FAST ALGORITHMS FOR SELECTION AND MULTIDIMENSIONAL SPARSE FOURIER TRANSFORM

by

André Rauh

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical and Computer Engineering

Fall 2015

FAST ALGORITHMS FOR SELECTION AND

MULTIDIMENSIONAL SPARSE FOURIER TRANSFORM

by

André Rauh

Approved: _____

Kenneth E. Barner, Ph.D.
Chair of the Department of Electrical and Computer Engineering

Approved: _____

Babatunde Ogunnaike, Ph.D.
Dean of the College of Engineering

Approved: _____

Ann L. Ardis, Ph.D.
Interim Vice Provost for Graduate and Professional Education

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Gonzalo R. Arce, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Javier Garcia-Frias, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Xiang-Gen Xia, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Jose Luis Paredes, Ph.D.
Member of dissertation committee

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor, Dr. Gonzalo Arce for giving me the opportunity to pursue this long and rewarding journey, and for his help and guidance. I would also like to thank my friends in Newark who have supported me throughout the years. I am especially grateful for the friendships I have in my home country in Germany which have continuously reached out to me even though I sometimes fell off the face off the earth. In particular I would like to mention my friend Sandra who has helped me when I most needed it. And my dear friend Violet who has enriched my life with so much with just being there.

Most of all I would like to thank my family in Germany who has supported me throughout the years and always offered help if I needed some.

Without the continuous support of all of these people this endeavour would have been a much tougher one, and for this I would like to say: Thank You!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ABSTRACT

In recent years, many applications throughout engineering, science, and finance have been faced with the analysis of massive amounts of data. This dissertation focuses on two algorithms ubiquitous in data analysis, with the goal of making them efficient in "big data" scenarios.

First, the selection problem also known as finding the order statistic of data is addressed. In particular, a fast weighted median (WM) algorithm, which is a more general problem than selection, is derived. The new algorithm computes the WM of N samples which has linear time and space complexity as opposed to $O(N \log N)$ which is the time complexity of traditional sorting algorithms. A popular selection algorithm often used to find the WM in large data sets is Quickselect whose performance is highly dependent on how the pivots are chosen. The new algorithm introduces an optimization based pivot selection strategy which results in significantly improved performance as well as a more consistent runtime compared to traditional approaches. In particular, the selected pivots are order statistics of subsets of the input data. In order to find the optimal order statistics as well as the optimal subset sizes, a set of cost functions are derived, which when minimized lead to optimal design parameters. The complexity is compared to Floyd and Rivest's algorithm SELECT which to date has been the fastest median algorithm and it is shown that the proposed algorithm requires 30% fewer comparisons. It is also shown that the proposed selection algorithm is asymptotically optimal for large N.

The second algorithm developed in this dissertation extends the concepts of the Sparse Fast Fourier Transform (sFFT) Algorithm introduced in 2012, to work with multidimensional input data. The multidimensional algorithm requires several generalizations to multiple key concepts of the 1D sparse Fourier transform algorithm.

It is shown that the permutation parameter is of key importance and should not be chosen randomly but instead can be optimized for the reconstruction of sparse real world data.

A connection is made between key steps of the algorithm and lattice theory, thus establishing a rigorous understanding of the effect of the permutation parameter on the algorithm performance. Lattice theory is then used to optimize the set of parameters to achieve a more robust and better performing algorithm.

The result improves the algorithm without penalty on sampling complexity. In fact other algorithms which use pseudorandom spectrum permutation can also benefit from this finding. This dissertation addresses the case of the exact k-sparse Fourier transform but the underlying concepts can be applied to the general case of finding a k-sparse approximation of the Fourier transform of an arbitrary signal.

Simulations illustrate the efficiency and accuracy of the proposed algorithm. The optimizations of the parameters and the improvements therewith are shown in simulations in such that the worst case and average case PSNR improves by several dB.

Chapter 1

INTRODUCTION

1.1 Motivation

One particularly important area of Electrical and Computer Engineering is signal processing. Both, the word *signal* as well as *processing* are very broad terms. A signal can come in many forms and can be interpreted in many different ways. The most prevalent signals are analog and digital signals. In recent decades, the so called digital age has taken over our lives with emerging technologies in communications, computing and data analysis. This technology often uses microprocessors such as CPUs, GPUs or FPGAs which execute instructions on discrete digital data. This discrete data are simply bits and bytes to the computer, but often it is interpreted as real world representations of analog signals. One example is a digital image which is represented as discrete pixels in the digital world and corresponds to photons in the analog world. Another popular example is an audio signal which is a sampled time varying signal in the digital world and corresponds to a continuous wave form in the analog world.

Frequently, digital data is not kept in its raw form but processed in some sense. Often this processing can be necessary or helpful to understand or improve the raw signal. One commonly known example is to generate a lower resolution thumbnail of a too large, high resolution image. Another example is compression algorithms such as the MP3 codec for audio and the JPEG file format for images. Both algorithms transform the digital data into another domain and perform data compression algorithms to discard information unimportant to the human eye or ear. In particular, MP3 uses the well-known Fourier transform, a method taught to most engineers during their undergraduate studies. The Fourier transform has many more application such as

correlation analysis, spectrum analysis, communication [CM13], image processing and MRI [CP54].

One very common method of data analysis is to calculate various statistics. The most popular statistic is the mean which is also known as the average which is simply the sum of the samples divided by the number of samples. Another statistic is the median which divides the ordered data set into two equally large sets. The median is often used when the input data contains outliers since an outlier skews the mean but not the median.

Extending the mean of an input sequence of samples, to the more general case of dealing with samples of different variances and the well-known linear filter is derived where the variances correspond to the reciprocal values of the weights of the filter. Furthermore, allowing negative weights, popular digital filters such as low pass, band pass and high pass can be derived. Applying the same concepts to the median the weighted median filters are derived. In recent years weighted median filters have become increasingly popular and are applied to compressive sensing [PA11,CRT06], audio processing [EF13], mechatronics [TLP+13] and have recently been ported to quantum computing [YMCX13].

This dissertation first focuses on an algorithm to find the weighted median. One disadvantage of the weighted median is that compared to a linear filter the computational cost is considerably higher. In the one dimensional case a linear filter exists of a sequence of filter coefficients. This sequence is then used to compute the inner product with the input signal in order to compute the output. In the weighted median case, however, the computation is much more complex. A naive implementation has to sort the input samples and computes the cumulative sum of the resulting array. This leads to the motivation to find a more performant algorithm to determine the weighted median. It turns out that the well-known recursive algorithm Quickselect can be modified to determine the weighted median.

In particular, one of this dissertation's focus is on the runtime complexity of algorithms. The runtime complexity of an algorithm is usually defined as the behavior

of the runtime on the input size. For instance, given an input vector of length N it is of interest to the user of an algorithm how the runtime changes if the input size is –for instance– $2N$. To answer this we introduce the so called *big O notation* also sometimes referred to as *Landau notation* [CLR+01] which describes the asymptotic behavior of functions:

$$f(x) = O(g(x)) \quad \text{as } x \to \infty$$

if and only if there exists constants C_1 and C_2 such that

$$|f(x)| \leq C_1|g(x)| \quad \text{for all } x > C_2 \tag{1.1}$$

Intuitively, this means that the function f does not grow faster than g. If the focus is runtime complexity then this definition allows us to compare different algorithms and judge them by how fast they solve a problem. For instance, a linear runtime complexity is said to run in time $O(N)$ and a exponential algorithm to run in time $O(2^N)$. Note that the notation is not limited to runtime complexity and can also be used to make statements about the sampling complexity or the memory requirements of an algorithm which may be of interest for other uses of an algorithm.

Quickselect is the focus for part of this dissertation. It has linear time complexity ($O(N)$) as opposed to $O(N \log N)$ of sorting. However, the performance and runtime of the algorithm highly depend on the choice of the pivots which are elements of the input signal that are used to partition the data. This dissertation's main goal is to derive, proof and demonstrate optimal parameters for the Quickselect algorithm such that the runtime is minimized. It should also be noted that this dissertation will treat the Median and Weighted Median problem synonymous sometimes, as both are found using the same algorithm. Further, the optimal parameters which are derived can be used with any version of the Quickselect algorithm, i.e. to find the order statistic or (weighted) median of an input set.

The second major focus of this dissertation is the Fourier transform and in particular algorithms which compute the discrete Fourier transform. For signal processing

the most common algorithm is the discrete Fourier transform which takes a discrete, finite input signal and produces a discrete finite output sequence — the spectrum. Similar to the weighted median problem, the Fourier transform has a naive algorithm: Generate a Fourier matrix of size $N \times N$ and multiply it by the input vector. This approach results in an $O(N^2)$ algorithm since a matrix vector multiplication has to perform $N \times N$ element wise multiplications. A much improved algorithm is the famous Fast Fourier Transform (FFT) method which uses properties of complex numbers to achieve an $O(N \log N)$ runtime complexity.

The term *sparse* in the context of signal processing means that the signal is populated with primarily zeros and only few non-zero samples. In many applications the signal that is processed is sparse in some other domain. For instance a recording of a music song is sparse in the Fourier domain since at a given interval only few frequencies are present that generate the output signal. Similarly, an image is sparse in the Wavelet domain. The sparsity depends on the basis functions that a transform utilizes and how well these basis functions match the real world signal. Intuitively, this allows to represent the signal by applying weights to these basis functions and adding the resulting functions together. If we omit some of these basis functions due to making little impact into the overall representation, then we lose only very little detail of the overall information. A signal is said to be compressible if we can omit many of these terms in some domain.

These ideas are the basis of file compression formats such as MP3 or JPEG 2000. One might wonder, if given a signal which has only few non-zero coefficients in the Fourier domain: Is it possible to exploit this sparsity in order to find an even faster performing algorithm to compute the Fourier transform? As it turns out, the answer to this question is yes. In recent years there has been considerate research about finding faster algorithms for computing the sparse Fourier coefficients of an input signal. Often these new algorithms are referred to as "sparse Fourier transform" or since dealing with discrete input data "sparse FFT". The most notable of which was published in 2012 by a group at M.I.T. [HIKP12a]. The achieved result was an average runtime complexity of

$O(k \log N)$ where k is the sparsity of the signal in the Fourier domain. This dissertation will use the one dimensional algorithm and extend it to work for d-dimensional data which requires extensions to multiple key concepts. Similarly to the weighted median problem, the algorithm is highly dependent on its parameters. Again, the parameters are to be optimized for real world signals which have not been studied by the academic community. Prior research has assumed purely random distribution of the Fourier coefficients of the input signals which fails to perform well for many real world input data. This dissertation proposed methods to find optimal parameters for structured signals.

1.2 Dissertation Overview

Both algorithms treated in this thesis –the sparse FFT and the weighted median– have one thing in common: The performance greatly depends on the parameters of each algorithm. For instance, the Quickselect algorithm takes the median of a subset as the pivot in each iteration. The overall runtime can vary greatly depending on the input but also depending on the subset size that the algorithm chooses. In the sparse FFT algorithm the performance is heavily dependent on the number of collisions after the input is permuted. The collisions in turn depend on the input spectrum and the permutation applied to it. Thus, the permutation –which is chosen by the algorithm– is the most important parameter in the performance of the algorithm.

In many academic findings, such as the sparse FFT algorithm of [HIKP12a] the theoretical analysis requires some assumptions in order to make the analysis mathematical tractable. For instance, the input distribution of the Fourier coefficients are assumed to be randomly distributed. If this assumption holds true, then it is easy to choose the parameters –such as the permutation– to be random as well. This turns out to work well for randomly generated signals and makes the analysis simple due to the simple properties of random variables.

However, if one were to use the proposed algorithm for real world signals, an important question needs to be asked: Do these assumptions actually hold for real

world data? The answer to this question is usually no. This means, these algorithms often require more attention with regards to their most important parameters in order to perform well enough to be useful for real world applications.

Similarly, the median finding algorithm is usually not applied to randomly generated data but often to real world data which follow a certain distribution. Again, this fact can be used to further improve upon a strategy that assumes uniformly distributed input data which is often employed by theoretical research findings.

This dissertation investigates both algorithms and spends significant effort in seemingly small details of these algorithms. As it turns out, this approach pays off and provides the following benefits to the research community as well as real world implementations of these algorithms:

1. Develop a rigorous *understanding* of the parameters in question. This helps to guide application developers to implement these algorithms which in turn helps adoption of these algorithms in the real world. It also allows other researchers to use the new findings to further advance the field by building upon the newly proposed methods.

2. Improve the *performance* of the algorithms by using the findings and the newly developed theoretical models. Here the word "performance" might stand for runtime complexity or a quality measure of the algorithm (such as reconstruction PSNR).

3. Improve the *robustness* of the algorithms by avoiding the worst cases which can happen when using real world data with the wrong theoretical model. This is crucial for real world applications which need a predictable performing algorithm. An algorithm that only sometimes performs well enough is not acceptable if used in real world products. The theoretical *understandings* allows us to make these algorithms more *robust* and thus become accessible to –for instance– the industry.

In this dissertation, the above points are addressed for both algorithms. A rigorous theoretical model is developed for both and each of the above items is demonstrated in detail.

Note that, the big-O notation (1.1) introduced earlier does not consider the exact runtime complexity of an algorithm. That is, the runtime is only considered as the asymptotic behavior of the function which is due to the constant C_1 in (1.1). In real world implementations however, the exact runtime matters just as much as the asymptotic complexity.

For instance, two algorithm might both have the runtime complexity $O(N \log N)$ but can differ quite pronounced in the actual runtime due to different constants in their actual runtime. That is, the number of operations might be $2N \log N$ for one algorithm whereas the other algorithm requires $15N \log N$ operations. Clearly, for real world usage of these algorithms more careful investigations are helpful.

Often enough, a theoretical result of the exact number of operations an algorithm performs until termination is prohibitively complex to calculate. Thus, the focus is often only on the big-O notation when in fact a real world usage requires a more nuanced understanding. This thesis investigates not only theoretical asymptotic complexity but also aims to provide a better understanding for the real world runtime.

1.3 Organization of the Dissertation

The organization of this thesis is as follows:

Chapter 2 deals with the Fast Weighted Median search algorithm. First, the problem formulation as well as the motivation for the problem is introduced. Subsequently, the chapter introduces the well known Quickselect algorithm and states the problem of the importance of the pivots on the runtime performance. A theoretical framework is introduced in order to model the problem mathematically. This model is then used to find the optimal parameters for the algorithm. In particular, the pivots used for each iteration of the algorithm are chosen very carefully with the provided model. The important subset size of the pivot selection process is also solved with this

model. A novel closed form solution to this problem is the result. This novel algorithm beats existing algorithms –the first time in over four decades. To show real world performance, the newly proposed algorithm's runtime is evaluated with simulations. Both, the number of comparisons –the main measure of complexity for selection type algorithms– as well as the actual runtimes are compared to the existing algorithms. To this end a low level implementation of the proposed algorithm in the C-programming language is used to do the comparison to existing algorithms. The chapter finishes with a summary and conclusions.

Chapter 3 of this dissertation closely examines the sparse FFT algorithm introduced in [HIKP12a]. First, the Fourier transform is introduced and the motivation behind finding an algorithm dealing with sparse input data is discussed. The parameters which have the most impact on the algorithmic performance are inspected in detail. In order to find a theoretical model for the most important parameter –the permutation matrix– the topic of lattices is introduced. Consecutively the dual lattice is shown to be an especially important part. Based on this novel connection an iterative algorithm is introduced to find an optimal permutation parameter. In particular, the proposed algorithm generates a sequence of potential permutation matrix candidates which are then evaluated in the multidimensional sparse FFT. Note that, this novel connection to Lattice theory can open new directions of research.

Preliminary simulations show the effectiveness of this algorithm by showing that the mean minimum inter-point distance increases. The chapter then continues with introducing the reader to the one dimensional sparse FFT algorithm by focusing on a high level overview of the concepts. This dissertation proceeds with extending the algorithm to multiple dimensions by carefully considering each part of the algorithm. Further, a rigorous solution to some of the sub problems within the sparse FFT are presented. One particular requirement for the sparse FFT algorithm is the permutation matrix to have an odd determinant. To this end, a novel algorithm is introduced which takes a square integer matrix and turns it into a similar matrix with an odd determinant. The algorithm works by only flipping the least significant bit of as few

entries of the permutation matrix as possible. A short complexity analysis of the proposed algorithm as well as an error analysis is presented. Thus, overall this chapter introduces three novel algorithms which are subsequently combined in order to achieve a consistently performing multidimensional sparse FFT. Again, a complexity analysis is given for the proposed multidimensional sparse FFT. The proposed algorithms are evaluated by implementing them in MATLAB and running simulations that demonstrate their working. The results are an improvement in the robustness and in the performance of the sparse FFT. A conclusion with an outlook is given in the final section of the chapter.

Chapter 4 concludes the dissertation in which the findings are summarized. Several future directions of this research are discussed.

The appendices A and B provide further details such as proofs which may be helpful to some readers.

Chapter 2

FAST WEIGHTED MEDIAN SEARCH

2.1 Introduction

Weighted medians (WM), introduced by Edgemore over two hundred years ago in the context of least absolute regression, have been extensively studied in signal processing over the last two decades [Arc05, Hoa61, MR02, AK97]. WM filters have been particularly useful in image processing applications as they are effective in preserving edges and signal discontinuities, and are efficient in the attenuation of impulsive noise properties not shared by traditional linear filters [Arc05, AK97].

The properties of the WM are inherited from the sample median — a robust estimate of location. An indicator of an estimator's robustness is its breakdown point, defined as the smallest fraction of the observations which when replaced with outliers will corrupt the estimate outside of reasonable bounds. The breakdown point of the sample mean, for instance, is 1 indicating that a single outlier present in the data can have a detrimental effect in the estimate. The median, on the other hand, has a breakdown point of $0.5N$ meaning that half or more of the data needs to be corrupted before the median estimate is deteriorated significantly [Mal80].

This is the main motivation to perform median filtering. Assuming that the data is contaminated with noise and outlying observations, the goal is to remove the noise while retaining most of the behavior present in the original data. The median filter does just that and does not introduce values that are not present in the original data. Significant efforts have been devoted to the understanding of the theoretical properties of WM filters [GW81, Arc05, Arc02, YYGN96, AM87, AG82, YHAN91, AP00, MA87, FAB98], their applications [AF89, Arc91, AG83, BA94, MA87, FPA02] and their

optimization. A number of generalizations aimed at improving the performance of WM filters have recently been introduced in the literature [AB07, KA98, GA01, KA00, PA99, KA99]. WM filters to date enjoy a rich theory for their design and application.

A limiting factor in the implementation of WM filters, however, is their computational cost. The most naive approach to computing the median, or any k^{th} order statistic, is to sort the data and then select the k^{th} smallest value. Once the data is sorted finding any order statistic is straightforward. Sorting the data leads to a computational time complexity of $O(N \log N)$ and since the traversing of the sorted array to find the WM is a linear operation, the cost of this approach is the cost of sorting the data. Several approaches to alleviate the computational cost have been proposed [HYT79, PH07].

In many signal processing applications the filtering is performed by a running window and the computation of a median filter benefits from the fact that most values in the running window do not change when the window is translated. In such case, a local histogram can be utilized to compute a running median since the median computation takes into account only the element values and not their location in the sliding window. Simply maintaining a running histogram at each location of the sliding windows enables the computation of the median [HYT79, PH07]. In a running histogram, the median information is indeed present since pixels are sorted out into buckets of increasing pixel values. Removing pixels from buckets and adding more is a simple operation, making it easier to keep a running histogram and updating it than to go from scratch for every move of the running window. The same idea can be used to build up a tree containing pixel values and the number of occurrences, or intervals and number of pixels. One can thus see the immediate benefit of retaining this information at each step [HYT79, AS87, PH07].

Other approaches to reduce the computation of running medians include separable approximations where 2D processing is attained by a 1D median filtering in two stages: the first along the horizontal direction followed by a second in the vertical direction [Nar09, AM87, MA87]. All of the above mentioned techniques focus on the

median computation of small kernels — a set of samples inside running windows. Depending on the signal's sampling resolution, the sample set may range from a small set of four or nine samples to larger windows that span a few hundred samples. However, emerging applications in signal processing are beginning to demand weighted median computation of much larger sample sets. In particular, weighted median computations are not simply needed to process data in running windows. WM are often needed for the solution of optimization problems where absolute deviations are used as distance metrics. While L_2 norms, based on square distances, have been used extensively in signal processing optimization, the L_1 norm has attracted considerable attention recently because of its attributes when used in regression. Firstly, the L_1 norm is more robust to noise, missing data, and outliers, than the L_2 norm [RL87,BLA79,LA04,BS80]. The L_2 norm sums the square of the residuals and thus places small weight on small residuals and strong weight on large residuals. The L_1 norm penalty, on the other hand, puts more weight on small residuals and does not weight as heavily large residuals. The end result is that the L_1 norm is more robust than the L_2 norm in the presence of outliers or large measurement errors.

Secondly, the L_1 norm has also been used as a sparsity-promoting norm in the sparse signal recovery problem, where the goal is to recover a high-dimensional sparse vector from its lower-dimensional sketch [CWB08]. In fact, the use of the L_1 norm for solving data fitting problems and sparse recovery problems traces back many years. In 1973, Claerbout et al. [CM73] proposed the use of the L_1 norm for robust modeling of Geophysical data. Later, in 1989, Donoho and Stark [DS89] used L_1 minimization for the recovery of a sparse wide-band signal from narrow-band measurements. Over the last decade, a wide use of the L_1 norm for robust modeling and sparse recovery began to appear. It turns out, that it is often the case that WMs are required to solve optimization problems when L_1 norms are used in the data fitting model.

For instance, the algorithm in [PA11] uses weighted medians on randomly projected compressed measurement to reconstruct the original sparse signal. For this application the input sizes on which a WM is performed are the size of the signals

which range from several thousands up to several million data points. In the examples described in Section 2.6, for instance, typical sample sets can approach millions of sample points. The computation of weighted medians for such very large kernels becomes critical and thus fast algorithms are needed. The data structures are no longer running windows and rough approximations are inadequate in optimization algorithms. To this end, fast and accurate WM algorithms are sought.

This chapter of the dissertation proposes a new algorithm which solves the problem of finding the WM of a set of samples. The algorithm is based on Quickselect which is similar to the well-known Quicksort algorithm. Even though the algorithm is explained and implemented for the WM problem it is straight forward to use similar concepts to construct a novel selection algorithm to find the order statistics of a set. Note that the median is a special case of an order statistic. In many applications of data processing it is crucial to calculate statistics about the data. Popular choices are quantiles such as quartiles or 2-quantile (for instance in finance time series) both of which reduce to a selection problem. Often, these consist of thousands or millions of samples for which a fast algorithm is of importance in order to allow quick data analysis.

Definition 1 *Let* $\{X_i\}_{i=1}^N$ *be a set of* N *samples and let* $\{W_i\}_{i=1}^N$ *be their associated weights. Now rearrange the samples in the form*

$$X_{(1)} \leq \cdots \leq X_{(N)}$$

then $X_{(k)}$ *is referred to as the* k^{th} *order statistic. Further denote* $W_{[k]}$ *as the associated weight of* $X_{(k)}$.

Moreover, W_0 is needed as a threshold parameter and is formally defined as

$$W_0 = \frac{1}{2}\sum_{i=1}^{N} W_i$$

Without loss of generality it is assumed that all weights are positive. All results can be extended to allow negative weights by coupling the sign of the weight to the corresponding sample and use the absolute value of the weight [Arc02].

The problem of estimating a constant parameter β under additive noise given N observations $\{X_i\}_{i=1}^N$ can be solved by minimizing a cost function under different error criteria:

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{N} f_e(X_i, \beta)$$

where f_e is a function that calculates the error between its arguments. The well-known sample average can be derived by choosing the L_2 error norm for the function f_e. Extending the idea by incorporating weights assigned to each sample into the equation results into the familiar weighted mean. In turn, the sample median follows from minimizing the error under the L_1 error norm. Conversely allowing the input samples to have different weights leads to the cost function of the weighted median:

$$T_{WM}(\beta) = \sum_{i=1}^{N} W_i \left| X_i - \beta \right| \tag{2.1}$$

where the weights satisfy $W_i > 0$. The WM $\hat{\beta}$ can be defined as the value of β in (2.1) which minimizes the cost function $T_{WM}(\beta)$. That is:

$$\hat{\beta} = \arg\min_{\beta} T_{WM}(\beta).$$

Figure 2.1 on page 15 depicts an example cost function $T_{WM}(\beta)$ for $N = 6$. It can be seen in the figure that $T_{WM}(\beta)$ is a piecewise linear continuous function. Furthermore, it is a convex function and attains its minimum at the sample median which is one of the input samples X_i. Figure 2.1 on page 15 also depicts the semi-derivative of the cost function T_{WM} where it is observed as a piecewise constant non-decreasing function with the limits $\pm 2W_0$ as $\beta \to \pm\infty$. Note that the WM is the input sample where the semi-derivative crosses the horizontal axis. Therefore the WM can alternatively be

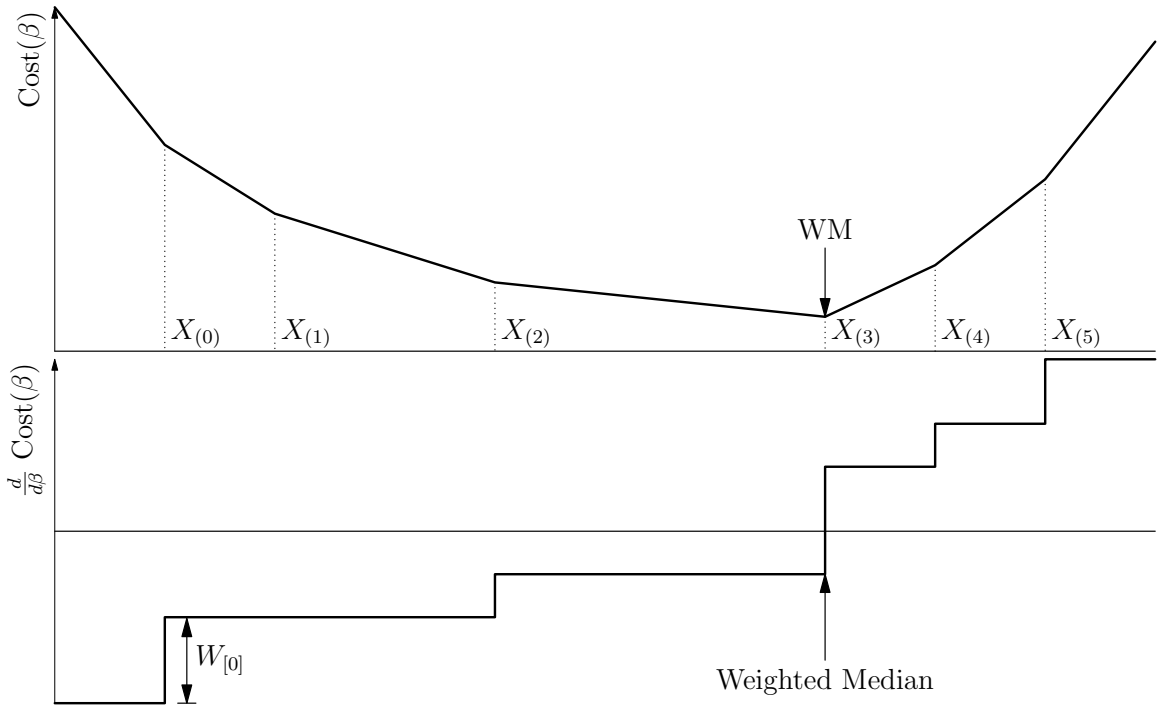

Figure 2.1: This figure visualizes the weighted median sample as the minimum of a cost function. *Top:* An example cost function $T_{WM}(\beta)$ with six input samples X_1 to X_6 and a minimum at $\beta = X_{(3)}$. *Bottom:* The derivative $T'_{WM}(\beta)$ of the cost function (top). Note the zero-crossing point at $\beta = X_{(4)}$ which coincides with the WM. In this example the weights W_i range from 0 to 1, however this is not required in general.

defined by the following formula:

$$\hat{\beta} = \left\{ X_{(k)} : \min k \quad \text{for which} \sum_{i=0}^{k} W_{[N-i]} \geq W_0 \right\}.$$

This first seemingly more complicated expression turns out to allow us to have a different view of the problem which in turn gives us an algorithm dealing more directly with the input samples to solve the weighted median problem. Note that finding the k^{th} order statistic is a special case of the above definition and can be found by replacing W_0 by $N - k$ and set all weights to 1.

Figure 2.1 on page 15 illustrates the algorithm to find the WM which is summarized as follows:

Step 1: Sort the samples X_i with their concomitant weights $W_{[i]}$, for $i = 1, \ldots, N$.

Step 2: Traverse the sorted samples summing up the weights.

Step 3: Stop and return the sample at which the sum is higher or equal to W_0.

It is well-known that sorting an array of N elements requires $O(N \log N)$ comparisons, both, in the average as well as in the worst case. However, using a similar approach as in Quicksort, Hoare [Hoa61] introduced the so called Quickselect algorithm which is an average linear time algorithm to find the k^{th} order statistic of a set of samples. This algorithm can be extended such that Quickselect solves the WM problem. This is further described in Section 2.3. The runtime of both algorithms greatly depend on the choice of the pivot elements which are used for partitioning the array. In Quicksort a pivot close to the median is best and in Quickselect a pivot close to the sought order statistic is best. Particularly, in Quicksort a good pivot is often sought in order to reduce the stack size of a program.

Moreover this dissertation extends the concept of Quickselect which seeks the k^{th} order statistic to the more general case of WM filters which is needed for many signal processing applications. This dissertation main contribution is a new concept to pivot selection. The idea is based on an optimization framework in order to minimize the expected cost of solving the WM problem. The cost functions are then minimized in order to determine the optimal parameters. In particular, the cost is defined as the number of comparisons needed until the algorithm terminates which is the standard measurement of runtime complexity for selection type algorithms.

My approach uses order statistics of a subset of samples to select the pivot. The optimization framework finds the optimal value of the parameter k, which determines which order statistic to choose, and the optimal value of the subset size M. For practical performance comparisons the proposed algorithm was implemented in the C programming language. Numerous simulations validate the theory and show the improvements gained by the proposed algorithm.

2.2 Preliminaries

The sample selection problem in essence, is equivalent to finding the k^{th} order statistic of a set of samples. The algorithms introduced in this dissertation to solve the selection problem can easily be extended to solve the WM problem. To this end, this dissertation focuses on the selection problem to simplify analysis but will go into the details of solving the weighted median problem when necessary. Before introducing the two major algorithms it is necessary to define what "fast" means in terms of algorithm runtime.

In algorithm theory a fast algorithm is one which solves the problem and at the same time has low complexity [CLR+01]. Complexity is defined in different ways which depends on the type of algorithm. In sorting and selection it is defined as the number of comparisons until termination. This is a sensible measure as the main cost of these algorithms is the partitioning step which compares all elements with the pivot. Furthermore the computational complexity is differentiated into worst case, average case and best case complexities. The best case complexity is of little interest since it is usually $O(N)$ for selection and $O(N)$ for sorting. The average case complexity is of most interest in practice since it is the runtime which can be expected in a real implementation with well distributed input data. Of similar importance in theory as well as practice is the worst case complexity as this can be exploited by malicious users to attack [CW03] an application which uses an algorithm whose worst case complexity is unexpectedly higher than the average case.

It was shown in [FR75] that a lower bound on the expected time of the selection problem is in $O(N)$, i.e. linear time. This result is not surprising since given a solution it takes linear time to verify if the solution is correct. For instance, given a sequence of purely random numbers and a location of the k^{th} order statistics (for instance in the form of an array index), it is still necessary to compare each element of the sequence with the given element in order to verify the location of the element if the sequence were ordered.

Additionally, in [BFP+73] it is shown that by choosing the pivots carefully it

is possible to avoid the $O(N^2)$ worst case performance of the traditional Quickselect and also achieve a $O(N)$ worst case complexity. In summary it is important to note that there exists a linear time worst case selection algorithm as well as a proof that the lower bound is also linear. For this reason I do not only compare the computational complexity of the algorithms in terms of their limiting behavior (i.e. asymptotic notation or *Landau* notation) but also analyze the behavior for low to moderate input sizes and obtain accurate numbers for the number of comparisons in order to compare the real world runtime performance.

Taking into account the constants involved in the equations is important especially in practice. A popular example is the fact that most programming language prefer Quicksort to Heapsort for their sorting routine: Despite Heapsort's advantageous behavior of having worst case as well as average case complexity of $O(N \log N)$, Quicksort –with its worst case of $O(N^2)$– is often preferred as the smaller constant term of Quicksort makes it outperform Heapsort on average.

To this date the fastest selection algorithm has been *SELECT* which was introduced by Floyd and Rivest in 1975 [FR75]. The algorithm is asymptotically optimal for large N and as shown by [FR75] the number of comparisons for selecting the k^{th} order statistic given N elements is $N + \min(k, N-k) + o(N)$. Where the notation $o(N)$ is similar to the already introduced big-O notation and is defined as:

$$f(N) \leq g(N) + o(N) \quad \text{means} \lim_{N\to\infty}((f(N) - g(N))/N) = 0.$$

Note that the proposed algorithm is asymptotically optimal as well. Furthermore, as can be seen by the simulations in Section 2.6 our algorithm outperforms *SELECT* and converges to the theoretical optimum more quickly. Quickselect, which will be introduced in Section 2.3 is another very popular selection algorithm due to its simplicity and similarity to Quicksort. Even though it is widely used in practice, its performance is always worse than *SELECT* except for very small input sizes. In particular, for a median-of-3 pivot selection approach, Quickselect needs on average

$2.75N$ comparisons for large N as shown by [MPV04].

Existing research has focused on two main subjects: Firstly, improving the overall runtime of the algorithm which is equal to lowering the constant term in the average case complexity. Secondly, improving the worst case runtime complexity by carefully designing the algorithm to avoid the worst case complexity of $O(N^2)$ often sacrificing average runtime. This dissertation adds a new measure to the equation which determines the reliability of the algorithm runtime under average input data. For instance, it might be that the algorithm performs very well on average but has few outliers which result in an undesirably long runtime. In practice it is often needed that an implementation performs well on average without large outliers. Even if this results in an increased average complexity, it is usually worth the lost performance for a more reliable runtime. An example measure is the well-known variance of the runtime distribution. This topic is further discussed in Section 2.4.

2.3 The Quickselect Algorithm

Quickselect was first introduced in 1961 in [Hoa61] as an algorithm to find the k^{th} order statistic. Note that the popular Quicksort algorithm and Quickselect are similar and the only difference between the two is that the former recurs on both sub-problems –the two sets after partitioning– and the latter only on one.

The original algorithm chooses a sample at random from the sample set $\mathbf{X}$ which is called the first pivot p_1. Later in this dissertation, the method of choosing the pivot will be made more accurately, than selecting a random sample, and instead optimal order statistics which minimize a set of cost functions are used. By comparing all other elements to the pivot, the rank r of the pivot is determined. The pivot is then put into the r^{th} position of the array and all other elements smaller or equal than the pivot are put into positions before the pivot and all elements greater or equal are put after the pivot. This step is called partitioning and can be implemented very efficiently by running two pointers towards each other. One from the beginning of the array and one from the end, swapping elements if necessary until both pointers cross each other.

After this partitioning step is completed there are three cases that can occur:

- If $r > k$ then the k^{th} order statistic is located in the first part of the array and Quickselect recurses on this part.
- If $r < k$ then Quickselect recurses on the second part but instead continues to seek the $(k-r)^{\text{th}}$ order statistic.
- If $k = r$ the recursion terminates and the pivot is returned.

A pseudocode description of the Quickselect algorithm is depicted in Algorithm 2.1 on page 20.

procedure QUICKSELECT($\mathbf{X}, k$) ▷ Returns the k^{th} order statistic of the set $\mathbf{X}$
  Select a pivot $p \in \mathbf{X}$ at random
  Partition the set into two disjoint sets:
  $\mathbf{X}_{\leq} \leftarrow \{X_i \in \mathbf{X} | X_i \leq p\}$
  $\mathbf{X}_{>} \leftarrow \{X_i \in \mathbf{X} | X_i > p\}$
  if $|\mathbf{X}_{\leq}| > k$ **then**
    return QUICKSELECT($\mathbf{X}_{\leq}$, k)
  else if $|\mathbf{X}_{>}| > k$ **then**
    return QUICKSELECT($\mathbf{X}_{>}$, $k - |\mathbf{X}_{\leq}|$)
  else
    return p
  end if
end procedure

Algorithm 2.1: Standard Quickselect algorithm using random pivots. The algorithm is given two parameters: The input set $\mathbf{X}$ and the integer k specifying the requested order statistic of the set $\mathbf{X}$. The algorithm then uses recursion to find the sought element by choosing a pivot and partitioning the set according to that pivot. Note that a actual implementation can transform the algorithm in a non-recursive version which is often desired in order to reduce stack size usage.

The case of $k = (N+1)/2$ (N odd) is the well-known median and is considered a special case of the WM with all weights equal to one. Small modifications of Quickselect lead to a WM-finding Quickselect algorithm in the general case with arbitrary weights:

Instead of counting the number of elements less than or equal to the pivot, the algorithm sums up the weights of all the samples which are less than or equal to the pivot. W_l is defined to be the sum of weights of the partition which contains all elements smaller than or equal to the pivot. Respectively, W_r contains the sum of weights of the other partition. The next step is to compare W_r and W_l to W_0 and either recurse on the partition which contains the WM or return the pivot which terminates the algorithm.

2.4 Optimal Order Statistics In Pivot Selection

In this section the pivots which were introduced in the previous section are studies in detail. As will be shown, the first two pivots are of essence for the performance of the algorithm and are thus the focus of this section.

2.4.1 First pivot p_1

The run time of Quickselect is mostly influenced by the pivot choice. A good pivot can significantly improve the performance. Consider an example: If the pivot is small compared to the sought weighted median then only elements which are less than the pivot are discarded. The worst case happens if the pivot is the smallest element of the entire set. This would results in no elements being discarded despite having performed and expensive partitioning step. The main cost of the partitioning step is to compare all $N_0 - 1$ elements to the pivot. Where N_0 is the number of elements of the original set before any reductions have been performed. Clearly, a pivot close to the actual WM is desired.

Assuming no prior knowledge of the sample distribution or their weights, the only good estimate for a pivot is to choose the median of the samples. The median –by its very definition– ensures that half of the samples are removed after partitioning. However, finding the median of the set is itself a selection type problem which would cost too much time to be computed and offset any possible gains. Instead, an approximation of the median is used as the pivot. An obvious and straightforward approach

is to take a random subset of the input set $\mathbf{X}$ and find the median of this smaller set and use it as the pivot. Let M_0 be the size of this subset with $M_0 \ll N_0$.

Intuitively, a though experiment helps gauging the values of a sensible choice of M_0. If the subset size M_0 is chosen to be close to 0 the time spent finding a pivot is very little, however the number of elements that are discarded after the partitioning step might be very little if the pivot was far from the sought value. If, on the other hand, the subset size M_0 is chosen very large (for instance $M_0 \to N$) then the time spent on finding the pivot is itself very expensive. In that case, however, the partitioning step can be sure to discard approximately $M_0/2$ elements. Concluding, we expect an optimal value of M_0 to lie between the two extremes carefully balancing the cost vs. benefit of choosing a larger subset size.

This intuition is depicted in Figure 2.2 on page 38. In this graph the performance which is measured in C/N is plotted. Where C is the number of comparisons performed until the algorithm terminates. The graph also shows the variance of C/N which is a measure of the reliability of the run time.

Martínez and Roura [MR02] studied the optimal subset size as a function of N_0 with the objective to minimize the average total cost of Quickselect. It was found however, that in practice the runtime was improved if M_0 was chosen larger. Also note that no prior work exists for finding a closed form solution to the optimal subset size. This dissertation derives a closed form solution to the problem.

To this end I introduce a model to obtain a closed form solution for the near optimal M_0. Consider a set of samples $X_1, X_2, \cdot, X_N$. Assume each sample $\{X_i\}_{i=1}^{N_0}$ is independent and identically distributed (i.i.d.). Furthermore consider the random subset $\mathbf{X}' \subset \mathbf{X}$ with $|\mathbf{X}'| = M_0$. The pivot $p_1 = \text{median}(\mathbf{X}')$ is sought as well as the optimal M_0 to minimize the expected samples left (N_1) after the partitioning step. This means that the cost is defined as an approximation to the expected number of comparisons needed until the algorithm terminates. The cost function has to differentiate between the three cases:

1. The pivot is less than the WM of $\mathbf{X}$

2. The pivot is greater than the WM of $\mathbf{X}$
3. The pivot is equal to the WM of $\mathbf{X}$.

The problem arises that both –pivot and WM– are not known beforehand and are in fact random variables. In order to obtain a simpler yet accurate enough cost function which can be solved, various assumptions and simplifications are applied to the model:

1. Each sample of the set $\mathbf{X}$ is modeled as uniformly distributed random variables. This approximation is in fact very accurate since the working point of the model is near the median of the true distribution function at which most distribution behave like a uniform distribution.

2. Finding median($\mathbf{X}'$) can be done in cM_0 comparisons where c is some constant independent of M_0. Additionally, solving the remaining WM problem after the partitioning step can be done in cN_1 comparisons as well. Since $M_0 \ll N_1$ this does not hold true as finding the constant c decreases with increasing samples. However, the difference is small enough and can be neglected.

3. The WM coincides with the median of the standard uniform distribution which is at 0.5. As stated earlier the WM is a random variable. However the variability of the WM can be accounted for in the pivot. Increasing the variance of the pivot distribution accordingly allows to perform this simplification.

Let R be the expected number of elements removed after the partitioning step and by using the assumptions (1)-(3) from above, R is derived as:

$$R \approx \int_0^{1/2} xN_0 \frac{x^{M_0/4-1}(1-x)^{M_0/4-1}}{B(M_0/4, M_0/4)} dx \tag{2.2}$$

$$= \frac{1}{2} N_0 I_{1/2}(\frac{M_0}{4} + 1, \frac{M_0}{4}) \tag{2.3}$$

where N_0 is the size of the original problem set $\mathbf{X}$, where B is the beta function, and I is the regularized incomplete beta function [OLBC10]. The first term (xN_0) of (2.2)

is the expected number of elements less than the pivot. The second term of (2.2) is the probability that the pivot is located at x. Assuming M_0 is odd, then the median of a random subset of size M_0 is beta distributed with the parameters $(M_0+1)/2$ and $(M_0+1)/2$ [DN03]. To account for the third item of the above approximation model the variance of the median is further changed. Reducing the number of samples of the beta distribution by a factor of 0.5 approximately doubles the variance. Furthermore the following approximation can be used:

$$B(\frac{M_0+1}{4}, \frac{M_0+1}{4}) \approx B(\frac{M_0}{4}, \frac{M_0}{4})$$

to obtain (2.2). Solving the integral of (2.2) cannot be done in a closed form. However since the resulting equation is again the p.d.f. of a beta distribution the result of the overall expression is the c.d.f. of the beta distribution evaluated at 0.5 in (2.3).

With the derived expression for R, the new cost function defined as the expected number of comparisons is given by:

$$\begin{aligned} T_{M_0} &= c(N_0 - R) + cM_0 \\ &= cN_0(1 - \frac{1}{2}I_{1/2}(\frac{M_0}{4}+1, \frac{M_0}{4})) + cM_0 \end{aligned} \tag{2.4}$$

where c is the constant mentioned in the second item of the above simplification model. The first summand is the expected number of comparisons necessary to solve the remaining problem after the partitioning step. The second summand accounts for the expected number of comparisons to find the pivot (i.e. the median of the subset). The minimum of T_{M_0} in (2.4) is defined to be M_0^*:

$$M_0^* = \underset{1 \le M_0 \le N}{\arg\min} \quad T_{M_0}.$$

Minimizing T_{M_0} cannot be done in an algebraic way hence further approximations are necessary before a closed for solution can be attained. First note that the division of the two parameters of the beta distribution $\frac{M_0/4+1}{M_0/4}$ is close to one as M_0 is

large. This fact allows to use the normal distribution to approximate the beta distribution. The variance of the beta distribution is $\frac{M_0}{2(M_0+2)^2}$ which can be approximated as $(2M_0)^{-1}$. The resulting approximate cost function is:

$$\tilde{T}_{M_0} = M_0 + N_0 - \frac{N_0}{4}\operatorname{erfc}(\frac{\sqrt{M_0}}{M_0+2}) \tag{2.5}$$

where *erfc* is the complementary error function. It is easy to show that (2.5) is convex for $M_0 > 4.397$. See Appendix B on page 98 for a computer algebra system aided proof.

Theorem 1 *For large N_0, the optimal subset size M_0^* for choosing the first pivot is approximately*

$$\tilde{M_0^*} \approx \frac{1}{\sqrt[3]{8\pi}} N_0^{2/3}.$$

Proof 1 *Differentiating (2.5) with respect to M_0 yields:*

$$\tilde{T}'_{M_0} = -\frac{(M_0-2)N_0}{4\sqrt{\pi M_0}(M_0+2)^2} e^{-\frac{M_0}{(M_0+2)^2}} + 1$$

using the two facts that:

$$\begin{aligned} e^{-M_0/(M_0+2)^2} &\approx 1 \\ (M_0-2)/(M_0+2)^2 &\approx 1/M_0 \end{aligned}$$

yields the result.

Note that in an implementation M_0^* is rounded to the nearest odd integer. Now the first pivot p_1 can formally be defined as:

$$p_1 = \text{MEDIAN}(X'_1, \ldots, X'_{\tilde{M_0^*}}).$$

The error introduced by the approximation is very small which can be seen by analyzing the error as well as the relative error. Figure 2.3 on page 39 depicts the

error $e_{M_0} = M_0^* - \tilde{M}_0^*$ as well as the relative error e_{M_0}/M_0^* for the input size range of $2^6 \leq N_0 \leq 2^{26}$.

The error is almost always zero except for very few N_0 for which the error is a small even number (due to rounding to odd numbers). In fact the number of values of N_0 where the error is not zero is 67964 between 2^6 and 2^{26}, i.e. approximately 99.999% of all samples between 2^6 and 2^{26} have zero error. For large $N_0 \approx 2^{23}$ however, the error starts to increase which makes it important to analyze the relative error e_{M_0}/M_0^*. As can be seen by the lower graph of Figure 2.3 on page 39 the relative error stays close to zero as the error increases. The simulations were run over all integer numbers N_0 between 2^6 and 2^{26}. As N_0 becomes larger it becomes difficult to compute the error. Random N_0 of up to 2^{50} were picked and the calculated error was bounded by 0.001 for the few chosen numbers which indicates that the error does not increase faster than M_0.

2.4.2 Second pivot p_2

For a large number of samples it is unlikely to find the exact WM with the first partitioning step. Thus it is assumed that the first pivot was either larger than or smaller than the WM. The next step is to choose the second pivot p_2. Two obvious choices come to mind:

1. Should the median of a subset be used again?
2. Should a pivot (again) aim to be as close as possible to the expected location of the weighted median?

As it turns out, both of these choices are not optimal. If the median of a subset is selected again, most likely the pivot will be far away from the WM, thus not discarding many elements. This is the result of a skewed sample median as many samples were discarded during the first step of the algorithm. If the pivot is chosen to be as close as possible to the expected location of the weighted median, then the second pivot p_2 will be on average 50% of the time on the same side of the weighted median as the side of

the first pivot. This however, would result in only very few elements being discarded after the second iteration. This turns out to be also not optimal.

That is, intuitively, if the first pivot was smaller but close to the WM then a good choice for the second pivot is an element close to the WM but slightly larger than it. It is natural to use the approach of using the k^{th} order statistic of a subset as the second pivot. The number of samples left after the first iteration of the algorithm is denoted as N_1. M_1 is the cardinality of the random subset $\mathbf{X}''$ of the remaining N_1 samples. Again the formula of Theorem 1 is used to determine the optimal subset size M_1 as a function of N_1. This intuition is depicted in Figure 2.4 on page 40.

To find the optimal order statistic k, the approach of minimizing the expectation of an approximate cost function is again used. Since the goal is to remove as many elements as possible by choosing the pivot appropriately the cost was defined as the expected number of elements left after the partitioning step. A chosen pivot can be either larger, smaller or equal to the WM. Since the cost is negligible if the pivot p_2 happens to be the WM there are only two term for the other two cases. For simplicity, only the case in which the pivot was smaller than the WM will be covered from here on, the other case follows from symmetry.

The cost is defined formally as the expected number of elements left after the partitioning step and is given by:

$$T_\beta = (kC - 1)P_\beta + (N_1 - kC)(1 - P_\beta) \tag{2.6}$$

where

$$C = \frac{N_1 + 1}{M_1 + 1}$$

and where $\beta = k/(M_1 + 1)$ is introduced to normalize k. The minimum of (2.6) is attained by the $k^{*\text{th}}$ order statistic

$$k^* = \underset{1 \le k \le M_0}{\arg\min} \quad T_\beta. \tag{2.7}$$

P_β is the probability that the expected order statistic of the WM is less than or equal to β and will be formally defined below in (2.9). β can be interpreted as the mean of the k^{th} order statistic of M_1 i.i.d. standard uniform distributed random variables. Equation (2.6) constitutes of two simple summands, the first of which accounts for the case that the pivot is greater than the WM and the second for the case that it is smaller. The first term of the first summand is the expected number of elements which are less than the pivot. Again, the WM is modeled as a beta distributed random variable with the parameters $\alpha(M_1+1)$ and $M_1-\alpha(M_1+1)+1$. Where α is the point at which the WM is expected to lie:

$$\text{WM} \approx X''_{(\lfloor \alpha(M_1+1)+0.5 \rfloor)}$$

with

$$\alpha = \frac{W_0 - W_{\leq}^1}{2W_0 - W_{\leq}^1}, \tag{2.8}$$

where $W_{\leq}^1$ is the sum of all weights which were lower than the first pivot p_1 and formally defined as

$$W_{\leq}^1 = \sum_{i=1}^{\text{rank}(p_i)} W_{[i]}.$$

Where $W_{[i]}$ are the concomitant weights as defined in the introduction. Note that the mean of the model is α as desired.

The terms for the second summand of (2.6) are similar to the first summand but cover the case when the pivot is smaller than the WM. For the above model to hold, it is assumed that the input samples are uniformly distributed at the vicinity of the sample median. This holds true for the vast majority of distributions.

The pivot is modeled as being the k^{th} order statistic drawn from a standard uniform distribution. Hence P_β can be expressed as [ABN08]:

$$\begin{aligned} P_\beta &= \Pr\{X''_{(\alpha(M_1+1))} \leq \beta\} && (2.9) \\ &= I_\beta(\alpha(M_1+1), M-\alpha(M_1+1)+1) && (2.10) \end{aligned}$$

where I is the incomplete beta function. Note that $X''_{(\alpha(M_1+1))}$ is not an order statistic since $\alpha(M_1+1)$ is most likely not an integer. Note however, that the formula can still be evaluated correctly since the incomplete beta function allows non-integer arguments.

An example plot of a cost function is depicted in Figure 2.5 on page 41. This figure shows that the 30^{th} order statistic ($k^* = 30$) of the set of $M_1 = 159$ samples should be chosen as the pivot in order to minimize the expected cost. This can be explained by looking at the parameters. α is 0.1 which means the WM is expected to lie close to $X''_{(20)}$. However if $X''_{(20)}$ was chosen as the pivot the probability that this pivot is again lower than the actual WM is higher than if $X''_{(30)}$ was chosen.

Similarly, there is no closed form algebraic solutions to (2.6) so that further approximations are necessary. First, P_β is approximated by the normal distribution with mean α and variance

$$\sigma^2 = \frac{\alpha(1-\alpha)}{(M_1+2)}.$$

Let this approximation be denoted as $\tilde{P}_\beta$. Replacing k by $\beta(M_1+1)$, taking the derivative with respect to β and division by the constant N_1+1 yields:

$$\tilde{T}'_\beta = 2\tilde{P}_\beta - 1 + (2\beta - 1)\tilde{P}'_\beta \tag{2.11}$$

Note that $\tilde{P}_\beta$ is a cumulative density function and $\tilde{P}'_\beta$ is a probability density function.

Lemma 1 *The function $\tilde{T}_\beta$ is quasiconvex for $\beta \in [0,1]$.*

Proof 2 *The proof is divided into three parts:*

for $0 \leq \beta \leq \alpha$: Since $\tilde{P}_\beta$ has median α it follows that (2.11) is strictly negative.

for $\alpha < \beta < 1/2$: Taking the second derivative of $\tilde{P}_\beta$:

$$\tilde{T}''_\beta = 4\tilde{P}'_\beta + (2\beta - 1)\tilde{P}''_\beta$$

shows that the function is convex on this interval as both terms are strictly positive.

for $1/2 \leq \beta \leq 1$: Since all terms are positive, $\tilde{P}_\beta$ is positive as well.

Combining the three intervals proves the quasiconvexity.

Since the cost function is quasiconvex as shown by Lemma 1 the function has only one global minima between $[0, 1]$. Minimizing (2.6) in order to find the optimal k^* yields:

Theorem 2 *The optimal order statistic which is to be chosen as the second pivot is*

$$\tilde{k}^* \approx (\alpha + \sqrt{2}\sigma \log(\frac{1+2\alpha}{\sqrt{2\pi}\sigma}))M_1 \tag{2.12}$$

where $\sigma = \sqrt{\frac{\alpha(1-\alpha)}{M_1+2}}$, $\alpha < 1/2$ *as defined in (2.8), and* M_1 *is the size of the random subset* $\boldsymbol{X}''$.

Proof 3 *Taking the derivative of (2.6) and setting it to zero yields the following equation:*

$$e^{x^2}\,\mathrm{erf}(x) - \frac{1 - 2\sqrt{2}\sigma x + 2\alpha}{\sqrt{2\pi}\sigma} = 0 \tag{2.13}$$

where $x = \frac{\beta-\alpha}{\sqrt{2}\sigma}$.

First, the approximation $\mathrm{erf}(x) \approx 1$ *is used to simplify equation (2.13). As* M_1 *increases so does* x *and hence the error function is close to 1 which justifies this step. Next, the approximation*

$$\begin{aligned} 2\sqrt{2}\sigma x &= 2\sqrt{2}\sigma\frac{\beta-\alpha}{\sqrt{2}\sigma} \\ &= 2(\beta-\alpha) \\ &\approx 0 \end{aligned}$$

can be used since $\beta \approx \alpha$ *for large* M_1. *Solving the resulting equation yields the desired result.*

Note that in an implementation $\tilde{k}^*$ is rounded up to the nearest integer. The reason is that the cost function is steeper towards smaller number and more flat towards larger number which makes this step plausible.

The error introduced by the approximations is very small as can be seen in Figure 2.6 on page 42. The error was defined as:

$$e_k = \max_{0<\alpha<1/2} \frac{\left|\tilde{k}^* - k^*\right|}{M_1}$$

Note that the maximum relative error is declining with increasing N. Figure 2.7 on page 43 shows an example of the relative error. Also note that the maximum occurs at $\alpha > 0.25$ and the error is close to zero for small α. Most likely α will be close to zero. This is true if the first pivot was close to the actual WM. Hence the maximum error has a low impact on the average performance.

Given that $\tilde{k}^*$ is now known, Quickselect is called recursively to compute the pivot. With high probability this pivot will be slightly larger than the WM and hence many samples will be discarded. However, it is not guaranteed that the pivot is smaller than the WM and in the other case very few samples are discarded. The case that the pivot is on the same side of $T'_{WM}(\beta)$ as the first pivot is to be avoided as it would only result in discarding the elements between p_1 and p_2. Since this approach is based on probabilistic analysis it will fail sometimes which will result in a repetition of this approach until the two pivots are located on different sides such that

$$T'_{WM}(p_1)T'_{WM}(p_2) < 0.$$

Note that in the successful case, the problem is bounded, i.e. it is known that the WM is between the first and the second pivot.

2.4.3 Subsequent Pivots

Given the first and second pivots, these pivots can be considered as the lower and upper bounds of the array since all elements outside of these bounds have been discarded. Therefore, at each iteration hereafter the pivot is defined as a convex combination of the maximum and minimum:

$$p_i = \beta X_{\min} + (1 - \beta) X_{\max}$$

where p_i is the i^{th} pivot, β is some constant between 0 and 1, and $X_{\min}$ and $X_{\max}$ are the minimum and maximum points left in the array. After each iteration the minimum and maximum will be updated and hence new bounds are established. This strategy is very different to the existing ones since the pivots are not selected but computed. This implies that the pivots are most likely not an element of the set. Note however, that the crucial step for the algorithm –partitioning the set– is still possible.

After the partitioning step, the set which does not contain the WM will be discarded and the new pivot takes over the weights of the discarded set. The pivot will act as a proxy of the samples of the discarded set. This will introduce new samples into the set which originally did not exist. However, this will not change the WM or the zero-crossing point and is therefore a valid operation.

An optimal β at some iteration can again be found by minimizing a cost function which will be of similar structure to the cost function (2.6). The result is similar to (2.12):

$$\beta \approx \alpha + \sqrt{2}\sigma \log(\frac{1 + 2\alpha}{\sqrt{2\pi}\sigma})$$

where α and is defined the same way as in (2.8):

$$\sigma^2 = \frac{\alpha(1 - \alpha)}{(N_i + 2)}$$

and N_i being the number of elements left in the array at the i^{th} iteration.

Intuitively, if there are many samples between $X_{\min}$ and $X_{\max}$ then this approach works very well. If, however, few samples are within the boundaries then the assumption of the data being uniformly distributed breaks down which results in degraded performance. Therefore the algorithm stops when the problem size gets below a certain threshold level and solves the remaining problem by the standard median-of-3 Quickselect implementation. In addition, in the case that this approach does not

remove any elements –which can happen for some inputs– the algorithm falls back to the median-of-3 Quickselect as well in order to protect from an infinite loop.

2.5 Complexity Analysis

It has been shown, e.g. in [Knu71], that the average number of comparisons to find the median using the standard Quickselect is approximately $3.39N$ and $2.75N$ for median-of-3 Quickselect. The space complexity of the algorithm is $O(N)$ since the partitioning step can be done in-place and hence does not need additional memory. For the first and second partition step the algorithm needs to choose the pivot from a random subset of $\mathbf{X}$. However, this approach would require additional memory and time to generate the random numbers. In practice it is more efficient to sample uniformly, i.e. using a set with equidistant indices. Selecting an order statistic from this subset can also be done in-place and hence there is no need to create a copy to hold the elements.

Since overall the most time of the algorithm is spent in the partitioning step, the time complexity is defined as the number of element-wise comparisons the algorithm makes until termination. In addition, the comparisons to select the pivots recursively have to be taken into account. Using a median-of-M samples strategy as the pivot, it was shown in [MR02] that the number of comparisons is $2N + o(N)$ on average. Simulations in the next section show that the proposed algorithm beats this result. This is mainly due to the strategy used for selecting the second pivot. As explained earlier, selecting just the median of a subset for the second pivot is not optimal in general.

The worst case runtime of the proposed algorithm is $O(N^2)$ trivially since the algorithm will always do better or equal than the median-of-3 Quickselect which has the same worst case runtime. It is shown in [MR02] that the worst case runtime for a median-of-M approach is $O(N^{3/2})$ for $M = \Theta(\sqrt{N})$ [1] which is a similar approach used

[1] The notation $\Theta(N)$ is used in the following way: $f(N) \in \Theta(g(N))$ means $\exists k_1, k_2 > 0, N_0 \forall N > N_0$: $|g(N)|k_1 \leq |f(N)| \leq |g(N)|k_2$

Table 2.1: This table shows the average number of normalized comparisons $\bar{C}$ for different alpha of alpha input distributions. Note that the number of comparisons are not affected by heavy tailed input distributions.

alpha	2	1.8	1.4	1.0	0.8	0.6	0.4
$\bar{C}$	1.72	1.72	1.72	1.72	1.72	1.72	1.72

for the proposed algorithm. This is an indication that the proposed algorithm's worst case complexity is also lower than $O(N^2)$.

Performing theoretical average complexity analysis is not possible due to the complex approach of choosing the subset size and the pivots. However, given the simulations from Section 2.6 the complexity can be approximated by fitting a function. An approximation is useful to give the reader or programmer an intuitive measure of runtime behavior of the proposed algorithm. Using the tool *Eureqa*, which was introduced in [SL09], the following formula for the complexity was obtained:

$$C(N) = 1.5N + 6.435N^{0.629}.$$

Eureqa was run with the basic building plots and the block "power". The data was not smooted prior and the most compact formula was chosen over the most accurate one.

2.6 Simulation Results

To compare the simulation results among different sample set sizes the number $\bar{C} = C/N$ is used. C is the sample mean of the number of comparisons and N the input data set size. The proposed algorithm was implemented in MATLAB and simulated with different input distributions of $\mathbf{X}$. Since weighted medians are often used with heavy tailed input distributions, the algorithm was run on alpha-stable distributions where alpha $\in [0, 2]$ [Arc05]. As alpha gets smaller the tails of the distribution get heavier which results in more outliers present in the input data. For alpha $= 2$ the data is Gaussian distributed. The resulting C/N for $N = 8193$ are depicted in Table 2.1.

Note that the mean does not change and is robust to heavy tailed inputs. This can be easily explained: After the first two steps the algorithm makes the problem bounded above and below and thus all outliers are already discarded. Furthermore due to the approach to pick the first two pivots it is highly unlikely to select an outlier as the pivot. In particular, the first pivot is taken as the median of a relatively large subset size. This median is unlikely to be an outlier. The second pivot is then taken in the vicinity of the first pivot which, again, is unlikely to be an outlier. Uniformly distributed input data does not change the complexity which is expected since the design of the proposed algorithm is based on the assumption that the input is uniformly distributed around the WM.

The performance is compared to two other selection type algorithms. The standard median-of-3 Quickselect and Floyd and Rivest's SELECT algorithm [FR75]. Since both algorithms only solve the more specific problem of finding the order statistic of a set of samples, a version of the proposed algorithm which finds the median was implemented in the C programming language. To this end each implementation keeps track of the number of comparisons it performs. In particular the proposed algorithm's pivot finding strategy is more complex and the comparisons performed during this phase is taken into account.

The results are depicted in Figure 2.8 on page 44. The proposed method clearly outperforms SELECT for smaller and medium input sizes and both methods converge to $\bar{C} = 1.5$ optimum for very large input sizes. The $\bar{C} = 1.5$ optimum can be explained easily: For very large input sizes the number of samples removed is almost $N_0/2$ after the first partitioning step which does N_0 comparisons. The second pivot will also remove almost $N_0/2$ samples which cost $N_0/2$ comparison to partition. All that is left is a small fraction of the input size and therefore the number of comparisons converge to $1.5N_0$. The asymptotic optimality is proved in Appendix A.

To explain why the proposed algorithm performs better it is necessary to look at how SELECT works: SELECT first chooses two pivots which lie with high probability just to the left and just to the right of the sought median. This however is suboptimal

unless the input size is very large since the two pivots have to be chosen to be relatively far away from the median and therefore many unnecessary comparisons are done. The proposed method on the other hand tries to get the first pivot as close as possible to the median and then chooses the second pivot such that the median is with high probability between the first and the second pivot. This explains Figure 2.8 on page 44.

However since the proposed algorithm is more complex and in turn requires more code than the other two algorithms, it is of interest to show how the actual runtime of the proposed algorithm compares to the other candidates. To this end, the C-implementations were run and timed and the sample averages are compared. The speedup is measured as $\frac{T_{slow}}{T_{new}}$ where T_{slow} is the runtime of the faster of the two existing algorithms SELECT or Quickselect. The tests were carried out on a Linux (CentOS) host with kernel version 2.6 on an *AMD Opteron 2376* processor and *GCC 4.6* compiler with all optimizations enabled. Figure 2.9 on page 44 shows the speedup in runtime as defined above. In addition the figure also depicts the theoretical speedup which is defined as the quotient of the comparisons: $\frac{C_{slow}}{C_{new}}$. Similar as above, C_{slow} is the number of comparisons for the faster of the two existing algorithms and C_{new} the number of comparisons for the proposed algorithm.

The proposed method is up to 23% faster than SELECT for a wide range of input sizes and converges to $\approx 5\%$ for very large input sizes. There are subtle differences in the theoretical and practical speedup which can be explained. First, the speedup for the input sizes 513, 1025 is smaller due to the involved overhead of the more complex implementation. This increased complexity however pays off nicely for larger input sizes. For this reason the proposed method is slower than SELECT for input sizes smaller than 513 samples. Secondly, even though the theoretical speedup approaches zero as N becomes large the practical speedup approaches 5%. It seems that the compiler can optimize the algorithm slightly better than SELECT.

For all simulations the input samples were i.i.d. normal distributed.

2.7 Conclusions

A fast new Weighted Median algorithm is introduced which is based on Quickselect. The algorithm is based on the optimality of several parameters used to design a set of pivots. In particular, the set of order statistics and the sample set size are the crucial parameters for optimal performance. Minimizing cost functions in combination with a well approximated model for the cost were used to develop this novel algorithm. Theory explains that the proposed method should be faster than Floyd and Rivest's SELECT unless input sizes are very small. This was backed up by experiments which showed a speedup of up to 23%.

Furthermore the proposed algorithm can compute the median as well as the WM but the same ideas can be applied to finding the k^{th} order statistic. In addition the proposed algorithm can be used to solve multidimensional selection problems which require medians of large data sets.

A C-implementation can be downloaded from my homepage at the University of Delaware at the URL: http://www.eecis.udel.edu/~rauh/fqs/

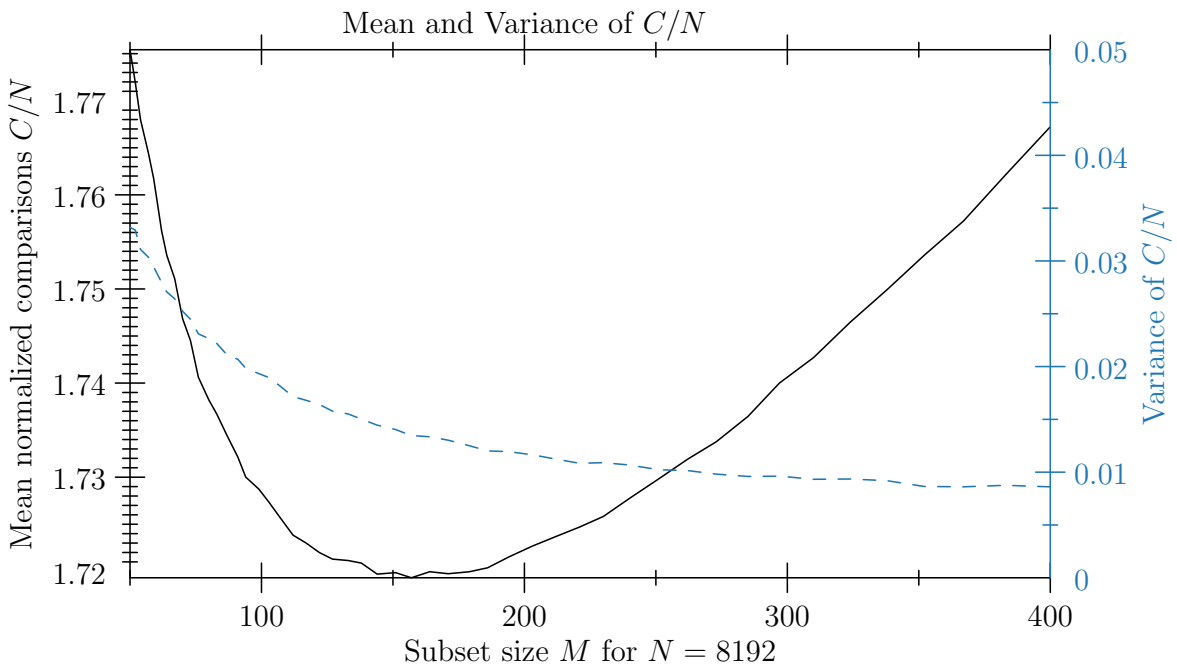

Figure 2.2: This graph show simulations of the number of comparisons that are needed depending on the subset size chosen for the first pivot. Very small subset sizes M_0 yield higher overall runtime due to more element-wise comparisons necessary. Very large subset sizes on the other hand are also not optimal since too much runtime is spent on finding a pivot. The optimal subset size is where the number of comparisons is minimized. This dissertation focuses on finding a closed form solution to this subset size. The higher M_0 is chosen the lower is the variance but the mean increases as well. The lower variance means that the runtime is more reliable which is due to a very good pivot that removes half the elements reliably. (For the graph: $N = 8192$, 100000 simulations averaged)

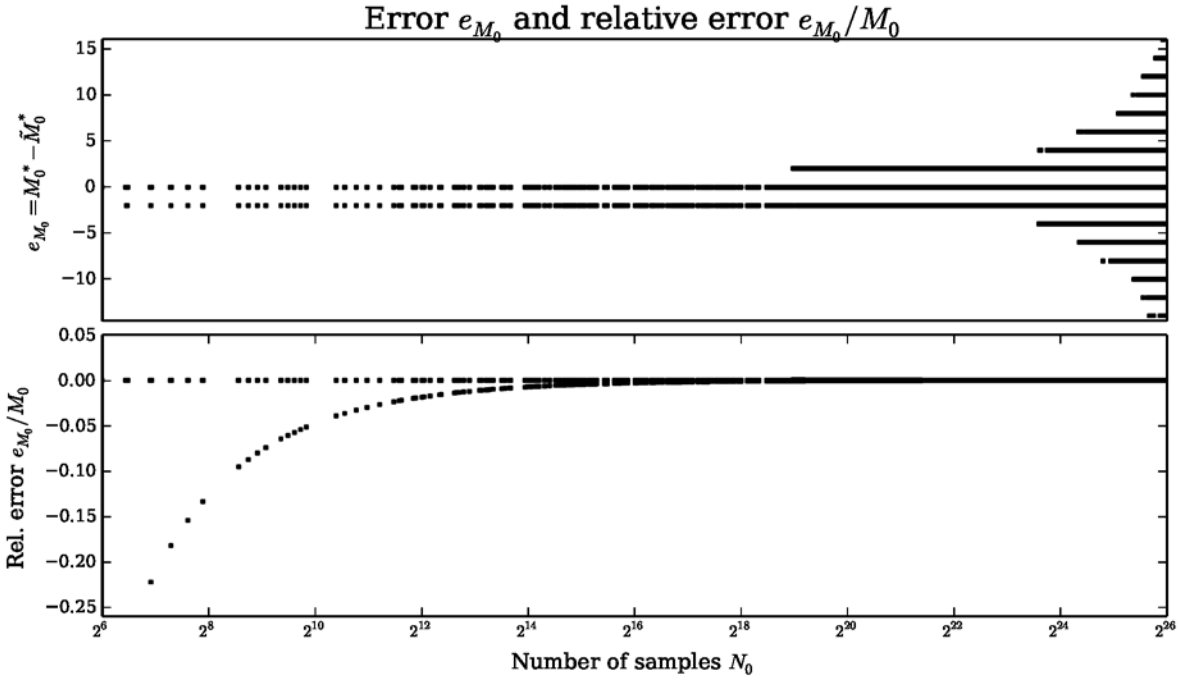

Figure 2.3: The error of the optimal M_0^* and the approximated $\tilde{M}_0^*$. The error increases as N_0 becomes increasingly large. However the relative error stays close to zero since the error grows slower than M_0. This result shows the applicability of the approximations.

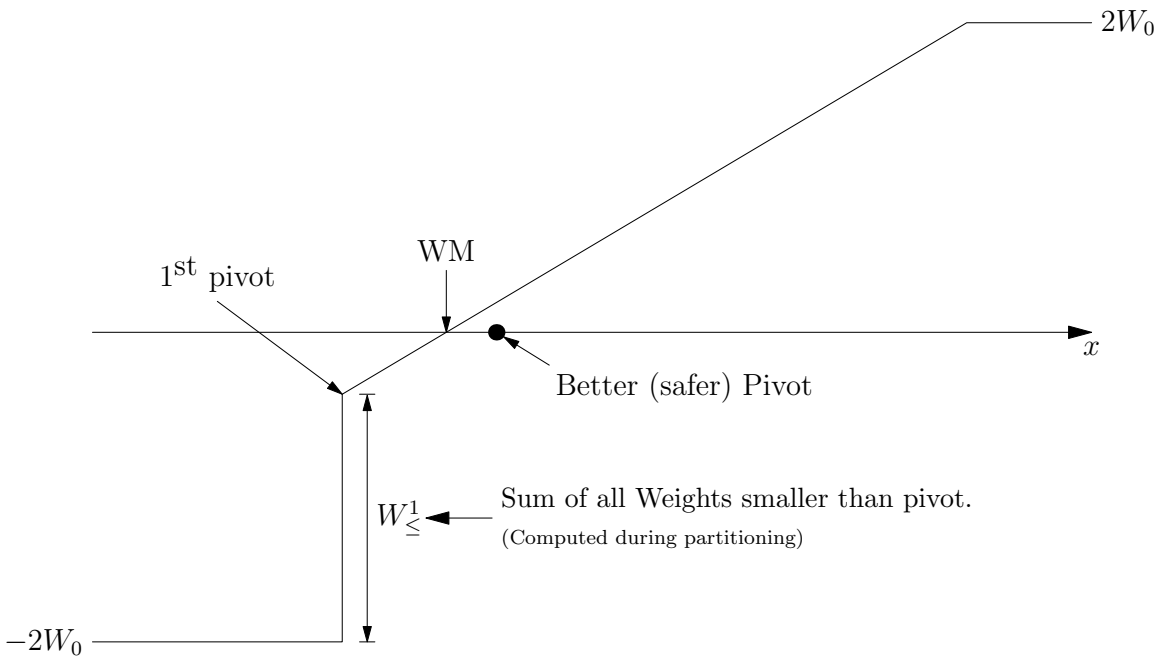

Figure 2.4: This figure shows a conceptual depiction of the state of the algorithm after the first pivot was chosen and the input sequence partitioned along the pivot p_1. In this case the first pivot happened to be less than the sought weighted median. During the partitioning step it is easy to calculate the sum of the weighted $W^1_{\leq}$ which is used to find the optimal second pivot p_2. The best strategy for the second pivot is *not* to choose a pivot as close as possible to the location where the weighted median is expected. This could potentially result in a "wasted" iteration when the second pivot is again less than the WM discard only very few elements. Thus, an overall better strategy is to choose a "safer" pivot which will result in a lower number of comparison on average.

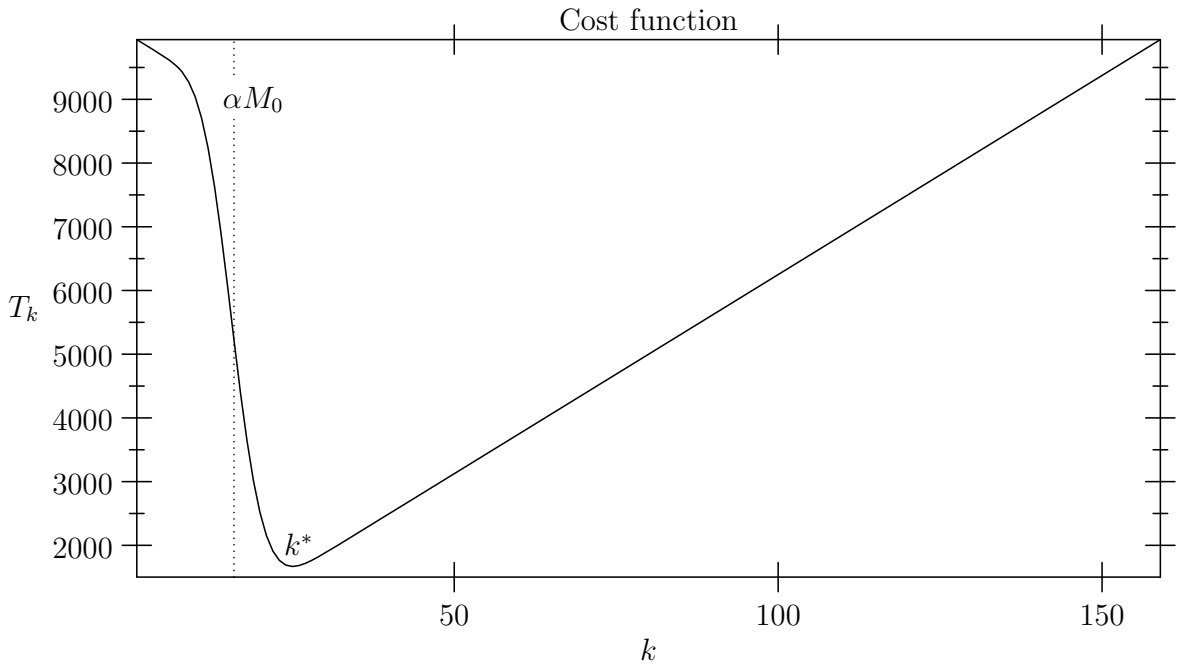

Figure 2.5: Expected cost T_k for choosing the k^{th} order statistic as the second pivot p_2. Note that the weighted median is expected to lie at αM. However, choosing this point as the second pivot is far from optimal. This is due the case that the pivot could end up below the weighted median which is highly undesirable. The minimum cost –and hence optimality– is achieved by choosing a slightly larger order statistics as the pivot. (For this plot: $N_1 = 10000, M_1 = 159, \alpha = 0.1$)

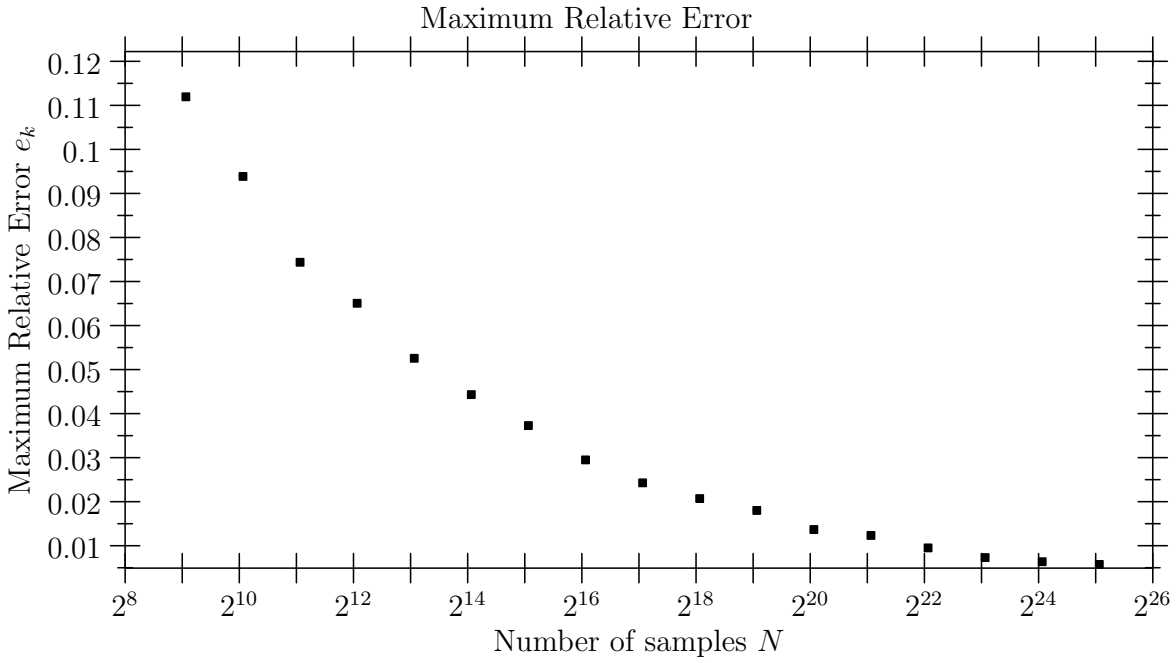

Figure 2.6: The maximum relative error of the optimal order statistic k^* and the approximation $\tilde{k}^*$. The error decreases as the input size increases.

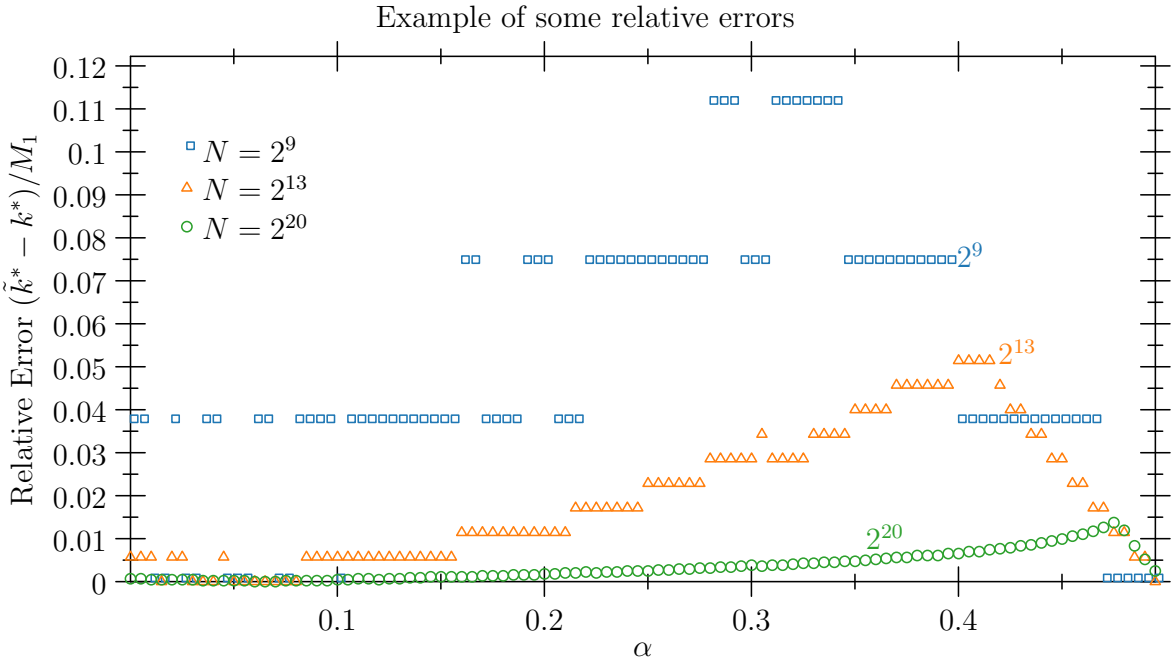

Figure 2.7: The relative error of the optimal order statistic k^* and the approximation $\tilde{k}^*$ for $N = 2^9$ ($M = 27$), $N = 2^{13}$ ($M = 175$) and $N = 2^{20}$ ($M = 4439$). It is important to note that the error is small for small α. This is crucial for the algorithm to work well as small α are more likely to occur in practice.

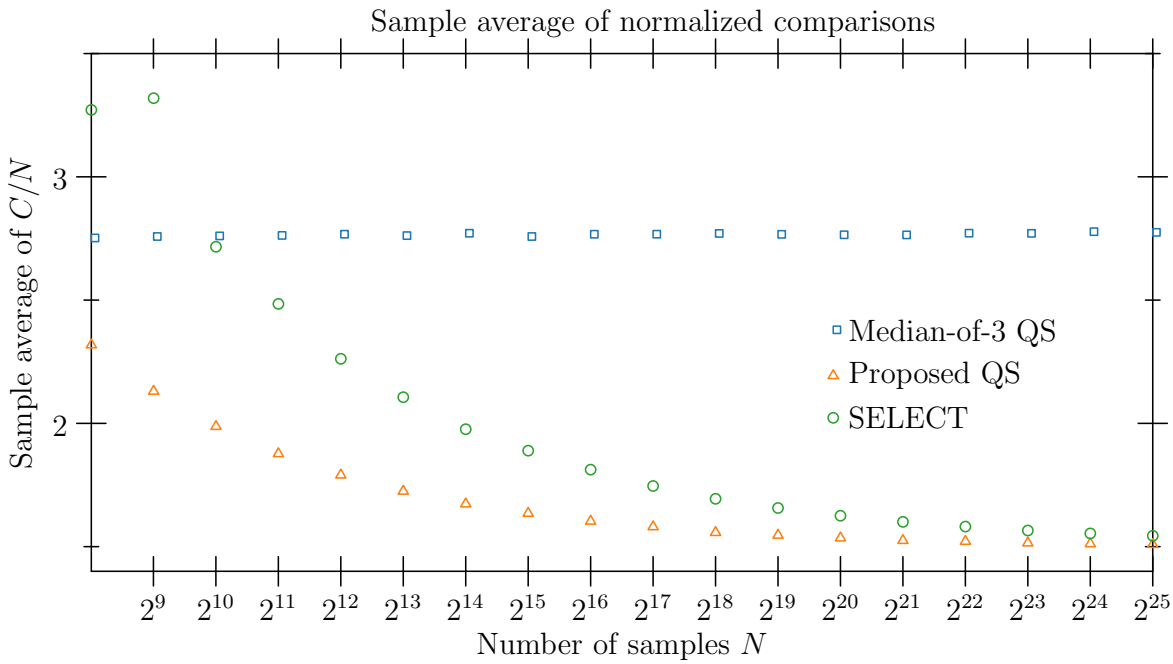

Figure 2.8: The sample average of the normalized number of comparisons (C/N) for the different algorithms.

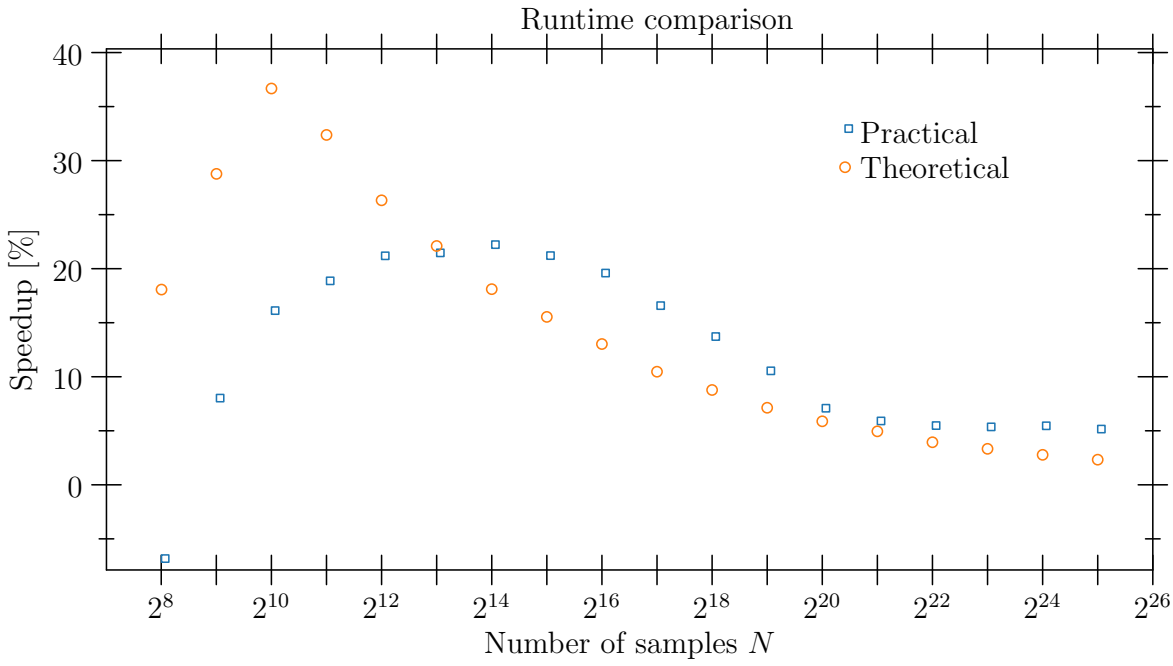

Figure 2.9: Speedup against Floyd and Rivest's SELECT.

Chapter 3

OPTIMIZED SPECTRUM PERMUTATION FOR THE MULTIDIMENSIONAL SPARSE FFT

3.1 Introduction

The Discrete Fourier Transform (DFT) calculates the spectrum representation of input signals and has become ubiquitous in signal processing applications. One famous such application is the popular MP3 audio codec in which case the input are samples in the time domain and the DFT is used to transform the signal into the audio frequency domain in which most audio signals are compressible. The most popular two dimensional signal is the digital photograph which contains the discretization of a spatial signal. Image processing tasks such as denoising use the 2D DFT transform in their algorithms. It is used broadly throughout digital signal processing, partial differential equations, polynomial multiplication [CLR+01] and audio processing.

In certain time constrained applications, there is a natural desire to accelerate the DFT. With the emergence of mobile computing which is often constrained in electrical and processing power it is natural to seek algorithms that perform increasingly faster and with less power usage [HAKI12].

In addition to improving the runtime complexity of the DFT, there is also an interest in reducing the sampling complexity. The sampling complexity is important when the sheer amount of data is impossible or prohibitive to sample or simply not available. This idea of reducing the sampling complexity is similar to the idea behind compressive sensing [CRT06].

Formally the DFT of a one dimensional signal of N elements is defined as:

$$\hat{x}_j = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i j n/N}, \tag{3.1}$$

where x_n is the signal in the time domain and $\hat{x}_k$ the DFT in the frequency domain. i is defined as $\sqrt{-1}$. The complexity of a naive and straightforward implementation of the DFT has a runtime complexity of $O(N^2)$ for a one dimensional signal of length N. This is readily noticed by analyzing (3.1) which can alternatively be expressed as a matrix vector multiplication:

$$\hat{\boldsymbol{x}} = \boldsymbol{F} \cdot \boldsymbol{x} \tag{3.2}$$

where $\boldsymbol{F}$ is the so called DFT matrix composed of the Fourier basis functions. The matrix has the following entries defined for $0 \leq n, j < N$:

$$\boldsymbol{F}_{n,j} = e^{-2\pi ijn/N}.$$

In 1965 the popular Fast Fourier Transform (FFT) was introduced in [CT65]. This highly celebrated finding exploited properties of complex numbers and reduced the runtime complexity of the DFT from $O(N^2)$ to $O(N \log N)$. The Fast Fourier Transform (FFT) exploits properties of complex numbers leading to a reduced runtime complexity of $O(N \log N)$. The idea behind the FFT algorithm is easy to understand since it only involves the well known divide and conquer approach taking into account special properties of complex numbers.

Following the approach of [FR13]:

$$\begin{aligned}
\hat{x}_j &= \sum_{n=0}^{N-1} x_n e^{2\pi ijn/N} \\
&= \sum_{l=0}^{N/2-1} x_{2l} e^{2\pi ij2l/N} + \sum_{l=0}^{N/2-1} x_{2l-1} e^{2\pi ij2l/N} \\
&= e^{2\pi ij/N} \sum_{l=0}^{N/2-1} x_{2l} e^{2\pi ijl/(N/2)} + \sum_{l=0}^{N/2-1} x_{2l-1} e^{2\pi ijl/(N/2)}.
\end{aligned} \tag{3.3}$$

Thus, it can be seen how the DFT of N elements can be expressed as two separate DFT transforms of size $N/2$. Applying the decomposition recursively yields the FFT algorithm and results in a runtime complexity of $O(N \log N)$.

Note that, in (3.3), the runtime complexity of the transformation algorithms is known to be determined only by the input size, and not affected by the value of the input.

While the FFT algorithm does not make any assumptions about the structure of the signal, very often the signals of interest are obtained from a structured source resulting in a nearly sparse Fourier spectrum. This fact is the basis for signal compression and is used among others in the popular MP3 codec. Other examples of structured signals include images, video and in general samples of most systems over time or space. In general it is likely that the signals encountered in nature are often structured unless the signal in question is purely random and thus just noise. These input signals frequently result in a sparse spectrum, i.e., most of the Fourier coefficients of a signal are very small or equal to zero.

Assuming that a signal of length N is k-sparse ($k < N$) in the Fourier domain, the signal can be described with only these k coefficients. Due to the fact that the signal is accurately described with just k coefficients it seems natural that there should be a better performing algorithm that exploits this property of the signal.

The last 20 years has seen advances in algorithms aimed at improving the runtime for the sparse Fourier transform. The first notable in [KM93] and several other algorithms have been proposed with this goal in mind [AGS03, GGI+02, Iwe10, Man92]. A recent approach is the so called sparse FFT (sFFT) which lowered the computational complexity significantly. It was introduced in [HIKP12a] and improved the runtime complexity to $O(k \log N)$ to make it faster than the FFT for a given sparsity factor $k \leq O(N/\log N)$.

The applicability of the sparse FFT is only limited by the sparseness of the signal in the Fourier domain. For instance, the standard FFT algorithm could be chosen to perform the compression of an audio signal for when a high audio quality is needed during playback. On the other hand, the sparse FFT could be used if the audio signal is highly compressed into a lower quality signal which may be desirable for speech.

There are also applications which inherently work on sparse signals. For instance GPS deals with extremely sparse signals and the sparse FFT can be used to significantly speed up signal acquisition [HAKI12]. Another application is GHz-Wide sensing [HSA+14] of signals. Additionally, a very popular field that deals with sparse signals is compressive sensing. Recovery algorithms for compressive sensing are often limited by the performance of the sparse basis transformation. This is where the sparse FFT would come into play and therefore allow faster recovery and recovery of large problems sizes. One particularly interesting application within compressive sensing is the reconstruction of Magnetic Resonance Imaging (MRI) images which rely on the two or three dimensional Fourier transform.

Another similar application is to apply the 2D sparse FFT to reconstruct a more sparsified image in 2D Magnetic Resonance Spectroscopy [SAH+13].

Computational Lithography uses very large scale two dimensional Fourier transforms to calculate the intensity of every source point of the Wafer [**?**]. A sparse Fourier transform of sizes up to 10^{33} or more is necessary for the detection of pulsars [AZJB06]. Various image registration techniques use the Fourier transform [ZF03]. Similarly these ideas can be applied to motion estimation which is used for video encoding among other applications [Tim99]. In general the DFT can be applied to multi dimensional signals used in engineering applications and is not limited to audio, image or video signals.

The algorithm introduced in [HIKP12a] handles only one dimensional signals and does not explicitly state how to extend the algorithm to solve the multidimensional sparse FFT problem. Being a separable transform the 1D DFT can easily be extended to multiple dimension by applying the 1D algorithm multiple times on each slice along each dimension. Applying the sparse FFT in this naive way, however, would negate all gains due to the repetitions necessary along one dimension.

Assuming a signal of dimensionality d with N elements along each dimension, the complexity of such a naive algorithm would be $kdN \log N$. The term N which is introduced by the repeated computation of the 1D sFFT is prohibitively fast growing compared to the desired logarithmic term $\log N$. Thus, a more sophisticated approach

is shown in our work by extending the sparse FFT algorithm itself to multiple dimensions.

In addition, the algorithm in [HIKP12a] introduces many parameters which are left to be chosen randomly. This works well for the assumption of a randomly distributed input spectra but is sub-optimal for signals inhibiting structure. Often however, choosing those parameters randomly is far from optimal and becomes a much more pronounced problem with the multi dimensional sFFT.

In this dissertation it is shown that the parameters in question should not be chosen randomly and Lattice theory is used to optimize these parameters. In fact, the findings are also applicable to other algorithms which use pseudorandom spectrum permutation [HIKP12b, HIKP12a, GGI+02, GMS05, GST+08]. Note that the proposed algorithm and finding in this chapter are not limited to the exact sparse FFT (sFFT-3.0) but are also applicable to other and more recent developed algorithms such as [**?**].

This chapter is structured in the following way: Section 3.2 introduces the concept of lattices and focus on the parts which are of interest for understanding the main idea of this chapter. Section 3.2.1 makes the connection of Dual Lattices and the Fourier transform. Section 3.2.2 introduces an iterative algorithm to optimize the permutation parameter when applied to commonly shaped spectra. Section 3.3 introduces a recursive algorithm to change a matrix of even determinant to a matrix of odd determinant which is needed for the proposed algorithm. Section 3.4 introduces the sparse FFT algorithm and Section 3.5 explains the modifications necessary to extend the algorithm to multiple dimensions. The chapter is finish up with some simulations in Section 3.6 and conclude the chapter in Section 3.7.

3.2 Lattices

Lattices have various mathematical definitions depending on the context and application field. In this work, lattices are used as a discrete additive subgroup of $\mathbb{R}^n$

also known as point lattices. An additive subgroup has the property:

$$\forall x,\, y \ \in \mathcal{L} \rightarrow -x, x+y \in \mathcal{L} \tag{3.4}$$

i.e. for any two points in the lattice $\mathcal{L}$, the points $-x$ and $x+y$ are also part of that lattice.

Equivalently, a lattice is the $\mathbb{Z}$-linear span of a set of n linearly independent vectors:

$$\mathcal{L} = \mathcal{L}(\boldsymbol{B}) = \boldsymbol{B} \cdot \mathbb{Z}^n = \{a_1\boldsymbol{b}_1 + a_2\boldsymbol{b}_2 + \cdots + a_n\boldsymbol{b}_n : a_1, a_2, \ldots, a_n \in \mathbb{Z}\}. \tag{3.5}$$

The vectors $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n$ are called the basis vectors of the lattice and similarly the matrix $\boldsymbol{B}$ is called the basis of the lattice. One simple 2-dimensional lattice is the set $\mathbb{Z}^2$ which is a simple rectangular shaped point lattice of all integer numbers. Note that in theory a lattice is an infinite set, however in practical applications the set of numbers will be on a finite domain.

The fundamental parallelepiped of a basis $\boldsymbol{B}$ is defined as:

$$\begin{aligned}\mathcal{P}(\boldsymbol{B}) &= \boldsymbol{B} \cdot \left[-\frac{1}{2}, \frac{1}{2}\right)^n \\ &= \left\{\sum_{i=1}^{n} c_i \boldsymbol{b_i} : c_i \in [-\frac{1}{2}, \frac{1}{2})\right\}.\end{aligned}$$

Another fundamental region of particular interest is the Voronoi cell of a lattice. It is defined as the set of all points in $\mathbb{R}^n$ that are closer to the origin than to any other lattice point:

$$\mathcal{V}(\boldsymbol{B}) = \{\boldsymbol{x} \in \mathbb{R}^n : ||\boldsymbol{x}|| < ||\boldsymbol{x} - \boldsymbol{v}|| \,\forall\, \boldsymbol{v} \in \mathcal{L} \setminus \{\boldsymbol{0}\}\}. \tag{3.6}$$

Note, that the basis of a lattice is not unique. For any matrix $\boldsymbol{U} \in \mathbb{Z}^{n\times n}$, $\det \boldsymbol{U} = \pm 1$ the matrix resulting from $\boldsymbol{B} \cdot \boldsymbol{U}$ will also be a basis for the lattice $\mathcal{L}(\boldsymbol{B})$.

Some basis are more desirable and interesting than other basis due to their

mathematical properties. Basis reduction is the process of taking a base and reducing it such that it still generates the same lattice. In lattice theory a vast effort has been made to find such reduced basis sets [LLL82].

Given a basis, the problem of finding the shortest near-orthogonal vectors is called the shortest vector problem (SVP). The exact problem of finding the shortest vector within a lattice is known to be NP-hard [Ajt98]. However, various approximate solutions with polynomial time exist. In fact, this NP-hardness result gave rise to a new field called lattice based cryptography. However, various algorithms to find good approximations in polynomial time complexity have been reported [LLL82, Kan83, Sch87].

The minimum basis of a lattice is the basis with the shortest and nearly orthogonal basis vectors. The fundamental parallelepiped of the minimum basis of a lattice is called the fundamental domain of the lattice. The determinant of the fundamental lattice is defined as the volume of the fundamental parallelepiped.

An example of a two dimensional lattice is the well known hexagonal tiling which is obtained from a lattice with basis vectors of equal length and an inner angle of 60 deg. Due to a lattice being an additive subgroup the tiling that is generated is a monohedral tiling, i. e. all tiles are congruent.

The rank of a lattice is the dimension of its linear span:

$$m = \dim(\mathrm{span}(\boldsymbol{B})). \tag{3.7}$$

In the case of $m = n$ the lattice is called a full rank lattice. The rank of the lattice is easily visualized: It is the number of vectors necessary to generate the lattice. In two dimensions, two well known full rank lattices are the square and the hexagonal lattice which are depicted in Figure 3.1.

Many engineering and science fields are connected to lattices in their applications. For instance, lattices find application in material science where they describe the atom structure of regular grid like structures such as crystals. In fact, the dual

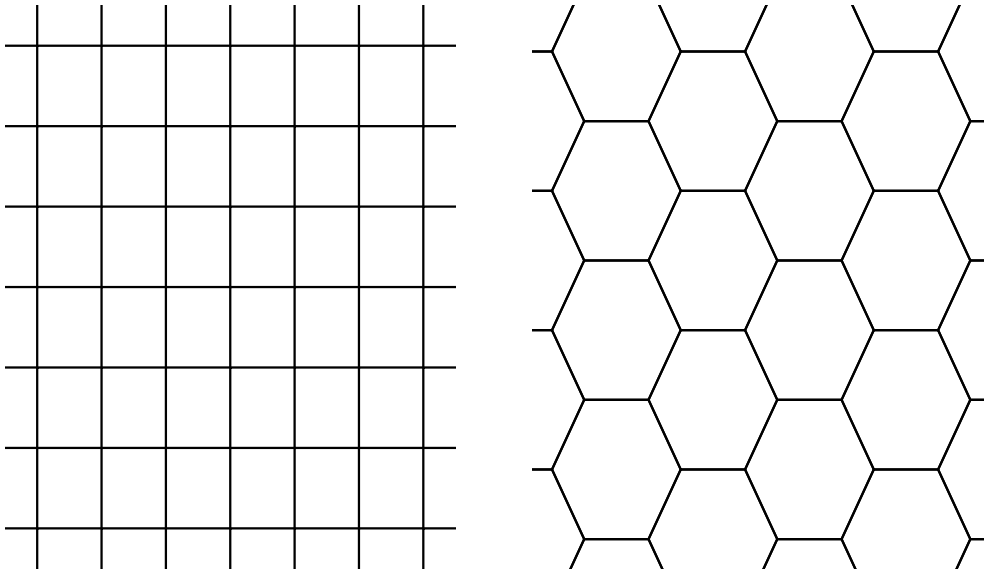

Figure 3.1: Examples of two dimensional lattice tilings. *Left:* A simple square lattice. *Right:* A hexagonal lattice.

lattice –which is often called the reciprocal lattice– is used in Crystallography to obtain information about the crystal after X-ray exposure [KMM76].

Another recently emerging application is the usage in public key cryptography due to the high computational complexity of the lattice basis reduction problem [NS01]. In communication, lattices find use in demodulation and quantization both of which seek to solve the closest vector problem. In computer graphics lattices are applied in sampling patterns and pixel anti-aliasing [Dam09]. Pseudo random number generators are closely connected to lattices and find applications in numerical integration [SB89].

This dissertation discusses lattices in the context of multidimensional sparse FFT. While the focus is on the two dimensional case, all arguments are straightforward to extended to arbitrary dimensions unless otherwise noted.

An example of a two dimensional lattice is depicted in Figure 3.2. It also depicts the Voronoi cell, the Delaunay tessellation and the two basis vectors that generate the lattice points. The fundamental parallelepiped is spanned by the two basis vectors and the origin.

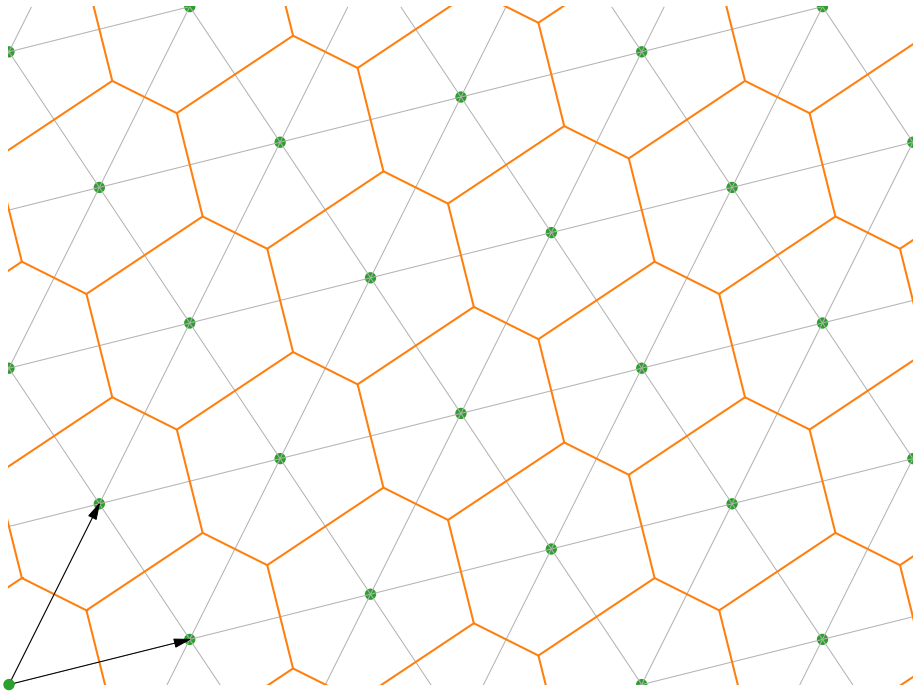

Figure 3.2: The solid dots (green) are the lattice points generated by the base vectors shown in the bottom left. The solid connected lines (orange) surrounding the lattice points is the Voronoi tessellation. The straight lines (grey) crossing through the lattice are the Delaunay tessellation also known as the dual graph. The fundamental region is spanned by the two basis vectors in the bottom left. Also note, the lattice points are the centroids of the Voronoi parallelepiped.

3.2.1 Dual Lattice

The Nyquist-Shannon sampling theorem is the corner stone of sampling a continuous signal. It states that a signal is completely determined when sampled at twice the bandwidth. Fulfilling this constraint, it is possible to reconstruct the signal perfectly. Violating this requirement yields to aliasing and artifacts in the signal. The original theorem is stated for the one dimensional case where the samples are equidistant. It is straightforward to extend the theorem to multiple dimension by treating each dimension independently. This is easy to see by using the fact that the Fourier transform is a separable transform. For instance, a two dimensional signal which has

a rectangular shaped spectrum with different bandwidths along each dimension, implies that the signal can be sampled at a lower rate along one dimension and must be sampled at a higher rate along the other dimension.

For instance, consider an image with a rectangular spectrum S that has a bandwidth of $1\frac{1}{m}$ along the x-dimension and a bandwidth of $2\frac{1}{m}$ along the y-dimension. This implies that the signal can be sampled at different rates along the x-dimension than the y-dimension thus departing from the commonly used square like sampling lattice. It is easy to see that the optimal sampling lattice for such a signal is again a rectangular shaped lattice with the difference that the sampling should be $0.5m$ along the x-dimension and $0.25m$ along the y-dimension. Thus, in this example the optimal sampling lattice is again a rectangular lattice. Note that a fat rectangular shaped spectrum will result in a tall and skinny shaped sampling lattice in the time domain.

Naturally, it is interesting to ask what the optimal sampling lattice is for a given arbitrary shaped spectrum? Note, that any kind of rectangular sampling in the time domain will result in the repetition of the spectrum along a rectangular shaped lattice in the frequency domain. Thus, the standard rectangular sampling grid method does not allow us to answer this question in an interesting way. This constraint of rectangular sampling is therefore relaxed and allow for more freedom. As introduced in the previous section, a lattice can be used to sample a continuous signal. This, in turn, raises the question of the effect on the signal spectra as it is sampled by a lattice. The answer to this question is the *Dual Lattice*.

One well known fact is that the optimal sampling lattice of circular shaped spectra is the hexagonal lattice. This is due to two facts:

1. The hexagonal lattice is self-dual, i.e. the dual of a hexagonal lattice is again a hexagonal lattice.
2. The hexagonal lattice optimizes the arrangement of spheres thus achieving the highest density. This fact was proven by Gauss in 1831 [Gau31].

It is also helpful to think of the Voronoi tessellation of the hexagonal lattice –which is a hexagon– as the best approximation to a circle among all two dimensional

lattices. Also note that the hexagonal lattice is popular due to the fact that it is optimal if the spectrum is assumed to be isotropic; i.e. of circular shape. This is in fact a good assumption for most real world signals.

In general, the signal is sampled in such a way that the resulting spectra of the samples signals do not overlap and thus the signal can be reconstructed perfectly.

If the sampling constraints are relaxed from ordinary rectangular lattices to arbitrary lattices however, the way the signal is sampled is still constrained. As shown later, the effect of signals that are sampled according to a lattice in the time domain is that their spectrum is reduplicated according to the dual lattice in the frequency domain.

Given this constraint it is natural to ask what the optimal sampling lattice is, given an *a priori* knowledge of the shape of the spectrum. Reduplication of the spectrum in the frequency domain along a lattice $\mathcal{L}$ means that overlapping of the spectrum occurs if the spectrum extends beyond the Voronoi cell of the lattice $\mathcal{L}$. This is a straightforward result from the definition of the Voronoi tessellation (3.6) which quasi tiles the space according to the proximity to a lattice point. Here, the term "quasi-tiling" is used since the Voronoi cell does not include the boundary points so the tiling is not complete. The above assumes symmetric spectra or else a more general approach must be taken which will be noted later in this section.

Formally, the dual lattice of the lattice $\mathcal{L}$ is denoted as $\mathcal{L}^T$.

It is defined as the set of real vectors $\boldsymbol{h} \in \mathbb{R}^n$ with:

$$\mathcal{L}^T := \{\boldsymbol{h} \in \mathbb{R}^n \,|\boldsymbol{h} \cdot \boldsymbol{x} \in \mathbb{Z} \text{ for all } \boldsymbol{x} \in \mathcal{L}\} \tag{3.8}$$

which means that the dual of a lattice $\mathcal{L}$ is the set of points whose inner product with any points of the lattice is an integer. The dual lattice is –as its name suggests– again a lattice. For instance the dual of the lattice $2\,\mathbb{Z}^n$ is the lattice $\frac{1}{2}\,\mathbb{Z}^n$. As this example shows, it is also common to refer to the dual lattice as the reciprocal lattice.

If the lattice is a full rank lattice, then the dual lattice can be generated with

the basis matrix of the inverse of the transpose of $\boldsymbol{B}$, i.e. $(\boldsymbol{B}^T)^{-1}$. This follows directly from the definition of the dual lattice. For the use-case of improving the sparse FFT, it should be noted that we always deal with full rank lattices.

Next it is shown that sampling a signal with a lattice $\mathcal{L}$ results in the spectrum of the signal being reduplicated according to the dual lattice $\mathcal{L}^T$ [EDM09]:

$$\begin{aligned}
\widehat{\mathcal{L}_{\boldsymbol{B}}}(\boldsymbol{\xi}) &= \int \mathcal{L}_{\boldsymbol{B}}(\boldsymbol{x}) e^{-2\pi i \boldsymbol{\xi}\cdot x} d\boldsymbol{x} \\
&= \sum_{\boldsymbol{k}\in\mathbb{Z}^n} e^{-2\pi i \boldsymbol{\xi}\cdot\boldsymbol{B}\boldsymbol{k}} \\
\widehat{\mathcal{L}_{\boldsymbol{B}}}(\boldsymbol{\xi}+\boldsymbol{B}^{-T}\boldsymbol{m}) &= \sum_{\boldsymbol{k}\in\mathbb{Z}^n} e^{-2\pi i (\boldsymbol{\xi}+\boldsymbol{B}^{-T}\boldsymbol{m})\cdot\boldsymbol{B}\boldsymbol{k}} \\
&= \sum_{\boldsymbol{k}\in\mathbb{Z}^n} e^{-2\pi i [\boldsymbol{\xi}\cdot\boldsymbol{B}\boldsymbol{k}+\boldsymbol{B}^{-T}\boldsymbol{m}\cdot\boldsymbol{B}\boldsymbol{k}]} \\
&= \sum_{\boldsymbol{k}\in\mathbb{Z}^n} e^{-2\pi i [\boldsymbol{\xi}\cdot\boldsymbol{B}\boldsymbol{k}+\boldsymbol{m}^T\boldsymbol{B}^{-1}\boldsymbol{B}\boldsymbol{k}]} \\
&= \sum_{\boldsymbol{k}\in\mathbb{Z}^n} e^{-2\pi i [\boldsymbol{\xi}\cdot\boldsymbol{B}\boldsymbol{k}+\boldsymbol{m}^T\boldsymbol{k}]} \\
&= \sum_{\boldsymbol{k}\in\mathbb{Z}^n} e^{-2\pi i \boldsymbol{\xi}\cdot\boldsymbol{B}\boldsymbol{k}} \quad \text{since } e^{-2\pi i \boldsymbol{m}\cdot\boldsymbol{k}} = 1 \\
&= \widehat{\mathcal{L}_{\boldsymbol{B}}}
\end{aligned} \tag{3.9}$$

A more rigorous proof can be found in [EDM09].

In general, the algorithm of finding an optimal sampling lattice provided an expected shape of the spectrum is the following:

1. Given the expected shape of the spectrum S_g (which, if no a-priori information is given, should be chosen as an n-sphere)
2. find a lattice whose Voronoi tessellation (3.6) best approximates the spectrum shape S_g. Let this lattice be called $\mathcal{L}_g$.
3. Calculate the dual lattice $\mathcal{L}_g^T$ of the lattice $\mathcal{L}_g$ and use the lattice $\mathcal{L}_g^T$ to sample the signal.

Step 2 of the algorithm can be done by a simple brute force search by generating lattices and their Voronoi tessellation and measuring the error. The error can be

measured according to some user defined error measure, for instance by calculating the area of the given spectrum that lies outside the Voronoi tessellation. This is a simple and effective approach if the dimensionality of the signal is low, otherwise this approach suffers from the popular curse of dimensionality and more sophisticated approaches must be employed [DDK09].

Lastly, it is noted that in some cases finding an optimal lattice requires a more sophisticated algorithm. For instance, if the spectrum of the signal is non-symmetric the step 2 of the algorithm tries to match a Voronoi cell –which is always symmetric– with the non-symmetric spectrum. This, in turn, will result in a poor fitting lattice which is larger than necessary and hence results in non-optimal lattice.

In this case it is recommended, that the optimal lattice should be found by brute force iterating over lattices and reduplicate the input spectrum around the neighborhood of the origin point of the lattice. The error measure used to find a good matching lattice is the intersection of the reduplicated spectra. In general, more elaborated methods to deal with non-symmetric spectral characteristics can yield slightly better lattices.

3.2.2 Permutation Candidates Algorithm

In this Subsection an iterative algorithm is introduced which is based on the dual lattice introduced in the previous section. The pseudo code of the algorithm is given in Algorithm 3.1 on page 58. The algorithm takes the basis $\boldsymbol{L}$ of a lattice $\mathcal{L}$ and returns a sequence of matrices of the same size as the basis. Note that, one step within the algorithm ensures that the determinant of the integer matrix is odd. An algorithm to ensure an odd determinant is given in Section 3.3.

To this end, the algorithm first calculates the dual and generates integer matrices based on this dual. This sequence of matrices can be used to permute the spectrum of the shape of the input lattice.

Each candidate matrix $\boldsymbol{M}$ is evaluated by applying the permutation operator

procedure DUALPERMCANDIDATES($\boldsymbol{L}$)
  $\boldsymbol{L'} \leftarrow \boldsymbol{L}^{-T}$
  $s \leftarrow ||\boldsymbol{L'}||_{\max}$ ▷ Maximum matrix element
  $S \leftarrow \emptyset$, $n \leftarrow 0$
  for $m \leftarrow 1, 2, \ldots$ **do**
    $\tilde{\boldsymbol{P}} \leftarrow \text{round}(ms\boldsymbol{L'})$
    Ensure $\det(\tilde{\boldsymbol{P}})$ is odd ▷ See Alg. 3.2
    if $\tilde{\boldsymbol{P}} \notin S$ **then**
      $S \leftarrow S \cup \tilde{\boldsymbol{P}}$
      $\mathcal{M}_n \leftarrow \tilde{\boldsymbol{P}}$
      $n \leftarrow n + 1$
    end if
  end for
  return $\mathcal{M}$
end procedure

Algorithm 3.1: The proposed iterative algorithm which generates an infinite sequence of candidates for permutation matrices $\tilde{\boldsymbol{P}}$. The only parameter needed is the lattice basis approximating the expected spectrum shape. Note that the evaluation of the "goodness" of the candidates is deferred until it is defined what constitutes a good permutation matrix. Also note that despite generating an infinite sequence an actual implementation would not realize the candidates in the sequence eagerly.

P' to the spectrum:

$$(P'\hat{x})_{\boldsymbol{j}} = \hat{x}_{\boldsymbol{Mj}}. \tag{3.10}$$

Each spectrum permutation is then evaluated by measuring the error. The error measure can vary by the application and is user defined. For instance it may be preferable to choose the overall PSNR or a simple inter point distance as an error measure.

Next, the algorithm is evaluated with simulations. To this end, multiple sparse input spectra are generated whose shape is derived from a multivariate Gaussian distribution. MATLAB was used for implementing the simulation. The setup was as follow: The input size was chosen to be $N \times N = 1024 \times 1024$ with a sparsity of 0.01%. One realization of such an input spectrum is depicted in Figure 3.3 (left).

For each input spectrum the simulation permutes the spectrum according to (3.10). The permutation matrix $\boldsymbol{M}$ is either chosen randomly or from the sequence of matrices obtained from Algorithm 3.1 (page 58).

One example of a random permutation is depicted in Figure 3.3 (middle) with an example of a good permutation obtained via the proposed algorithm depicted on the right. Note the structured layout of the coefficients which are unfavorable to avoid collisions in the sparse FFT algorithm.

In order to measure the "goodness" of a permuted spectrum the distance to the closest neighbor is calculated for each nonzero coefficient:

$$D_{\boldsymbol{k}}^{\min} = \min_{\boldsymbol{j} \neq \boldsymbol{k}} ||\hat{x}_{\boldsymbol{k}} - \hat{x}_{\boldsymbol{j}}||_2. \tag{3.11}$$

Note that, this is different from the minimum distance of a lattice. This error measure was chosen due to being correlated to the number of collisions in the sparse FFT algorithm. Close neighbors can lead to collisions which in turn yield to non-optimal performance.

A total of 400 simulations were run and the results summarized with a histogram which is depicted in Figure 3.4. The result is a 21% improvement in the mean closest neighbor distance.

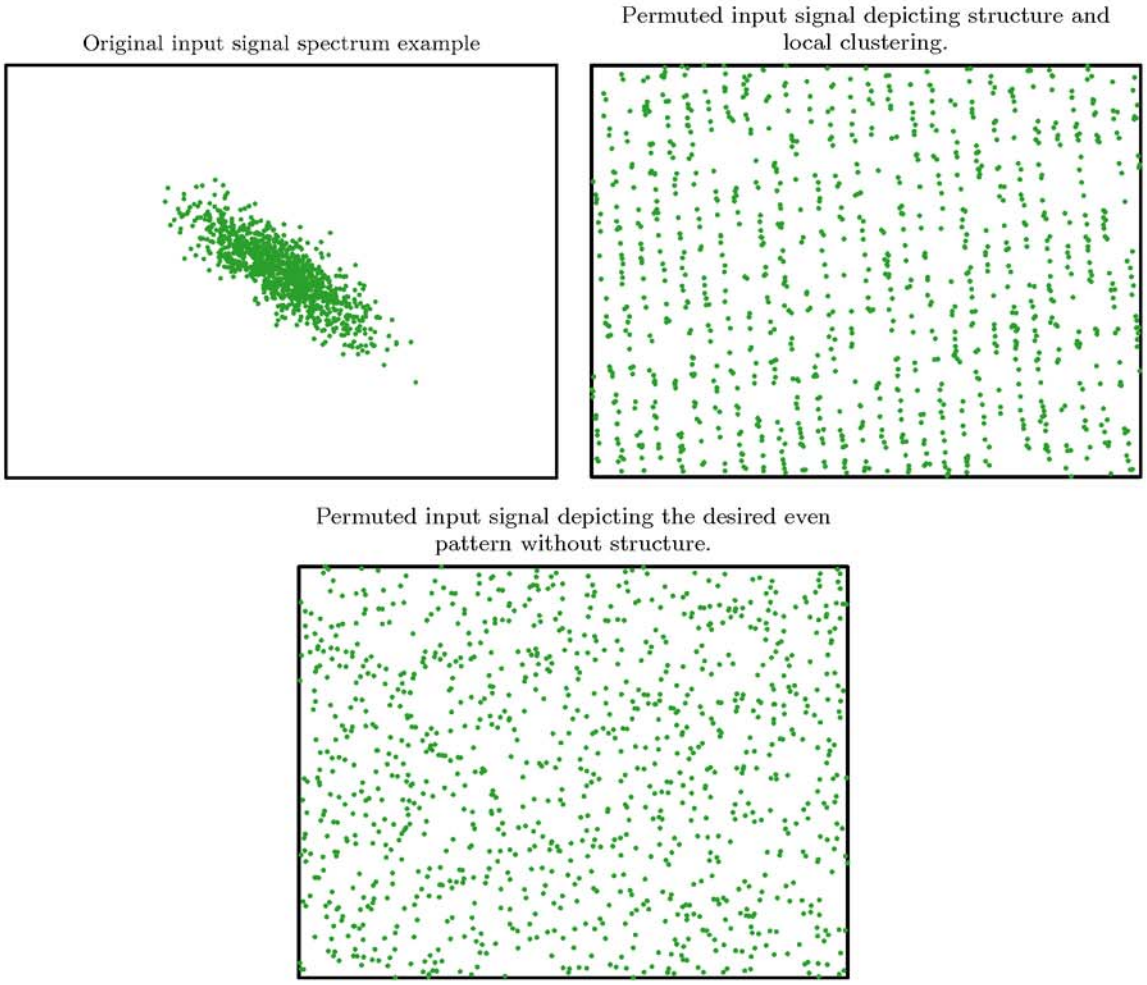

Figure 3.3: Good and bad permutation of a permuted input spectrum. *Top left:* The original input spectrum generated with a multivariate Gaussian distribution. *Top right:* An example of a bad permutation due to choosing the parameters randomly. Notice the clustering of the lattice points leading to collisions in the sparse FFT algorithm. *Bottom:* An example of using a permutation obtained from using the optimal permutation with Algorithm 3.1. Note the very uniform distribution of the coefficients which is desirable to reduce collisions in the sparse FFT.

3.2.2.1 Complexity Analysis

The proposed algorithm generates a sequence of permutation matrices. To this end the dual matrix is calculated and subsequently multiplied by a a factor. The input size is a matrix of $d \times d$ integer elements. However, the complexity is not measured in

term of d since d is usually small and considered constant for the sparse FFT algorithm. The actual complexity comes from the dimension of the input spectrum that is to be permuted. That is, the elements of the returned permutation matrix are at most N, where N is the number of elements in each dimension. It does not make sense for the elements to be $> N$ since the input spectrum is a cycling signal due to the definition of the DFT.

A naive algorithm would try all possible permutation matrices which has a complexity of $O(N^{dd})$. This means that the complexity is exponential in the number of dimensions. This is no surprise due to the well known curse of dimensionality. However, evaluating all possible matrices is impossible even for $d = 2$ unless N is small which is of no interest to the sparse FFT algorithm. The proposed Algorithm 3.1 on page 58 reduces this complexity for any dimensionality d to a linear search algorithm of $O(N)$ by only evaluating the matrices that are likely to yield a good performance. This massive reduction in complexity allows for a much more sensible subset of possible permutation matrices which yield good performance as will be shown later. It also makes the problem tractable for higher dimensions due to being independent of d.

3.3 Odd Determinant Algorithm

In this Section a novel algorithm is introduced that ensures that the determinant of a square matrix is odd. This constraint is to be fulfilled in order for the permutation generated by the matrix to be invertible. Normally a matrix is invertible if the determinant is nonzero. However, the permutation applies a modulo N operation to the indices due to the implicitly cyclic DFT. This results in a system of linear congruences which are invertible if the determinant is relatively prime to N [Apo13]. Due to the further constraint of N being a power of two, the constraint is for the determinant to be odd. The algorithm works with any square matrix $\boldsymbol{M} \in \mathbb{N}^{d \times d}$. The approach is to flip bits of as few of the matrix entries as possible thus changing some entries from even to odd and vice versa.

procedure EnsureOddDeterminant($\boldsymbol{M}$)

Assert $\boldsymbol{M} \in \mathbb{N}^{d \times d}$

if $|\boldsymbol{M}|$ is odd **then**

return $\boldsymbol{M}$

end if

if $d = 1$ **then**

return $\boldsymbol{M} + 1$

end if

Let m and $\boldsymbol{N}$ be the expansion such that:

$|M| = \sum_{i=1}^{d} (-1)^{i+1} m_{1,i} \boldsymbol{N}_{1,i}$

$T_o = \sum_{i=1}^{d} [m_{1,i} \boldsymbol{N}_{1,i} \text{ is odd}]$ ▷ T_o: Terms odd

if $T_o \geq 2$ **then**

Find the i for which $m_{1,i}$ is odd

return $\boldsymbol{M}$ with the entry $m_{1,i} \leftarrow m_{1,i} - 1$

end if

Find i for which $|\boldsymbol{N}_{1,i}|$ is odd

if i was found **then**

return $\boldsymbol{M}$ with the entry $m_{1,i} \leftarrow m_{1,i} + 1$

end if

if There is no i for which $m_{1,i}$ is odd **then**

Set $m_{1,1} \leftarrow m_{1,1} + 1$

end if

Find i for which $m_{1,i}$ is odd

Update $\boldsymbol{N}_{1,i} \leftarrow$ EnsureOddDeterminant($\boldsymbol{N}_{1,i}$)

return EnsureOddDeterminant($\boldsymbol{M}$)

end procedure

Algorithm 3.2: Recursive algorithm to turn an integer matrix with even determinant into a similar matrix with odd determinant. Note that the notation $[P]$ is the Iversion bracket which is 1 if P is true and 0 otherwise. Note that the algorithm potentially recurses on a matrix of the same input size but guarantees termination after only one more call due to the conditions preceding the recursion. The algorithm works by flipping bits carefully such that the Laplace expansion of the determinant has an odd number of odd terms.

The basis for the algorithm approach is the Laplace expansion of the determinant calculation of a matrix:

$$|\boldsymbol{M}| = \sum_{i=1}^{d} (-1)^{i+j} m_{i,j} \boldsymbol{N}_{i,j}$$

where $m_{i,j}$ are the matrix entries and $\boldsymbol{N}_{i,j}$ are the minors of $\boldsymbol{M}$ obtained by generating a matrix of the elements of $\boldsymbol{M}$ omitting the entries of the row i and column j. Thus effectively crossing-out the entries of the i^{th} row and j^{th} column. Note that the size of the minor $\boldsymbol{N}$ is $d-1 \times d-1$. This technique is also sometimes referred to as the expansion by minors.

The other obvious observation is that only an odd number of odd terms yield an odd number. The algorithm is most easily understood with recursion which reduces the problem size in each iteration by one and ends in the trivial case of a 1×1 matrix which is just an integer. Such a matrix can be made odd by subtracting or adding one in case the number is even.

To describe the algorithm we further introduce the Iversion bracket which is a more general version of the well known indicator function and defined as:

$$[P] = \begin{cases} 1 & \text{if } P \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

A full pseudo code description of the algorithm is given in Algorithm 3.2 on page 62 .

A detailed description of the algorithm is given walking through the algorithm from the top to the bottom: First, the algorithm handles two trivial cases which stop the recursion:

1. The determinant of the input is odd
2. The input size is a single number ($d = 1$)

The second case can be solved by adding one to the input number which makes the number odd. The algorithm then proceeds with inspecting the input's Laplace expansion as introduced earlier. Subsequently, the number of odd terms in the expansion are counted. Note that, at that stage of the algorithm the number of odd terms are even or else the determinant would have been odd. If the number of odd terms is two or more, the solution is easy: Make one of the terms even and the overall determinant will become odd. Note, however, that the minor matrices of the terms are not disjoint and have common elements. Thus, flipping one of the elements of the minor matrices can result in a change in multiple terms. This in turn means that a flip in the matrices could result in the determinant staying even. For this reason, the factor $m_{1,i}$ of the expansion is used to make the overall term odd. It is know –due to the term being odd– that the factor $m_{1,i}$ must be odd as well. This stage solves the problem and terminates the recursive algorithm.

The next stage is to handle the case if the number of odd terms is zero. This means that the determinant can be made odd by making one single term odd. Overall, it is desirable to change change the least amount of matrix entries which implies that the focus is on changing the factors $m_{1,i}$ as opposed to the minor matrices which could require multiple changes for it be become odd. Thus, the algorithm analyses each term of the expansion and tries to find a term with a minor matrix which has an odd determinant. If this is successful, the factor $m_{1,i}$ is made odd as well. Again, this terminates the algorithm since this will make one term odd and thus the overall determinant odd.

At this stage, there is no minor matrix with odd determinant, which implies that recursion on one minor matrix is necessary. The first step however, is to ensure that at least one factor $m_{1,j}$ exists which is odd. If this fails, then the first factor $m_{1,1}$ of the matrix is set to become odd. The next step is to find an odd factor $m_{1,j}$ and its accompanying minor matrix $\boldsymbol{N}_{1,j}$. It is known that the minor matrix $\boldsymbol{N}_{1,j}$ has an even determinant. Thus, the algorithm calls itself recursively which ensures that minor matrix has an odd determinant. Note that the problem size is now reduced by one

since the minor matrix has size $d-1 \times d-1$.

The final line of the algorithm requires some attention: As noted earlier, changing a minor matrix's entries affects the other $d-2$ minor matrices' entries. This, in turn, can lead to other terms of the expansion become odd. For instance, if there were no terms which were odd and one minor matrix is modified then this modification could yield two 2 odd terms. Thus effectively negating what was desired: An overall odd determinant. Interestingly, the case of two or more odd terms was already covered earlier in the algorithm. This means that this case can be "fixed" by simply calling the algorithm recursively. Note that, the recursive call is *not* being done with a reduced input size but with the same $d \times d$ input matrix size. It first seems that this could potentially result in an infinite loop. However, the recursive call will *never* call itself recursively and is thus safe to do. This observation also proves the termination of the algorithm since the other recursion reduces the problem size each time.

Note that, there is a slightly more complex version of this algorithm possible. The proposed algorithm only does one Laplace expansion along the first row of the input matrix. A more sophisticated algorithm could potentially reduce the number of bit flips even more by considering other expansion along the rows and columns. This, however, does increase the complexity of the algorithm and was not considered for the use-case of our main algorithm: The sparse FFT.

3.3.1 Error and Complexity Analysis

In this Section the runtime complexity of Algorithm 3.2 as well as the error introduced by the bit flipping is analyzed. The worst case runtime of the proposed algorithm is $O(d!)$. This is due to the complexity of the Laplace expansion which also has a complexity of $O(d!)$. In the worst case the algorithm recurses on one smaller minor matrices and expands each. Note, however, that for the use case of generating permutation matrices this seemingly high complexity is not an issue. For instance for a 3-dimensional sparse FFT the input matrix is only 3×3 and in fact considered constant for the algorithmic complexity of the sparse FFT algorithm.

Since the algorithm changes the entries of the input matrix it is of interest how much the output of the algorithm differs from the input. The absolute maximum error is bound by one:

$$||M_{\text{in}} - M_{\text{out}}||_\infty = 1$$

This is due to the careful choice of adding one to the even entries and subtracting one from the odd entries which effectively can be implemented as a bit flipping operation. This means, that even if the recursive algorithm changes entries multiple times the error is guaranteed to be at most one.

3.3.2 Examples

In this Section some examples input and output of the proposed algorithm are given.

$$\begin{aligned}
\textsc{EnsureOddDeterminant}\left(\begin{bmatrix} 62 & \mathbf{50} & 97 \\ \mathbf{41} & 87 & 18 \\ 10 & 53 & \mathbf{39} \end{bmatrix}\right) &= \begin{bmatrix} 62 & \mathbf{51} & 97 \\ \mathbf{40} & 87 & 18 \\ 10 & 53 & \mathbf{38} \end{bmatrix} \\
\textsc{EnsureOddDeterminant}\left(\begin{bmatrix} \mathbf{33} & 35 \\ 43 & \mathbf{44} \end{bmatrix}\right) &= \begin{bmatrix} \mathbf{32} & 35 \\ 43 & \mathbf{45} \end{bmatrix} \\
\textsc{EnsureOddDeterminant}\left(\begin{bmatrix} \mathbf{33} & 25 \\ 67 & 19 \end{bmatrix}\right) &= \begin{bmatrix} \mathbf{32} & 25 \\ 67 & 19 \end{bmatrix}
\end{aligned}$$

Note how the algorithm successfully only flips one element in the last case. A more naive algorithm would potentially have flipped more elements to achieve an odd determinant.

Table 3.1: Overview of the most recent existing sparse FFT algorithms.

Algorithm	Complexity	Robust	Permutation
sFFT 2.0 (Non-Iterative)	$O(\log N\sqrt{Nk\log N})$	Yes	Yes
sFFT 3.0 (Exact k Sparse)	$k\log N$	No	Yes
sFFT 4.0 (General k Sparse)	$k\log N\log(N/k)$	Yes	Yes
Prony Based [HKPV13]	$O(k^{5/3}\log^2 N)$	Yes	Yes

3.4 Sparse FFT Algorithm

In this Section the core ideas of the one dimensional sparse FFT algorithm as introduced by [HIKP12a] in 2012 are described. A more in depth explanation of all steps is described in Section 3.5. Specifically, the proposed algorithm is based on what is often referred to as the exact k-sparse algorithm also called sFFT-3.0. However, the proposed changes to the algorithm are not specific to this version and are also applicable to the general k-sparse algorithm which is also sometimes referred to as sFFT-4.0. An overview of a few selected sparse FFT algorithms is presented in Table 3.1.

Firstly, the notation is introduced. Note however that the notation will partly be re-used for the multidimensional version of the algorithm in Section 3.5.

Given a signal x of length N its discrete Fourier transform is denoted as $\hat{x}$. A signal is considered to be k-sparse if there are only k non-zero components in $\hat{x}$. Furthermore $\omega = e^{-2\pi\imath/N}$ is defined as the Nth root of unity. The set $\{0,\dots,N-1\}$ is defined as $[N]$ and further $[N]\times[N]$ as $[N]^2$. The number of bins that are used to hash the Fourier coefficients is denoted by B.

The following paragraph describes the main ideas of one iteration of the sFFT-3.0 algorithm. The key idea of the sFFT algorithm is to hash the k coefficients into few buckets in sub-linear time. This is achieved by using a carefully designed filter that is concentrated in time as well as in the frequency domain. Due to the sparsity of the signal and the careful selection of the number of bins, each bin is likely to only contain one coefficient after being hashed. After the coefficients of each bin are obtained the

actual positions in the frequency domain are recovered by *locating* and *estimating*. The algorithm does this hashing twice and "encodes" the frequency of the coefficient into the phase difference between the two hashed coefficients. This technique achieves the *locating* part of the algorithm by decoding the phase and obtaining the frequency. Before the coefficients are hashed into buckets, the procedure (HashToBins) permutes the signal x in the time domain by applying the permutation operator $P_{\sigma,a,b}$ which is defined as

$$(P_{\sigma,a,b}x)_i = x_{\sigma(i-a)}\omega^{\sigma bi}, \tag{3.12}$$

where the parameter b is uniformly random between 1 and N, σ is uniformly random odd between 1 and N, and a is 0 for the first hashing operation (HashToBins) and 1 for the second call to HashToBins. The constraining to odd values for σ is necessary in order for the permutation to be invertible.

With the use of some basic properties of the Fourier transform the following can be proved (page 5 of [HIKP12a]):

$$\widehat{P_{\sigma,a,b}x}_{\sigma(i-b)} = \hat{x}_i\omega^{a\sigma i}. \tag{3.13}$$

Later a multidimensional version of this equation is derived. Informally, this equation states the following: A permutation, defined by equidistant sub-sampling in the time domain in addition to applying a linear phase, results in a (different) permutation in the frequency domain with a (different) linear phase. By carefully choosing the parameters of (3.13) it is possible to design the permutation such that the phase difference between the two hashed coefficients is linear in frequency. This property is then used to recover the coefficient exactly by using the quotient of two measurements with different parameter a.

A high level overview of the functions that divide the key steps of the sFFT-3.0 algorithm are the following [HIKP12a]:

- HashToBins permutes the spectrum of $\widehat{x-z}$, then hashes to B bins. Where z is the already recovered signal which is initially all zero.

- NoiselessSparseFFTInner runs HashToBins twice and *estimates* and *locates* "most" of $\widehat{x-z}$'s coefficients.
- NoiselessSparseFFT runs NoiselessSparseFFTInner multiple times until it finds $\hat{x}$ exactly.

The function NoiselessSparseFFTInner generates the random parameters for the permutation (among others) and passes it to HashToBins. The permutations are $P_{\sigma,0,b}$ for the first call of HashToBins and $P_{\sigma,1,b}$ for the second call respectively. The number of bins is denoted by B and gradually reduced with each call of NoiselessSparseFFTInner. HashToBins performs the low pass filtering on the signal which has a complexity of $O(B \log N)$. By carefully reducing B per iteration the 1D sFFT algorithm runs in time $O(k \log N)$. The reader is advised to see [HIKP12a] for a more in depth description and proves of the 1D sFFT algorithm.

3.5 Multidimensional sparse FFT Algorithm

In this Section it is described how to extend the one dimensional sparse FFT from Section 3.4 to multiple dimensions. A comprehensive pseudo code description of the d-dimensional algorithm is given in Algorithm 3.3 on page 70. First it is described what extensions to the concepts are necessary. Consecutively the algorithm is described in more detail.

As stated earlier, for simplicity the symbols are reused and the notation is redefined for the d-dimensional case. Let x be an $[N]^d$-dimensional signal with sparsity k. It is assumed that each Fourier coefficient $\hat{x}_{\boldsymbol{i}} \in -L, \ldots, L$ where $L \leq N^c$ for some constant $c > 0$. Let the number of bins that are used to hash the coefficients be $[B]^d$.

The low pass filter –which has a general form approximating a rectangular in one dimension– needs to be extended to multiple dimensions. There are two popular and straightforward options:

1. hypersphere
2. hypercube

procedure HASHTOBINS(x, $\hat{z}$, $P_{M,a,b}$, B, δ, α)
  Compute $\hat{y}_{\boldsymbol{j}N/B}$ for $\boldsymbol{j} \in [B]^d$, where $y = G_{B,\alpha,\delta} \cdot (P_{M,a,b}x)$
  $\hat{y}'_{\boldsymbol{j}N/B} = \hat{y}_{\boldsymbol{j}N/B} - (\widehat{G'_{B,\alpha,\delta}} * \widehat{P_{M,a,b}z})_{\boldsymbol{j}N/B}$, $\boldsymbol{j} \in [B]^d$
  return $\hat{u}$ given by $\hat{u}_{\boldsymbol{j}} = \widehat{y'}_{\boldsymbol{j}N/B}$
end procedure
procedure NOISELESSSPARSEFFTINNER(x, k', $\hat{z}$, α)
  Let $B = k'/\beta$, for sufficiently small constant β.
  Let $\delta = 1/(4N^2L)$
  Choose $\boldsymbol{M}$ by Algorithm 3.1 ▷ precomputed
  Choose $\boldsymbol{b}$ uniformly at random from $[N]^d$.
  for $i \leftarrow 0, 1, \ldots, d$ **do**
    $a \leftarrow$ according to (3.17)
    $\hat{\boldsymbol{u}}_i \leftarrow$ HASHTOBINS$(x, \hat{z}, P_{M,a,b}, B, \delta, \alpha)$
  end for
  Compute $J = \{j : |\hat{u}_{0,j}| > 1/2\}$
  for $j \in J$ **do**
    for $p \leftarrow 1, 2, \ldots, d$ **do**
      $\boldsymbol{a}_{p-1} \leftarrow \phi(\hat{u}_{0,j}/\hat{u}_{p,j})$ ▷ $\phi(\cdot)$ denotes the phase
    end for
    $\nu \leftarrow \text{round}(\boldsymbol{a}\boldsymbol{M}^{-1}\frac{n}{2\pi})$ ▷ round$(\cdot)$ applied to each element
    Find v in $\boldsymbol{M}^T(v - b) \mod N = \nu$ ▷ See (3.18)
    $\hat{w}_v \leftarrow \text{round}(\hat{u}_o)$
  end for
  return $\hat{w}$
end procedure
procedure NOISELESSSPARSEFFT(x, k)
  $\hat{z} \leftarrow 0$
  for $t \in 0, 1, \ldots, \log k$ **do**
    $k_t \leftarrow k/2^t$
    $\alpha_t \leftarrow \Theta(2^{-t})$
    $\hat{z} \leftarrow \hat{z} +$ NOISELESSSPARSEFFTINNER$(x, k_t, \hat{z}, \alpha_t)$
  end for
  return $\hat{z}$
end procedure

Algorithm 3.3: Exact k-sparse d-dimensional algorithm.

The extension to multiple dimensions need to be performed very carefully due to the constraints of the filter. It is crucial that the filter has limited support in *both*, time domain as well as Fourier domain. It turns out that this poses a significant problem when defining the multi-dimensional filter which transition from one to zero not along a principal axis. For instance, a circular shaped low pass filter contains transitions along each direction (0° to 180°) whereas the rectangular filter only contains transitions along 0° and 90°.

Due to this unique limitation the low pass filter is defined as the dyadic product of the one dimensional low pass filter:

$$\boldsymbol{G}_{B,\alpha,\delta} = \bigotimes_{i=1}^{d} \boldsymbol{g'}_{B,\alpha,\delta} \tag{3.14}$$

where $\boldsymbol{g'}_{B,\alpha,\delta}$ denotes the same filter vector as described in Section 7 of [HIKP12a]. Thus approximating a hypercube.

For two dimensions this is defined as:

$$\boldsymbol{G}_{B,\alpha,\delta} = \begin{bmatrix} & & \\ \boldsymbol{g'}_{B,\alpha,\delta} & \cdots & \boldsymbol{g'}_{B,\alpha,\delta} \\ & & \end{bmatrix} \odot \begin{bmatrix} \boldsymbol{g'}^{T}_{B,\alpha,\delta} \\ \vdots \\ \boldsymbol{g'}^{T}_{B,\alpha,\delta} \end{bmatrix} \tag{3.15}$$

where $\odot$ denotes the element wise matrix multiplication. Note that the actual complexity of this is much less than the seemingly $O(N^d)$ due to the limited support of the vector $\boldsymbol{g'}$ which is $O(B \log N)$ in the one dimensional case. In d dimensions the support of $\boldsymbol{G}_{B,\alpha,\delta}$ is thus $O(B^d \log^d N)$

The fact that the phase difference between the two hashes is always a one dimensional entity even in a d-dimensional sample poses a problem. To be able to recover the frequencies in d-dimensions it is necessary to hash a total of $d+1$ times and encode each dimension in the calls $1, 2, \ldots, d$ to HashToBins. This allows to *locate* the coefficient in d-dimensions by decoding each frequency component along each dimension separately.

The most interesting part of the algorithm and the focus of this dissertation is the very first part of each outer iteration: The permutation. It is necessary to extend the permutation (3.13) to multiple dimensions which is done with the following definition of the permutation operator $P_{\boldsymbol{M},\boldsymbol{a},\boldsymbol{b}}$:

$$\left(P_{\boldsymbol{M},\boldsymbol{a},\boldsymbol{b}}\boldsymbol{x}\right)_v = x_{\boldsymbol{M}(\boldsymbol{v}-\boldsymbol{a})}\omega^{\boldsymbol{v}^T\boldsymbol{M}^T\boldsymbol{b}} \tag{3.16}$$

where $\boldsymbol{M}$ is a matrix of size $d \times d$ that stretches the input signal $\boldsymbol{x}$. And $\boldsymbol{a}$ and $\boldsymbol{b}$ are the d-dimensional vectors counterparts of the one dimensional definition in (3.13). Note also that all of the vectors and matrices in (3.16) only contain integers. The definition works for arbitrary dimensions. In order to recover the original Fourier coefficients, i.e. perform a reversible permutation the determinant of the matrix $\boldsymbol{M}$ needs be odd. For two dimensions, the matrix $\boldsymbol{M}$ can be interpreted as applying a shear and scale to the multidimensional input signal. This interpretation allows for some intuition regarding the optimal parameter: If the parameters are chosen randomly, and the shear along one dimension happens to dominate the transformation, what happens to an isotropic input spectrum? The answer is that in such an unfortunate case the permuted spectrum would results in a "banding" like accumulation of Fourier coefficients. This can be fatal for the performance of the algorithm since such bands of accumulated coefficients results in many collisions. The answer in choosing the optimal matrix $\boldsymbol{M}$ lies in the proposed Algorithm 3.1.

Furthermore, the parameter $\boldsymbol{a}$ is the only parameter that differs among each call to HashToBins. For a given iteration i and vector index q:

$$a_{i,q} = \begin{cases} 0 & \text{if } i = 0 \\ 0 & \text{if } i \neq 0 \text{ and } q \neq i \\ 1 & \text{if } i \neq 0 \text{ and } q = i \end{cases} \tag{3.17}$$

where the iteration index i ranges from 0 to d.

Another part of the multi-dimensional algorithm requiring special attention is one step within NoiselessSparseFFTInner where the inverse to above permutation is needed. Again, the extension to multiple dimensions is not straightforward. Firstly, note that in the one dimensional sparse FFT algorithm the inverse is found with the extended Euclidean algorithm with the constraint of σ being odd which is the counterpart of the constraint that the determinant of the permutation matrix is odd. In order to simplify this Section the focus is only on the expression $\boldsymbol{M}\boldsymbol{v}$. The result is straightforward to extend to the form of (3.16). Remember that all elements in $\boldsymbol{M}$ are integers and further $\det \boldsymbol{M}$ must be odd for the expression to be a bijection. We are interested in finding $\boldsymbol{v}$ for

$$\boldsymbol{M}\boldsymbol{v} \mod N = \boldsymbol{y}$$

Where the modulo appears due to the simple fact that the DFT of a signal is implicitly periodic with the signal length N.

This problem turns out to be a congruence equation which can be solved by reducing this problem to a linear Diophantine equation [Mor69]. First, a trick is used to get rid of the modulo operation:

$$\begin{aligned}
\begin{bmatrix} m_{1,1} & \cdots & m_{1,d} \\ \vdots & \ddots & \vdots \\ m_{d,1} & \cdots & m_{d,d} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} \mod N &= \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix} \\
\begin{bmatrix} m_{1,1} & \cdots & m_{1,d} & \mu_1 \\ \vdots & \ddots & \vdots & \vdots \\ m_{d,1} & \cdots & m_{d,d} & \mu_d \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_d \\ N \end{bmatrix} &= \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix} \\
\boldsymbol{M}'\boldsymbol{v}' &= \boldsymbol{y}
\end{aligned} \tag{3.18}$$

where $\boldsymbol{\mu}$ is a vector of arbitrary integers. In this simple form the solution to the linear Diophantine equation can be found by computing the Hermite normal form of the matrix $\boldsymbol{M}'$ [Mar99]. Note that the complexity of computing the Hermite normal form is

quite high with $O(d^6)$. However, the dimensionality is usually quite low and considered constant for the complexity of the sparse FFT algorithm. Thus, it does not increase the complexity of the overall sparse FFT algorithm. In fact, the permutation matrices as well as its inverse permutation obtained with the HNF form can be precomputed by an application.

Next the relationship between the original signal $\boldsymbol{x}$ and the effect of the permutation (3.16) to the spectrum is derived:

$$\begin{aligned}
(\widehat{P_{M,a,b}x})_{M^T(v-b)} &= \sum_{u\in[N]^d} \omega^{u^T M^T (v-b)} (P_{M,a,b}x)_u \\
&= \sum_{u\in[N]^d} \omega^{u^T M^T (v-b)} x_{M(u-a)} \omega^{u^T M^T b} \\
&= \omega^{v^T Ma} \sum_{u\in[N]^d} \omega^{v^T M(u-a)} x_{M(u-a)} \\
&= \hat{x}_v \omega^{v^T Ma}
\end{aligned} \tag{3.19}$$

Again, this states that the applied permutation (3.16) in the time domain results in a permutation in the Fourier domain. Both, the frequency as well as the time domain, also apply a (different) phase, which is exactly what is needed for the algorithm to function by encoding the frequency in the phase component.

To conclude this Section, a high level overview of the key steps of the 2D sparse FFT algorithm is depicted in Figure 3.5.

3.5.1 Complexity Analysis

In this Section the time complexity of the d-dimensional sparse FFT is analyzed. The time complexity of the d-dimensional sparse FFT algorithm is similar to the one dimensional algorithm. Besides the straightforward extensions of the various parameters and entities of the one dimensional sparse FFT to multiple dimensions, there are a few steps that require more careful attention.

First, the $d+1$ calls to HashToBins which encode the d dimensions of each coefficient need to be taken into account. Secondly, the d-dimensional FFT within

HashToBins results in a complexity of $O(B^d \log B)$. Taking into account that B is chosen as $O(k^{1/d})$ the result is a complexity of $O(k \log k^{1/d})$ for the FFT calculation within HashToBins.

The main cost of the algorithm is the first iteration. Within this first iteration the application of the time domain filter is the dominating cost with $O(B^d \log^d N)$. Again, taking into account that B is chosen as $O(k^{1/d})$ the result is a complexity of $O(k \log^d N)$

Thus, given that $d+1$ iterations of HashToBins is necessary for one iteration and the fact that the complexity is dominated by the first iteration of NoiselessSparseFFTInner, the algorithm has an overall time complexity of $O(k \log^d N)$.

3.6 Results

The proposed algorithms are validated by simulating the sparse FFT algorithm in two dimensions. To this end the 2D sFFT algorithm was implemented in MATLAB. The simulation setup was as follows: Input spectra were generated with a given sparsity k. Each spectrum had an isotropic shape which is the most common shape among real world signals and should be assumed if no a-priori knowledge of the input signal is given. One iteration of the overall algorithm is run, i.e. one call to NoiselessSparseFFTInner. This allows one to compare the performance of the permutation. As an error measure the PSNR of the recovered signal after one iteration is calculated. A good permutation will reduce the number of collisions and result in a higher PSNR. Also note that the algorithm runtime is dominated by the first iteration.

Algorithm 3.1 is used to generated candidates for the permutation matrix of Algorithm 3.3. Each candidate is evaluated with 40 generated input spectra. An example of the evaluation of 50 candidates is depicted in Figure 3.6. As a comparison 60 random candidates where evaluated and are depicted on the top image of Figure 3.6. It can be seen that our proposed algorithm performs better finding a better maximum PSNR.

Figure 3.7 shows the improvements for the proposed algorithm for different input sizes N. Again, the PSNR is compared to choosing random permutation matrices. It can be seen that the proposed algorithm finds permutation matrices that improve the PSNR of the sparse FFT algorithm by roughly 2dB across the shown input sizes.

Similarly, Figure 3.8 depicts the PSNR improvement over different sparsity k for and input spectrum of size $N \times N = 8192 \times 8192$. Again, the improvement is around 2dB compared to a random permutation strategy.

Figure 3.9 depicts the histogram of PSNR values over 2000 simulations. The top histogram shows the distribution of the PSNR with the permutation obtained from the proposed algorithm whereas the bottom shows the optimal permutation obtained from randomly selected permutations. Again, 40 iterations where employed. This shows that the PSNR values follow a Gaussian like distribution which is desirable for real world applications that require predictable performance.

Next, a different approach is investigated. Instead of iterating over the sequence generated by Algorithm 3.1 and finding the optimal permutation the strategy of using a random sample of the sequence is used and is compared to taking a completely random permutation matrix. This turns out to be a better strategy since it avoids very poor permutations which in turn result in very poor PSNR. The result is depicted in Figure 3.10. The histogram shows the PSNR of 600 simulations of an input spectrum of 8192×8192 and a sparsity of $k = 400$. This histogram shows that the very poor PSNR values are avoided. Note that this does not increase the runtime since picking a random element of the sequence of Algorithm 3.1 does not require generating the entire sequence and is thus a very cheap (constant time) operation. Thus, this strategy could be employed by libraries which have no prior knowledge of the input data.

Furthermore, a comparison of the minimum PSNR is shown in Table 3.2 on page 77. This table shows that a randomly chosen permutation can result in extremely poor performance. The proposed method mitigates these poor permutations successfully which is crucial for real world applications of the sparse FFT.

Concluding, an example of applying the two dimensional sparse FFT to an

Table 3.2: This table show the improvement of the proposed algorithm by avoiding collision generating parameters which can result in a very poor PSNR. 600 input spectra were generated and one iteration of the algorithm was run. The table show the minimum PSNR across all 600 simulations. σ_p and σ_r show the corresponding standard deviations of the proposed method and the random method respectively.

Sparsity k	Min. PSNR proposed	Min. PSNR random
200	22.93	8.35
300	24.49	13.47
400	23.50	10.05
600	22.52	9.85

image is shown. Figure 3.11 on page 85. shows a crop of 1000×1000 pixels of an image of 32768×32768. Note that the sparse FFT was independently applied to each RGB channel.

3.7 Conclusion

In this chapter the one dimensional exact sparse FFT introduced in [HIKP12a] is extended to multiple dimensions. Further the focus is on the permutation part of the algorithm which is the main subtlety and crucial to achieve good performance. This is well known in the Computer Science community where good performance for many algorithms are obtained by minimizing the collision rate of the hashing functions [CLR+01]. The focus is on the shape of the spectra of real world signals and it is shown that the performance can be optimized by carefully choosing the permutation parameters as opposed to the widely spread notion of randomly choosing the parameters.

The permutation operation is interpreted as generating a lattice which helps to argue the optimal parameters for the shape of many real world signal spectra. The results showed successfully that the proposed method avoids poor hashing permutation which can result in many collision. This is crucial in real world applications which often require a reliable performing algorithm.

Furthermore, the novel connection of lattice theory to the permutation step of the algorithm could lead to more research in this area. A clear understanding of the permutation and its inverse for the general d-dimensional case was established by solving a system of linear congruence equations.

Further, an algorithm was proposed which modifies a matrix to ensure that the determinant is odd. This novel algorithm was necessary in order for the congruence equations to be invertible.

Practical guidelines were established for the permutation parameters in our proposed algorithm which are optimized for real world signals. The proposed method successfully avoids "bad" hashing permutation parameters which results in a more robust and consistent performing algorithm.

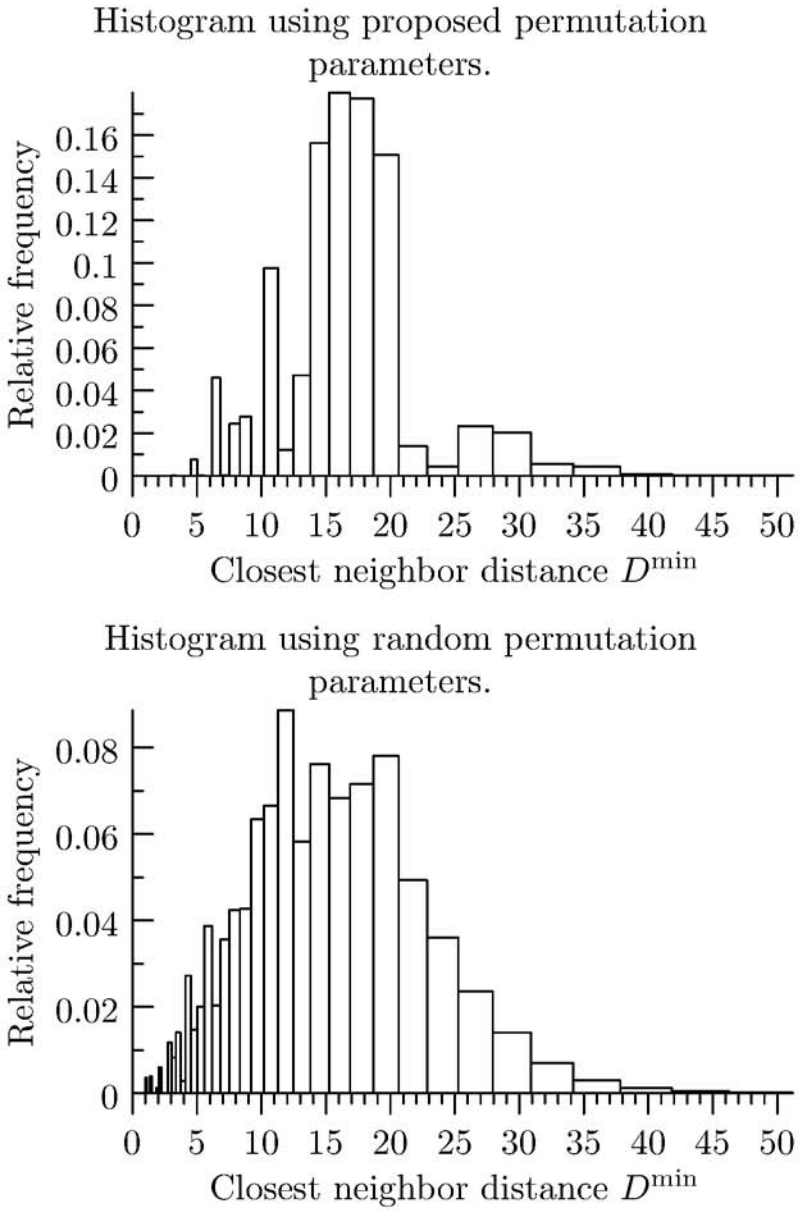

Figure 3.4: Two histograms depicting the difference between using different methods of choosing permutation parameters. Over 400 input spectra are generated in the shape of Figure 3.3 and permuted them either randomly (bottom) or according the dual lattice method (top). The distance between each non-zero coefficient in the spectrum and its nearest neighbor is measured of how "good" a permutation is. For this the Euclidean norm is used. The top image has a mean minimum distance of 16.1 which is a 21% improvement over using random permutation parameters (bottom) which has a mean minimum distance of 13.3.

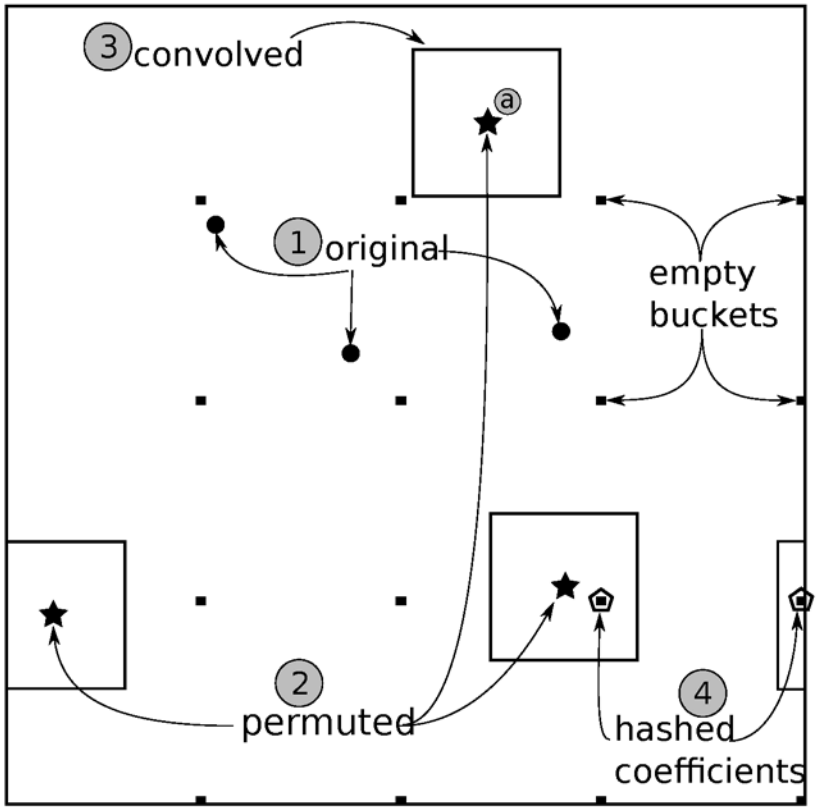

Figure 3.5: Conceptual depiction of the steps performed in HashToBins. The original spectrum (1) has only three non-zero coefficients ($k = 3$) which are then permuted (2) and convolved with the low pass filter (3). Note that only two coefficients are hashed (4) and the third (a) is missed. There is no collisions in this particular example which could occur if the spectrum overlaps with neighboring coefficients and the area is hashed.

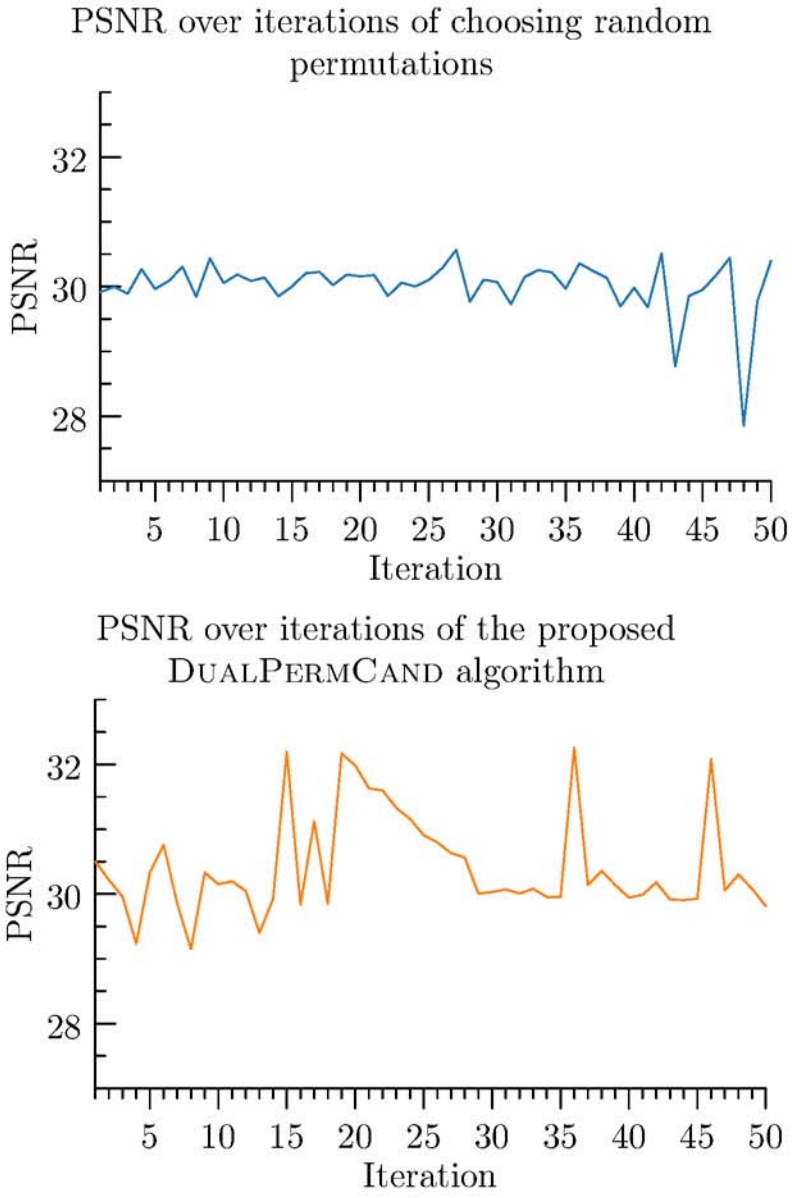

Figure 3.6: An example graph depicting the PSNR over 50 iterations. The PSNR was calculated as the average over 40 generated input spectra. The input size $N \times N = 8192 \times 8192$ and the sparsity $k = 1600$. *Top*: In each iteration a randomly generated permutation *Bottom*: A subset of the proposed method DualPermCandidates was used. Note: With only very few iteration the proposed algorithm finds a very good permutation matrix. The proposed algorithm also avoids bad permutation with low PSNR. A random strategy might or might not find a good permutation. This non-deterministic behavior is undesirable for real applications.

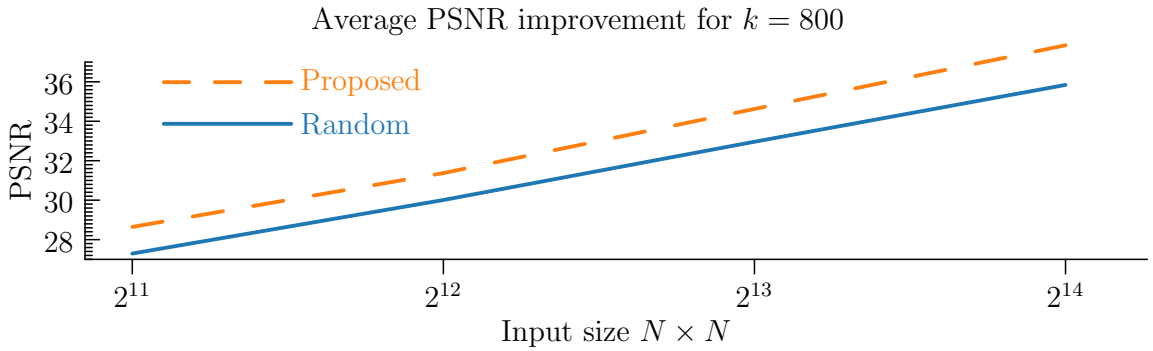

Figure 3.7: 40 iterations of the proposed algorithm are used and compared it with a random strategy. The PSNR was calculated as the average over 40 generated input spectra. For the each iteration the average PSNR was calculated and the best performing permutation matrix chosen. For the shown graph the sparsity k was kept constant at 800. The graph shows that the proposed method improves the PSNR by roughly 2dB.

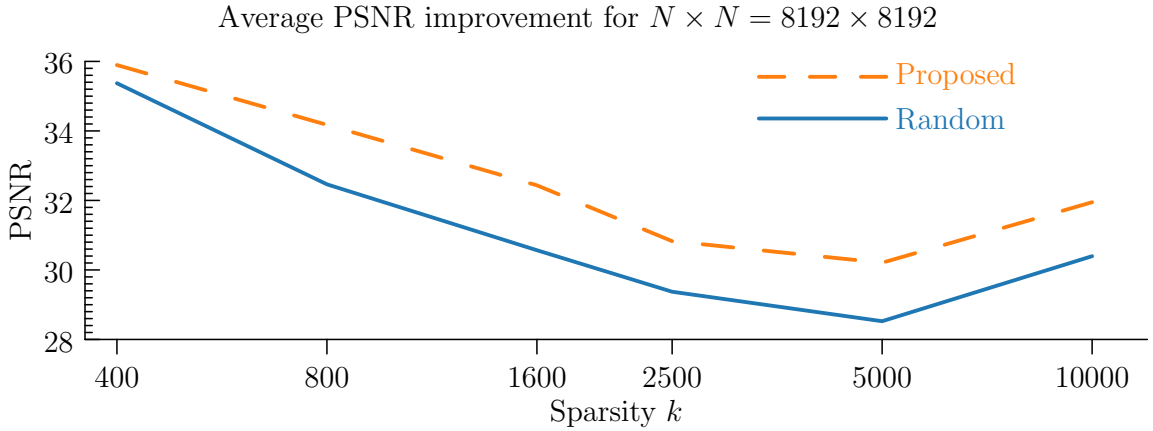

Figure 3.8: This graph shows the improvement in PSNR for an input spectrum of $N \times N = 8192 \times 8192$ with different signal sparsity k ranging from 400 to 10000. The test setup is the same as the one of Figure 3.7.

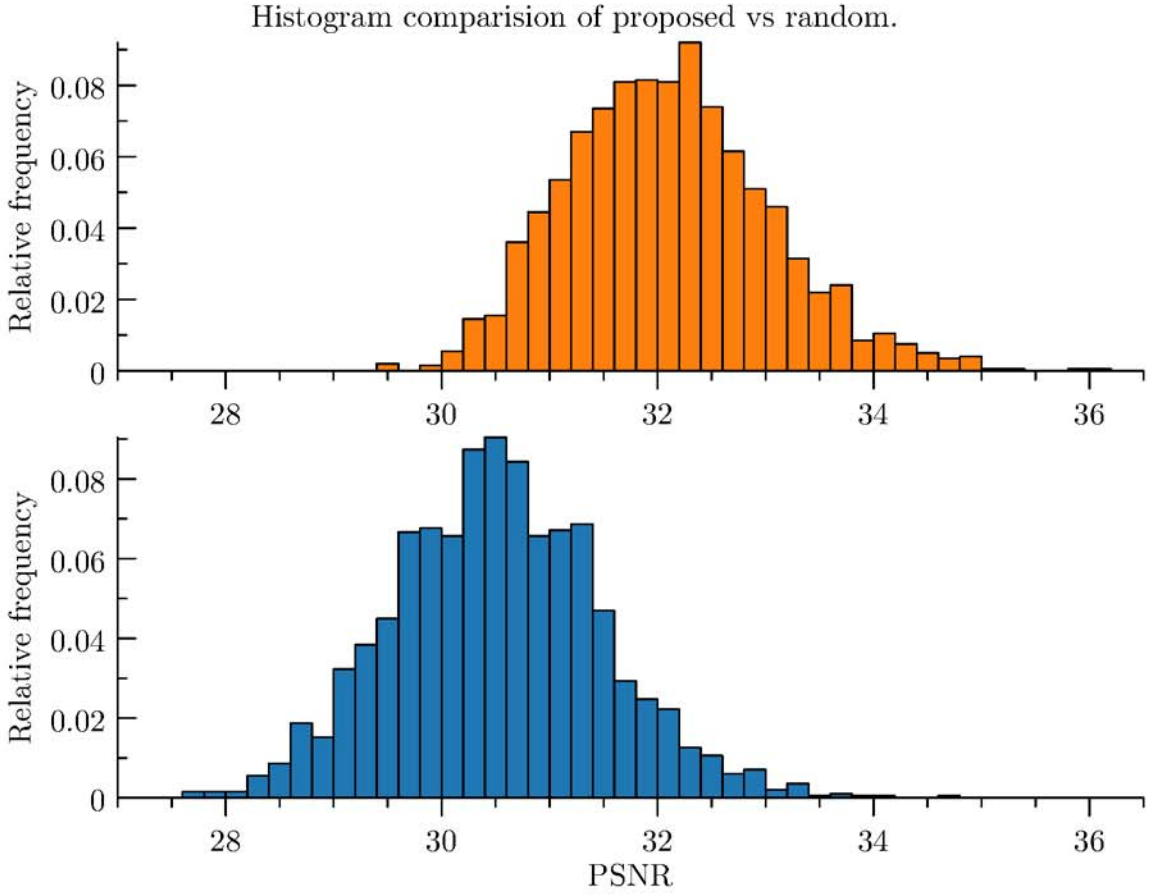

Figure 3.9: This graph compares two histograms obtained from the optimal permutation matrix from the proposed method (top) and the optimal matrix obtained from randomly generating permutation matrices (bottom). The histogram shows the distribution of 2000 generated input spectra. It can be seen that the PSNR of the proposed method is well contained and improves upon the random permutation by roughly 2dB.

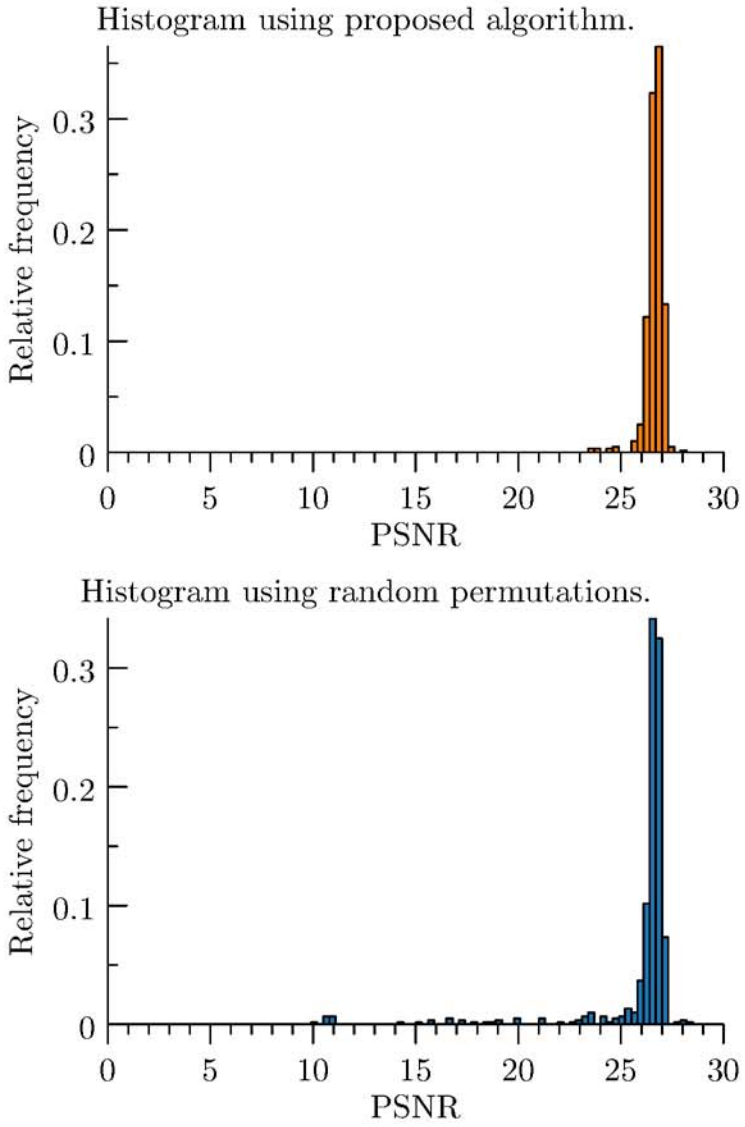

Figure 3.10: Two histograms depicting the difference between using different methods of choosing permutation parameters. Over 600 input spectra are generated with an isotropic input spectrum with a sparsity $k = 400$ and dimensions of 8192×8192. The PSNR is measured after one iteration of the sFFT algorithm in order to compare the performance of the permutation. *Top*: Choosing a random element of the DUALPERM-CANDIDATES procedure as the permutation matrix. *Bottom*: Choosing a completely random permutation element. The top histogram shows that the PSNR is concentrated and successfully avoids very poor permutations which can occur with random parameters (bottom) and result in unpredictable algorithmic performance. The average PSNR is 26.65dB on the top and 25.97dB on the bottom. The minimum PSNR is 23.50dB on the top and 10.05dB on the bottom.

Figure 3.11: *Top*: A 1000×1000 pixel crop of a 32768×32768 image with a sparsity of 1%. *Bottom*: The image after running the proposed sFFT algorithm. The PSNR is 27.81dB when compared to the original image.

Chapter 4

DICUSSION AND CONCLUDING REMARKS

The main contributions of this dissertation are as follows:

- Introduce four novel algorithms.
- Develop a theoretical understanding of the inner parts of two exiting algorithms (weighted median, sparse FFT).
- Develop theoretical models.
- Optimize these models in order to optimize these algorithms.

The essence of the presented work was to optimize existing algorithm through the development of theoretical models. To this end, novel algorithms were introduced in order to achieve this optimization of the existing algorithms.

In particular, in the first part of this dissertation a new algorithm has been proposed to solve the weighted median problem. The weighted median is an increasingly used tool to solve signal processing problems containing impulsive noise. This fact motivated the development of an algorithm which is asymptotically optimal and is the fastest known algorithm to this date, beating an algorithm that has been the fastest for over four decades. A theoretical framework was developed which was based on random variables and a cost function. The idea of optimizing a cost function is an old one and has been suggested many times in the past decades. However, a closed form solution for the optimal parameters has never been discovered due to the complexity of the cost functions. This dissertation achieved the feat of finding a closed form solution by approximating the cost function. Careful approximation lead to a very accurate closed

form solution which was verified by investigating the error of the approximations. Further, the model turned out to be viable for finding the optimal order statistic of a subset to be used as the second pivot of the weighted median finding algorithm. The result was a novel algorithm which is based on the well known Quickselect algorithm having closed form solutions for the optimal parameters. Simulations for the error and the runtime showed the effectiveness of the proposed algorithm. The algorithm is not limited to weighted median but can also be used to calculate the order statistics of a given input set. This is due to the simple fact that the weighted median algorithm is a more general version of the Quickselect algorithm. Thus by assuming all weights as equal one can implement a specific version of the proposed algorithm to find the k^{th} smallest value of a set.

The sparse FFT algorithm performs the well known DFT algorithm on sparse signals. This problem is motivated by the fact that most natural signals are sparse in the frequency domain. In addition, the Fourier coefficients of such natural signals often inhibit a structure and are not randomly distributed. This fact motivated a key observation: Given a non-random distribution of the Fourier coefficients, what are the best parameters for the sparse FFT? The main focus was thus shifted on the permutation step of the sparse FFT algorithm. In general, for hashing type algorithms, the main performance measure for the "goodness" of a hashing operation is the number of collisions. In order to minimize the collisions the distribution of the Fourier coefficients after the hashing operation is desired to be as uniform as possible. The permutation step was thus identified to be the main step of the algorithm to focus on. Similar to the weighted median algorithm, a theoretical model was developed in order to optimize the permutation step. To this end, lattice theory was connected to the permutation step and a novel iterative algorithm was developed which reduces the search space for possible permutation matrices from exponential to linear.

In order to evaluate the new findings, the existing one dimensional sparse FFT was extended to multiple dimensions. To this end, each step of the sparse FFT was

investigated and generalizations developed for solving the multidimensional sparse FFT problem. For instance, the Hermite normal form is now used to invert the permutation. Another novel algorithm was proposed to turn an integer matrix with even determinant into a similar integer matrix with odd determinant. This algorithm was proposed due to the need for the determinant to be odd for the permutation matrix. To this end the algorithm flips as few entries' bits of the matrix to achieve an odd determinant.

Concluding, the proposed findings and algorithms allowed for a novel multidimensional sparse FFT algorithm which performs significantly better than with using random permutations. This was shown by combining and simulating the proposed algorithms. For each algorithm the complexity and error was analyzed.

4.1 Recommendations for Future Work

This dissertation has proposed multiple new algorithms as well as presented multiple new findings and theoretical models. In this section, we discuss possible future direction that could be of interest for future research.

As it was noted in the dissertation, finding the (weighted) median of a set is at least a linear algorithm even in the best case. This can become prohibitive with even larger data sets. It is thus of interest to develop an algorithm to find an approximate solution to the (weighted) median problem. This is quite similar to the sparse FFT which does exactly that: Solving the problem by working with only a subset of the input data. I believe there is merit in finding good guarantees for an approximate algorithm.

Another interesting topic that is becoming increasingly important with nowadays' big-data movement is distributed computing. For a distributed algorithm the permutation step happens in parallel and communication should be kept to a minimum to minimize the delay. The pivot question then becomes more interesting: Should the pivots be chosen as close as possible to the sought element? Should the pivots try to contain (i.e. achieve a bounded problem) the sought elements?

For the sparse FFT the introduced connection to lattice theory for the optimal permutation may allow other researchers to further combine these ideas to other algorithms. In particular, a closed form solution –similar to the one I give for the weighted median problem– is desirable. Given an optimal permutation which minimizes the number of collisions, what other trade offs can we achieve? For instance, we can trade the improved performance in PSNR with a reduction in the number of bins which in turn reduce the runtime. How much faster can we thus solve the sparse FFT problem?

BIBLIOGRAPHY

[AB07] T. C. Aysal and K. E. Barner. Meridian filtering for robust signal processing. *IEEE Transactions on Signal Processing*, 55(8):3949–3962, 2007.

[ABN08] Barry C. Arnold, N. Balakrishnan, and H. N. Nagaraja. *A First Course in Order Statistics (Classics in Applied Mathematics)*. SIAM, 2008.

[AF89] G. R. Arce and R. E. Foster. Detail-preserving ranked-order based filters for image processing. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(1):83–98, jan 1989.

[AG82] G. R. Arce and N. C. Gallagher. State description for the root-signal set of median filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 30(6):894–902, 1982.

[AG83] G. R. Arce and N. C. Gallagher. BTC image coding using median filter roots. *IEEE Transactions on Communications*, 31(6):784–793, 1983.

[AGS03] Aadi Akavia, Shafi Goldwasser, and Shmuel Safra. Proving hard-core predicates using list decoding. In *Annual Symposium on Foundations of Computer Science*, volume 44, pages 146–159. IEEE COMPUTER SOCIETY PRESS, 2003.

[Ajt98] Miklós Ajtai. The shortest vector problem in l2 is np-hard for randomized reductions (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 10–19, New York, NY, USA, 1998. ACM.

[AK97] J. Astola and P. Kuosmanen. *Fundamentals of nonlinear digital filtering*. CRC, 1997.

[AM87] G. R. Arce and M. P. McLoughlin. Theoretical analysis of the max/median filter. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(1):60 – 69, January 1987.

[AP00] G. R. Arce and J. L. Paredes. Recursive weighted median filters admitting negative weights and their optimization. *IEEE Transactions on Signal Processing*, 48(3):768–779, 2000.

[Apo13] Tom M Apostol. *Introduction to analytic number theory*. Springer Science & Business Media, 2013.

[Arc91] G. R. Arce. Multistage order statistic filters for image sequence processing. *IEEE Transactions on Signal Processing*, 39(5):1146–1163, 1991.

[Arc02] G. R. Arce. A general weighted median filter structure admitting negative weights. *IEEE Transactions on Signal Processing*, 46(12):3195–3205, 2002.

[Arc05] G. R. Arce. *Nonlinear Signal Processing: A Statistical Approach*. John Wiley & Sons, Inc., New York, NY, USA, 2005.

[AS87] M. Ahmad and D. Sundararajan. A fast algorithm for two dimensional median filtering. *Circuits and Systems, IEEE Transactions on*, 34(11):1364 – 1374, November 1987.

[AZJB06] WB Atwood, M Ziegler, RP Johnson, and BM Baughman. A time-differencing technique for detecting radio-quiet gamma-ray pulsars. *The Astrophysical Journal Letters*, 652(1):L49, 2006.

[BA94] K. E. Barner and G. R. Arce. Permutation filters: A class of nonlinear filters based on set permutations. *IEEE Transactions on Signal Processing*, 42(4):782–798, 1994.

[BFP+73] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection*. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

[BLA79] V. Barnett, T. Lewis, and F. Abeles. Outliers in statistical data. *Physics Today*, 32:73, 1979.

[BS80] P. Bloomfield and W. Steiger. Least absolute deviations curve-fitting. *SIAM Journal on Scientific and Statistical Computing*, 1(2):290–301, 1980.

[CLR+01] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

[CM73] J. F. Claerbout and F. Muir. Robust modeling with erratic data. *Geophysics*, 38:826, 1973.

[CM13] Anirban Chatterjee and GK Mahanti. Combination of fast fourier transform and self-adaptive differential evolution algorithm for synthesis of phase-only reconfigurable rectangular array antenna. *annals of telecommunications-annales des télécommunications*, pages 1–13, 2013.

[CP54] Herman Y Carr and Edward M Purcell. Effects of diffusion on free precession in nuclear magnetic resonance experiments. *Physical Review*, 94(3):630, 1954.

[CRT06] Emmanuel J Candes, Justin K Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on pure and applied mathematics*, 59(8):1207–1223, 2006.

[CT65] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[CW03] Scott A Crosby and Dan S Wallach. Denial of service via algorithmic complexity attacks. In *Usenix Security*, volume 2, 2003.

[CWB08] E. J. Candes, M. B. Wakin, and S. P. Boyd. Enhancing sparsity by reweighted L1 minimization. *Journal of Fourier Analysis and Applications*, 14(5):877–905, 2008.

[Dam09] Sabrina Dammertz. *Rank-1 Lattices in Computer Graphics*. PhD thesis, Ulm University, Germany, 2009.

[DDK09] Sabrina Dammertz, Holger Dammertz, and Alexander Keller. Efficient search for two-dimensional rank-1 lattices with applications in graphics. In *Monte Carlo and Quasi-Monte Carlo Methods 2008*, pages 271–287. Springer, 2009.

[DN03] H. A. David and H. N. Nagaraja. *Order statistics*. Wiley series in probability and mathematical statistics. John Wiley & Sons, Inc., 2003.

[DS89] D. L. Donoho and P. B. Stark. Uncertainty principles and signal recovery. *SIAM Journal on Applied Mathematics*, 49(3):906–931, 1989.

[EDM09] Alireza Entezari, Ramsay Dyer, and Torsten Möller. From sphere packing to the theory of optimal lattice sampling. In *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, pages 227–255. Springer, 2009.

[EF13] Anders Elowsson and Anders Friberg. Modelling perception of speed in music audio. *Forthcoming for Proc. of SMC*, 2013.

[FAB98] A. Flaig, G. R. Arce, and K. E. Barner. Affine order-statistic filters: Medianization of linear FIR filters. *IEEE Transactions on Signal Processing*, 46(8):2101–2112, 1998.

[FPA02] M. Fischer, J. L. Paredes, and G. R. Arce. Weighted median image sharpeners for the World Wide Web. *IEEE Transactions on Image Processing*, 11(7):717–727, 2002.

[FR75] R. W. Floyd and R. L. Rivest. Expected time bounds for selection. *Commun. ACM*, 18(3):165–172, 1975.

[FR13] Simon Foucart and Holger Rauhut. *A mathematical introduction to compressive sensing*. Springer, 2013.

[GA01] J. G. Gonzalez and G. R. Arce. Optimality of the myriad filter in practical impulsive-noise environments. *IEEE Transactions on Signal Processing*, 49(2):438–441, 2001.

[Gau31] Carl Friedrich Gauß. Besprechung des buchs von la seeber: Intersuchungen über die eigenschaften der positiven ternären quadratischen formen usw. *Göttingsche Gelehrte Anzeigen*, 2:188–196, 1831.

[GGI+02] Anna C. Gilbert, Sudipto Guha, Piotr Indyk, S. Muthukrishnan, and Martin Strauss. Near-optimal sparse fourier representations via sampling. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 152–161. ACM, 2002.

[GMS05] Anna C Gilbert, S Muthukrishnan, and Martin Strauss. Improved time bounds for near-optimal sparse fourier representations. In *Optics & Photonics 2005*, pages 59141A–59141A. International Society for Optics and Photonics, 2005.

[GST+08] Anna C Gilbert, Martin J Strauss, Joel Tropp, et al. A tutorial on fast fourier sampling. *Signal processing magazine, IEEE*, 25(2):57–66, 2008.

[GW81] N. C. Gallagher and G. L. Wise. A theoretical analysis of the properties of median filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 29(6):1136 – 1141, December 1981.

[HAKI12] Haitham Hassanieh, Fadel Adib, Dina Katabi, and Piotr Indyk. Faster gps via the sparse fourier transform. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 353–364. ACM, 2012.

[HIKP12a] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly optimal sparse fourier transform. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 563–578. ACM, 2012.

[HIKP12b] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Simple and practical algorithm for sparse fourier transform. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1183–1194. SIAM, 2012.

[HKPV13] Sabine Heider, Stefan Kunis, Daniel Potts, and Michael Veit. A sparse prony fft. 2013.

[Hoa61] C. Hoare. Find (algorithm 65). *Commun. ACM*, pages 4:321–322, 1961.

[HSA+14] Haitham Hassanieh, Lixin Shi, Omid Abari, Ezzeldin Hamed, and Dina Katabi. Ghz-wide sensing and decoding using the sparse fourier transform. In *INFOCOM, 2014 Proceedings IEEE*, pages 2256–2264. IEEE, 2014.

[HYT79] T. Huang, G. Yang, and G. Tang. A fast two-dimensional median filtering algorithm. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 27(1):13 – 18, February 1979.

[Iwe10] Mark A Iwen. Combinatorial sublinear-time fourier algorithms. *Foundations of Computational Mathematics*, 10(3):303–338, 2010.

[KA98] S. Kalluri and G. R. Arce. Adaptive weighted myriad filter algorithms for robust signal processing in α-stable noise environments. *IEEE Transactions on Signal Processing*, 46(2):322–334, 1998.

[KA99] S. Kalluri and G. R. Arce. A general class of nonlinear normalized adaptive filtering algorithms. *IEEE Transactions on Signal Processing*, 47(8):2262–2272, 1999.

[KA00] S. Kalluri and G. R. Arce. Fast algorithms for weighted myriad computation by fixed-point search. *IEEE Transactions on Signal Processing*, 48(1):159–171, 2000.

[Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 193–206, New York, NY, USA, 1983. ACM.

[KM93] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

[KMM76] Charles Kittel, Paul McEuen, and Paul McEuen. *Introduction to solid state physics*, volume 8. Wiley New York, 1976.

[Knu71] D. E. Knuth. *Mathematical Analysis of Algorithms*. Storming Media, 1971.

[LA04] Y. Li and G. R. Arce. A maximum likelihood approach to least absolute deviation regression. *EURASIP Journal on Applied Signal Processing*, 2004:1762–1769, 2004.

[LLL82] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[MA87] M. P. McLoughlin and G. R. Arce. Deterministic properties of the recursive separable median filter. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(1):98–106, 1987.

[Mal80] C. L. Mallows. Some theory of nonlinear smoothers. *The Annals of statistics*, 8(4):695–715, 1980.

[Man92] Yishay Mansour. Randomized interpolation and approximation of sparse polynomials. In *Automata, languages, and programming: 19th international colloquium, Wien, Austria, July 13-17, 1992: proceedings*, volume 623, page 261. Springer, 1992.

[Mar99] Richard Kipp Martin. *Large Scale Linear and Integer Optimization: A Unified Approach: A Unified Approach*. Springer Science & Business Media, 1999.

[Mor69] Louis Joel Mordell. *Diophantine equations*, volume 30. Academic Press, 1969.

[MPV04] C. Martínez, D. Panario, and A. Viola. Adaptive sampling for quickselect. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 447–455. SIAM Philadelphia, PA, USA, 2004.

[MR02] C. Martínez and S. Roura. Optimal sampling strategies in quicksort and quickselect. *SIAM Journal on Computing*, 31(3):683–705, 2002.

[Nar09] P. M. Narendra. A separable median filter for image noise smoothing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-3(1):20–29, 2009.

[NS01] Phong Q Nguyen and Jacques Stern. The two faces of lattices in cryptology. In *Cryptography and lattices*, pages 146–180. Springer, 2001.

[OLBC10] F.W. Olver, D.W. Lozier, R.F. Boisvert, and C.W. Clark. *NIST handbook of mathematical functions*. Cambridge Univ. Press NY, 2010.

[PA99] J. L. Paredes and G. R. Arce. Stack filters, stack smoothers, and mirrored threshold decomposition. *IEEE Transactions on Signal Processing*, 47(10):2757–2767, 1999.

[PA11] J. L. Paredes and G. R. Arce. Compressive sensing signal reconstruction by weighted median regression estimates. *IEEE Transactions on Signal Processing*, 59(6), June 2011.

[PH07] S. Perreault and P. Hébert. Median filtering in constant time. *IEEE Trans. on Image Processing*, 16(9):2389–2394, September 2007.

[RL87] P. J. Rousseeuw and A. M. Leroy. *Robust regression and outlier detection*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 1987.

[SAH+13] Lixin Shi, O Andronesi, Haitham Hassanieh, Badih Ghazi, Dina Katabi, and Elfar Adalsteinsson. Mrs sparse-fft: Reducing acquisition time and artifacts for in vivo 2d correlation spectroscopy. In *ISMRM'13, Int. Society for Magnetic Resonance in Medicine Annual Meeting and Exhibition*, 2013.

[SB89] Josef Stoer and Roland Bulirsch. *Numerische Mathematik*, volume 8. Springer, 1989.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical computer science*, 53(2):201–224, 1987.

[SL09] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81, 2009.

[Tim99] Samson J Timoner. *Subpixel motion estimation from sequences of video images*. PhD thesis, Massachusetts Institute of Technology, 1999.

[TLP+13] Kok Kiong Tan, Wenyu Liang, Le Phuong Pham, Hsueh Yee Lim, and Chee Wee Gan. Mechatronic design of an office-based ventilation tube applicator for patients with otitis media with effusion. In *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on*, pages 1448–1453. IEEE, 2013.

[YHAN91] O. Yli-Harja, J. Astola, and Y. Neuvo. Analysis of the properties of median and weighted median filters using threshold logic and stack filter representation. *IEEE Transactions on Signal Processing*, 39(2):395–410, 1991.

[YMCX13] Suzhen Yuan, Xia Mao, Lijiang Chen, and Yuli Xue. Quantum digital image processing algorithms based on quantum measurement. *Optik-International Journal for Light and Electron Optics*, 2013.

[YYGN96] L. Yin, R. Yang, M. Gabbouj, and Y. Neuvo. Weighted median filters: a tutorial. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 43(3):157–192, 1996.

[ZF03] Barbara Zitova and Jan Flusser. Image registration methods: a survey. *Image and vision computing*, 21(11):977–1000, 2003.

Appendix A

OPTIMALITY PROOF FOR MEDIAN SEARCH

This proofs that $C(N_0 \to \infty) = 1.5N_0$ for $k = (N_0 + 1)/2$ (median):

Proof 4 *As $N_0 \to \infty$ then $M_0 \to \infty$ and hence $p_1 \to$ MEDIAN($\boldsymbol{X}$). This removes $N_0/2$ elements with cost N_0.*

As $M_0 \to \infty$ then $M_1 \to \infty$. As $p_1 \to$ MEDIAN($\boldsymbol{X}$) then either $\alpha \to 0$ or $\alpha \to 1$. Hence either $k^* \to 1$ (since $\alpha \to 0$) or $k^* \to M_1$ (since $\alpha \to 1$). Hence $p_2 \to$ MEDIAN($\boldsymbol{X}$). This removes $N_0/2$ elements with cost $N_0/2$.

Since the first and second pivot each removed $N_0/2$ elements, the number of remaining samples $N_2 \to 0$. Hence the total cost $\to 1.5N_0$. ■

Appendix B

PROOF OF CONVEXITY FOR COST FUNCTION

In the following section the input source code is Mathematica 10.0 source code. Mathematica is a well known Computer Algebra System which can aid in solving symbolic equations.

First, the second derivative of the cost function (2.5) is computed:

```
In[1]:= D[m + n - n/4 Erfc[Sqrt[m]/(m + 2)], {m, 2}]
```

$$\frac{1}{4}\left(-\frac{2e^{-\frac{m}{(m+2)^2}}\left(-\frac{1}{4m^{3/2}(m+2)}+\frac{2\sqrt{m}}{(m+2)^3}-\frac{1}{\sqrt{m}(m+2)^2}\right)}{\sqrt{\pi}} - \frac{2e^{-\frac{m}{(m+2)^2}}\left(\frac{2m}{(m+2)^3}-\frac{1}{(m+2)^2}\right)\left(\frac{1}{2\sqrt{m}(m+2)}-\frac{\sqrt{m}}{(m+2)^2}\right)}{\sqrt{\pi}}\right) n$$

Next, we show that the second derivative is greater than zero. For this, it is necessary to help Mathematica by providing it with constraints:

```
In[2]:= Assuming[n > 0,
          Reduce[{$$Assumptions,
            D[m + n - n/4 Erfc[Sqrt[m]/(m + 2)], {m, 2}] > 0}]]

Out[2]= m > Root[-16 - 72 #1 - 32 #1^2 - 2 #1^3 + 3 #1^4 &, 2] && n > 0
```

This states that the value m needs to be larger than the 2^{nd} root of the polynomial $-16-72x-32x^2-2x^3+3x^4$.

The exact solution can be converted to a numerical value:

```
In[3]:= N[Root[-16 - 72 #1 - 32 #1^2 - 2 #1^3 + 3 #1^4 &, 2], 20]

Out[3]= 4.3968437061310307376
```

Thus, the function is convex for all $m \geq 4.397$. ■

Appendix C

COPYRIGHT NOTICE

This appendix states the various copyright notices from the material in this dissertation.

Most of Chapter 3 of this dissertation is submitted to IEEE *Transactions on Signal Processing* which, when published will hold the Copyright © 2015, IEEE.

Most of Chapter 2 was published in IEEE journal *Transactions on Signal Processing* and is Copyright © 2012, IEEE.

In both cases, I –André Rauh– am the senior author of the publications and give full permission to reprint my work.

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of University of Delaware's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.