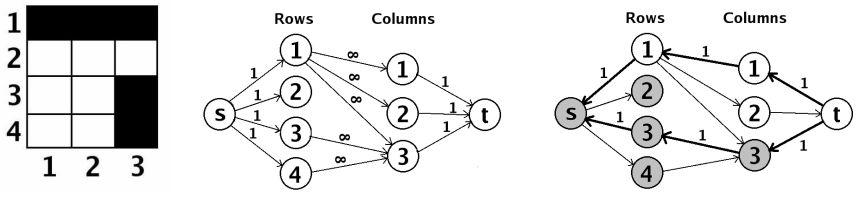

Figure B.1: An Example Problem, The Problem's Related Scan Graph, and a Scan Graph Solution

In the Figure B.1 example image we would like to scan the five black pixels. Hence,

our set of interesting pixels is $P = \{(1,1),(1,2),(1,3),(3,3),(4,3)\}$. Our task is to find the minimum number of columns and/or rows to scan in order to image all 5 black pixels.

We next compare three methods for obtaining feasible covers. They all take a set P of p interesting pixels, and return a set of columns C and/or rows R to be scanned in order to cover P. The three methods are:

B.3.1 Push Broom

Let $x = min\{j \mid \exists i \in \mathbb{N} \text{ with } (i,j) \in P\}$ and $y = max\{j \mid \exists i \in \mathbb{N} \text{ with } (i,j) \in P\}$. Scan $C = \{x, x+1, \ldots, y-1, y\}$ and $R = \emptyset$.

The Push Broom method is essentially the current standard method for scanning a small number of interesting pixels. After quickly obtaining a low fidelity first-pass image, a set of interesting pixels is obtained. The entire region from leftmost to rightmost column containing interesting pixels is then rescanned from left to right with a higher integration time.

B.3.2 Optimal Columns

Scan column set $C = \{j \mid \exists i \in \mathbb{N} \text{ with } (i,j) \in P\}$ and row set $R = \emptyset$. In effect, scan every column containing an interesting pixel.

B.3.3 Optimal Rows + Columns

Scan any cover of P that is Optimal.

It is straightforward to implement the Push Broom and Optimal Columns methods. Algorithms for Optimal Rows + Columns have been known [106]; we include a brief discussion for completeness and to illustrate the computational cost.

We omit the proof of the following.

Algorithm B.1 Plan: Plan Detail Imaging

1: **Input:** Pixels to image P.
2: **Output:** Optimal Rows + Columns cover of P.
3: Construct a **scan graph** for P. The scan graph of P is a directed weighted graph, G, with node set $\{s,t\} \cup \{1,2,\ldots,n\} \cup \{1,2,\ldots,m\}$ and edge set $\{(s,i) \mid 1 \leq i \leq n\} \cup P \cup \{(j,t) \mid 1 \leq j \leq m\}$. All edges from the source node s and into the termination node t have a weight of 1. All remaining P edges are given a weight of ∞.
4: Use the Ford-Fulkerson method [31] to find a minimum cut of G.
5: Using the final resulting residual network we let C be the set of columns reachable from s and R be the set of rows not reachable from s.

Theorem B.2. *Algorithm* B.*1 produces an Optimal Rows + Columns cover of its input,* P.

Example B.3. Recall the Figure B.1 example image. Figure B.1's middle graph is the scan graph for the 4×3 image with $P = \{(1,1),(1,2),(1,3),(3,3),(4,3)\}$. Figure B.1's rightmost graph gives the *residual network* that arises using the Ford-Fulkerson algorithm for a minimum cut in the scan graph. In the rightmost graph all gray nodes are reachable from the source node s. All white nodes are unreachable from s. Note that the gray(reachable) column 3 and white(not reachable) row 1 nodes provide us with an Optimal Rows + Columns cover of P. By inspecting the example image we can see that scanning row 1 and column 3 is indeed a minimal way of imaging P. Furthermore, we can see that if we only use columns or rows alone it will require 3 scans to cover P as opposed to only 2 scans.

The computational cost to run Algorithm B.1 is polynomial in the size of the input, P. Note that the size of P is at most the total number mn of *possible* pixels; in the context where this algorithm is used, we expect that $|P| \ll mn$. For a 256×256-pixel image, we expect that the time to compute an Optimal Rows + Columns cover of P will be less than the time to acquire data in the Detail Imaging step. In any case, our focus in this appendix is minimizing the data acquisition time, which we equate with patient discomfort; we mention that computation time is acceptable.

B.4 Empirical Evaluation

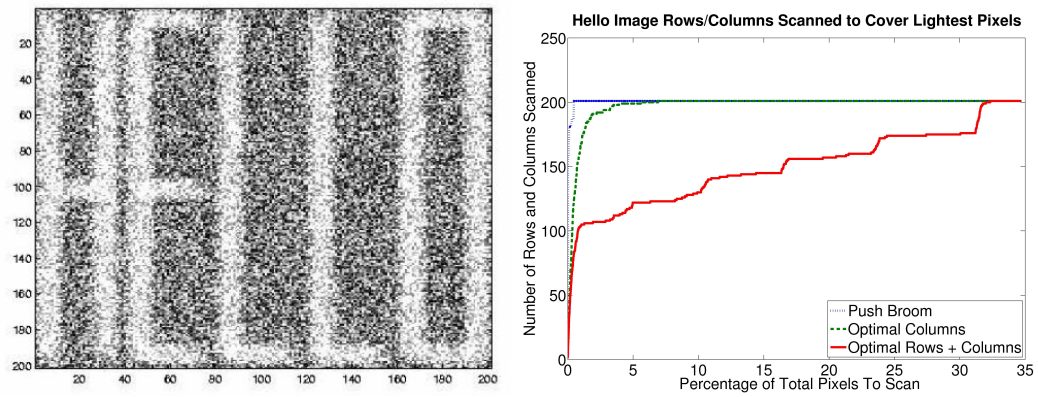

Figure B.2: Test Image Along with the Number of Rows+Columns Required to Cover Its Lightest Pixels

We compare the performance of Push Broom, Optimal Columns, and Optimal Rows + Columns on two test problems. For both test problems we assume that scanning any row and/or column is just as costly as scanning any other. All non-P scan graph edges are given a weight (cost) of one.

See Figure B.2 for the first test image and results. For our first test we let I be the noisy Figure B.2 "HELLO" image and let the set of interesting pixels, P, be the lightest p pixels in I. Note that this first test contains a variety of both horizontal and vertical bands of light (i.e., interesting) pixels. As a result we can see in Figure B.2's results graph that the Optimal Rows + Columns method requires substantially fewer columns and rows than the other two methods to cover the lightest $p \leq 30\%$ of I's pixels. Between the Optimal Columns and Push Broom methods we can see that the Optimal Columns method outperforms the Push Broom method for covering a very small (i.e., less than about 2%) number of the lightest pixels. However, both Push Broom and Optimal Columns are about the same cost for larger p.

See Figure B.3 for the second test image. In Figure B.3 our image I is a first-

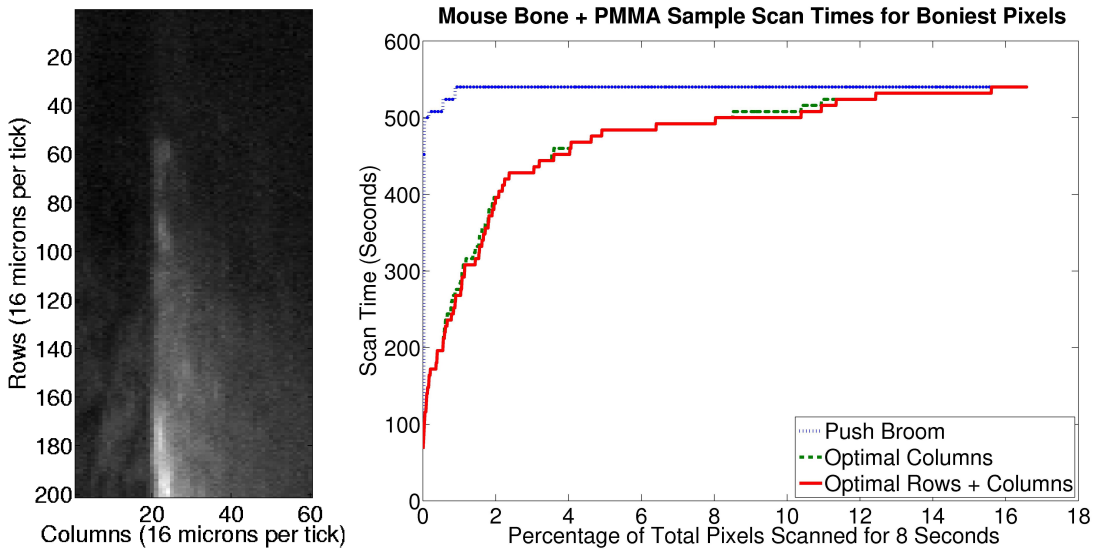

Figure B.3: Bone + PMMA Image, and The Total Time Required to Image Its Boniest (Lightest) Pixels

pass Raman image of test sample consisting of mouse bone embedded in PMMA plastic. Here the lighter pixels correspond to bone while darker pixels correspond to PMMA. Gray pixels indicate bone covered by a thin layer of PMMA. Here our pixels of interest, P, are the p boniest (lightest) pixels in I. Here we *assume* that choosing the p boniest pixels, for various p, according to the low-fidelity First-Pass image is a good way to do Detail Identification; properly addressing this question is beyond the scope of this appendix.

Figure B.3's first-pass bone + PMMA image, I, was produced by scanning each of the 60 image columns with a 1 second integration time. We would like, however, to scan each bony (interesting) pixel for 8 seconds. Hence, Figure B.3's result graph reports $60 + 8(\#$ columns/rows to cover $P)$ seconds for each method. There we can see that both the Optimal Columns and Optimal Columns + Rows methods outperform Push Broom for scanning the lightest p at most about 15% of I's pixels.

Finally, note that Figure B.3's first-pass bone + PMMA image, I, is biased toward

a strong Optimal Columns performance over the Optimal Columns + Rows method. Not only does each of I's columns cover more than three times as many pixels as each row, but all of I's boniest (i.e. lightest) features are aligned vertically. However, even for this very difficult test image, Optimal Rows + Columns still requires less scan time than Optimal Columns for most small $|P|$ values (i.e. less than $\approx 5\%$ pixels scanned).

B.5 Generalizations and Future Work

In the Optimal Rows + Columns method there is some flexibility with respect to the edge weights assigned in the scan graph. Although all P pixel edges should always be given a weight of ∞, the remaining edges from the s node and into the t node need not all have weight 1. In general the weight assigned to an edge (s, i) should correspond to the cost of scanning row i. Likewise, the weight assigned to an edge (j, t) should correspond to the cost of scanning column j. If, as above, all non-P edges are assigned the weight 1 it means that all rows and columns require the same unit of cost to scan. However, each non-P column/row scan graph edge can indeed be given any desired positive real cost. This leaves the user a good deal of flexibility in assigning row and column costs based on the first-pass image I. Brighter pixels require less integration time.

Angles other than 90° can be considered as well. If each pixel is in more than two possible frames (horizontal and vertical), we know of no efficient computation of an optimal cover. There are, however, fast approximate algorithms [31] for the set-cover problem, including a greedy algorithm, with an approximation ratio of $\ln(\max(m, n))$. An example is the greedy algorithm that repeatedly chooses a frame that covers the maximum number of as-yet-uncovered pixels, until all pixels are

covered. The number of frames selected by this algorithm is guaranteed to be at most $\ln(\max(m, n))$ times the optimal number of frames. There are implementations of this algorithm with runtime close to $O(k|P|)$, where k is the number of angles allowed. One can also use an Optimal Columns cover under the best possible rotation. Preliminary experiments on limited data were inconclusive. There is also inherent approximation involved in using data from one pass in order to predict the outcome of a second pass rotated by an angle that is not a multiple of 90°. In particular, if the pixels are square, pixels of one pass do not line up exactly with pixels of the second pass. We do not discuss that further here.

Jitter and hysteresis effects on the scanner realignments necessitated by the Optimal Columns and Optimal Rows + Columns methods should also be more thoroughly investigated. However, we don't expect these effects to be important. The spectrometer used to produce the Figure B.3 test image utilizes a mirror which can be positioned to better than 0.1 micron (small in comparison to Figure B.3's 16 micron length scale). Stages exist with similar precision. Furthermore, hysteresis effects can be mitigated by beginning detailed imaging behind the starting point and progressing with column/row scans in only one direction along each axis.

B.6 Conclusion

In this appendix we demonstrated that two proposed scanning methods, Optimal Columns and Optimal Columns + Rows, may be useful in decreasing the total integration time required to rescan a small set of interesting image pixels.

Bibliography

Bibliography

[1] *Available at* http://www-personal.umich.edu/∼markiwen/.

[2] *Available at* http://sdmc.i2r.a-star.edu.sg/rp/.

[3] Compressed sensing resources. http://www.dsp.ece.rice.edu/cs/.

[4] The dprep package. *http://cran.r-project.org/doc/packages/dprep.pdf*.

[5] l_1-magic website. http://www.acm.caltech.edu/l1magic/.

[6] R. Agrawal, T. Imielinski, and A. Swami. Mining associations between sets of items in large databases. *SIGMOD*, pages 207–216, 1993.

[7] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *VLDB*, pages 487–499, 1994.

[8] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. of Comput. and System Sci.*, 58:137–147, 1999.

[9] C. Anderson and M. D. Dahleh. Rapid computation of the discrete Fourier transform. *SIAM J. Sci. Comput.*, 17:913–919, 1996.

[10] R. Baraniuk, M. Davenport, R. DeVore, and M. Wakin. A simple proof of the restricted isometry property for random matrices. *Constructive Approximation*, to appear.

[11] M.-A. Belabbas and P. Wolfe. On sparse representations of linear operators and the approximation of matrix products. *Conference on Information Sciences and Systems (CISS)*, 2008.

[12] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. *Proc. Twentieth Annual ACM Symp. Theory Comput.*, pages 301–309, 1988.

[13] R. Berinde, A. C. Gilbert, P. Indyk, H. Karloff, and M. Strauss. Combining geometry and combinatorics: A unified approach to sparse signal recovery. *preprint*, 2008.

[14] L. I. Bluestein. A Linear Filtering Approach to the Computation of Discrete Fourier Transform. *IEEE Transactions on Audio and Electroacoustics*, 18:451–455, 1970.

[15] G. Box and M. Muller. A note on the generation of random normal deviates. *Ann. Math. Stat.*, 29:610–611, 1958.

[16] J. P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover Publications, Inc., 2001.

[17] L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001.

[18] E. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Inform. Theory*, 52:489–509, 2006.

[19] E. Candes, J. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):1207–1223, 2006.

[20] A. Carden, M. D. Morris, R. M. Rajachar, and D. H. Kohn. Ultrastructural changes accompanying the mechanical deformation of bone tissue: A raman imaging study. *Calcified Tissue International*, 72(2):166–175, 2003.

[21] V. Chandar. A negative result concerning explicit matrices with the restricted isometry property. *preprint*, 2008.

[22] C. Chang and C. Lin. Libsvm: a library for support vector machines, 2001.

[23] B. Chazelle. *The Discrepancy Method: Randomness and Complexity*. Brooks/Cole Publishing Company, 1992.

[24] H. Cohn, R. Kleinberg, B. Szegedy, and C. Umans. Group-theoretic algorithms for matrix multiplication. *preprint*.

[25] G. Cong, K. L. Tan, A. K. H. Tung, and X. Xu. Mining top-k covering rule groups for gene expression data. *SIGMOD*, 2005.

[26] G. Cong, A. K. H. Tung, X. Xu, F. Pan, and J. Yang. Farmer: Finding interesting rule groups in microarray datasets. *SIGMOD*, 2004.

[27] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, 19:297–301, 1965.

[28] D. Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, pages 467–471, 1982.

[29] D. Coppersmith. Rectangular matrix multiplication revisited. *J. Complexity*, pages 42–49, 1997.

[30] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.*, pages 251–280, 1990.

[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithms. *2nd Edition*, 2001.

[32] G. Cormode and S. Muthukrishnan. Combinatorial Algorithms for Compressed Sensing. *Technical Report DIMACS TR 2005-40*, 2005.

[33] G. Cormode and S. Muthukrishnan. Combinatorial Algorithms for Compressed Sensing. *Conference on Information Sciences and Systems*, March 2006.

[34] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[35] I. Daubechies, O. Runborg, and J. Zou. A sparse spectral method for homogenization multiscale problems. *Multiscale Model. Sim.*, 2007.

[36] R. A. DeVore. Deterministic constructions of compressed sensing matrices. *http://www.ima.umn.edu/2006-2007/ND6.4-15.07/activities/DeVore-Ronald/Henrykfinal.pdf*, 2007.

[37] G. Dong and J. Li. Efficient mining of emerging patterns: discovering trends and differences. *KDD*, pages 43–52, 1999.

[38] G. Dong, X. Zhang, L. Wong, and J. Li. Caep: Classification by aggregating emerging patterns. *Proc. 2nd Int. Conf. Discovery Science (DS)*, 1999.

[39] D. Donoho. Compressed Sensing. *IEEE Trans. on Information Theory*, 52:1289–1306, 2006.

[40] D. Donoho and X. Huo. Uncertainty principles and ideal atomic decomposition. *IEEE Transactions on Information Theory*, 47:2845–2862, 2001.

[41] D. Donoho and P. Stark. Uncertainty principles and signal recovery. *SIAM J. Appl. Math.*, 49:906–931, 1989.

[42] D. L. Donoho and J. Tanner. Thresholds for the recovery of sparse solutions via l1 minimization. In *40th Annual Conference on Information Sciences and Systems (CISS)*, 2006.

[43] P. Drineas, R. Kannan, and M. W. Mahoney. Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM J. Comp.*, 2006.

[44] D. Z. Du and F. K. Hwang. *Combinatorial Group Testing and Its Applications*. World Scientific, 1993.

[45] A. Dutt and V. Rokhlin. Fast Fourier transforms for nonequispaced data. *SIAM J. Sci. Comput.*, 14:1368–1383, 1993.

[46] D. Eppstein, M. T. Goodrich, and D. S. Hirschberg. Improved combinatorial group testing algorithms for real-world problem sizes, May 2005.

[47] J. A. Fessler and B. P. Sutton. Nonuniform Fast fourier transforms using min-max interpolation. *IEEE Trans. Signal Proc.*, 51:560–574, 2003.

[48] P. Flajolet and G. Martin. Probabilistic counting algorithms for data base applications. *J. of Comput. and System Sci.*, 31:182–209, 1985.

[49] G. B. Folland. *Fourier Analysis and Its Applications*. Brooks/Cole Publishing Company, 1992.

[50] M. Frigo and S. Johnson. The design and implementation of fftw3. *Proceedings of IEEE 93 (2)*, pages 216–231, 2005.

[51] S. Ganguly and A. Majumder. CR-precis: A deterministic summary structure for update data streams. *ArXiv Computer Science e-prints*, Sept. 2006.

[52] C. F. Gerald and P. O. Wheatley. *Applied Numerical Analysis*. Addison-Wesley Publishing Company, 1994.

[53] A. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss. Near-optimal sparse Fourier estimation via sampling. *ACM STOC*, pages 152–161, 2002.

[54] A. Gilbert, S. Muthukrishnan, and M. Strauss. Improved time bounds for near-optimal sparse Fourier representations. *SPIE*, 2005.

[55] A. C. Gilbert and M. J. Strauss. Group testing in statistical signal recovery. *submitted*, 2006.

[56] A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin. Algorithmic linear dimension reduction in the l_1 norm for sparse vectors. *submitted*, 2006.

[57] K. Golcuk, G. S. Mandair, A. F. Callender, N. Sahar, D. H. Kohn, and M. D. Morris. Is photobleaching necessary for raman imaging of bone tissue using a green laser? *Biochimica et Biophysica Acta*, 1758(7):868–873, 2006.

[58] N. B. Haaser and J. A. Sullivan. *Real analysis*. Dover Publications, Inc., 1991.

[59] S.-Y. Ho, C.-H. Hsieh, H.-M. Chen, and H.-L. Huang. Interpretable gene expression classifier with an accurate and compact fuzzy rule base for microarray data analysis. *Biosystems*, 85:165–176, 2006.

[60] P. Indyk. Explicit constructions of selectors and related combinatorial structures, with applications. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 697–704, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[61] P. Indyk. Explicit constructions for compressed sensing of sparse signals. In *Proc. of ACM-SIAM symposium on Discrete algorithms (SODA'08)*, 2008.

[62] P. Indyk. Personal correspondence, 2008.

[63] P. Indyk and M. Ruzic. Near-optimal sparse recovery in the l_1 norm. *preprint*, 2008.

[64] M. A. Iwen. Unpublished Results. http://www-personal.umich.edu/ markiwen/.

[65] M. A. Iwen. A deterministic sub-linear time sparse fourier algorithm via non-adaptive compressed sensing methods. In *Proc. of ACM-SIAM symposium on Discrete algorithms (SODA'08)*, 2008.

[66] M. A. Iwen, A. C. Gilbert, and M. J. Strauss. Empirical evaluation of a sub-linear time sparse DFT algorithm. *Communications in Mathematical Sciences*, 5(4), 2007.

[67] M. A. Iwen, W. Lang, and J. Patel. Scalable rule-based gene expression data classification. In *IEEE International Conference on Data Engineering (ICDE'08)*, 2008.

[68] M. A. Iwen, G. S. Mandair, M. D. Morris, and M. Strauss. Fast line-based imaging of small sample features. In *IEEE International Conference on Acoustics, Speech, and Signal Processing(ICASSP)*, April 2007.

[69] M. A. Iwen and C. V. Spencer. Improved bounds for a deterministic sublinear-time sparse fourier algorithm. In *Conference on Information Sciences and Systems (CISS)*, 2008.

[70] W. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Conf. in Modern Analysis and Probability*, pages 189–206, 1984.

[71] E. Kaltofen and L. Yagati. Improved sparse multivariate polynomial interpolation algorithms. *International Symposium on Symbolic and Algebraic Computation*, 1988.

[72] S. Kirolos, J. Laska, M. Wakin, M. Duarte, D. Baron, T. Ragheb, Y. Massoud, and R. Baraniuk. Analog-to-information conversion via random demodulation. *Proc. IEEE Dallas Circuits and Systems Conference*, 2006.

[73] R. Kress. *Numerical Analysis*. Springer-Verlag, 1998.

[74] S. Kunis and H. Rauhut. Random Sampling of Sparse Trigonometric Polynomials II - Orthogonal Matching Pursuit versus Basis Pursuit. *Foundations of Computational Mathematics*, to appear.

[75] J. Lafferty and L. Wasserman. Rodeo: Sparse nonparametric regression in high dimensions. *preprint*, 2008.

[76] J. M. Landsberg. Geometry and the complexity of matrix multiplication. *Bulletin of the American Mathematical Society*, 45(2), April 2008.

[77] J. Laska, S. Kirolos, Y. Massoud, R. Baraniuk, A. Gilbert, M. Iwen, and M. Strauss. Random sampling for analog-to-information conversion of wideband signals. *Proc. IEEE Dallas Circuits and Systems Conference*, 2006.

[78] J.-Y. Lee and L. Greengard. The type 3 nonuniform FFT and its applications. *J Comput. Phys.*, 206:1–5, 2005.

[79] J. Li and L. Wong. Identifying good diagnostic genes or gene groups from gene expression data by using the concept of emerging patterns. *Bioinformatics*, 18:725–734, 2002.

[80] J. Li, X. Zhang, G. Dong, K. Ramamohanarao, and Q. Sun. Efficient mining of high confidence association rules without support thresholds. *Principles of Data Mining and Knowledge Discovery (PKDD)*, pages 406 – 411, 1999.

[81] W. Li, J. Han, and J. Pei. Cmar: Accurate and efficient classification based on multiple class-association rules. *ICDM*, 2001.

[82] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. *KDD*, 1998.

[83] M. Lustig, D. Donoho, and J. Pauly. Sparse MRI: The application of compressed sensing for rapid MR imaging. *Submitted for publication*, 2007.

[84] R. Maleh, A. C. Gilbert, and M. J. Strauss. Signal recovery from partial information via orthogonal matching pursuit. *IEEE Int. Conf. on Image Processing*, 2007.

[85] S. Mallet. *A Wavelet Tour of Signal Processing*. China Machine Press, 2003.

[86] Y. Mansour. Learning boolean functions via the fourier transform. *Theoretical Advances in Neural Computation and Learning*, pages 391–424, 1994.

[87] Y. Mansour. Randomized approxmation and interpolation of sparse polynomials. *SIAM Journal on Computing*, 24:2, 1995.

[88] T. McIntosh and S. Chawla. On discovery of maximal confident rules without support pruning in microarray data. *SIGKDD Workshop on Data Mining in Bioinformatics (BIOKDD)*, 2005.

[89] M. D. Morris, W. F. Finney, and R. M. R. et al. Bone tissue ultrastructural response to elastic deformation probed by raman spectroscopy. *Faraday Discussions*, 126:159–168, 2004.

[90] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[91] S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1, 2005.

[92] S. Muthukrishnan. Some Algorithmic Problems and Results in Compressed Sensing. *Allerton Conference*, 2006.

[93] D. Needell and J. A. Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *preprint*, 2008.

[94] D. Needell and R. Vershynin. Signal recovery from incomplete and inaccurate measurements via regularized orthogonal matching pursuit. *preprint*, 2007.

[95] D. Needell and R. Vershynin. Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit. *submitted*, 2007.

[96] J. R. Quinlan. Bagging, boosting, and c4.5. *AAAI*, 1:725–730, 1996.

[97] L. Rabiner, R. Schafer, and C. Rader. The Chirp z-Transform Algorithm. *IEEE Transactions on Audio and Electroacoustics*, AU-17(2):86–92, June 1969.

[98] F. Rioult, J. F. Boulicaut, B. Cremilleux, and J. Besson. Using transposition for pattern discovery from microarray data. *DMKD*, pages 73–79, 2003.

[99] M. Rudelson and R. Vershynin. Sparse reconstruction by convex relaxation: Fourier and gaussian measurements. In *40th Annual Conference on Information Sciences and Systems (CISS)*, 2006.

[100] R. Salem and D. C. Spencer. On sets of integers which contain no three terms in arithmetical progression. *Proc. Nat. Acad. Sci.*, pages 561–563, 1942.

[101] S. Sudarsky. Fuzzy satisfiability. *Intl. Conf. on Industrial Fuzzy Control and Intelligent Systems (IFIS)*, 1993.

[102] C. P. Tarnowski, M. Ignelzi, and W. W. et al. Earliest mineral and matrix changes in force-induced musculoskeletal disease as revealed by raman microspectroscopic imaging. *Journal of Bone and Mineral Research*, 19(1):64–71, 2004.

[103] L. N. Trefethen. *Spectral methods in MatLab*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[104] J. Tropp and A. Gilbert. Signal recovery from partial information via orthogonal matching pursuit. *Submitted for Publication*, 2005.

[105] S. Vinterbo, E. Kim, and L. Ohno-Machado. Small, fuzzy and interpretable gene expression based classifiers. *Bioinformatics*, 21:1964–1970, 2005.

[106] D. Wagner. Efficient algorithms and intractable problems, April 2003. UC Berkeley CS 170 Handout 20.

[107] J. Wang, J. Han, and J. Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. *KDD*, 2003.

[108] M. Zaki and C. Hsiao. Charm: An efficient algorithm for closed association rule mining. *Proc. of the 2nd SIAM Int. Conf. on Data Mining (SDM)*, 2002.