UNIVERSITY OF THE WITWATERSRAND

MASTERS DISSERTATION

Computational Approaches In Compressed Sensing

Author:
Matthew WOOLWAY

Supervisor:
Dr. Byron JACOBS
Prof. Ebrahim MOMONIAT

A dissertation submitted in fulfilment of the requirements for the degree of Master of Science

School of Computational and Applied Mathematics

June 2014

Declaration of Authorship

I, Matthew WOOLWAY, declare that this thesis titled, ‘Computational Approaches In Compressed Sensing’ and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Matthew Woolway

Date: 20/06/2014

UNIVERSITY OF THE WITWATERSRAND

Abstract

Faculty of Science
School of Computational and Applied Mathematics

Master of Science

Computational Approaches In Compressed Sensing

by Matthew WOOLWAY

This thesis aims to provide a summary on computational approaches to solving the Compressed Sensing problem. The theoretical problem of solving systems of linear equations has long been investigated in academic literature. A relatively new field, Compressed Sensing is an application of such a problem. Specifically, with the ability to change the way in which we obtain and process signals. Under the assumption of sparse signals, Compressed Sensing is able to recover signals sampled at a rate much lower than that of the current Shannon/Nyquist sampling rate. The primary goal of this thesis, is to describe major algorithms currently used in the Compressed Sensing problem. This is done as a means to provide the reader with sufficient up to date knowledge on current approaches as well as their means of implementation, on central processing units (CPUs) and graphical processing units (GPUs), when considering computational concerns such as computational time, storage requirements and parallelisability.

Acknowledgements

I would like to thank my supervisors, Dr. Byron Jacobs and Prof. Ebrahim Momoniat for all their help, support and advice throughout the undertaking of this MSc.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Our continued voyage into an ever deepening digital world shows no sign of slowing down. Armed with consumers possessing insatiable technological thirst and businesses demanding the competitive edge, the digital revolution spreads to every corner of the globe.

With such demands increasing day by day, significant pressure has been placed on the development of new and continually improving technology.

One such area, is the development of new sensing systems. Today's society has become accustomed to a manner and means of instant information acquisition as well as high definition media consumerism. Many of these enjoyed liberties are provided off the back of various forms of Signal Processing.

Numerous Signal Processing based applications, are under severe strain to keep up with this modern deluge of data and digitisation. However, a timeous and emerging field has garnered a significant amount of attention in the fields of Signal/Image processing and Information theory. Compressed Sensing or Compressive Sampling, a field in its relative infancy, has shown to have tremendous potential for future applications.

Compressed Sensing has a plethora of applications, such as tomography, radar, communication and astronomy to name a few [3]. This notion of Compressed Sensing is a new form of sampling theory. Deviating from the established theory developed by Shannon/Nyquist by making use of sparse signals, and thus allowing for the reconstruction of signals and images from what was previously understood to be insufficient information.

The traditional approach to sampling was one that obeyed the Shannon/Nyquist theorem, which states that an analogue signal can be reconstructed perfectly from its samples: providing it was sampled at a rate at least twice the highest frequency present in the signal[1].

Although the Shannon/Nyquist sampling theorem specifies to avoid a loss of information in the signal, we are required to sample at this aforementioned *Nyquist rate* [5, 6]. This potentially has serious repercussions. A prominent concern for example, is the application of digital imagery and high-definition video, where the *Nyquist rate* is simply too high. The current process of acquiring the entire signal and subsequently compressing it, often requires vast resources when considering extremely large signals. To which the majority of the captured information is to be thrown away during compression. The quintessential question then would be, "*why don't we simply combine these processes and just sense the essential information in the signal directly?*" (ie. fewer measurements). Compressed Sensing appeals to this notion, with results showing promising indications of success [7].

1.2 What Is Compressed Sensing

"Compressed Sensing," a term coined by David Donoho in [8], attempts to allow for exact signal reconstruction at sample rates well below the expected *Nyquist rate*.

Particularly, Compressed Sensing deals with sparse signals. Generally speaking, signals aren't sparse. However, it is possible for such a signal to be sparse in some predetermined basis (where most of the coefficients are zero). The use of a traditional measurement technique would heavily oversample such a signal since the bulk of the signal has little to no important information.

The ground breaking and pioneering work conducted in this area belonged not only to Donoho, but also the work done by Candés, Tao and Romberg [7, 9]. The fundamental papers by Candés and Tao [10, 11] and Donoho [8], showed the use of linear programming to efficiently and successfully reconstruct signals with high levels of accuracy.

Following these initial strides forward, numerous alternative methods have been developed. Claiming faster, possibly superior techniques to those of pioneering linear programming algorithms. Tropp and Gilbert [12] proposed Orthogonal Matching Pursuit (OMP), an extention of the original Matching Pursuit techniques developed by Mallat and Zhang [13] already in 1993. This was advanced further with Stagewise Orthogonal Matching Pursuit

[1] This rate is known as the *Nyquist rate* [4].

(StOMP) [14], and even more so by Needell and Tropp [15] who proposed Compressive Sampling Matching Pursuit (CoSaMP). Even gradient pursuit methods have been developed by Blumensath and Davies [16], who also proposed the idea of Threshold Based Algorithms [17].

1.2.1 Importance

The above techniques represent the immense speed at which the field of Compressed Sensing is expanding. In the space of 7 years the spectrum and size of published papers is simply staggering. Combining the disciplines of mathematics, applied mathematics, computer science amongst others has shown massive cross-discipline usage and definitely provides credence to Compressed Sensing as a viable theory to combat the present day concern of data deluge.

Compressed Sensing proposes numerous advantages to a multitude of fields. Areas such as error correction, image processing, radar, seismology, tomography and astronomy are just few [3, 18], which may benefit from what Compressed Sensing has to offer.

Further a major advantage to Compressed Sensing is how these potentially improvable fields can all be formulated down to the same mathematical problem. Thus, solving this problem in a general case, solves arrays of problems in completely different fields.

1.3 Important Theory

Before we may begin formally describing the Compressed Sensing problem, it may be frugal to address some important mathematical requirements.

1.3.1 Sparsity and Compressibility

The concept of *sparsity* can be illustrated by introducing the set $\{1,2,...,N\}$ or notionally $[N]$, along with the corresponding cardinally of set S, $\text{card}(S)$[2]. Formally, the definition follows,

Definition 1.1. ([19]) The Support of a vector $x \in \mathbb{R}^N$ is the index set of its non-zero entries, ie;

$$\text{supp}(x) = \{j \in [N] : x_j \neq 0\}. \tag{1.1}$$

Further, the vector $x \in \mathbb{R}^N$ is called s-sparse if at most s of its entries are non-zero, ie, if;

$$||x||_0 = \text{card}(\text{supp}(x)) \leq s. \tag{1.2}$$

Often, sparsity itself is a strong constraint to impose. Therefore, it is perhaps shrewd to make use of a weaker concept entitled *compressibility*. This then, allows for the consideration of vectors which are close to those of s-sparse nature. The *compressibility* of a vector may be measured by the error of *best s-term approximation*. Consider the following definition [19].

Definition 1.2. For $p > 0$, the ℓ_p error of best s-term approximation to a vector $x \in \mathbb{R}^N$ is defined by;

$$\sigma_s(x)_p = \inf\{||x-z||_p, z \in \mathbb{R}^N \text{is } s\text{-sparse }\}. \tag{1.3}$$

In the above definition, $\sigma_x(x)_p$ achieves its infimum by an s-sparse vector $z \in \mathbb{R}^N$ whose nonzero elements are equal to the s largest absolute value elements of x.

Thus we may call $x \in \mathbb{R}^N$ a *compressible* vector if the error of its best s-term approximation decays in s. That is to say sparsity decays according to a power law described by the ℓ_p-ball to be discussed later.

In generality, signals found and used in reality are often not exactly sparse. Importantly, both sparse and compressible signals can be represented with a high degree of accuracy by only preserving the values and locations of the largest coefficients of the signal. Thus,

[2] The cardinality of a set is a quota of the number of entries or elements found within the set

should x be sparse, then this signal can be efficiently approximated from only a few significant coefficients [20]. This process is called *sparse approximation* which forms the basis of transform coding [21].

1.4 Organisation

This thesis aims at to provide the reader with a comprehensive review and knowledge base in terms of the main Compressed Sensing problem. This is then built upon, by approaching major recovery algorithms and methods with a computational concern. This allows for direct comparison and analysis of the various methods taking into account the importance of speed, computational complexity and storage costs.

More specifically, Chapter 1 considers the outlying idea behind Compressed Sensing. This is done on a more conceptually based approach and slowly introduces the fundamental problem in its simplest mathematical form.

Chapter 2 deals with the many recovery algorithms previously mentioned. A thorough literature review on these techniques is executed. This is done to provide the reader with knowledge on the specific tools to be wielded when tackling Compressed Sensing.

Chapter 3 involves the implementation of these methods and algorithms with specific results targeted. These can then be analysed to classify the specific use and effectiveness of each approach.

Chapter 4 allows for a discussion of the above implemented methods and draws some concessions and possible extensions through the use of different computational hardware. A final conclusion for a qualitative performance metric is drawn.

1.5 Problem Formulation

1.5.1 Goal of Compressive Sensing

Consider a real valued, finite length, one-dimensional, discrete time signal $x \in \mathbb{R}^N$ (which can be regarded as an $N \times 1$ column vector). Compressive Sensing's claim is then: given M measurements, where $M \ll N$, often we may reconstruct the original signal x in manner which the measurements are chosen at random or rather non-adaptively.

Now given that any signal within $\mathbb{R}^N$ may be represented in terms of some basis of $N \times 1$ vector $\{\psi\}_{i=1}^N$ [2]. For the purpose of simplicity we may assume the basis to be orthonormal[3]. We may now construct an $N \times N$ orthonormal basis matrix $\Psi = [\psi_1, \psi_2, .., \psi_N]$ where the

[3] i.e. two vectors in an inner product space are orthonormal if they are orthogonal.

$i-$th column is the $i-$th basis vector ψ_i [2]. This allows us to express any signal $x \in \mathbb{R}^N$ as a linear combination of these basis vectors by,

$$x = \sum_{i=1}^{N} z_i \psi_i \quad \text{or} \quad x = \Psi z, \tag{1.4}$$

where $z \in \mathbb{R}^N$ is the vector of inner products $z_i = \langle x, \psi_i \rangle$. Note that x and z are equivalent representations of the same signal, however in different domains [2]. Typically we say that x is in the *time domain* (time dependent signals such as audio) or in the *spatial domain* (for spatially dependent signal like images), z is referred to be in the Ψ *domain*.

Measuring the given signal x is undertaken by sampling it with respect to a measurement matrix $\Phi \in \mathbb{R}^{M \times N}$, where Φ pertains to rows ϕ_i for $1 \leq i \leq M$. As such, there is a corresponding observation y_i which relates to the respective rows from Φ [2]. That is it say,

$$y_i = \langle \phi_i, x \rangle, \tag{1.5}$$

which written in vector notation returns,

$$y = \Phi x. \tag{1.6}$$

Clearly, if $M \geq N$ and given that the rows of Φ span $\mathbb{R}^N$ then we can completely reconstruct the signal x from its observations y. Thus substituting $x = \Psi z$ into equation (1.6) we obtain,

$$y = \Phi x = \Phi \Psi z = \Theta z, \tag{1.7}$$

where $\Theta = \Phi \Psi$.

The major question asked by Compressed Sensing is the case when $M \ll N$? This case outlines the fundamental problem of an under-determined system as can be seen in Figure (1.1).

Since we are assuming the inner products z are sparse, specifically that it is a linear combination of only $s \ll N$ basis vectors (i.e a linear combination of s columns of Φ as can be seen in Figure (1.2)), then we may say that z is s-*sparse with respect to the basis* Ψ. These columns correspond to the location of non-zero entries. Note the equivalence to equations (1.4) and (1.7). By solving equation (1.7) for z, we are equivalently solving for x as Ψ is a predetermined basis.
One of the simplest ways theoretically to recover a vector from its measurements $y = \Phi x$ is to solve the ℓ_0-minimisation problem [18],

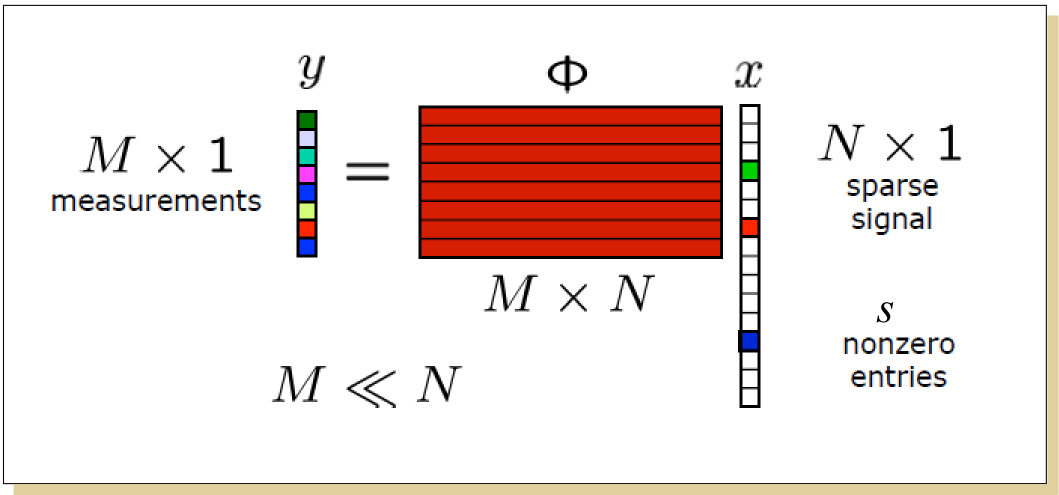


FIGURE 1.1: **Undetermined System** [1]

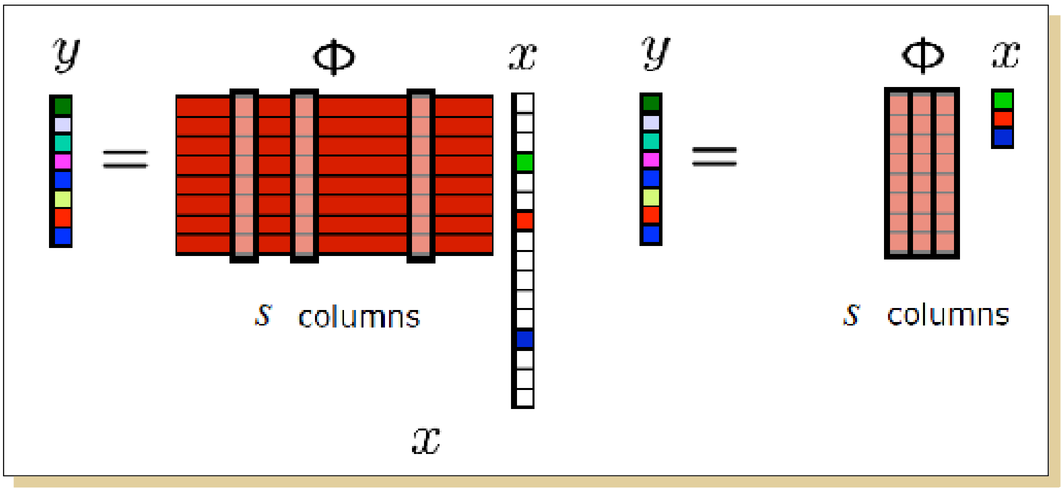


FIGURE 1.2: **Linear Combination** [1]

$$\min_{x} ||x||_0 \quad \text{subject to} \quad y = \Phi x. \tag{1.8}$$

This provides precise reconstruction since the ℓ_0 norm calculates the sparsity of every vector x and finds the sparest vector. However, due to the unavoidable combinatorial search, this algorithm is NP-Hard [22].

1.5.2 NP-Hardness

Since the major vein of interest of this thesis involves the computational approach to the Compressive Sensing problem. A small discussion on the notions of computational complexity would prove fruitful. To begin with, the idea of an algorithm with a polynomial runtime, is one in which the algorithm performs its task in a number of steps which are bounded by a polynomial expression within the size of the input [19].

The various classes of decision problems can be listed as follow [19],

- P- problems consist of problems where there exists a polynomial-time algorithm finding a solution.
- NP-problems pertain to problems where there exists a polynomial-time algorithm certifying a solution.
- NP-hard problems encompass all problems for which a solving algorithm could be transformed in polynomial time into a solving algorithm for any NP-problems.
- NP-complete problems consist of all problems which belong to both NP and NP-hard classes. This can be seen in figure 1.3.

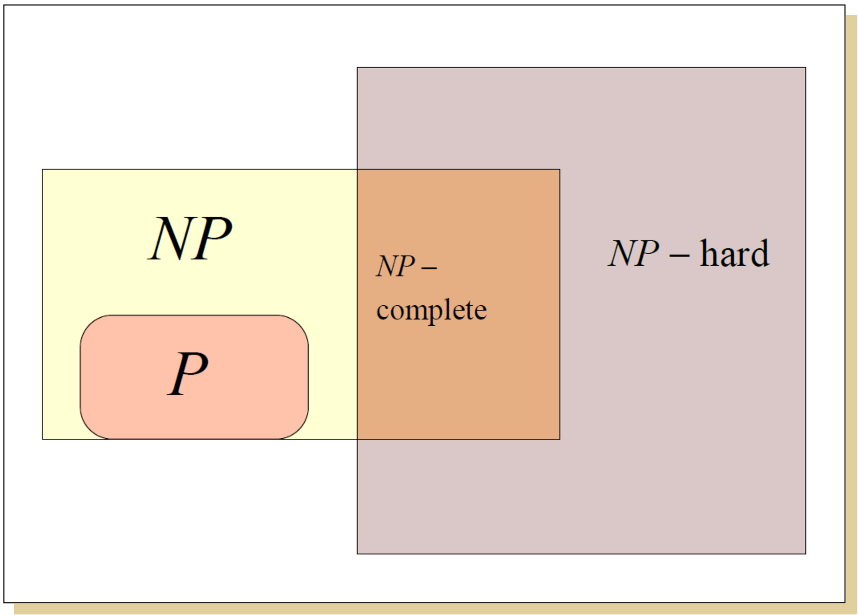


FIGURE 1.3: **Visual Summary of Decision Problem Classes**

We may now reduce the goals of Compressed Sensing to two pertinent points. To design a matrix Φ, along with a reconstruction algorithm, for which s-sparse signals require a much smaller number of measurements ($M \approx s$) to reconstruct the given signal.

Chapter 2

Main Algorithmic Approaches

Compressed Sensing has provided numerous methods and techniques to solve the sparse recovery problem outlined above. The initial work, undertaken by Donoho, Candés, Tao, Romberg, *et al.*, was essentially an optimisation problem solved using linear programming. While this differs from the later developed methods such as greedy algorithms, all approaches have their own advantages and disadvantages. The first method developed in this field, *basis pursuit*, was essentially a linear programming method. We consider this first.

2.1 Basis Pursuit

It was briefly mentioned with equation (1.8), that sparse recovery problems can be formulated as an NP-Hard problem. In the late eighties, Donoho and others [14, 23], proved that in the case of certain measurement matrices Φ, the NP-Hard problem can be relaxed to the equivalent problem,

$$\min_x ||x||_1 \quad \text{subject to} \quad y = \Phi x. \tag{2.1}$$

Later Candés and Tao proved that equations (1.8) and (2.1) are equivalent for certain measurement matrices providing the satisfaction of a particular property [10].

2.1.1 The Recovery Algorithm

Knowing that the vector x is sparse, how does one aim to recover x from the corresponding observations y? An intuitive approach when generally dealing with problems such as these is to obtain the best possible x. The common avenue for such an objective would be the involvement of a least squares approximation through the minimisation of the ℓ_2 norm. In terms of the problem at hand. The minimisation of the ℓ_2 norm would essentially be searching for the smallest energy found on the hyperplane of the underdetermined system as can be seen in Figure (2.1).

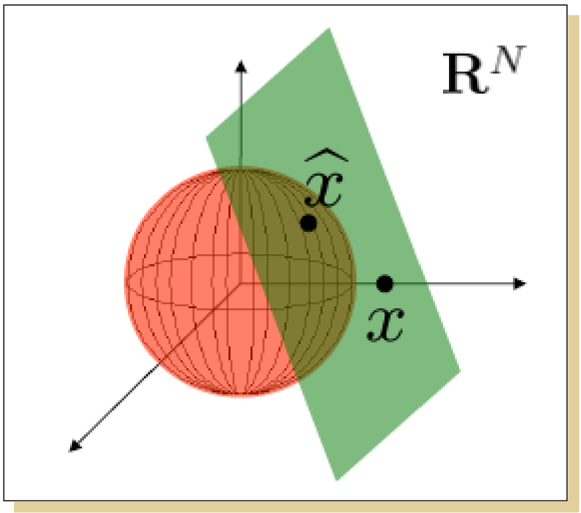


FIGURE 2.1: **Insufficient ℓ_2 Representation** [2]

Unfortunately, this approach yields the incorrect answer as can be seen in Figure (2.1). Due to the fact that sparse signals are always found close to or on the coordinate axes. This is in part due to the fact that the ℓ_2 geometry is not the optimal norm to construct the signal x. Since the closer the ℓ_2 norm approaches the hyperplane, the worse the reconstruction becomes.

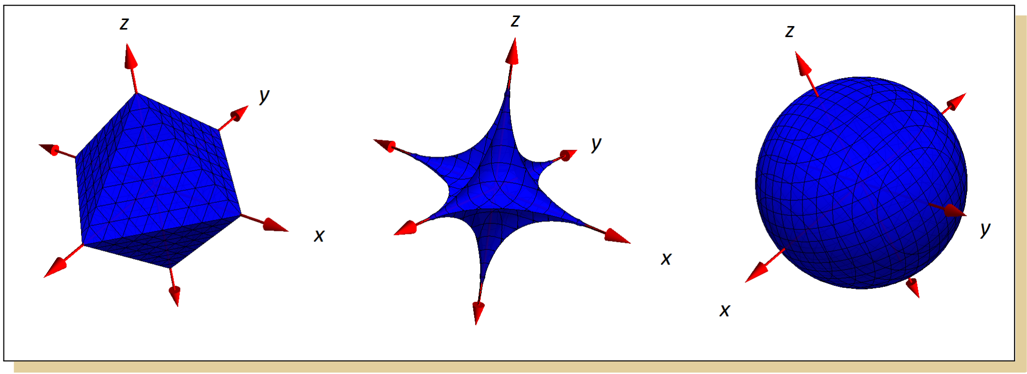


FIGURE 2.2: **3D Norm Representations. ℓ_1 left, ℓ_p centre and ℓ_2 right.**

Clearly then this is not the norm to overcome our initial NP-Hard problem. Chen, Donoho and Saunders were the first to establish the idea of substituting the ℓ_0-norm with the closet convex norm, the ℓ_1-norm [24]. This paper lead to the minimisation problem described by equation (2.1).

Thus the shape of the ℓ_1 ball and its subsequent minimisation promotes sparsity. This can be seen in Figure (2.3), with the ℓ_1 ball intersecting the hyperplane directly on the coordinate axis.

Therefore, *Basis Pursuit* employs the geometry of the octahedron to recover the required sparse signal x whilst the measurement matrices Φ satisfy the deterministic property to follow.

2.1.2 The Measurement Matrix

Being a primary goal of Compressed Sensing. We require an apt description on the design of the matrix Φ. The main property of the matrix Φ, is the guarantee that any original information in the signal x is not destroyed through measurement. However, due to the fact that $M < N$ and we have an underdeterminded system, the process of solving x is an

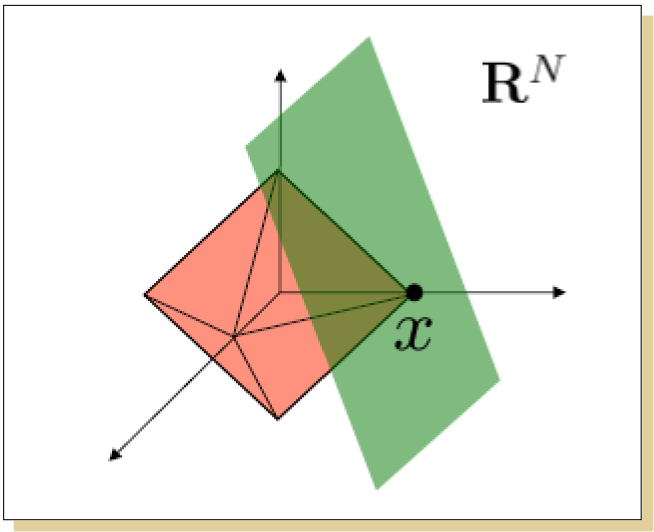


FIGURE 2.3: **Correct ℓ_1 Representation** [2]

ill-posed one.

As already mentioned, by restricting the problem to s-sparse signals, we can reduce the ill-posed problem to a more manageable one as seen in the right of Figure (1.2). The concern here is that we assume the positions of s non-zero entries of s are known beforehand. In this particular case, we could then form the $M \times s$ matrix Φ where $M \gg s$ and solve the least squares problem associated to the non-zero positions of x. A sufficient condition for any s-sparse vector $v \in \mathbb{R}^N$ to be well conditioned is,

$$1 - \epsilon \leq \frac{||\Phi v||_2}{||v||_2} \leq 1 + \epsilon, \tag{2.2}$$

for some $\epsilon > 0$, with the matrix Φ preserving the length of these s-sparse vectors. Again, as previously mentioned, the positions of the non-zero coefficients are not known *a priori*. Fortunately, it can be shown that a sufficient condition for a stable inverse for s-sparse signals is for Φ to satisfy not only equation (2.2) but also the **Restricted Isometry Property** [9].

2.1.3 Restricted Isometry Property

In an attempt to providing a parameter to determine the quality of the measurement matrix, a primary aim of Compressed Sensing. Candés and Tao proposed a concept refered to as the **Restricted Isometry Property** [10, 11, 25]. The definition follows;

Definition 2.1 (**Restricted Isometry Property, RIP**). dummy text
For all x so that $||x||_0 \leq s$, it is said that Φ satisfies the RIP with the isometry constant δ_s, if δ_s

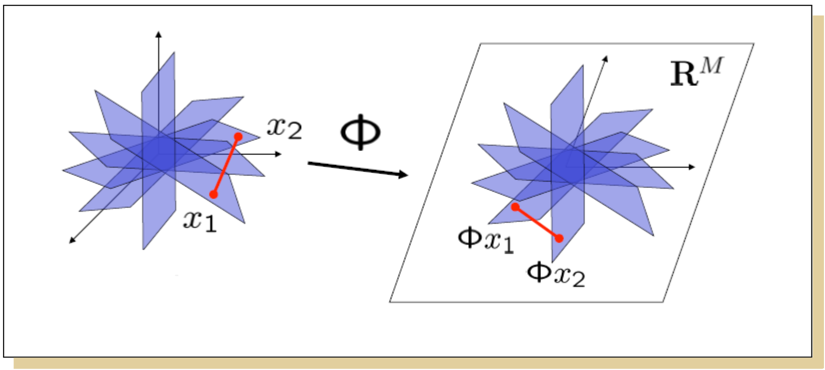


FIGURE 2.4: **Distance Preservation through Dimensionality Reduction**

is the smallest value satisfying,

$$(1-\delta_s)||x||_2^2 \le ||\Phi x||_2^2 \le (1+\delta_s)||x||_2^2. \tag{2.3}$$

This measurement matrix Φ, can be thought of as the mapping of the signal from the higher dimensional *signal space*, down to the lower dimensional *measurement space*. More specifically, we can consider the RIP as the property that dictates distance preservation or lack there of between two particular entries x_1 and x_2. Visually we can see this in Figure (2.4). Through the dimensionality reduction, we ideally would like $||x_1 - x_2||_2 \approx ||\Phi x_1 - \Phi x_2||_2$. That is to say, when reducing from N to M dimensions, we would like the distance between two arbitrary signals to remain roughly the same.

Now given this RIP, we are still not able to know if a matrix Φ has this property since testing the matrix computationally for the RIP is combinatorial.

Fortunately, it has been shown that many types of matrices satisfy the RIP with high probability. More specifically, that $M \times N$ matrices can be randomly generated in accordance to the following:

- The entries of Φ must be i.i.d. normal.
- Or the entries of Φ are i.i.d. symmetric Bernoulli distributed (i.e. $\pm$ 1 matrix), or any other subgaussian or Fourier distribution.

Any matrix adhering to the above satisfy the RIP with high probability so long as the measurements taken satisfy,

$$M \ge cs\log\left(\frac{N}{s}\right), \tag{2.4}$$

where c is just a constant. The proofs corresponding to above matrices satisfying the RIP are credited to Baranuik *et al.* [26], and Mendelson *et al.* [27].

It is ideal to use the RIP as a means of analysing the performance of different Compressed Sensing recovery algorithms. This however, is not easily done due to the difficulty in finding δ_s for any given measurement matrix [28]. An alternative is to establish a bound on δ_s with mutual coherence.

2.1.4 Mutual Coherence

Donoho and Hou introduced the concept of *mutual coherence* as another property of the measurement matrix. Essentially, *mutual coherence* is a measure of the ability of suboptimal algorithms[1] to efficient and correctly identify the true representation of a sparse signal [29, 30]. The definition of mutual coherence follows.

Definition 2.2. (Mutual Coherence)

Let ϕ_i and ϕ_j be two columns of Φ. Then the mutual coherence is defined as,

$$\eta(\phi_i, \phi_j) = \sup\{|\langle\phi_i, \phi_j\rangle| : \forall i, j, \text{ where } i \neq j\}. \tag{2.5}$$

Sufficiently small values of functional $\eta(\phi_i, \phi_j)$ guarantees the possibility of ideal atomic decomposition. So should two bases (columns) have a small value of η, then they are mutually incoherent. Since $0 \leq \eta \leq 1$, if two orthobases have an element in common then $\eta = 1$ [31]. This restriction is easier to calculate than the RIP parameter due to its computational complexity scaling exponentially with the number of columns in Φ [28].

2.1.5 Stability

In practice, the vectors required to recover via basis pursuit, or others for that matter, as already previously mentioned are sparse only in ideal cases. Those cases not included above involve the recovery of x with an error controlled by its distance to s-sparse vectors [19].

This property then, can be considered as the stability of the reconstruction algorithm with respect to the sparsity defect. Firstly, it can be shown that basis pursuit proves stable under a more rigorous adaptation of the null space property (NSP). The definition follows,

Definition 2.3. ([19]) A matrix $\Phi \in \mathbb{R}^{M\times N}$ is said to satisfy the *stable null space property* with constant $0 < \rho < 1$ relative to a set $S \subset [N]$ if,

$$||v_S||_1 < \rho ||v_{\bar{S}}||_1 \quad \text{for all } v \in \ ker\ \Phi, \tag{2.6}$$

where $v \in \mathbb{R}^N$, while v_s is the vector in $\mathbb{R}^S$ (i.e. the restriction of v to the indices in S) and $v_{\bar{s}}$ its complement. It is also said to satisfy the null space property of order s, if it satisfies the null space property relative to any set $S \subset [N]$ with $\text{card}(S) \leq s$.

[1] e.g matching pursuit and basis pursuit.

Fortunately, as shown by Cohen, Dahmen and DeVore in [32], the RIP and NSP are linked closely. Now, a major concern is that of creating a matrix Φ given the basis Ψ, resulting in Θ pertaining to high order.

As such that stability of the measurement matrix can also be ensured by demanding a high level of incoherence between the measurement matrix Φ and the basis matrix Ψ. We may formally define this as follows:

Definition 2.4. (Stability Coherence)

The coherence between the sensing(measurement) basis Φ and the basis matrix Ψ is defined as,

$$\eta(\Phi,\Psi) \triangleq \sqrt{N} \max_{i\leq j,k\leq N} |\langle \phi_i, \psi_j \rangle|. \tag{2.7}$$

For any pair of orthonormal matrices Φ and Ψ, $1 \leq \eta(\Phi,\Psi) \leq \sqrt{N}$. Conceptually, this coherence is measuring the largest correlation between any two elements of Φ and Ψ. Should this coherence value be small, then this implies that the basis vectors cannot sparsely represent the vectors in Φ and similarly the converse is also true [8, 9, 33].

2.1.6 Remarks on Basis Pursuit

We know that although solving the ℓ_0 minimisation problem is guaranteed to return the correct solution, being an NP-compete problem, it is infeasible computationally. Thus, the pertinent point here is that Compressed Sensing shows there exists substantially faster algorithms, that with high probability, solve this problem.

A large amount of the early work in Compressed Sensing involved the use of the ℓ_1 norm as a substitute in the ℓ_0 minimisation problem. Candés and Romberg showed [34], that if x is s-sparse and provided the number of measurements M taken satisfy,

$$M \geq cs\log\left(\frac{N}{\delta_s}\right), \tag{2.8}$$

then with probability exceeding $1-\delta_s$, the solution to the problem,

$$x = \arg\min||\hat{x}||_1, \qquad \text{subject to } \Phi\hat{x} = y, \tag{2.9}$$

is $\hat{x} = x$ [33]. So far, all the cases have dealt with the noiseless scenario. Fortunately Compressed Sensing is robust in the ℓ_1 minimisation problem, as small error or noise does not increase throughout the ℓ_1 minimisation. Generally speaking, error and noise propagation before and after the use of Compressed Sensing is often of the same order.

Mathematically, in the case with noise, $||\Phi\hat{x} - y||_2 \leq \epsilon$, where ϵ is a tolerance parameter. The ℓ_1 minimisation problem can be reformulated as the convex optimisation problem,

$$\underset{\hat{x}\in\mathbb{R}^N}{\operatorname{argmin}} ||y - \Phi\hat{x}||_1, \tag{2.10}$$

accountable from a theorem by Candés *et al* [35]. This can be stated as a linear program,

$$\min \sum_{i=1}^{N} t_i \qquad \text{subject to} \qquad -t_i \leq \hat{x}_i \leq t_i, \quad y = \Phi\hat{x} \tag{2.11}$$

which is officially known as *basis pursuit.* The above problem has a computational complexity of $\mathcal{O}(N^3)$ [33, 35, 36].

2.2 Greedy Algorithms

The earlier work covered in Section 2.1, showed that Compressed Sensing could rely on the solution to the ℓ_1 minimisation problem being the correct solution as well as providing this within computationally acceptable runtimes. However, an alternative approach using different algebraic tools has also been developed. "*Greedy Algorithms,*" are alternative algorithms that attempt to find faster or superior performance in the signal reconstruction.

Most greedy algorithms attempt to first find indices which correspond to the non zero values in x and then assign the correct values to theses indices [28]. The majority of these techniques are extremely robust in the presence of noise and work with considerable efficiency. A major advantage to greedy algorithms is speed. However, this alternative approach is not without challenges of its own. Greedy algorithms often rely on signal processing heuristics, a result of this is difficulty in proving the performance of these methods over the convex relation based approaches as discussed in Section 2.1.

The large majority of these techniques can be classified into two main groups. Those that belong to variations of the matching pursuit technique or those belonging to thresholding algorithms [33].

2.2.1 Some Mathematical Requirements

As already said, greedy algorithms are harder to analyse than the convex optimisation methods. However, these greedy algorithms offer computational efficiency and easy implementation, often with the potential for better performance.

Before we can formally define our first greedy method, it would be prudent to dispense with some important theory. Revisiting the idea of the support set mentioned in Chapter 1 Section 1.3, we redefine the notion of the support set to effectively describe greedy algorithms. The definition follows:

Definition 2.5. (Support Set)
The support set I is a set of indices corresponding to the non-zero elements in the sparse signal vector x [28],

$$I \triangleq \{\, i \,:\quad x_i \neq 0\}, \tag{2.12}$$

with the complement,

$$\bar{I} \triangleq \{\, i \,:\quad x_i = 0\}. \tag{2.13}$$

The union of the support set, $I \cup \bar{I} = \{1,2,...,N\}$ is the entire set of indices, while the intersection of the $I \cap \bar{I}$ is just the empty set $\emptyset$.
As such, we are able to pick all the non-zero elements in x and place them sequentially into the column vector $x_I = \{x_i : x_i \neq 0\}$. The ℓ_0 norm of this vector, $||x||_0 = |s| \leq s$, is equal to the magnitude of the support set [28].

2.2.2 Solution Approaches

The main approach by greedy methods, is that given some measurement vector, it attempts to either detect or estimate the support set of a sparse signal vector and subsequently evaluating the associated signal values [28].

Throughout the array of various greedy algorithms, two main approaches are used to estimate the support set. One such approach iteratively detects the support set elements to be added to the support set estimate sequentially, one at a time, until the support set itself is full. More specifically, the algorithm introduces an initial estimate for x ($\hat{x}^{[0]} = \bar{0}$), an initial residual error $r^{[0]} = y - \Phi\hat{x}^{[0]} = y$ and the empty set $I = \varnothing$ (since the number of non-zero entries in $\hat{x}^{[0]} = 0$).
Thus each iteration will update these values by adding respective entries to the support I and subsequently updating the estimate for the signal $\hat{x}$, whilst decreasing the overall residual error r. Algorithms using this approach are known as *serial pursuit* (s-pursuit) algorithms.

The other approach, establishes an initial guess for the entire support set and iteratively refines it until such a point that the support set no longer improves through additional iterations [28]. Algorithms employing such an approach are known as *parallel pursuit* (p-pursuit) algorithms.

2.2.3 Matching Pursuit

The first look at greedy algorithms, takes form in a procedure known as *Matching Pursuit* (MP). Matching Pursuit iteratively selects elements which correlate most with the signal.

Matching Pursuit brings the possibility of speed to the table but can require numerous additional vectors (columns), should it have selected non ideal elements in previous iterations [37].

Matching Pursuit makes use of matched filter detection. We can define this as follows [28].

Definition 2.6. Let Φ be a matrix containing normalised (orthogonal) columns ϕ_i and given the measurement vector y. Then the matched filter detection calculates the magnitude of the corresponding correlation vector,

$$|\phi_i^T y|, \quad \forall i, \tag{2.14}$$

and systematically selects one or more indices which index the largest elements.

Although we were not able to use the ℓ_2 norm in the original Compressed Sensing problem due to the underdetermined system, if we were able to have the correct support set I of x, then we are able to make use of a least squares approximation [28]. Thus, $y = \Phi_I x_I$ constitutes an overdetermined system. A system which least squares obtains a unique solution to. Since our vector x pertains to not just non-zero coefficients, we can see that we can obtain the full reconstruction by filling the remaining indexed entries with zeros, or rather the complement of x_I ($\bar{x}_I = 0$).

We may now formally define the least squares as the subsequent definition [28],

Definition 2.7. The least squares estimation of a signal is the $\hat{x}_I$ which minimises the following,

$$\min_{\hat{x}_I} ||y - \Phi_I \hat{x}_I||_2^2. \tag{2.15}$$

Since we are concerned with Φ_I possessing full column rank, an estimate can be reconstructed as,

$$\hat{x}_I = \Phi_I^{\dagger} y, \tag{2.16}$$

where we can define the pseudoinverse as,

$$\Phi_I^{\dagger} = (\Phi_I^T \Phi_I)^{-1} \Phi_I^T. \tag{2.17}$$

Clearly in application the true support set I is not known but instead an estimate $\hat{I}$ is generated through the respective greedy algorithm.
In the unique case of having the true support set I, then the reconstruction from equation (2.16) returns the correct non-zero components, $\hat{x}_I = x_I$. As a result in such a case, equation (2.15) would return zero, with the optimal solution on the hyperplane being found.

Finally, by multiplying equation (2.16) by Φ_I,

$$\begin{aligned} \Phi_I \hat{x}_I &= \Phi_I^\dagger y \Phi_I \\ &= \left(\Phi_I^\dagger \Phi_I\right) y \\ &= y, \end{aligned} \tag{2.18}$$

the orthogonal projection, y_p of y is found.

The MP algorithm is summarised in Algorithm 1.

Algorithm 1 Matching Pursuit (MP)

```
procedure MP($\Phi, y, s$)                                         ▷ Required Inputs
    Initialisation: $r^{[0]} = y, \hat{x}^{[0]} = \bar{0}$          ▷ Initialising the residual and support set
    for $i = 1; i = i+1$ till stopping criteria is met do
        $g^{[i]} = \Phi^T r^{[i-1]}$
        $j^{[i]} = \underset{j}{\text{argmax}} |g_j^{[i]}| / ||\Phi_j||_2$
        $\hat{x}_{j^{[i]}}^{[i]} = \hat{x}_{j^{[i]}}^{[i-1]} + g_{j^{[i]}}^{[i]} / ||\Phi_{j^{[i]}}||_2^2$
        $r^{[i]} = r^{[i-1]} - \Phi_{j^{[i]}} g_{j^{[i]}}^{[i]} / ||\Phi_{j^{[i]}}||_2^2$
    end for
    return $\hat{x}^{[i]}, r^{[i]}$                                  ▷ Required Outputs
end procedure
```

Analysing the algorithm above we can see that the approximation of MP is incremental and one column of Φ is selected at a time and subsequently at each iteration, only the coefficient associated with the selected column is updated [38].

More specifically, at each iteration, the update $\hat{x}_{j^{[i]}}^{[i]} = \hat{x}_{j^{[i]}}^{[i-1]} + g_{j^{[i]}}^{[i]} / ||\Phi_{j^{[i]}}||_2^2$ will minimise the approximation cost of $||y - \Phi \hat{x}^{[i]}||_2^2$ with respect to that particular coefficient.

Importantly, MP often will repeat the selection of the columns of Φ in attempt to improve the quality of approximation. Should the norm of residual $r^{[i]}$ be used as the stopping criteria then the algorithm will terminate within finite iterations. This is due to the fact that the norm of the residual converges linearly to zero in the cases where the columns of Φ span $\mathbb{R}^M$ [38].

In terms of computational implementation. The matching pursuit algorithm involves heavy use of matrix multiplication. Thus, it is advisable to impose MP with matrices which allow for fast implementation such as those based on fast Fourier transforms (FFT). Presently, there exist packages with impressively quick implementations of MP for problems involving columns with restricted support [38, 39].

2.2.4 Orthogonal Matching Pursuit

An extension of Matching Pursuit is Orthogonal matching Pursuit (OMP). Initially proposed by Mallat *et al* [13]. OMP rose to early prominence in the 1950's, hailing from the statisitics community, albeit under the name stagewise regression [28].
Used in many fields such as machine learning and signal processing, it was first used to solve the Compressed Sensing problem by Gilbert and Tropp [12].

OMP reduces the disadvantages of MP. Although it is similar to MP, it makes use of a Gram-Schmidt process orthogonalising the dictionary after each iteration [37]. This ensures that no elements in the direction of a previously selected column are selected, something which may happen through use of Matching Pursuit.

Importantly, OMP unlike its predecessor MP, will never reselect a given entry with the residual orthogonalised to all current entries throughout all iterations.
The algorithm itself fulfils a large computational bias towards matrix-vector multiplication. However, many computational package exist where these operations are optimised, leaving the step of orthogonalisation particularly taxing in terms of computational time [38].

More work has been undertaken in regards to the least squares solution in the algorithm. Methods involving Cholesky factorisation [40], QR factorisation [41] and gradient methods have been proposed. OMP provides a fast algorithm, both computationally as well as theoretically but cannot offer guarantees to match those of Basis Pursuit [18]. OMP however, still has the property to recover a s-sparse signal completely, provided the number of measurements taken are closely proportional to s [42].

Orthogonal Matching Pursuit is summarised in Algorithm 2 below.

Algorithm 2 Orthogonal Matching Pursuit (OMP)

```
procedure OMP($\Phi, y, s$)                                        ▷ Required Inputs
    Initialisation: $I^{[0]} = \varnothing, r^{[0]} = y, \hat{x}^{[0]} = \bar{0}$
                        ▷ Initialising the residual and support set and estimate for $\hat{x}$
    for $i = 1, i = i + 1$ till the stopping criteria do
        $g^{[i]} = \Phi^T r^{[i-1]}$
        $j^{[i]} = \underset{j}{\text{argmax}} |g_j^{[i]}| / ||\Phi_j||_2$
        $I^{[i]} = I^{[i-1]} \cup j^{[i]}$
        $\hat{x}^{[i]} = \Phi^{\dagger}_{I^{[i]}} y$
        $r^{[i]} = y - \Phi \hat{x}^{[i]}$
    end for
    return $\hat{x}^{[i]}, r^{[i]}$                                 ▷ Required Outputs
end procedure
```

Elaborating on the OMP algorithm, the approximation for x gets updated per iteration by the projection of y orthogonally onto the columns of Φ accompanied with the present support set $I^{[i]}$. Importantly, this allows OMP to minimise $||y - \Phi\hat{x}||_2$ over all $\hat{x}$ with support $I^{[i]}$. Therefore the major difference from MP to OMP, is the fact that minimisation is undertaken with respect to all currently selected coefficients [38].

Tropp and Gilbert successfully showed that OMP recovers a fixed signal with high probability by proving the following [12]:

Theorem 2.8. ***(OMP) Signal Recovery*** *[12]*
Fix the RIP parameter $\delta \in (0, 0.36)$ and let Φ be an $N \times s$ Gaussian measurement matrix with measurements $M \geq cM\log(s/\delta)$. Let x be a s-sparse signal in $\mathbb{R}^N$. Then with probability exceeding $1 - 2s$, OMP correctly reconstructs signal x from its measurements Φx.

Similar results are found in the case of a subGaussian measurement matrix Φ. A disadvantage is that the above probability holds only for fixed signals. It is also unknown as to whether OMP works in the case of random Fourier matrices [18].

Additional concerns with the use of OMP, especially to that of large scale data, is that of computational and storage costs of a single iteration being quite high for large scale problems.

Recently there has been progress in the use of the RIP to analyse the performance of OMP in regards to non sparse signals [43], however this RIP analysis remains an area of open work. Performance improvements can also be made by considering restrictions imposed on the smallest non zero value in a signal, done in [29].

2.2.5 Stagewise Orthogonal Matching Pursuit

Stagewise Orthogonal Matching Pursuit (StOMP) is an extension of OMP. Devised by Donoho *et al* [14]. Inspired by ideas used in the wireless communications industry [44]. StOMP signifies an improvement over OMP and has established itself as one of the frontier algorithms within Compressed Sensing [45].

Its main difference with OMP, is the manner in which it selects columns from the measurement matrix. Specifically by allowing for multiple coefficients to be added within one iteration. This is done by fixing a specific threshold value. Any column whose correlation value is found to be over this value is selected as a matching column. As such, this highlights the paramount importance in the selection of a correct threshold value, with the performance of the algorithm dependent upon it. More specifically, a threshold parameter $\lambda^{[i]}$ regarding StOMP may be defined with,

$$\lambda^{[i]} = t^{[i]} \frac{||r^{[i-1]}||_2}{\sqrt{M}}. \tag{2.19}$$

Donoho [14] *et al.*, suggest ideal values for $t^{[i+1]}$ hold in the domain $2 \leq t^{[i]} \leq 3$, as well as providing specific formulae for the calculation and derivation of $t^{[i]}$ [14].

The idea behind the thresholding strategy is that it allows many terms to enter at each iteration, with the algorithm halting after a fixed number of iterations.

The noise level ξ is proportional to the Euclidean norm of the residual at each respective iteration [18]. Formally, should the measurement matrix Φ have columns sampled from the unit sphere and provided M and N are large, then the entries $z = \Phi^T y - x = \Phi^T \Phi x - x$ have a histogram which is approximately Gaussian with the standard deviation,

$$\xi \approx \frac{||x||_2}{\sqrt{M}}, \tag{2.20}$$

as proved by Donoho and his collaborators in [14].

We may now discuss the algorithm with more rigour. Consider I, the support set of x, and I_s the support set of $\hat{x} = x_s$. Then the elements of I_s are known as *discoveries*, conversely, the complement of $\bar{I}_s$ have elements known as *false discoveries*. Should an element of the true support set I not appear in I_s, then this referred to as a *missed detection*. On the other hand, should an element be present in I_s but not I, then this is aptly named a *false alarm*.

As a result of this distinction, the determination of the threshold parameter can be done with one of two ways. As detailed by Abramovich *et al* [46]. The selection may be either,

1. A guarantee made on the number of false alarms, Ensuring the number does not exceed $M-s$ over all iterations. This allows for a threshold parameter to enforce the false alarm rate by means of a per-iteration allowance.

2. Alternatively, the use of false discovery can be used to determine the threshold, By choosing a value which does not exceed a particular fraction of the total number of elements summed over all iterations.

Algorithm 3 Stagewise Orthogonal Matching Pursuit StOMP

```
procedure StOMP($y, t, s$)
    Initialisation: $r^{[0]} = y, I^{[0]} = \varnothing, x^{[0]} = \bar{0}$
    for $i = 1, i = i + 1$ till the stopping criteria do
        $g^{[i]} = \Phi^T r^{[i]}$
        $\alpha^{[i]} = \{j : |g^{[i]}_{[j]}|\}$          ▷ Selection of elements larger the given threshold value $t$
        $I^{[i]} = I^{[i-1]} \cup \alpha^{[i]}$          ▷ Update the support set
        $x^{[i]} = \Phi^{[T]}_{I^{[i]}} y$
    end for
    return $\hat{x}^{[i]}$          ▷ Required Output
end procedure
```

Although StOMP has a structure similar to that of OMP. Due to the fact that StOMP selects many coefficients at each iteration, the computational complexity is vastly improved. Specifically, the problem can be reduced to a runtime of $KnkN + \mathcal{O}(N)$, where n is the fixed number of iterations and K a constant dependent upon the accuracy level of the least squares problem [18].

2.2.6 Iterative Hard Thresholding

Although we have seen that greedy methods are simple in terms of implementation as well as the advantages of fast speeds. They suffer in terms of recoverability guarantees unlike the more stringent convex optimisation methods.

Thresholding offers to bridge the gap between the two above approaches. An overview of thresholding algorithms can be found in [47]. The primary algorithm under question in this section is that of Iterative Hard Thresholding (IHT) found in [17].

The IHT algorithm surfaces separately in [17] and [48] after initial developments by Kingsbury and Reeves found in [49].

Although IHT does fall under the umbrella of greedy algorithms, it attempts to iteratively solve a local approximation to the following [38],

$$\min_{\hat{x}} ||y - \Phi\hat{x}||_2^2 \qquad \text{such that } ||\hat{x}||_0 \leq s. \tag{2.21}$$

Instead of solving (2.21) however, a substitute objective function is found. This local approximation is derived upon optimisation transfer framework found in [50]. The cost function,

$$C_s^S(\hat{x}, z) = \mu||y - \Phi\hat{x}||_2^2 - ||\Phi\hat{x} - \Phi z||_2^2 + ||\hat{x} - z||_2^2, \tag{2.22}$$

or alternatively,

$$C_s^S(\hat{x}, z) \propto \sum[\hat{x}_j^2 - 2\hat{x}_j(z_j + \mu\Phi_j^T y - \Phi_j^T \Phi z)], \tag{2.23}$$

where Φ_j represents the columns of Φ.

Equation (2.23) shows that the optimisation of each $\hat{x}$ can be done independently. Further, the consideration of the $||\hat{x}||_0 \leq s$ constraint implies that (2.22) has the minimiser [38],

$$x^* = z + \mu\Phi^T(y - \Phi z). \tag{2.24}$$

Interestingly, at this minimum value, the cost function assumes a value proportional to,

$$C_s^S(\hat{x}, z) \propto ||x^*||_2^2 - 2\langle x^*, (z + \mu\Phi^T(y - \Phi z))\rangle = -||x^*||_2^2. \tag{2.25}$$

The constraint can then be reintroduced by selecting the s largest coefficients of x^* and assigning zero to all other coefficients.

The cost function found in (2.22) is therefore minimised to the constraint $||x||_0 \le s$ at,

$$\hat{x} = H_s(z + \mu\Phi^T(y - \Phi z)), \tag{2.26}$$

with the operator H_s being the nonlinear projection that assigns all except the s largest elements to zero [38].

Algorithmically speaking, the required conversion from the optimisation to an iterative approach is done by substituting $z = \hat{x}^{[i]}$. The Iterative Hard Thresholding algorithm follows;

Algorithm 4 Iterative Hard Thresholding (IHT)

```
procedure IHT($\Phi, y, s, \mu$)
    Initialisation $x^{[0]} = \bar{0}$
    for $i = 0; i = i + 1$ until stopping criteria do
        $\hat{x}^{[i+1]} = H_s(\hat{x}^{[i]} + \mu\Phi^T(y - \Phi\hat{x}^{[i]}))$
    end for
    return $\hat{x}^{[i]}$
end procedure
```

IHT shows to be a computational efficient algorithm . Major computational steps involve vector-matrix multiplication. As such, it provides a low storage approach with Φ and Φ^T multiplication done extremely efficiently when well constructed [38].

When considering the convergence of the IHT algorithm, the prominent conditional parameter is that of the step size μ. This step size is reliant on the matrix Φ and the value β_{2s} [38], which is the smallest value such that,

$$||\Phi(x_1 - x_2)||_2^2 \le \beta_{2s}||x_1 - x_2||_2^2, \tag{2.27}$$

holds for all s-sparse vectors x_1 and x_2. This is related back to the RIP constant where,

$$\beta_{2s} \le (1 + \delta_{2s}) \le ||\Phi_2^2||. \tag{2.28}$$

In [17], proof within, Blumensath and Davies state the convergence formally as;

Theorem 2.9. *if $\beta_{2s} < \mu^{-1}$, $s \le M$ and assume Φ is of full rank, then the sequence $\{\hat{x}^{[i]}\}$, defined by the IHT algorithm converges to a local minimum of* (2.21).

[1] Note that $C(Y) = C^S(Y, \Phi)$

2.2.7 Compressive Sampling Matching Pursuit

So far this dissertation has covered two major categories concerning the Compressed Sensing recovery problem. The mathematically elegant optimisation approach provides robustness and strong guarantees but lacks the computational speed compared to the greedy methods. Conversely, the greedy methods have so far been unable to provide the guarantees of the Basis Pursuit method.

Needell and Vershynin changed this however, by introducing a new algorithm possessing the speed of the greedy approaches, while also providing strong guarantees akin to that of the optimisation approach. They did this through *Regularised Orthogonal Matching Pursuit* (ROMP) [51], the first algorithm to bridge the divide between the two major approaches.

Although ROMP made massive strides towards an ideal recovery algorithm, due to its heavy requirements imposed upon the RIP (see [51]), this opened weaker bounds on error when dealing with noisy signals.
This was rectified however, with the development of Compressive Sampling Matching Pursuits (CoSaMP) [18].

The difference between ROMP and OMP was that at each iteration, ROMP selects more than just one element to join the support set. Although this allowed for mistakes to be present within the support set, it is still able to reconstruct the signal correctly. This was achievable due to bounds on the number of mistakes the algorithm can make [51].

Similarly to Subspace Pursuit (to be discuss later), CoSaMP monitors the required support set I, while adding and removing elements each iteration.

Each iteration begins with an s-sparse estimate $\hat{x}^{[i]}$, which in term is used to calculate the residual, $y - \Phi\hat{x}^{[i]}$, with the inner products of the column vectors of Φ calculated [38]. The particular indices of columns found within Φ related to the largest inner products are selected and added to the support set $I^{[i+0.5]}$ for the estimate $\hat{x}^{[i]}$. An additional intermediate step is made with an estimate for $\hat{x}^{[i+0.5]}$, the solution to the least squares problem $\underset{\tilde{x}_{I^{[i+0.5]}}}{\text{argmin}} ||y - \Phi\tilde{x}^{[i+0.5]}_{I^{[i+0.5]}}||_2$. Finally, the largest s entries of $\hat{x}^{[i+0.5]}$ are selected and used as the new support set $I^{[i+1]}$ [38]. The main CoSaMP algorithm outlined by Needell and Tropp can be written as found in algorithm 5 below.

Needell and Tropp [15] also introduced faster implementations of CoSaMP, primarily with the replacement of the least squares problem $\hat{x}^{[i+0.5]}_{I^{[i+0.5]}} = \Phi^{\dagger}_{I^{[i+0.5]}} y$ with either a conjugate gradient technique or specifically a three iteration gradient decent.

Algorithm 5 Compressive Sampling Matching Pursuit (CoSaMP)

```
procedure CoSaMP($\Phi, y, s$)
    Initialisation $x^{[0]} = \bar{0}, I^{[0]} = \text{supp}(H_s(\Phi^T y))$
    for $i = 0, i = i + 1$, until stopping criteria is met do
        $g^{[i]} = \Phi^T(y - \Phi\hat{x}^{[i]})$
        $I^{[i+0.5]} = I^{[i]} \bigcup \text{supp}\big(g^{[i]}_{2s}\big)$
        $\hat{x}^{[i+0.5]}_{I^{[i+0.5]}} = \Phi^{\dagger}_{I^{[i+0.5]}} y$
        $I^{[i+1]} = \text{supp}\big(\hat{x}^{[i+0.5]}_s\big)$
        $\hat{x}^{[i+1]}_{I^{[i+1]}} = \hat{x}^{[i+0.5]}_{I^{[i+1]}}$
    end for
    Output $\hat{x}^{[i]}$ and $r^{[i]}$
end procedure
```

Notably, the original algorithm as well as Needell and Tropp's proposed alternative implementations all possess the guarantees given by the following theorem.

Theorem 2.10. *[15] For any x, given $y = \Phi x + \epsilon$ where Φ satisfies the RIP with,*

$$0.9 \leq \frac{||\Phi x_1 - \Phi x_2||_2^2}{||x_1 - x_2||_2^2} \leq 1.1, \tag{2.29}$$

for all $2s$-sparse vectors x_1 and x_2 after,

$$i^* = \big(log\Big(\frac{||x_s||_2}{||e||_2}\Big) log 2\big), \tag{2.30}$$

iterations, the CoSaMP algorithm calculates a solution $\hat{x}^{[i]}$ which satisfies,

$$||x - \hat{x}^{[i^*]}||_2 \leq 21\left(||y - x_s||_2 + \frac{||x - x_s||_1}{\sqrt{s}} + ||e||_2\right). \tag{2.31}$$

Although a negligible difference in the Euclidean norm of two successive estimates, $||\hat{x}^{[i]} - \hat{x}^{[i+1]}||_2$ could be used as a stopping criterion, this does not guarantee convergence. Alternatively, a stricter approach guaranteeing no presence of instability would be in the case where the error, ($||y - \Phi\hat{x}^{[i]}||_2 < ||y - \Phi\hat{x}^{[i+1]}||_2$) begins to increase [38].

2.2.8 Subspace Pursuit

Subspace Pursuit marks a p-pursuit algorithm. Developed by Dai and Milenhovic [52]. It hails from the initial algorithmic idea used in the A^* order statistic algorithm [53]. Subspace Pursuit (SP) is similar to the CoSaMP algorithm and was under development simultaneously.

The difference between SP and CoSaMP arrives in the extension phase of the algorithm. Although SP has larger computational work per iteration as opposed to OMP, it has been empirically shown to require fewer iterations [28].

In SP, Dai and Milenhovic [52] apply the stopping criteria on the difference of two successive iterations; $||y - \Phi\hat{x}^{[i]}||_2 - ||y - \Phi\hat{x}^{[i+1]}||_2$, thus ensuring stability even in cases when the RIP condition doesn't hold [38].

The primary difference from CoSaMP to that of SP, lies in the addition of entries to the support set $I^{[i]}$ within each iteration. A further difference, requires the SP algorithm to have an additional least-squares solution. This leads to an implementation efficiency matter, whereby the least-squares solution used in the CoSaMP algorithm has the potential to be replaced by three gradient-based updates [38], a distinct advantage over SP.
The SP algorithm follows in algorithm 6.

Algorithm 6 Subspace Pursuit (SP)

```
procedure SP($\Phi, y, s$)
    Initialisation $x^{[0]} = \Phi^{\dagger}_{I^{[0]}} y, I^{[0]} = \text{supp}(H_s(\Phi^T y))$
    for $i = 0, i = i + 1$, until $||y - \Phi\hat{x}^{[i+1]}||_2 \geq ||y - \Phi\hat{x}^{[i]}||_2$ do
        $g^{[i]} = \Phi^T(y - \Phi\hat{x}^{[i]})$
        $I^{[i+0.5]} = I^{[i]} \bigcup \text{supp}\big(g^{[i]}_{2s}\big)$
        $\hat{x}^{[i+0.5]}_{I^{[i+0.5]}} = \Phi^{\dagger}_{I^{[i+0.5]}} y$
        $I^{[i+1]} = \text{supp}\big(\hat{x}^{[i+0.5]}_s\big)$
        $\hat{x}^{[i+1]} = \Phi^{\dagger}_{I^{[i+1]}} y$
    end for
    Output $\hat{x}^{[i]}$ and $r^{[i]}$
end procedure
```

SP, CoSaMP as well as IHT all offer near optimal performance guarantees provided the conditions of the RIP are satisfied. Dai and Milenhovic [52] have the performance derived and proved in [52] with,

Theorem 2.11. *[19] For any x, given $y = \Phi x + \epsilon$ where Φ satisfies, for all s-sparse vectors x_1 and all $2s$-sparse vectors x_2, the RIP with,*

$$0.927 \leq \frac{||\Phi x_1 - \Phi x_2||_2^2}{||x_1 - x_2||_2^2} \leq 1.083, \tag{2.32}$$

then SP calculates a solution $\hat{x}$ satisfying,

$$||x - \hat{x}|| \leq 1.18\big(||y - x_s||_2 + \frac{||x - x_s||_1}{\sqrt{s}} + ||\epsilon||_2\big) \tag{2.33}$$

2.3 Alternative Approaches

Although the algorithms covered in Sections 2.1 and 2.2 are discussed in detail, they certainly are not the only family and groups of algorithms that are available for use in the Compressed Sensing problem. This Section briefly makes mention of these additional approaches for the purpose of establishing the already large depth of the field. Since these algorithms are not following with implementation in the numerical results, their specific algorithms are not listed for the sake of brevity.

2.3.1 Combinatorial Algorithms

Combinatorial algorithms historically developed in the computer science community prior to the inception of Compressed Sensing are highly relevant to sparse signal recovery.

These combinatorial methods make use of group testing to recover sparse signal. Real world examples of these algorithms can be found in [54–57]. Further usage of combinatorial algorithms have surfaced within the field of data streams [58, 59]. These problems often relate to the recovery of some x from a system Φx, presenting its similarity to the Compressed Sensing problem [21].

Coming back to the Compressed Sensing problem however, combinatorial algorithms have some important inherent differences. Namely, the reconstruction algorithms allow for the choice of Φ be one which reduces the computation required to achieve recovery. That is to say, a case where Φ has minimal non-zero elements, i.e the sensing matrix is itself sparse [21, 60].

Algorithms making use of such an idea are Fourier Sampling Algorithm [61], Chaining Pursuits [62], and Heavy Hitters on Steroids (HHS) [63].

Although these methods involve specific construction of the sensing matrix, there does exist methods which can make use of general sparse matrices as can be found in [64].

Combinatorial algorithms unlike its convex optimisation and greedy algorithm counterparts, whose computational complexity is at best minimally linear in terms of N due to the fact that the recovery of x requires at least the computational cost of reading out all N entries of x. These methods suffer immensely once the signal length N becomes extremely large. Rather, one would invest in an algorithm whose complexity is only linear in the length of representation of the signal, for example the signal sparsity [21]. Algorithms such as these do not return a signal reconstruction of x but instead return the largest s largest entries and their specific indices, requiring a simple remaining step to the construction of the signal approximation. Examples of such methods can be found here [65–67].

2.3.2 Bregman Iterative Algorithms

Yin, Osher *et al* [68], proposed a simple yet efficient method for solving the Basis Pursuit problem. Originating from *Bregman Iteration Regularisation*, it poses the idea of achieving the exact solution of constrained problems by iteratively solving a succession of sub-problems which are unconstrained. Yin, Osher *et al* [68], show that in most cases, no more than two to six iterations are required to achieve an exact solution. Theoretical and numerical guarantees can be found in [68].

2.3.3 Non Convex Minimisation Algorithms

Non convex optimisation algorithms are heavily used in real world applications. *Focal Underdetermined System Solution* (FOCUSS) [69, 70], for example is a Compressed Sensing framework used in medical MRIs. Other algorithms, such as Iteratively Reweighed Least Squares along with Sparse Bayesian Learning Algorithms [71], have also found merit in the medical tomography community [72, 73].

Further examples of real world applications of non convex minimisation algorithms include the processing of sparse music and audio signals making use of Monte-Carlo based algorithms as found in [74].

2.3.4 Message Passing Algorithms

It has already been mentioned that convex optimisation methods scale heavily in expense when considering large scale applications, as such quick iterative thresholding algorithms have been under heavy study as alternatives when dealing with large set problems. Often however, the investment of speed comes with poorer sparsity-undersampling tradeoffs relative to convex optimisation approaches like that of BP.

Fortunately, Donoho *et al,* proposed a new idea of algorithms based off a modification of iterative thresholding [75]. Animated by idea of belief propagation, they developed *Message Passing Algorithms* (MPA). These MPA techniques allowed for a sparsity-undersampling tradeoff to match that of convex optimisation approaches.

This approach has numerous advantages such as its low computational complexity and importantly, the potential to be easily implemented in parallel computation, an area which is becoming of increasing importance as the GPU programming paradigm establishes itself as a computational norm.

Some examples of these algorithms can be found with Expander Matching Pursuits [76], as well as Sparse and Sequential Sparse Matching Pursuits [77, 78].

Chapter 3

Results

The results of implementation now follow, Matching Pursuit (MP) has not been considered due to its extension in OMP.
All algorithms have been implemented in MATLAB. The hardware used for empirical runs consisted of the following: ASUS Rampage Extreme 4 motherboard, Core i7-3930K CPU @ 4.25 GHz, 64 GB RAM @ 1666 MHz and 1× Nvidia GTX 680.

Firstly, the algorithms were implemented on the CPU. Algorithm recovery and computation time was investigated as well as the Kolmogorov-Smirnov test for analysing a *goodness-of-fit*. An extension of the computation time was undertaken with the GPU.

3.1 CPU Results

Numerical results were obtained using the CPU to investigate the efficiency and ability of the methods listed in Chapter 2.

The investigations listed pertain to a small sample of empirical studies undertaken. Further extensions and results that are not listed here are available for each algorithm. Major results for BP can be found in the following [7, 10, 79]. Tropp and Gilbert probe OMP deeply in [12]. Needell lists large numerical investigation into CoSaMP with [18]. Subspace Pursuit (SP) is empirically analysed by Dai and Milenkovic in [80]. StOMP is detailed in [14] by Donoho *et al.* While IHT is covered in detail in [17, 38, 81].

The recovery itself was tested with the following computational runs:

We denote the number of measurements by M and a signal length ny N. The original signal x, is generated to be a random vector containing s nonzero elements. These nonzero elements are generated randomly from the standard normal distribution and placed in random locations within the signal vector. Although numerous other signals could be used for testing, and undoubtedly performance of the algorithms to follow may differ from signal to signal. The aim of the empirical results recorded here is to serve only as a guideline to their performance. For extensive experimentation could be conducted with alternative signal types such as, periodic/aperiodic, ± 1 signals, causal, anti casual signals etc, with useful comparisons to be drawn.

The measurement matrix Φ of size $M \times N$ is also generated randomly in accordance to i.i.d. Gaussian entries. That is to say the matrix has entries populated from $\mathcal{N}(0, M^{-1})$.

Next, three specific configurations were chosen. Setting the number of measurements M to a fraction of N, we consider $M/N = 1/8, 1/4, 1/2$, while keeping the length of the signal N fixed with a value of 256.

Following the selection of these configurations. Numerical trials are run over varying sparsity to test the efficiency and accuracy of the algorithms. In configuration 1, where $M/N = 1/8$, we consider a sparsity s ranging between 1 and 20. The second configuration $M/N = 1/4$ is run with sparsity changing from 1 to 40. While the third configuration, $M/N = 1/2$ is a sparsity range of 1 to 80. All three configurations ran with sparsity increasing in steps of 1.

Recovery for the above configurations was considered successful when the relative error between the recovered signal and original signal is less than 10^{-5}, i.e. $||\hat{x} - x||/||x|| < 10^{-5}$, where $\hat{x}$ is the reconstructed signal and x is the exact signal. Each configuration was run for

200 trials and the recovery measured accordingly. This was repeated for each algorithm under investigation.

Another area under investigation was the speed of each algorithm. To test this, trials were computed and timed with varying signal lengths. Using the above ratios of M and N, we increase the signal length in the base 2 power from 2^8 to 2^{15} (i.e. $N = 256$ to $N = 32768$). The number of measurements M increases accordingly to the configurations being tested (i.e. 1/8,1/4,1/2). The sparsity in all timing trials is set to $s = 1/8 \times M$ (1/8 of the measurements taken). Again 200 trials were run for all values of N and the results averaged for each algorithm.

3.1.1 Algorithm Recovery With $\dfrac{M}{N} = \dfrac{1}{8}$

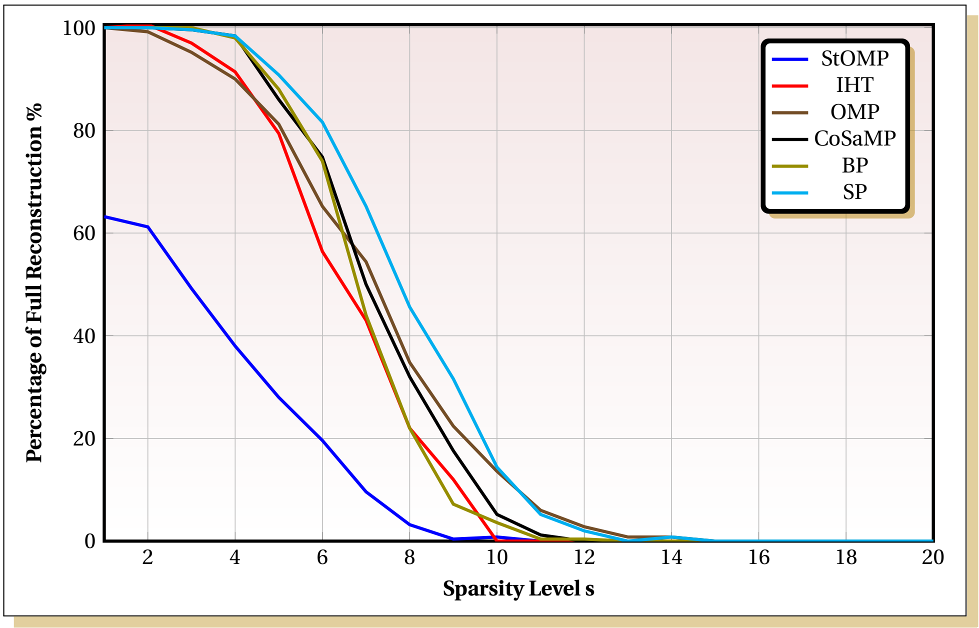


FIGURE 3.1: **The percentage of signals recovered with the signal length N = 256 and fixed measurements M = 32, with varying sparsity levels.**

When dealing with the above configuration, it is important to point out that this is the sparsest numerical trial. Importantly however, since the number of measurements is the lowest here as well, this configuration would represent the most ideal recovery case. That is to say, when dealing with extremely sparse signals, the best performing algorithm corresponding to the least number of measurements is sought. Consider Figure 3.1, the immediate result of interest is the performance of StOMP. Compared to all other methods under consideration, it under performs immensely. The implementation of StOMP itself involved the usage of the SparseLab [82, 83], by Donoho *et al.* The SparseLab code makes use of a threshold parameter as discussed in Chapter 2. This parameter is defaulted to 0.5 in the code, however, as mentioned the ideal range for this value is on the domain $[2 \leq t \leq 3]$. As such, choosing alternative values for this in the code found even worse results. Judging by this it would seem that this value is in fact the reciprocal. Thus certain refinement on this parameter could lead to some better performance. SparseLab itself is a MATLAB toolbox consisting of numerous sparse optimisation techniques. Indeed the BP implementation is also a modified run from SparseLab, this seemingly provided better results (at least computational run times) over the ℓ_1-magic toolbox [84, 85], written by Romberg *et al.* With regard to the other algorithms, SP was implemented with a modified version of Igor Carron's code found here [86]. As we can see, SP out performs all other techniques overall. This holds true with results obtained by Maleki and Donoho showing SP outperforming CoSaMP [87]. IHT however, appears to under perform under the above conditions. Blumensath and Davies show in [38], that IHT both out performs and under performs against CoSaMP depending on whether CoSaMP is implemented with a pseudo inverse or alternatively a conjugate gradient step respectively. The implementation of IHT here is taken from a toolbox written by Blumensath named sparsify [88], sparsify is a collection of MATLAB files which may be used to implement a wide variety of different Compressed Sensing algorithms, ranging from greedy methods to thresholding algorithms. The hard thresholding itself keeps exactly M elements in each iteration. As such, with further numerical trials no doubt this result could be improved with an optimal number of elements kept per iteration. McCoy provides updated code based off the original CoSaMP and OMP implementation of Needell and Tropp [89]. Modifying this to the specific numerical configuration above, we see that OMP performs extremely well and only begins to lose out to CoSaMP, BP and SP once the sparsity has reached approximately 1/5 of the measurements taken. Basis Pursuit (BP) also performs adequately above too, however, as will be shown later, BP struggles in regards to computational time as the signal length increases.

3.1.2 Computational Time With $\frac{M}{N} = \frac{1}{8}$

The following table and plots illustrate the computational runtimes involved in the first numerical configuration. Each algorithm was run with 200 trials and the time averaged. Table 3.1 follows with the reported times.

M/N(1/8)								
Signal Length (N)	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}
BP	0.0202	0.0445	0.1578	0.5877	3.5797	18.8230	103.7727	762.0937
OMP	0.0012	0.0034	0.0106	0.0229	0.1045	0.4693	3.0236	22.8810
CoSaMP	0.0009	0.0013	0.0027	0.0059	0.0220	0.0858	0.3697	3.7458
StOMP	0.0011	0.0019	0.0085	0.0486	0.1973	0.6917	3.2107	17.1235
IHT	0.0036	0.0076	0.0133	0.0282	0.1171	0.4258	1.6181	6.0168
SP	0.0029	0.0032	0.0052	0.0168	0.0638	0.2927	1.4904	9.210

TABLE 3.1: **Computational Times in Seconds (s)**

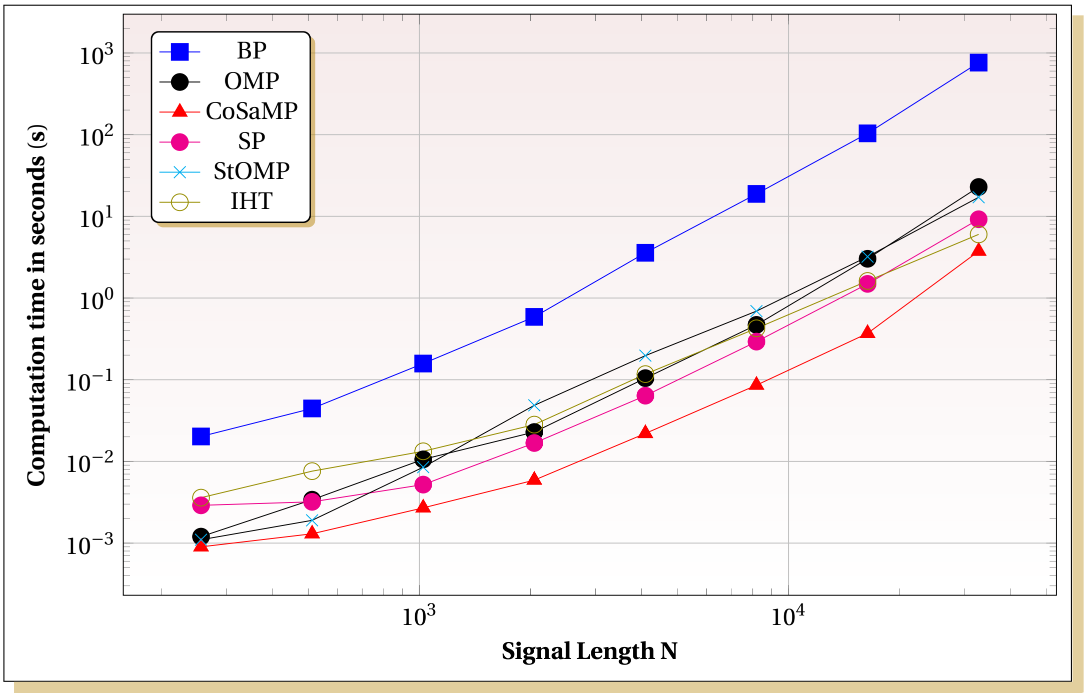


FIGURE 3.2: **Log Scale of Computation Time for M/N = 1/8**

As expected, all the algorithms appear efficient on small signal lengths (i.e. $N = 2^8 - 2^{10}$) however, from 2^{11} we begin to see massive computational differences. BP begins to show a sharp increase in time compared to all other methods. Indeed relatively speaking so does StOMP and OMP. It is also worth noting other interesting behaviour, CoSaMP appears the strongest overall, but IHT scales well with an increase in signal length, becoming the second fastest by $N = 2^{15}$. This seems to imply that IHT is an algorithm that definitely improves when larger signals are considered, while CoSaMP reigns supreme in the case of extremely sparse signals.

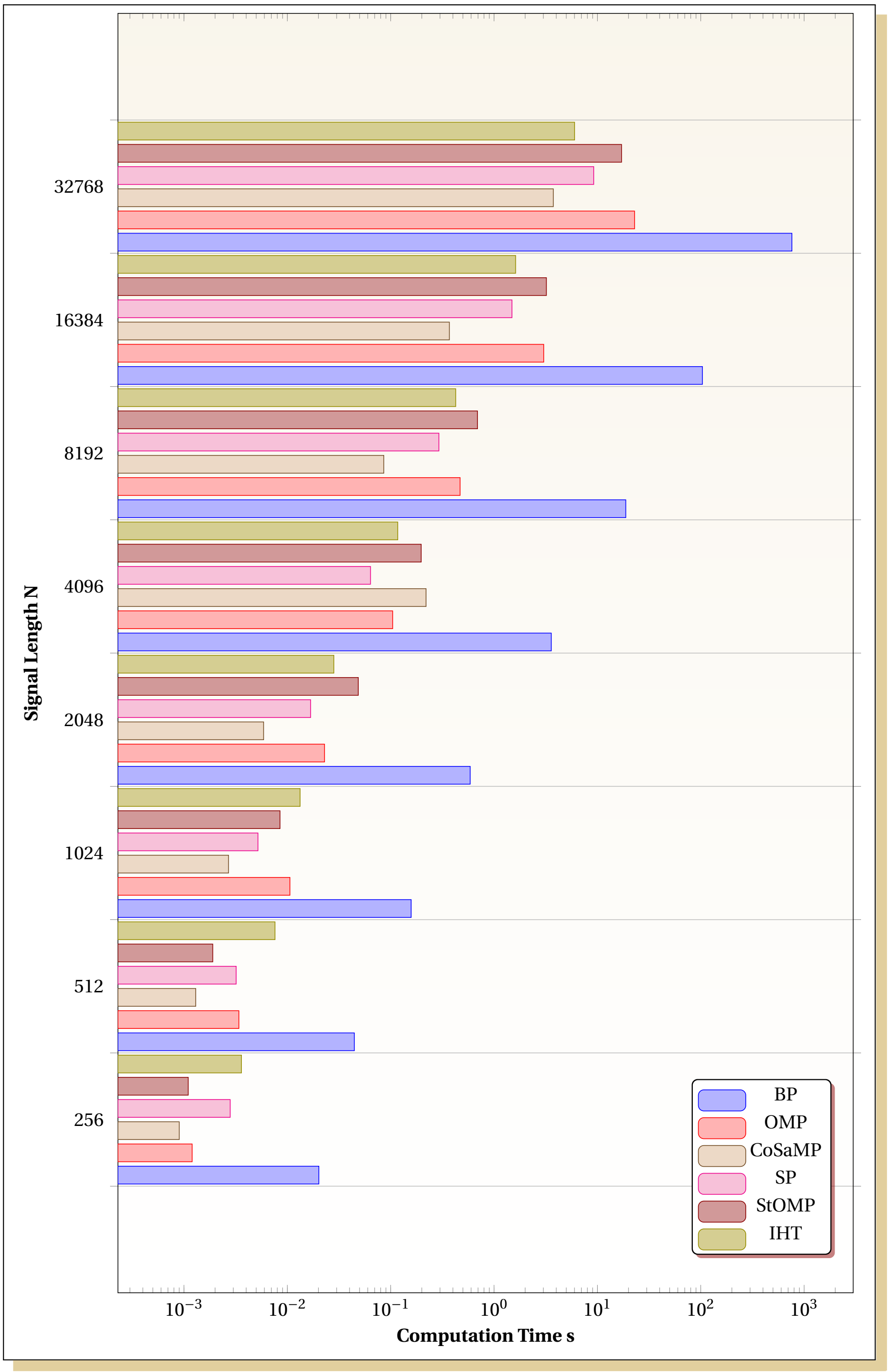


FIGURE 3.3: **Log Scale Bar Graph for Computation Times When M/N = 1/8**

Figure 3.3 shows the scaling between each algorithm as the signal size increases. It is interesting to see the greedy algorithms and thresholding algorithms clump together within some neighbourhood while the convex optimisation is orders of magnitude larger in runtime.

3.1.3 Algorithm Recovery With $\frac{M}{N} = \frac{1}{4}$

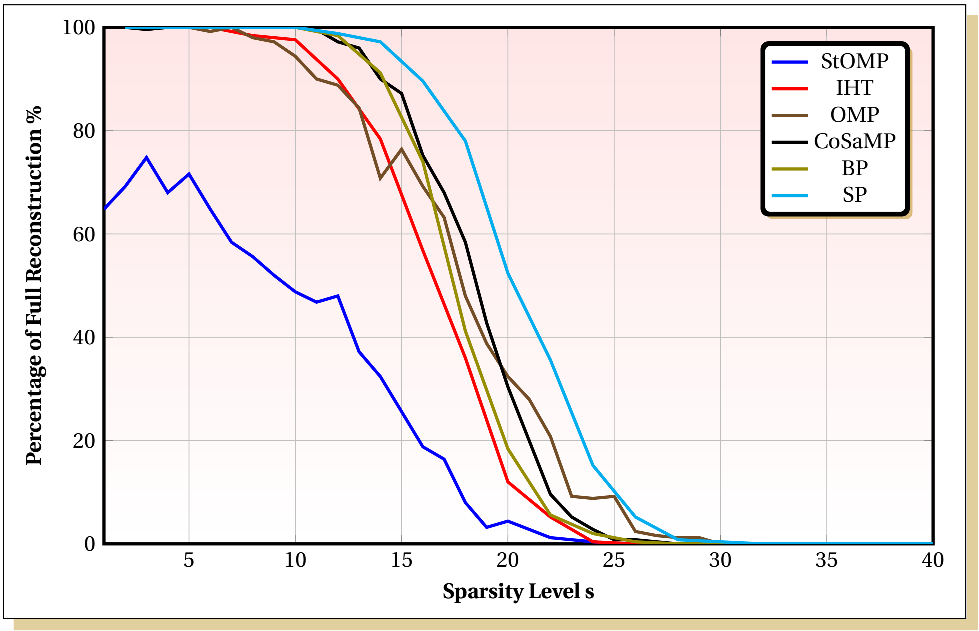


FIGURE 3.4: **The percentage of signals recovered with the signal length N = 256 and fixed measurements M = 64, with varying sparsity levels.**

We now consider the configuration of $M/N = 1/4$. In this case sparsity ranged from $s = 1$ to $s = 40$. Again we notice that StOMP is the worst performer. Subspace Pursuit (SP) again performs the best overall. We can also see that CoSaMP has now began to pull away from OMP, with it achieving the second best results. BP still continues to perform well, albeit at much longer computation time. IHT performs better here as opposed to the first configuration but still lags behind the others, again suggesting the importance of the selection of the number of nonzero elements to keep per iteration.

3.1.4 Computational Time With $\frac{M}{N} = \frac{1}{4}$

Again we tabulate and plot the resulting computational times with regard to the signal/measurement configuration. We begin to notice some expected trends as well as some new interesting developments.

M/N(1/4)								
Signal Length (N)	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}
BP	0.0273	0.0834	0.3318	1.8457	10.1041	56.4383	385.3201	2908.2
OMP	0.005	0.0093	0.0226	0.0689	0.3410	1.9466	14.6318	120.5985
CoSaMP	0.0013	0.0019	0.0045	0.0175	0.0701	0.3139	3.1661	26.8067
StOMP	0.0025	0.0041	0.0322	0.1140	0.3964	1.6751	12.2695	39.0968
IHT	0.0061	0.0081	0.0134	0.0345	0.1876	0.7030	2.6377	10.1864
SP	0.0032	0.0039	0.0133	0.0507	0.2437	1.3390	7.3405	44.3120

TABLE 3.2: **Computational Times in Seconds (s)**

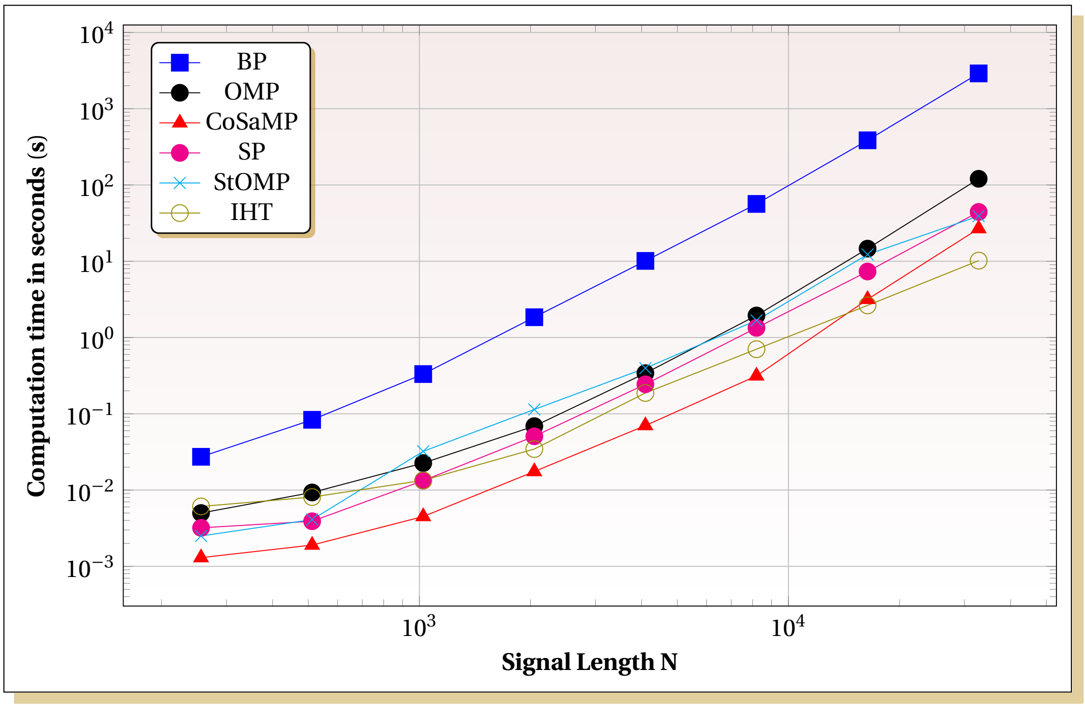


FIGURE 3.5: **Log Scale of Computation Time for M/N = 1/4**

Unsurprisingly, BP continues to scale badly with computational times orders of magnitude larger than the others under investigation. OMP worsens slightly when considering larger signals, while CoSaMP yet again continues to shine. The slight increase in performance, in regards to runtime saw in the first configuration by IHT is amplified further here as it becomes the quickest algorithm by $N = 2^{14}$ onwards, suggesting its performance ties in better with a larger number of measurements and signal length.

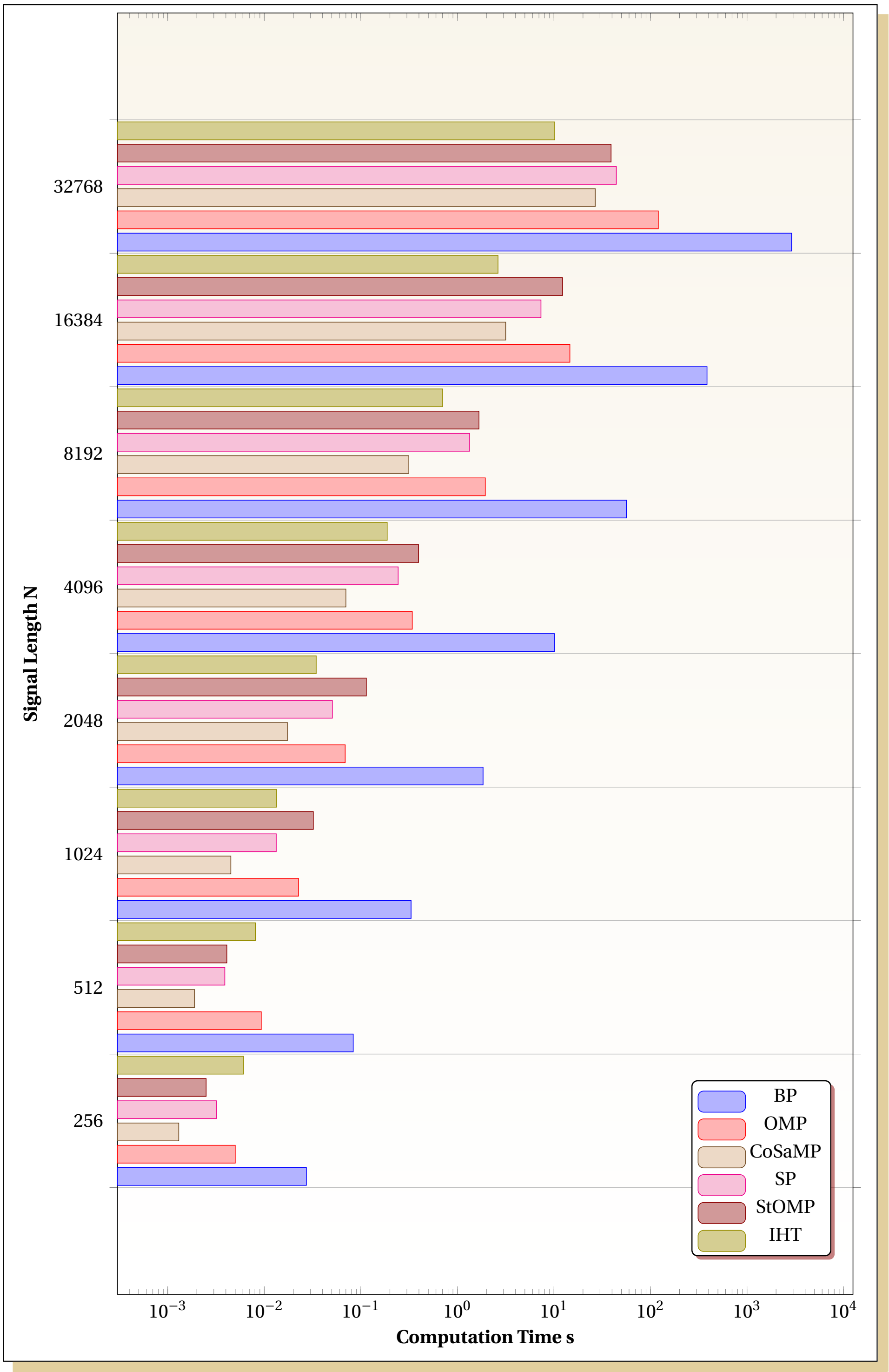


FIGURE 3.6: **Log Scale Bar Graph for Computation Times When M/N = 1/4**

Figure 3.6 again highlights what is happening in Figure 3.5 in a more visual manner, especially the increase in performance of the IHT algorithm as a small discrepancy in the graph corresponds to a large discrepancy because of the log scale. Again the greedy algorithms continue to cluster with one another while BP is magnitudes slower.

3.1.5 Algorithm Recovery With $\frac{M}{N} = \frac{1}{2}$

We now consider the final configuration. This case deals with the largest number of measurements taken and as such considered a much larger array of sparsity values. Figure 3.7 shows the recovery efficiency of each algorithm. Yet again, we see that SP achieves the best results and similarly StOMP the worst. With regard to the other algorithms, we finally see OMP being clearly overtaken by the others, while BP and CoSaMP achieving strikingly similar results. IHT again trails behind the top three although it does appear to be reaching a better level of performance. It is interesting to see that in all 3 configurations, StOMP constantly landing between 60 to 80% recovery, while all the others in eventuality reach 100%. Possible reasons for this are discussed later in more detail.

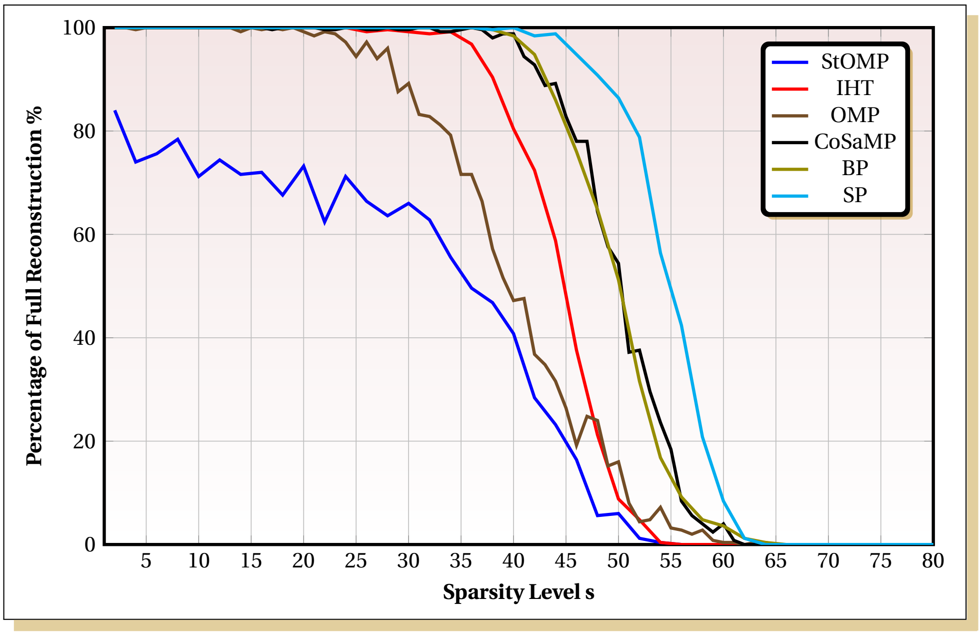


FIGURE 3.7: **The percentage of signals recovered with the signal length N = 256 and fixed measurements M = 128, with varying sparsity levels.**

3.1.6 Computational Time With $\frac{M}{N} = \frac{1}{2}$

Table 3.3 lists the computational times recorded for the final configuration on the CPU. We can now really begin to see the increased workload on the algorithms as the times are substantially larger than that of the first configuration which is to be expected. BP in particular has reached, relatively speaking, an immense amount of time per trial. Even the greedy methods have begun to reach large computational times. Interestingly however, IHT now show its superiority in regards to computation time when dealing with large signal sizes and increased measurement value.

M/N(1/2)								
Signal Length (N)	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}
BP	0.0297	0.0857	0.4205	2.3542	12.6919	82.0171	617.2454	4732.4
OMP	0.0128	0.0224	0.0633	0.2628	1.3753	10.2961	87.6122	694.5822
CoSaMP	0.0017	0.004	0.01045	0.0620	0.2826	2.9926	22.5482	174.5490
StOMP	0.0054	0.0189	0.0758	0.2791	1.2920	8.4087	44.1666	192.1980
IHT	0.0065	0.0091	0.0167	0.0792	0.3185	1.2249	4.6273	17.6595
SP	0.0065	0.0106	0.0407	0.2078	1.0572	6.4544	39.6360	261.1395

TABLE 3.3: **Computational Times in Seconds (s)**

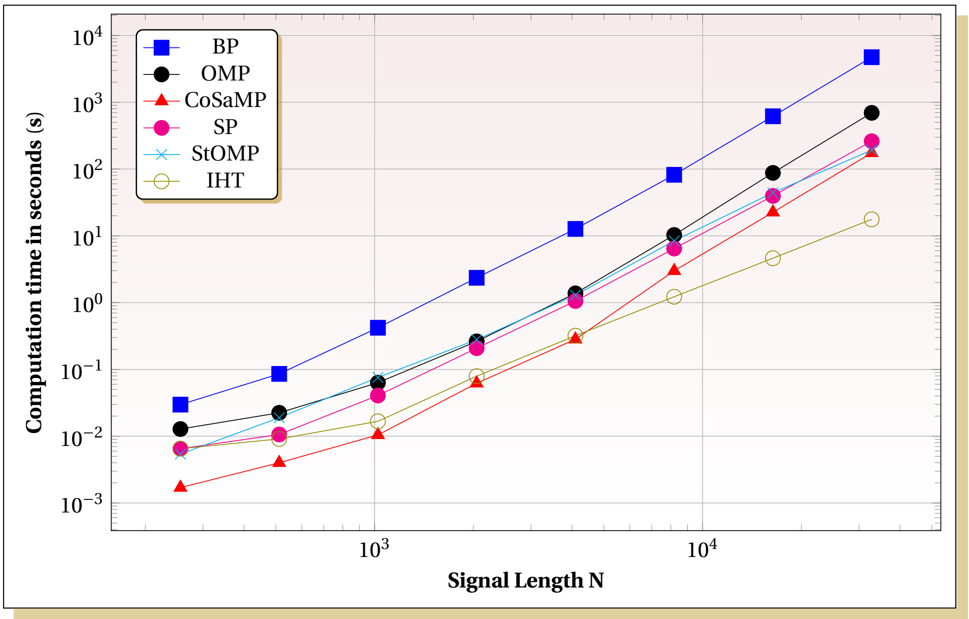


FIGURE 3.8: **Log Scale of Computation Time for M/N = 1/2**

Figure 3.8 shows this improved IHT result nicely. CoSaMP still maintains an edge over IHT until the signal length reaches $N = 2^{12}$, after that IHT vastly outperforms CoSaMP with IHT 10 times faster by the time the signal is of length $N = 2^{15}$. From the trend observed in Figure 3.8, it would seem acceptable to say that this behaviour should be expected to continue, hinting that IHT would be the method of choice when confronted with large signal lengths.

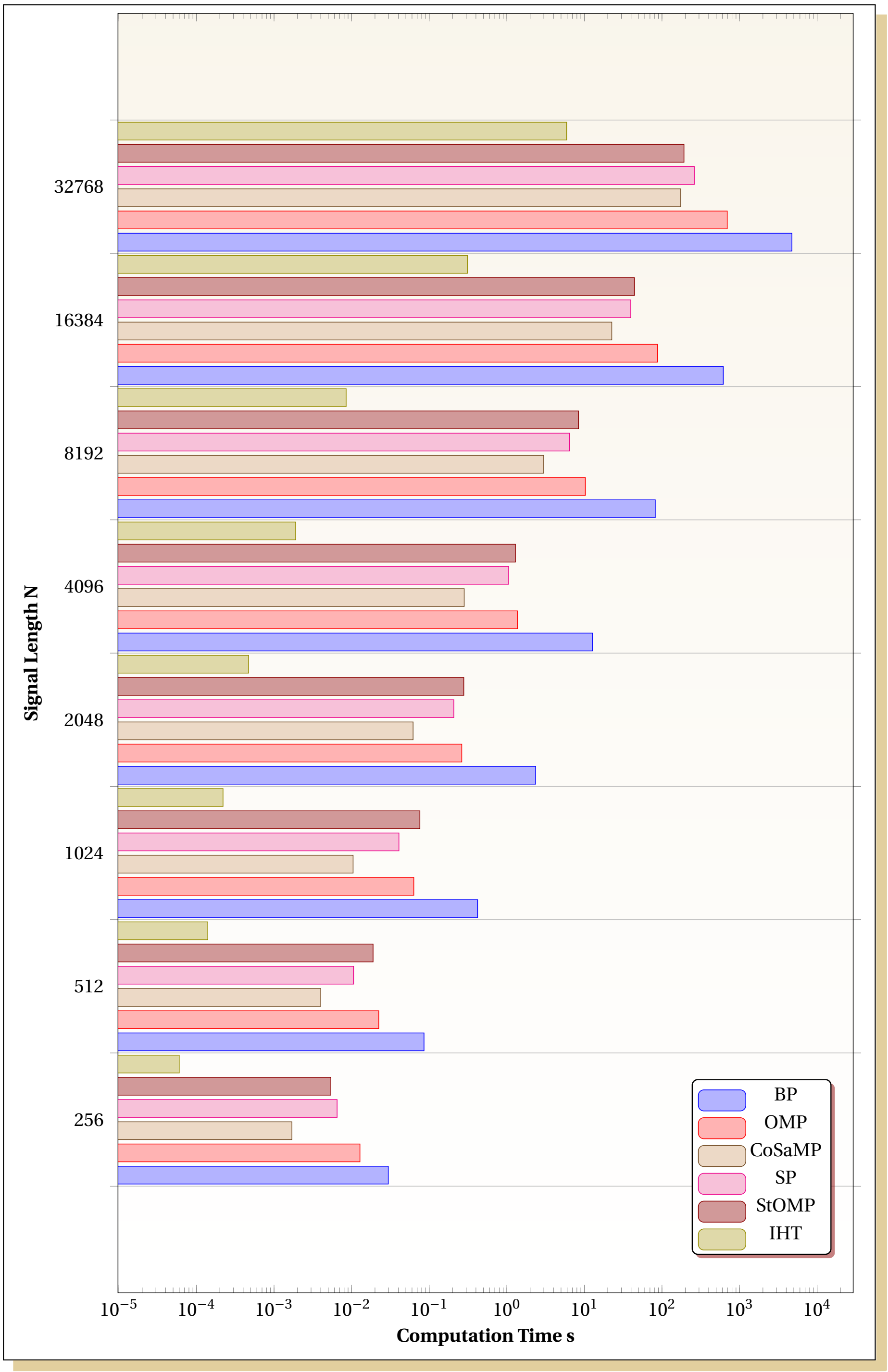


FIGURE 3.9: **Log Scale Bar Graph for Computation Times When M/N = 1/2**

3.2 Kolmogorov-Smirnov Test

In the reconstruction trials we measured success by relative error. This is indeed the most useful way when measuring the distance between an original signal and a reconstructed one.

We can however, consider a statistical analysis. Using the Kolmogorov-Smirnov test for a "*goodness-of-fit*", we can test whether two samples come from the same distribution [90]. In terms of our signals, this means we can see how well the reconstructed signals fit the original, or rather summarise the discrepancy between the two.

The test works by comparing the empirical distribution functions of the two signals. The comparrison is calculated with,

$$D^* = \sup_x |F_1(x) - F_2(x)|, \tag{3.1}$$

where F_1 and F_2 are empirical distribution functions of the original and reconstructed signals respectively. The **Null Hypothesis** under test is rejected at some significance level α if,

$$D^* > C_v(\alpha)\sqrt{\frac{n_1 + n_2}{n_1 n_2}}, \tag{3.2}$$

where n_1 and n_2 are the lengths of the original and reconstructed signal [90].

The significance levels under investigation for our tests are $\alpha = 0.05(5\%)$ and $\alpha = 0.01(1\%)$. The critical value $C_v(\alpha)$ is 1.36 at $\alpha = 0.05$ and 1.63[1] at $\alpha = 0.01$. For consistency we investigate the goodness-of-fit for all algorithms tested with reconstruction. Again the signal length was fixed at $N = 256$. The signal was generated in the same way, with the measurements taken at the ratios of $M/N = 1/8, 1/4$ and $1/2$. For each of these ratios the sparsity was also varied with, $s/M = 1/8, 1/4$ and $1/2$. These values were chosen to see if any conclusions could be drawn against the reconstruction results, while also investigating a wider array of configurations.

MATLAB was used to compute the test on all the aforementioned configurations, with the corresponding graphs found in Appendix A. The graphs plot the results of 200 trials per configuration.

The magenta curve represents the difference between distributions (i.e. D^*). The blue and red vertical lines are the critical values for $\alpha = 0.05(0.12)$ and $\alpha = 0.01(0.144)$[2] respectively. Should D^* pass these values then the Null hypothesis (that the signal x and the

[1] These values are obtained using tables found in [90].

[2] These values are calculated using equation 3.2.

approximation $\hat{x}$ are from the same distribution) is rejected.
The critical values indicate the maximum deviation possible between F_1 and F_2 at each level of significance. That is to say a significance level of $\alpha = 0.05$ implies that 95% of the time, the maximum deviation will be 0.12.
The blue curve found on the graphs, plots the probability of each particular trial passing the Hypothesis test. It is noteworthy to add, that if no curves are visible, that $D^*= 0$ and consequently the Hypothesis will always be accepted. This implies the signals are identical.

Looking at the results, we can see that BP starts well, passing the test for $M/N = 1/8$ and $s/M = 1/8, 1/4$ and $1/2$. With Figure A.2, we see that BP begins to struggle at $M/N = 1/4$, failing the test often when sparsity is $1/4M$. Interestingly this improves when the sparsity increases to $M/N = 1/2$. The final configuration for BP sees a 128 measurements taken and as such at $s = 1/8M$ and $s = 1/4M$ appears to perform well. Once $s = 1/2M$ however, BP fails at both significant levels.
With regards to OMP, we see good performance at $M/N = 1/8$ with acceptance at all sparsity levels. With $M/N = 1/4$, we see a perfect fit at $s = 1/8M$, with a few rejections at $s = 1/4M$ and $1/2M$. At $M/N = 1/2$, full acceptance is seen at $s = 1/8M$ and $1/4M$ while there are some accepts at $\alpha = 0.05$ but almost complete failure at $\alpha = 0.01$ when $s = 1/2M$.
CoSaMP has vastly superior performance over OMP and BP in this test, with the only rejections occurring at $M/N = 1/2$ with $s = 1/2M$.
StOMP, while it may appear to perform decently at $M/N = 1/8$ and $M/N = 1/4$, suffers immensely at $M/N = 1/2$ with numerous rejections at all sparsity. Again, the thresholding value must play a large role here. No doubt tuning this parameter would lead to a better goodness-of-fit.
By far, the two superior algorithms with regards to goodness-of-fit are IHT and SP. IHT appears to have the slight edge at $M/N = 1/2$, however both perform exceptionally well.

Comparing these results with those of the reconstruction, we note the similar tread with BP and StOMP seemingly the worst performers. Interestingly, IHT shows the most improvement in this test, possibly due to its thresholding of the reconstructed signal $\hat{x}$. OMP and CoSaMP while similar in performance with regards to the reconstruction results, differ with goodness-of-fit. Here CoSaMP clearly has the edge. SP performs the best overall but as previously mentioned, many of the other algorithms under investigation can be tuned and improved under certain conditions and parameters.

3.3 GPU Results

In this section we consider the results obtained through the use of the GPU. The reconstruction efficiency would be expected to be the same, regardless whether this is computed on the CPU or GPU and as such is not presented again in this chapter. The computational times however, should be expected to differ tremendously. The GPU involved in the computational runs was a Nvidia GTX 680 with 1536 shader cores, thus large speed-ups should be expected.

Furthermore, not all algorithms were implemented on the GPU. Rather the best performers were selected. IHT, CoSaMP and SP are investigated. Blanchard and Tanner have written a software package and MATLAB toolbox especially for the GPU implementation of these algorithms with [91–93]. **GAGA** or in full, GPU Accelerated Greedy Algorithms was developed for Compressed Sensing problems involving millions of unknowns, most likely incapable or too slow to be computed on conventional CPU hardware. The package itself does not implement CoSaMP or SP directly, but rather a two stage projection that is essentially equivalent to CoSaMP and SP, aptly named CSMPSP [92].

3.3.1 GPU Parallelisation and Implementation

When considering any iteration of any of the algorithms under investigation, at least one support set identification and one matrix-vector multiplication is computed. As a result, all the major algorithms discussed and by extension the algorithms found in the GAGA toolbox follow a similar structure:

- **The initialisation:** A routine for the generation of the measurement matrix Φ, and the original signal x.
- **Support Set Detection and Thresholding:**
    1. **Support Detection:** A routine returning the index set I of s largest (in magnitude) elements found in the signal x.
    2. **Thresholding:** A routine where all elements not indexed by I are set to zero.
- **Iterative Procedure:** An iterative procedure of ever increasing approximations until a designated stopping criteria is met. Each algorithm differs in the manner in which they update the approximation.

3.3.1.1 Initialisation

GAGA generates measurement matrices in three different ways. Firstly, it can generate a DCT (discrete cosine transform) matrix given inputs M and N. Secondly, sparse matrices can be randomly generated with M and N. These matrices have p nonzero elements per column and with each nozero element ± 1 in value. The columns are then normalised by dividing by $\sqrt{p}$. This option could offer significant speedups for algorithms which make use of sparse sensing matrices such as the combinatorial algorithms discussed in Section 2.3.1. The third option is the generation of generic Gaussian matrices given inputs M and N. These generic matrices are used in the GPU implementation to provide consistency with the CPU results.

For comparison with CPU computation times. The same configurations are tested. Starting with $N = 2^8$ to N^{15}. The measurements are set to the ratios $M/N = 1/8, 1/4$ and $1/2$. The sparsity of the original signal x is fixed at $s = 0.125M$, with these nonzero elements sampled from the standard normal distribution. The initial measurement vector, y is generated with the matrix-vector multiplication $y = \Phi x$. Tables 3.4, 3.5, 3.6 list the recorded results of each algorithm on the same configurations as the CPU averaged over 200 trials.

3.3.1.2 Support Detection and Thresholding

The general approach to support detection involves a sorting routine to obtain the sth largest magnitude of the signal x and construct its corresponding support set. Previously, finding the support set was not a computational concern as the vector-matrix operations were far more taxing. With GAGA however, the computational burden of vector-matrix operations is reduced to that of the support set identification. Importantly, for larger speedups, GAGA has reduced the cost of the support detection with their GPU implementation. They initially did this using the highly efficient radix sort routine by Merrill and Grimshaw [94]. The radix sort is available with the Thrust library and is included in CUDA 4.0 and all subsequent releases[3] [95, 96]. With this, the act of thresholding and identifying the support set is done simultaneously [91].

Blanchard and Tanner improved on the performance of the support set identification by making use of a novel method. Similar to distributive partitioning [97], a sorting algorithm which makes use of n buckets of equal length to partition n data. This new approach identifies the support set using the approximate s-selection technique found in [92]. This is done by taking a linear projection of elements in the vector x into linearly spaced bins,

[3] The latest version being CUDA 6.0 at the time of writing.

where the number of bins is the maximum value of either 1000 or $N/2$. The partitioning of the bins itself is done on the interval of 0 to the maximum absolute value of x (i.e. $[0, max(abs(x))]$).

With regards to the iterative procedure of each algorithm, the CUDA based implementations depend upon GPU-accelerated libraries. Specifically, GAGA makes heavy use of the CUBLAS library (the GPU library within the Basic Linear Algebra Subroutines (BLAS) library). The BLAS libray consists of a set of subroutines which perform common linear algebra operations such as vector dot product, matrix multiplication, vector scaling and linear combinations. The CUBLAS variant is an optimised version of BLAS for NVIDIA based GPU cards.

The GPU implementation itself manifests in the form of kernels. These kernels assign work to the respective GPU cores. The work to be done is divided into blocks in which each block is given an equal number of threads. CUDA has built in functions to povide the user with the maximum number of threads available, dependent to the user's hardware.

3.3.2 GPU Computational Times With $\frac{M}{N} = \frac{1}{8}$

We begin again with the first configuration of $M/N = 1/8$, with the results tabulated in 3.6 and plotted in Figure 3.10.

M/N(1/8)								
Signal Length (N)	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}
IHT	0.0036	0.0076	0.0133	0.0282	0.1171	0.4258	1.6181	6.0168
IHT-GPU	0.0058	0.0103	0.0114	0.013937	0.028851	0.073104	0.232367	0.937411
CoSaMP	0.0009	0.0013	0.0027	0.0059	0.0220	0.0858	0.3697	3.7458
SP	0.0029	0.0032	0.0052	0.0168	0.0638	0.2927	1.4904	9.210
CSMPSP-GPU	0.0074	0.00827	0.016001	0.01732	0.041457	0.110370	0.431023	1.697272

TABLE 3.4: **Computational Times in Seconds (s)**

Looking at the times and specifically the above plot, we see that the GPU is actually outperformed by the CPU for smaller signal lengths. Although this is interesting, it perhaps is not surprising. The hexacore i7 used is an extremely powerful 12 core processor and it is not uncommon for powerful CPU's to beat GPU counterparts on smaller computational problems. However, once the signal length begins to increase, i.e. $N = 2^{13}$ the GPU begins to overtake the CPU in speed. This trend will no doubt continue on larger signal sizes, with the GPU brining larger and larger speed-ups the bigger the signal gets.

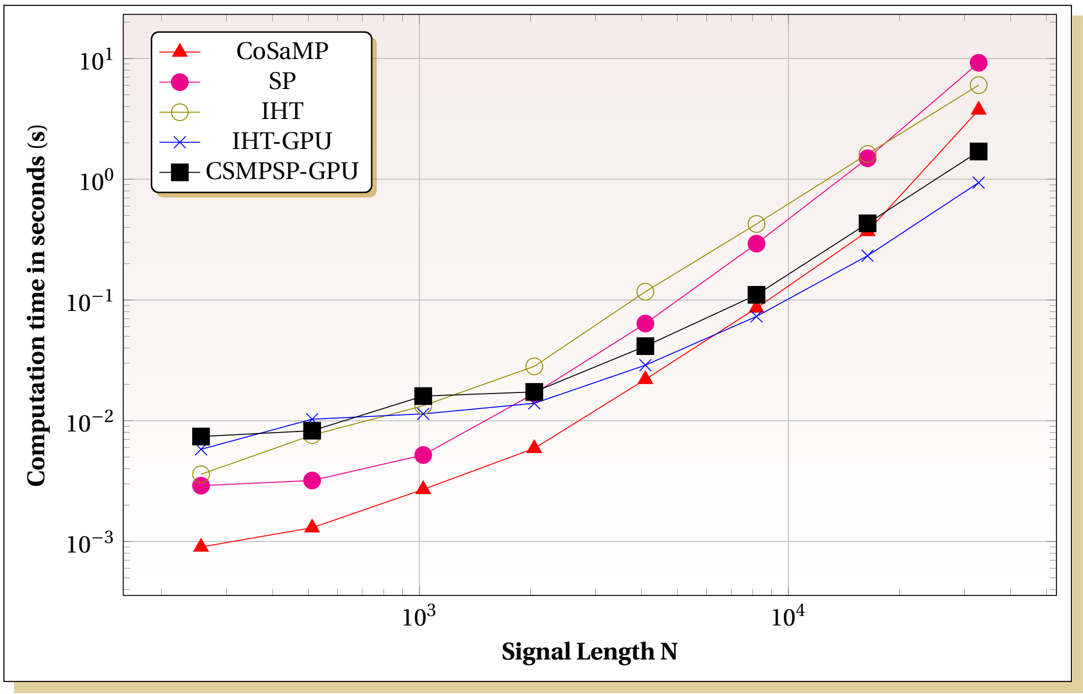


FIGURE 3.10: **Log Scale of Computation Time for M/N = 1/8**

3.3.3 GPU Computational Times With $\dfrac{M}{N} = \dfrac{1}{4}$

Moving onto the second configuration we find the tabulated results in Table 3.5 and plotted in Figure 3.11.

M/N(1/4)								
Signal Length (N)	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}
IHT	0.0061	0.0081	0.0134	0.0345	0.1876	0.7030	2.6377	10.1864
IHT-GPU	0.00878	0.01241	0.01549	0.015997	0.037781	0.12052	0.456144	1.37641
CoSaMP	0.0013	0.0019	0.0045	0.0175	0.0701	0.3139	3.1661	26.8067
SP	0.0032	0.0039	0.0133	0.0507	0.2437	1.3390	7.3405	44.3120
CSMPSP	0.008068	0.008456	0.01614	0.021477	0.056959	0.197231	0.78846	2.8145

TABLE 3.5: **Computational Times in Seconds (s)**

We can now clearly see the performance of the GPU coming through. From signal lengths $N = 2^{11}$ onwards the GPU times appear to be substantially lower than to that of its serial CPU counterpart.

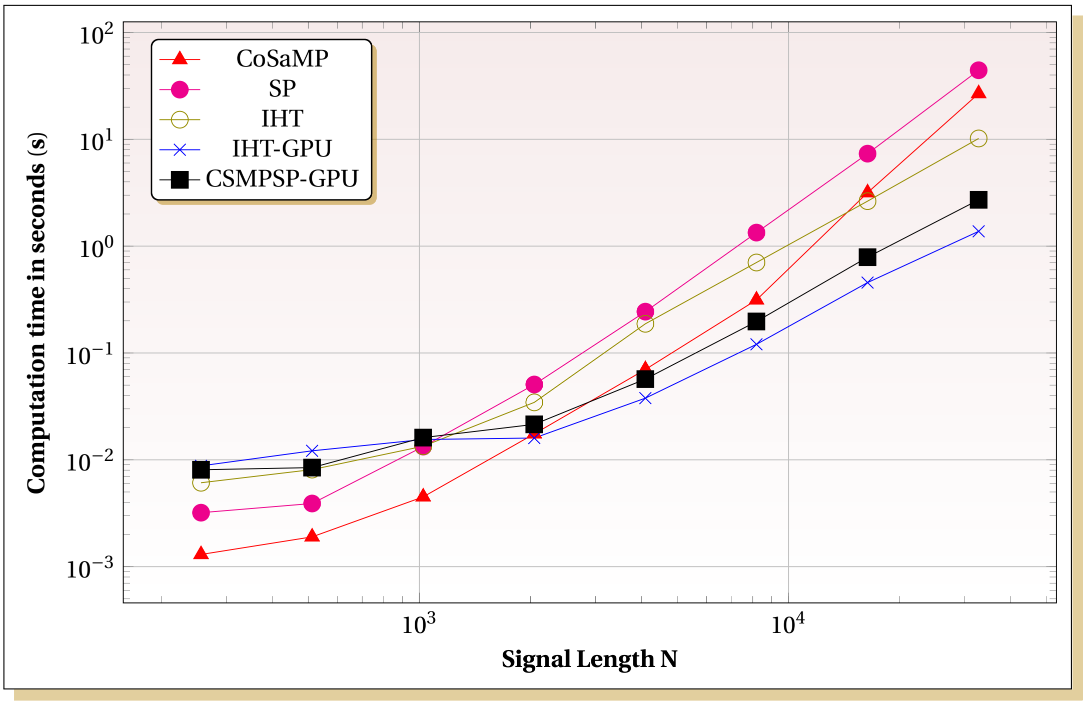


FIGURE 3.11: **Log Scale of Computation Time for M/N = 1/4**

3.3.4 GPU Computational Times With $\frac{M}{N} = \frac{1}{2}$

We now consider the final configuration. It is here where we should expect to see the greatest performance of the GPU over the CPU as the problem size begins to become too large for the CPU to handle in manageable time. The results are tabulated in Table 3.6 and are plotted in Figure 3.12.

M/N(1/2)								
Signal Length (N)	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}
IHT	0.0065	0.0091	0.0167	0.0792	0.3185	1.2249	4.6273	17.6595
IHT-GPU	0.008982	0.012479	0.01579	0.02018	0.075414	0.2369	0.9248	2.7926
CoSaMP	0.0017	0.004	0.01045	0.0620	0.2826	2.9926	22.5482	174.5490
SP	0.0065	0.0106	0.0407	0.2078	1.0572	6.4544	39.6360	261.1395
CSMPSP	0.009229	0.009556	0.017051	0.030581	0.091979	0.322689	1.516474	4.13562

TABLE 3.6: **Computational Times in Seconds (s)**

As expected the GPU results have now begun to differ by orders of magnitude on larger signals as we begin to see the large speed-ups capable on the GPU. Again this speed-up ratio will only continue to grow as the signal lengths increase.

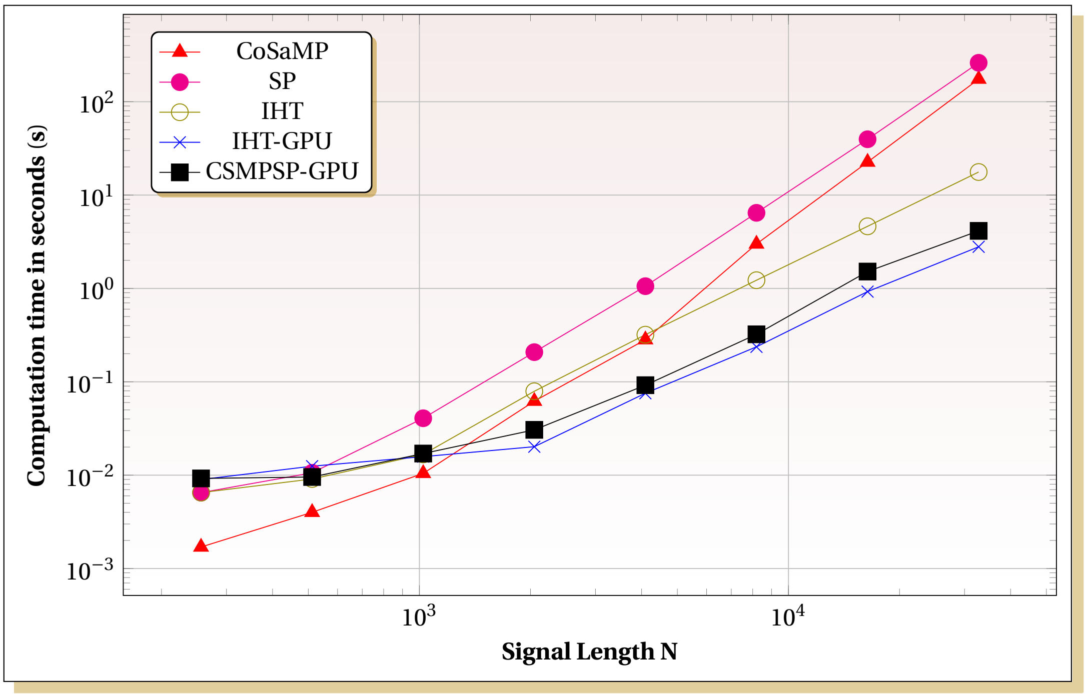


FIGURE 3.12: **Log Scale of Computation Time for M/N = 1/2**

3.3.5 Average Ratio Speed-Ups

To highlight how much of a speed-up the GPU provides, we list the Tables 3.7, 3.8 and 3.9. At the largest tested signal length of 2^{15} we observe speed-ups of 6 to 63 times faster than the serial implementation. These ratios also seem to lay in agreement with Blanchard and Tanner's results in [92], in which larger signal lengths ($N = 2^{20}$) are reported to be over 1000 times faster.

Speed-Up For IHT			
Signal Length (N)	IHT (s)	IHT-GPU (s)	Speed-Up Ratio
2^{10}	0.0167	0.01579	1.050
2^{11}	0.0792	0.02018	3.925
2^{12}	0.3185	0.075414	4.223
2^{13}	1.2249	0.2369	5.171
2^{14}	4.6273	0.9247	5.004
2^{15}	17.6595	2.792594	6.324

TABLE 3.7: **GPU Speed-Up Ratios for IHT when $\frac{N}{M} = \frac{1}{2}$**

Speed-Up For CoSaMP			
Signal Length (N)	CoSaMP (s)	CSMPSP (s)	Speed-Up Ratio
2^{10}	0.01045	0.017051	0.613
2^{11}	0.0620	0.030581	2.027
2^{12}	0.2826	0.091979	3.072
2^{13}	2.9926	0.322689	9.274
2^{14}	22.5482	1.516474	14.869
2^{15}	174.549	4.13562	42.206

TABLE 3.8: **GPU Speed-Up Ratios for CoSaMP when $\frac{N}{M} = \frac{1}{2}$**

Speed-Up For SP			
Signal Length (N)	SP (s)	CSMPSP (s)	Speed-Up Ratio
2^{10}	0.0407	0.017051	2.387
2^{11}	0.2078	0.030581	6.795
2^{12}	1.0572	0.091979	11.494
2^{13}	6.4544	0.322689	20.002
2^{14}	39.6360	1.516474	26.137
2^{15}	261.1395	4.13562	63.144

TABLE 3.9: **GPU Speed-Up Ratios for SP when $\frac{N}{M} = \frac{1}{2}$**

Chapter 4

Discussion of Results

All the results listed in Chapter 3 for the most part provide expected results. We saw that the greedy algorithms were much faster than traditional Basis Pursuit albeit if some were slightly less efficient by means of recovery. StOMP underperformed throughout all configurations. This may perhaps be improved through use of a different threshold parameter. Alternatively, this poor performance is possibly linked to the idea of recovery itself. The assumption that recovery is consider complete when the relative error is less than 10^{-5} is problem specific. In the computational runs conducted here, this was chosen for two reasons. Firstly, this value is commonly used and accepted as a stopping criterion and secondly, the influence it had on BP. Interestingly, BP could not manage any form of success rate below 10^{-8}, StOMP as well suffered with regard to this. Algorithms such as OMP, CoSaMP, SP and IHT were able to obtain similar reconstruction behaviour as listed in Chapter 3 even when the recovery criterion was set to 10^{-15}. It seems fair to conclude that the success rate of the algorithms in question is then a matter dependent upon each respective problem. If a problem allows for a looser bound on the relative error then this will be relevant in which algorithm to make use of.

With regards to the recovery results obtained under the above conditions, it is noted that SP performs the best in each configuration, maintaining dominance over the other algorithms regardless of signal size and measurements taken as is seen with Table 4.1. OMP performs well with low measurement and sparsity but struggles as the measurement matrix increases in size. CoSaMP and BP appear to be highly competitive with each other in all three configurations. IHT does not perform as well as expected but it seems highly likely that numerical investigation into the correct parameters for this specific problem set could see large improvement.

Reconstruction Performance			
Sparsity Range (s)	**M/N = 1/8**	**M/N = 1/4**	**M/N = 1/2**
$[1-20]$	SP	SP	SP
$[1-40]$	-	SP	SP
$[1-80]$	-	-	SP

TABLE 4.1: **Best Performing Algorithm over Sparsity Ranges Considered**

The use of the Kolmogorov-Smirnov (KS) test provided some interesting results. For the most part, the KS plots (as seen in Figures A.1 through A.18) echoed the performance we saw with reconstruction. However, there were some differences. IHT for example, experienced a massive improvement in this test, outperforming all other algorithms. SP continued to impress and certainly established itself as one of the most successful algorithms by this measure. Another interesting result, was the difference between OMP and CoSaMP. They performed quite closely when considering the reconstruction trials, but the KS test shows CoSaMP providing much better goodness-of-fit. StOMP and BP continued to under perform with BP showing more promise than StOMP.

Considering tests which measure the reconstruction as a distance metric such as the relative error used here. It seems recoveries which have small values oscillating at indices of zero elements with respect to the original signal, that performance isn't impacted too severely. However, the use of the KS test of similar recoveries doesn't appear to agree with this, as these small oscillations seem to have a large affect on the difference between the two empirical distribution functions.

Computational Complexity	
Algorithm	Complexity
BP	$\mathcal{O}(N^3)$
OMP	$\mathcal{O}(sMN)$
StOMP	$\mathcal{O}(N\text{log}N)$
CoSaMP	$\mathcal{O}(sMN)$
SP	$\mathcal{O}(sMN)$
IHT	$\mathcal{O}(MN)$

TABLE 4.2: **Typical Computational Complexities For Implemented Algorithms**

Table 4.2 lists the expected computational complexity of each algorithm investigated. Considering this as well as the computational runtimes obtained we can see for the most part that the results are consistent. The complexity of BP being $\mathcal{O}(N^3)$ is definitely noted as the worst performing algorithm when considering time per run both theoretically and with the empirical results above. StOMP theoretically should be the second worst in regards to time taken which for the most part is true from the results. SP does appear to require more time when $M/N = 1/4$ and $M/N = 1/2$ but this was influenced heavily by MATLAB'S struggle to calculate inverses. The other greedy methods of order $\mathcal{O}(sMN)$ appeared to behave accordingly except for OMP which did not scale duly with an increase in signal length. Finally, IHT of order $\mathcal{O}(MN)$, definitely satisfied its theoretical expectation by running substantially faster than all the other algorithms.

Importantly, it is worth noting that the algorithms making use of matrix-vector products Φx and $\Phi^T x$, will have had this dominate the computational cost regardless of fast transforms used. It is therefore paramount to keep the number of these operations to a minimum. Another concern when making use of these algorithms is the storage cost involved. Table 4.3 lists the typical storage costs of the algorithms involved. Again IHT shows its worth by having the lowest storage requirement. The signal x excluded, IHT need only store 2s non zero elements.

Storage Cost	
Algorithm	**Storage**
OMP	$2(M+1)s+0.5s(s+1)+\Phi+N$
StOMP	$2M+\Phi+2s+N$
CoSaMP	$2M+\Phi+2s+N$
SP	$2M+\Phi+2s+N$
IHT	$2s$

TABLE 4.3: **Typical Storage Cost For Implemented Algorithms**

Finally, the implementation of the GPU in Chapter 3 showed the potential of massive speed-ups when considering large signal sizes. Blanchard and Tanner's package provides the potential to solve large Compressed Sensing problems in reasonable amounts of time. This package in tandem with large GPU arrays or SLI/CrossFireX systems could be used to obtain solutions to massive values of N. The speed-ups seen in [91, 92], show massive acceleration. The results themselves were obtained on a Nvidia Telsa GPU. Nvidia's new Kepler and Maxwell architecture found on their flagship cards could see these results improved dramatically.

Chapter 5

Conclusion

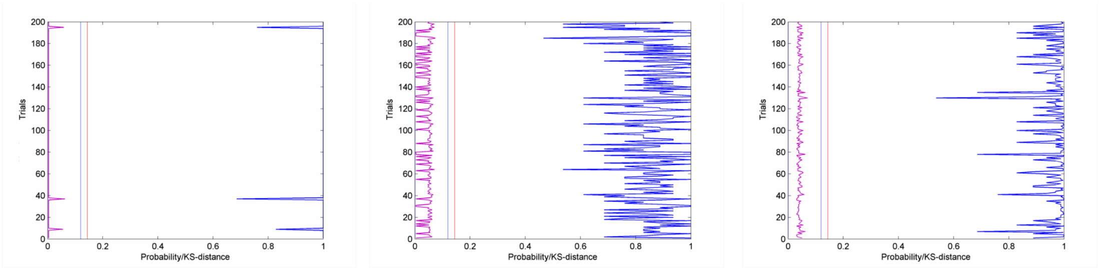
In conclusion, Compressed Sensing appears to be a modern and sensible approach to combat the ever present data deluge. A multitude of algorithms have already been developed as the field is investigated by numerous different disciplines, that in itself must highlight its importance.

This thesis took interest into the computation of CS algorithms specifically and attempted to provide a broad spectrum study. The discussion and results were not meant to be major investigations into the depths of each method, but rather provide an overview of expected and empirical results. Although a few perturbations were observed, the results themselves satisfied numerical expectancy. We saw that greedy algorithms, as expected achieved much more manageable computation times as well as strong recovery performance opposed to that of the convex optimisation found in BP. It was also noted how well IHT algorithm performed in computation time but surprisingly did not do as well in terms of reconstruction recovery. However, it saw vast improvement with the KS test implying strong goodness-of-fit. Looking at all the results across the board, SP is the best overall performer especially at reconstruction as was concluded with Table 4.1, while it also maintained decent computation times. Importantly, as mentioned numerously already, these tests are extremely sensitive input configurations. I hope the interested reader seeking more detail on any specific algorithm has enough references provided so that they may continue their own investigations accordingly. In addition I hope that the importance of GPU technology is emphasized as a modern shift in computation begins. The computational speed ups illustrate this with orders of magnitude increases in computational time. The greedy algorithms in particular were improved immeasurably with up 63 times speed ups.

Furthermore, I hope that this thesis provides a solid overview of the computational concerns involved with Compressed Sensing, while also providing a modicum of theoretical knowledge. As big data continues to grow and society continually demands more, the use of GPUs will become paramount to the success of meeting these modern requirements. The improvement of hardware cannot be expected to satisfy this alone. However, the collusion of new, exciting and clever mathematics with the exponential surge in modern architecture posses the potential to reach these objectives.

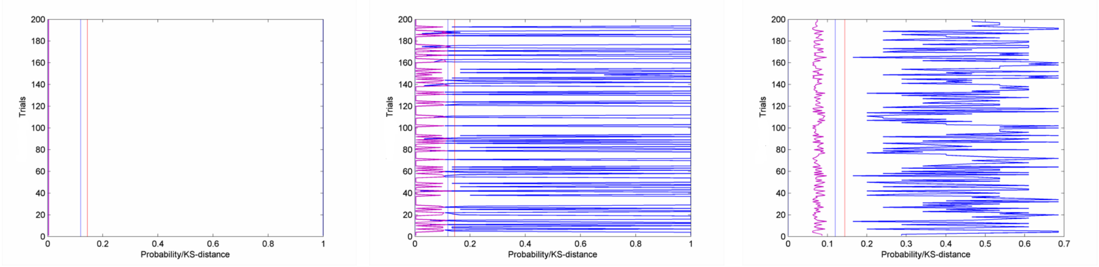
Appendix A

Kolmogorov-Smirnov Tests



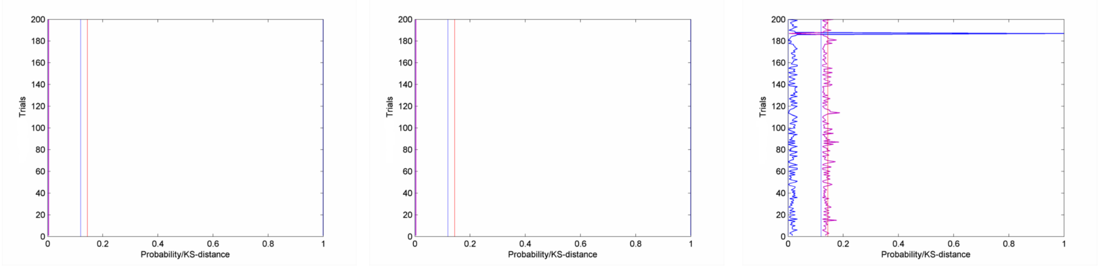
(A) $s = 1/8$ (B) $s = 1/4$ (C) $s = 1/2$

FIGURE A.1: **BP KS Test with M/N = 1/8 and varying sparsity.**



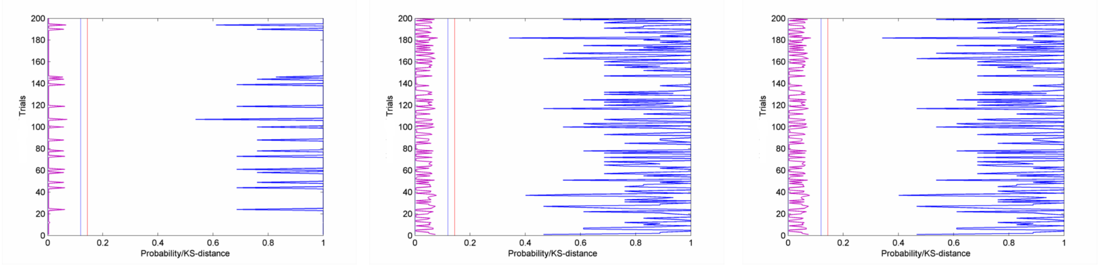
(A) $s = 1/8$ (B) $s = 1/4$ (C) $s = 1/2$

FIGURE A.2: **BP KS Test with M/N = 1/4 and varying sparsity.**



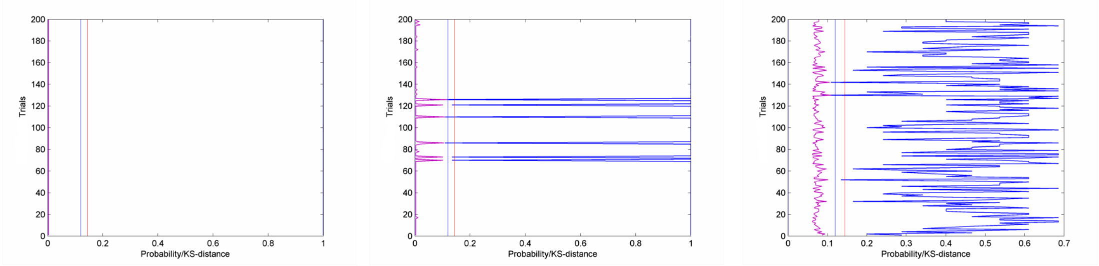
(A) $s = 1/8$ (B) $s = 1/4$ (C) $s = 1/2$

FIGURE A.3: **BP KS Test with M/N = 1/2 and varying sparsity.**



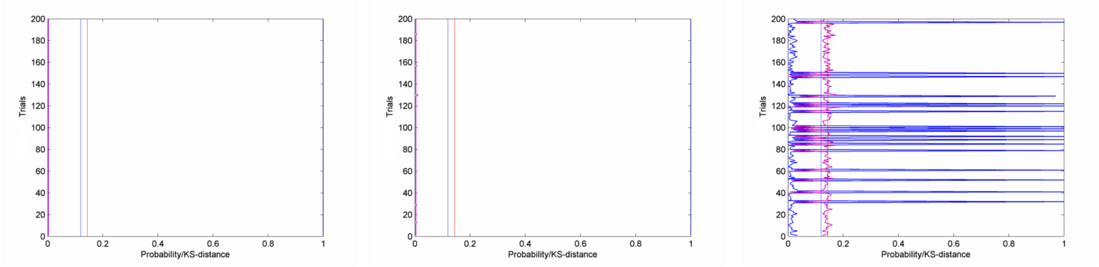
(A) $s = 1/8$ (B) $s = 1/4$ (C) $s = 1/2$

FIGURE A.4: **OMP KS Test with M/N = 1/8 and varying sparsity.**



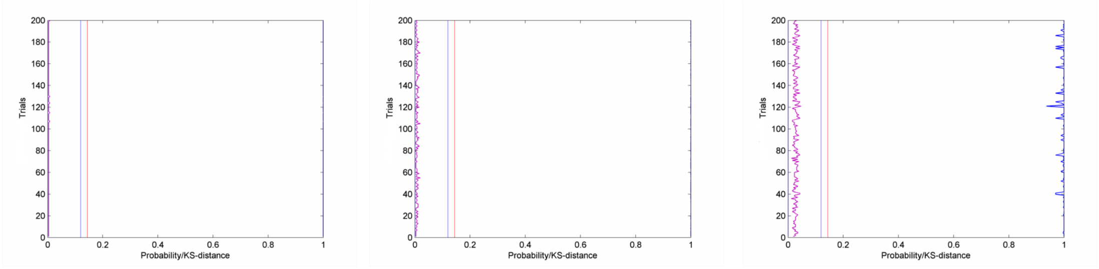
(A) $s = 1/8$ (B) $s = 1/4$ (C) $s = 1/2$

FIGURE A.5: **OMP KS Test with M/N = 1/4 and varying sparsity.**



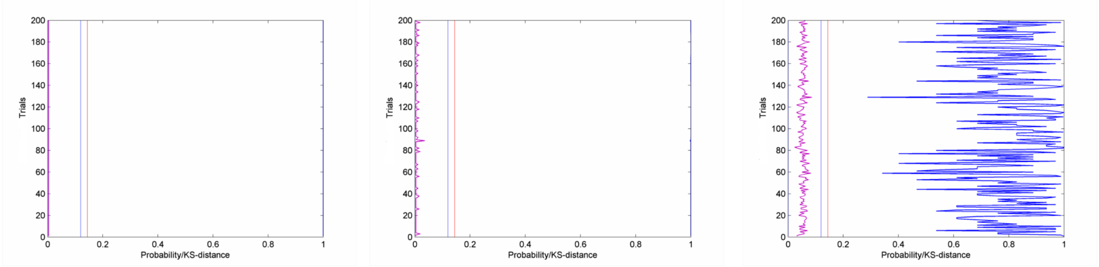
(A) $s = 1/8$ (B) $s = 1/4$ (C) $s = 1/2$

FIGURE A.6: **OMP KS Test with M/N = 1/2 and varying sparsity.**



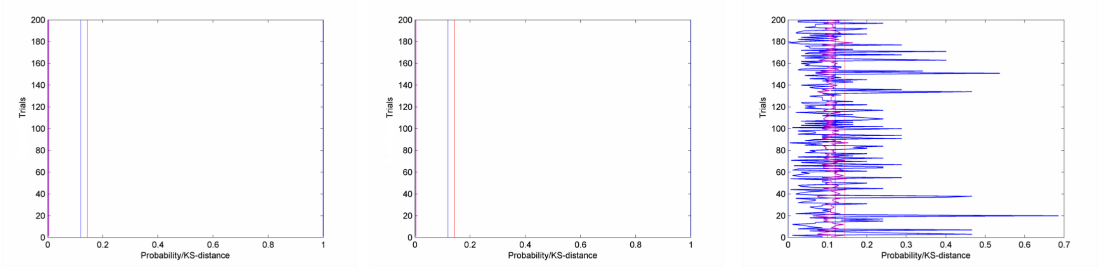
(A) $s = 1/8$ (B) $s = 1/4$ (C) $s = 1/2$

FIGURE A.7: **CoSaMP KS Test with M/N = 1/8 and varying sparsity.**



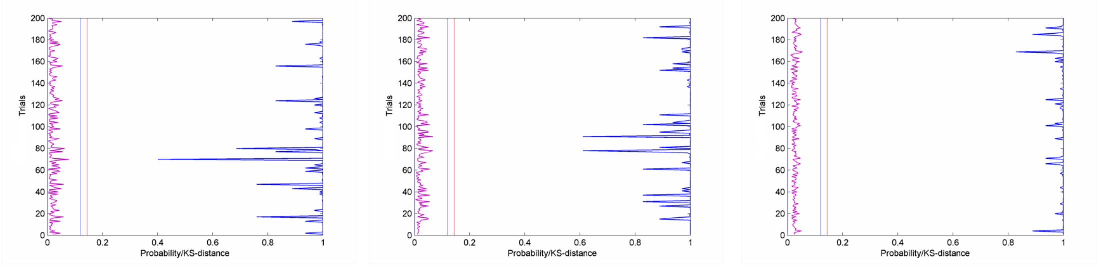
(A) $s = 1/8$ (B) $s = 1/4$ (C) $s = 1/2$

FIGURE A.8: **CoSaMP KS Test with M/N = 1/4 and varying sparsity.**



(A) $s = 1/8$ (B) $s = 1/4$ (C) $s = 1/2$

FIGURE A.9: **CoSaMP KS Test with M/N = 1/2 and varying sparsity.**

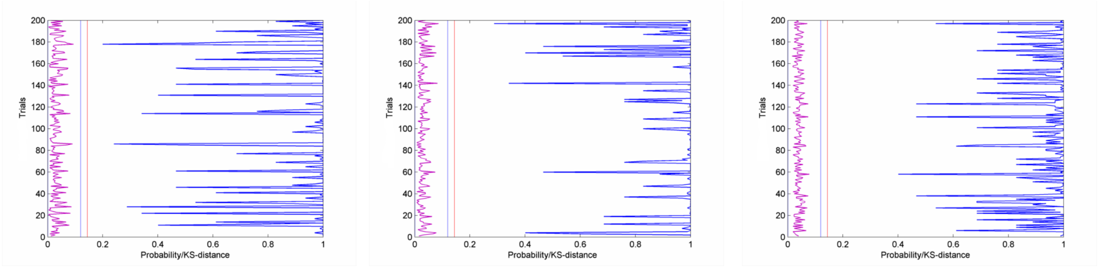


(A) $s = 1/8$

(B) $s = 1/4$

(C) $s = 1/2$

FIGURE A.10: **StOMP KS Test with M/N = 1/8 and varying sparsity.**

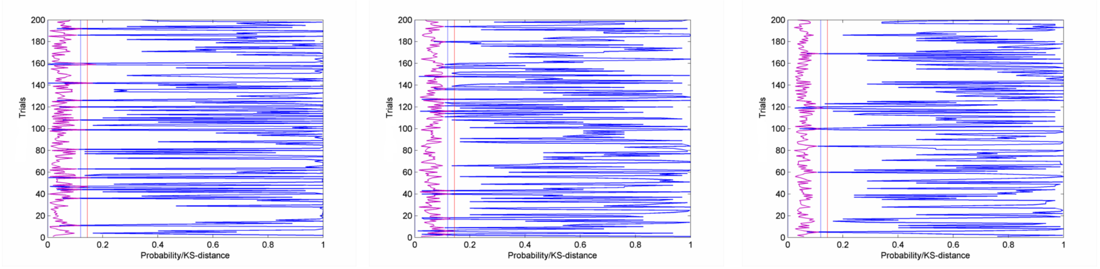


(A) $s = 1/8$

(B) $s = 1/4$

(C) $s = 1/2$

FIGURE A.11: **StOMP KS Test with M/N = 1/4 and varying sparsity.**

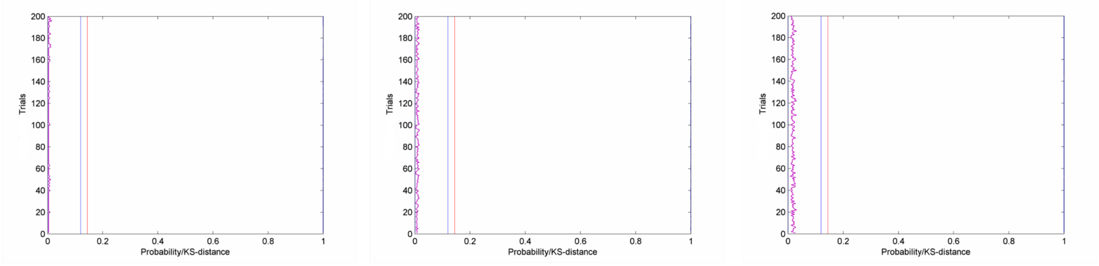


(A) $s = 1/8$

(B) $s = 1/4$

(C) $s = 1/2$

FIGURE A.12: **StOMP KS Test with M/N = 1/2 and varying sparsity.**

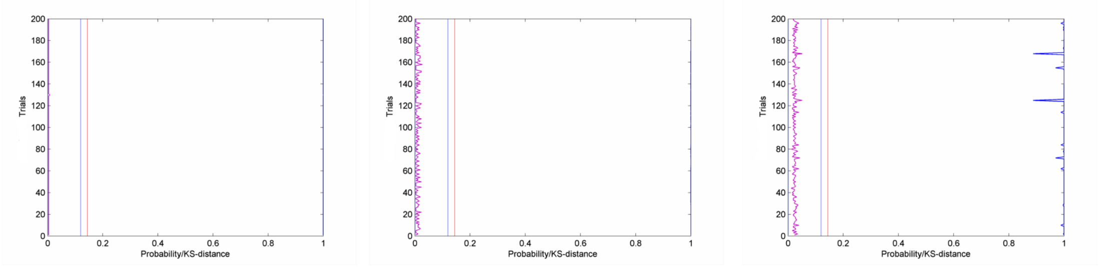


(A) $s = 1/8$

(B) $s = 1/4$

(C) $s = 1/2$

FIGURE A.13: **IHT KS Test with M/N = 1/8 and varying sparsity.**

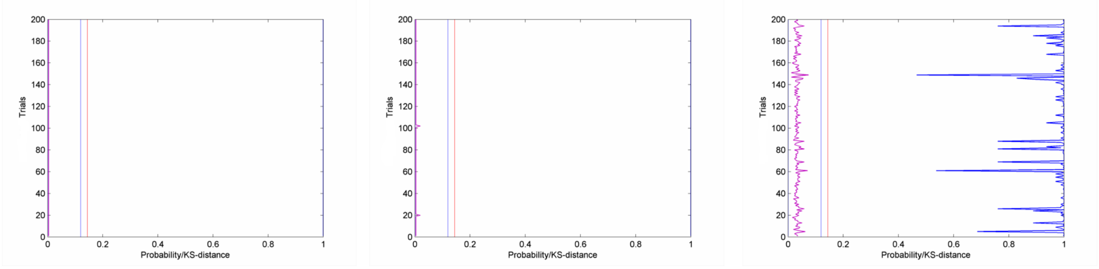


(A) $s = 1/8$

(B) $s = 1/4$

(C) $s = 1/2$

FIGURE A.14: **IHT KS Test with M/N = 1/4 and varying sparsity.**

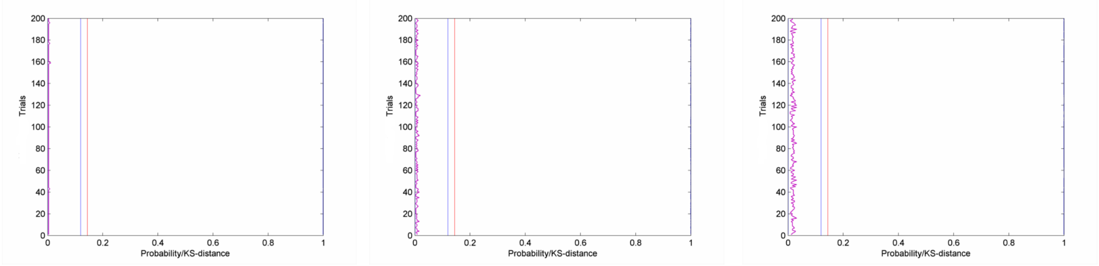


(A) $s = 1/8$

(B) $s = 1/4$

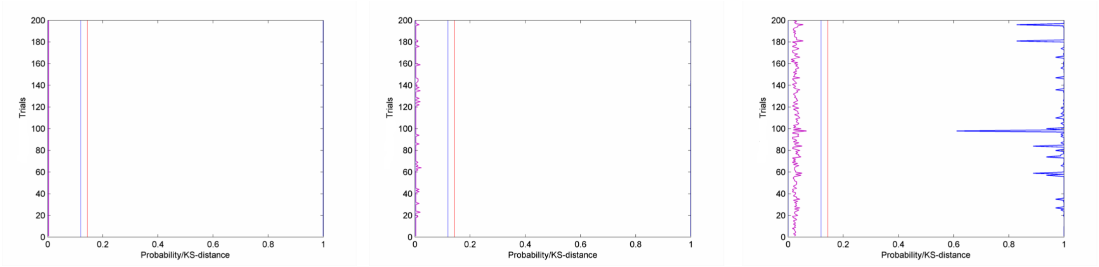
(C) $s = 1/2$

FIGURE A.15: **IHT KS Test with M/N = 1/2 and varying sparsity.**



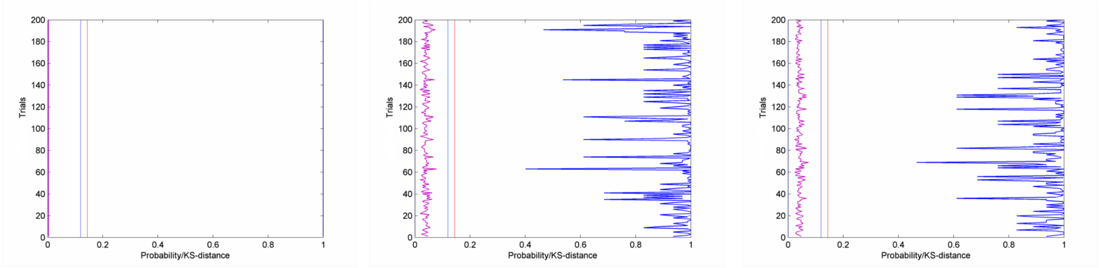
(A) $s = 1/8$ (B) $s = 1/4$ (C) $s = 1/2$

FIGURE A.16: **SP KS Test with M/N = 1/8 and varying sparsity.**



(A) $s = 1/8$ (B) $s = 1/4$ (C) $s = 1/2$

FIGURE A.17: **SP KS Test with M/N = 1/4 and varying sparsity.**



(A) $s = 1/8$ (B) $s = 1/4$ (C) $s = 1/2$

FIGURE A.18: **SP KS Test with M/N = 1/2 and varying sparsity.**

Appendix B

MATLAB Code

The following appendix lists the code used along with any external code referenced in the Chapters above.

B.1 Basis Pursuit

B.1.1 Recovery

```
% The following script tests the computation time involved with BP.
% Makes use of the outside function SolveBP found in SparseLab
% written by Donogo et al.
N = 256;
M = 64;%32,128
% number of spikes in the signal
T = 1:1:20;%1:1:40 1:1:80
% number of observations to make
maxIters=300;
lambda=0;% Default value
OptTol=1e-3;
trials = 200;
disp('Creating measurment matrix...');
A = (1/sqrt(M))*randn(M,N);
disp('Done.');
successrate=zeros(length(T),1);
tic
for i = 1:length(T)
normval = zeros(trials,1);
count = 0;
```

```
for j = 1:trials
x = zeros(N,1);
q = randperm(N);
x(q(1:T(i))) = randn(T(i),1);
% observations
y = A*x;
% solve the LP
xp = SolveBP(A, y, N, maxIters, lambda, OptTol)
normval(j,1) = norm(xp-x)/norm(x);
if normval(j,1) <= 10^-5
    count = count +1;
end
end
successrate(i,1) = 100*(count/trials);
end
results = [successrate,T'];
toc
```

B.1.2 Computational Time

```
% The following script tests the computation time involved with BP.
% Makes use of the outside function SolveBP found in SparseLab
% written by Donogo et al.
N = 256;% 512 1024 2048 4096 8192 16384 32768
M = 0.125*N;% 0.25*N 0.5*N
% number of spikes in the signal
T = 0.125*M
maxIters=300;
lambda=0;% Default value
OptTol=1e-3;
trials = 200;
disp('Creating measurment matrix...');
A = (1/sqrt(M))*randn(M,N);
disp('Done.');
tic
for i = 1:length(T)
normval = zeros(trials,1);
count = 0;
for j = 1:trials
x = zeros(N,1);
q = randperm(N);
x(q(1:T(i))) = randn(T(i),1);
% observations
```

```
y = A*x;
% solve the LP
xp = SolveBP(A, y, N, maxIters, lambda, OptTol)
normval(j,1) = norm(xp-x)/norm(x);
if normval(j,1) <= 10^-5
    count = count +1;
end
end
end
time = toc/trials
```

B.2 Orthogonal Matching Pursuit OMP

B.2.1 Recovery

```
% This scrip tests the recovery performance of CoSaMP.
% It makes use of an outside function written by
% Micheal McCoy.
N = 256;
ntest = 200;
sVals = 1:1:20;%1:1:40 1:1:80
mVals = 32;%64 128;


[success,results]=...
    run_test('cosamp_vs_omp',{N,ntest,mVals,sVals},global_debug_opts{:});

% Required Outputs


FinalOMP_N256M32SR = results.test_output.psOMP
%FinalOMP_N256M64SR = results.test_output.psOMP
%FinalOMP_N256M128SR = test_output.psOMP
```

B.2.2 Computational Times

```
% This scrip tests the recovery performance of CoSaMP.
% It makes use of an outside function written by
% Micheal McCoy.
N = 256;
ntest = 200;
sVals = 1:1:20;%1:1:40 1:1:80
mVals = 32;%64 128;

tic
[success,results]=...
    run_test('cosamp_vs_omp',{N,ntest,mVals,sVals},global_debug_opts
    {:});
time=ntest/trials
% Required Outputs


FinalOMP_N256M32SR = results.test_output.psOMP
%FinalOMP_N256M64SR = results.test_output.psOMP
%FinalOMP_N256M128SR = test_output.psOMP
```

B.3 Compressive Sampling Matching Pursuit CoSaMP

B.3.1 Recovery

```
% This scrip tests the recovery performance of CoSaMP.
% It makes use of an outside function written by
% Micheal McCoy.
N = 256;
ntest = 200;
sVals = 1:1:20;%1:1:40 1:1:80
mVals = 32;%64 128;


[success,results]=...
    run_test('cosamp_vs_omp',{N,ntest,mVals,sVals},global_debug_opts{:});

% Required Outputs


FinalCosamp_N256M32SR = results.test_output.psCosamp
%FinalCosamp_N256M64SR = results.test_output.psCosamp
%FinalCosamp_N256M128SR = results.test_output.psCosamp
```

B.3.2 Computational Times

```
% This scrip tests the recovery performance of CoSaMP.
% It makes use of an outside function written by
% Micheal McCoy.
N = 256;% 512 1024 2048 4096 8192 16384 32768
ntest = 200;

mVals = 0.125*N;
sVals = 0.125*M;
tic
[success,results]=...
    run_test('cosamp_vs_omp',{N,ntest,mVals,sVals},global_debug_opts{:});
time = toc/ntest
% Required Outputs


FinalCosamp_N256M32SR = results.test_output.psCosamp
%FinalCosamp_N256M64SR = results.test_output.psCosamp
%FinalCosamp_N256M128SR = results.test_output.psCosamp
```

B.4 Stagewise Orthogonal Matching Pursuit StOMP

B.4.1 Recovery Performance

```
% This script evaluates the recovery of StOMP
% It calls the outside function SolveStOMP found in SparseLab
% written by Donoho et al.
N = 256;
M = 32;%64 128
Phi = (1/sqrt(M))*(randn(M,N));
K = 1:1:20;%1:1:40,1:1:80
trials = 200;
successrate = zeros(length(K),1);

for j =1:length(K)
count = 0;
for i =1:trials
S = K(j);
T    = randperm(N);
T    = T(1:S);
xsig    = zeros(N,1);
xsig(T)= randn(S,1);
y = Phi*xsig;
thresh = 'FDR';
param = 0.5;
maxiter = 10;
optTol = 10^-16;
verbose = 1;
[sol, numIters] = SolveStOMP(Phi, y, N, thresh, param, maxIters,
    verbose, OptTol);

if norm(sol-xsig)/norm(xsig) <= 10^-5
    count = count +1;
end

end
successrate(j,1) = 100*(count/trials);
end
plot(K,successrate,'--ro')
```

B.4.2 Computational Time

```
% This script evaluates the recovery of StOMP
% It calls the outside function SolveStOMP found in SparseLab
% written by Donoho et al.
N = 256;% 512 1024 2048 4196 8192 16384 32768
M = 0.125*N;% 0.25*N 0.5*N
Phi = (1/sqrt(M))*(randn(M,N));
K=0.125*M
trials = 200;

tic
for j =1:length(K)
count = 0;
for i =1:trials
S = K(j);
T    = randperm(N);
T    = T(1:S);
xsig    = zeros(N,1);
xsig(T)= randn(S,1);
y = Phi*xsig;
thresh = 'FDR';
param = 0.5;
maxiter = 10;
optTol = 10^-16;
verbose = 1;
[sol, numIters] = SolveStOMP(Phi, y, N, thresh, param, maxIters,
    verbose, OptTol);

if norm(sol-xsig)/norm(xsig) <= 10^-5
    count = count +1;
end
end
end
time = toc/trials
plot(K,successrate,'--ro')
```

B.5 Subspace Pursuit

B.5.1 Recovery Performance

```matlab
% This script evaluates the recovery of SP.
% It calls the function SP(K, phi, y,nn) written
% by Ogor Carron
% % -- Measurement matrix "A"
N   = 256;
M   = 128; %32,64
A   = (1/sqrt(M))*randn(M,N);
S = 1:1:80; %1:1:20,1:1:40
tol = 10^-5;
trials = 200;
successrate = zeros(length(S),1);
KvalnormMat = zeros(length(S),1);
 tic
for j = 1:length(S)
Kvalnorm = zeros(trials,1);
count = 0;
for i = 1:trials
K   = S(j)
T   = randperm(N);
T   = T(1:K);
x   = zeros(N,1);
x(T)= randn(K,1);
y   = A*x;
[xh,That]=SP(K, A, y)
Kvalnorm(i,1) = norm(xh-x)/norm(x)
if Kvalnorm(i,1) < tol
    count = count + 1;
    disp('Satisfactory Reconstruction')
else
    disp('Reconstruction Fail')
end
end
count
successrate(j,:) = 100*((count/trials))
results=[successrate,S']
toc
end
```

B.5.2 Computational Time

```
% This script evaluates the recovery of SP.
% It calls the function SP(K, phi, y,nn) written
% by Ogor Carron
% % -- Measurement matrix "A"
N    = 256;%512,1024,2048,4096,8192,16384,32768
M    = 0.125*N;%0.25*N,0.5*N
A    = (1/sqrt(M))*randn(M,N);
S = 0.125*M;
tol = 10^-5;
trials = 200;
KvalnormMat = zeros(length(S),1);
 tic
for j = 1:length(S)

Kvalnorm = zeros(trials,1);
count = 0;
for i = 1:trials
K    = S(j)
T    = randperm(N);
T    = T(1:K);
x    = zeros(N,1);
x(T)= randn(K,1);
y    = A*x;
[xh,That]=SP(K, A, y)
Kvalnorm(i,1)  = norm(xh-x)/norm(x)
if Kvalnorm(i,1)  < tol
    count = count + 1;
    disp('Satisfactory Reconstruction')
else
    disp('Reconstruction Fail')
end
end
time = toc/trials
end
```

B.6 Iterative Hard Thresholding IHT

B.6.1 Recovery

```
% This code implements the IHT algorithm. It calls a outside function
% named hard_l0_Mterm from the toolbox sparsify written by Blanchard
% and Tanner
N = 256;
M = 0.125*N; % 0.25*N, 0.5*N

Phi = 1/sqrt(M)*(randn(M,N));

K = 1:1:20;%1:1:40 1:1:80
successrate = zeros(length(K),1);
trials = 1;

for j = 1:length(K)

count = 0;
iter = 0;
for i = 1:trials


S    = K;
T    = randperm(N);
T    = T(1:S);
x    = zeros(N,1);
x(T)= randn(S,1);
y = Phi*x;
Meas  =S;

[s, err_mse,correct]=hard_l0_Mterm(y,Phi,N,Meas,'maxIter',N^2,'verbose'
    ,true);
%if correct == 1
%     count = count +1;
%end
 if err_mse <1e-5
     count = count + 1;
 end

end

successrate(j,1) = 100*(count/trials)
end
%time = toc/trials
```

```
results = [successrate,K']
```

B.6.2 Computational Time

```
% This code implements the IHT algorithm. It calls a outside function
% named hard_l0_Mterm from the toolbox sparsify written by Blanchard
% and Tanner
N = 256;
M = 0.125*N; % 0.25*N, 0.5*N

Phi = 1/sqrt(M)*(randn(M,N));

K = 1:1:20;%1:1:40 1:1:80
successrate = zeros(length(K),1);
trials = 200;
tic
for j = 1:length(K)

count = 0;
iter = 0;
for i = 1:trials


S    = K;
T    = randperm(N);
T    = T(1:S);
x    = zeros(N,1);
x(T)= randn(S,1);
y = Phi*x;
Meas  =S;

[s, err_mse,correct]=hard_l0_Mterm(y,Phi,N,Meas,'maxIter',N^2,'verbose',true);
%if correct == 1
%     count = count +1;
%end
 if err_mse <1e-5
     count = count + 1;
 end

end

%successrate(j,1) = 100*(count/trials)
```

```
end
time = toc/trials
%results = [successrate,K']
```

Bibliography

[1] R.G Baraniuk. Compressive sensing: A new framework for sparse signal acquisition and processing. URL http://www.eecis.udel.edu/~cimini/dl/Baraniuk_20110504.pdf.

[2] R.G. Baraniuk. Compressive sensing. *IEEE Signal Processing Magazine*, pages pp. 118–124, July 2007.

[3] G. Kutyniok. “Theory and applications of compressed sensing”. pages pp. 1–25, August 2012.

[4] M.B Wakin E.J Candès. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, pages pp. 21–30, March 2008.

[5] H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the A.I.E.E*, pages pp. 617–644, February 2008.

[6] C.E. Shannon. Certain topics in telegraph transmission theory. *Proc. Institute of Radio Engineers*, 37(1):pp. 617–644, January 2008.

[7] J. Romberg E.J Candès and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):pp. 1207–1223, 2006.

[8] D.L Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):pp. 1289–1306, April 2006.

[9] J. Romberg E.J Candès and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *Information Theory, IEEE Transactions on*, 52(2):pp. 489–509, 2006.

[10] E.J Candès and T. Tao. Decoding by linear programming. *Information Theory, IEEE Transactions on*, 51(12):pp. 4203–4215, December 2005.

[11] E.J Candès and T. Tao. Near-optimal signal recovery from random prejections: Universal encoding strategies. *Information Theory, IEEE Transactions on*, 52(12):pp. 5406–5425, December 2006.

[12] J.A. Tropp and A.C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *Information Theory, IEEE Transactions on*, 53(12):pp. 4655–4666, December 2007.

[13] S.G. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *Signal Processing, IEEE Transactions on*, 41(12):pp. 3397–3415, December 1993.

[14] I. Drori D.L. Donoho, Y. Tsaig and J-L. Starck. Sparse solution of undetermined linear equations by stagewise orthogonal matching pursuit. *Information Theory, IEEE Transactions on*, pages pp. 1–39, March 2006.

[15] D. Needell and J.A. Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3):301 – 321, 2009. ISSN 1063-5203. doi: http://dx.doi.org/10.1016/j.acha.2008.07.002. URL http://www.sciencedirect.com/science/article/pii/S1063520308000638.

[16] T. Blumensath and M.E. Davies. Gradient pursuits. *Signal Processing, IEEE Transactions on*, 56(6):pp. 2370–2382, June 2008.

[17] T. Blumensath and M.E. Davies. Iterative hard thresholding for compressed sensing. May 2008.

[18] D. Needell. *Topics in Compressed Sensing*. PhD thesis, University of California.

[19] Simon Foucart and Holger Rauhut. A mathematical introduction to compressive sensing. *Appl. Numer. Harmon. Anal. Birkhäuser, Boston*, 2013.

[20] J-L. Starck J. Bobin and R.Ottensamer. Compressed sensing in astronomy. *IEEE Journal of Selected topics in Signal Processing*, 2(5):pp. 718–726.

[21] *Compressed Sensing: Theory and Applications*. Cambridge University Press, 2012.

[22] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers, Bonston, MA, 2005.

[23] Shaobing Chen and David Donoho. Basis pursuit. In *Signals, Systems and Computers, 1994. 1994 Conference Record of the Twenty-Eighth Asilomar Conference on*, volume 1, pages 41–44. IEEE, 1994.

[24] S.S Chen D.L Donoho and M.A. Saunders. Atomic decompostion by basis pursuit. *SIAM J. Sci. Comput.*, 20:pp. 33–61, 1998.

[25] E.J Candès. The restricted isometry property and its implications for compressed sensing. *Comptes Redus Mathematique*, 346:pp. 589–592, May 2008.

[26] R. DeVore R.G. Baraniuk, M. Davenport and M. Wakin. A simple proof of the restricted isometry property for random matrices. *Constructive Approximation*, 2007.

[27] A. Pajor S. Mendelson and N. Tomczak-Jaegermann. Uniform uncertainty principle for bernoulli and subgaussian ensembles. 2006.

[28] D. Sundman. *Compressed Sensing Algorithms and Applications*. PhD thesis, Royal Institute of Technology, 2012.

[29] V.N. Templyakov D.L Donoho, M. Elad. Stable recovery of sparse overcomplete representations in the presence of noise. *IEEE Transactions on Information Theory*, 52 (1):pp. 6–18, 2006.

[30] J.A. Tropp. Just relax: Convex programming methods for identifying sparse signals in noise. *Information Theory, IEEE Transactions on*, 52(3):pp. 1030–1051, March 2006.

[31] X. Huo D.L Donoho. Uncertainty principles and ideal atomic decomposition. *IEEE Transactions on Information Theory*, 47(11):pp. 2845–4862, 2001.

[32] Albert Cohen, Wolfgang Dahmen, and Ronald DeVore. Compressed sensing and best -term approximation. *Journal of the American Mathematical Society*, 22(1):211–231, 2009.

[33] Graeme Pope. Compressive sensing a summary of reconstruction algorithms. Master's thesis, Eidgenössische Technische Hochschule, 2009.

[34] J. Romberg E.J Candès. Sparsity and incoherence in compressive sampling. *Inverse Problems*, 23(3):pp. 969–985, 2007.

[35] T. Tao E.J Candès, M. Rudelson and R. Vershynin. *In FOCS*.

[36] *Convex Optimization*. Cambridge University Press, 2004.

[37] Markus B. Schenkel. Sparsity based correlation models for joint reconstruction of compressed images. Master's thesis, Swiss Federal Institute of Technology, 2010.

[38] Mike E. Davies, Thomas Blumensath, and Gabriel Rilling. Greedy Algorithms for Compressed Sensing. In Y. C. Eldar and G. Kutyniok, editors, *Compressed Sensing: Theory and Applications*. Cambridge University Press, 2011. URL http://hal.inria.fr/inria-00574440.

[39] Sacha Krstulovic and Rémi Gribonval. Mptk: Matching pursuit made tractable. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 3, pages III–III. IEEE, 2006.

[40] S.F. Cotter, R. Adler, R.D. Rao, and K. Kreutz-Delgado. Forward sequential algorithms for best basis selection. *Vision, Image and Signal Processing, IEE Proceedings -*, 146(5): 235–244, 1999. ISSN 1350-245X. doi: 10.1049/ip-vis:19990445.

[41] S. Mallat and Z. Zhang. Adaptive time-frequency decomposition with matching pursuits. In *Time-Frequency and Time-Scale Analysis, 1992., Proceedings of the IEEE-SP International Symposium*, pages 7–10, 1992. doi: 10.1109/TFTSA.1992.274245.

[42] Daniel Zahonero Inesta. Compressed sensing for ultra wideband (uwb) systems. Master's thesis, Tennessee Technological University, 2009.

[43] Paweł Bechler and Przemysław Wojtaszczyk. Error estimates for orthogonal matching pursuit and random dictionaries. *Constructive Approximation*, 33(2):273–288, 2011.

[44] *Multiuser Detection*. Cambridge University Press, 1998.

[45] Shihao Ji and Lawrence Carin. Bayesian compressive sensing and projection optimization. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 377–384, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273544. URL http://doi.acm.org/10.1145/1273496.1273544.

[46] Y. Benjamini D. L. Donoho Abramovich, F. and I. M. Johnstone. Adapting to unknown sparsity by controlling the false discovery rate. *Annals of Statistics*, 2000.

[47] Michael Elad, Boaz Matalon, Joseph Shtok, and Michael Zibulevsky. A wide-angle view at iterated shrinkage algorithms. In *Optical Engineering+ Applications*, pages 670102–670102. International Society for Optics and Photonics, 2007.

[48] J. Portilla. Image restoration through l0 analysis-based sparse optimization in tight frames. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 3909–3912, 2009. doi: 10.1109/ICIP.2009.5413975.

[49] N. G. Kingsbury and T. Reeves. Iterative image coding with overcomplete complex wavelet transforms. In T. Ebrahimi and T. Sikora, editors, *Visual Communications and Image Processing 2003*, volume 5150 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 1253–1264, June 2003. doi: 10.1117/12.509901.

[50] Kenneth Lange, David R Hunter, and Ilsoon Yang. Optimization transfer using surrogate objective functions. *Journal of computational and graphical statistics*, 9(1):1–20, 2000.

[51] Deanna Needell and Roman Vershynin. Uniform uncertainty principle and signal recovery via;regularized orthogonal matching pursuit. *Found. Comput. Math.*, 9(3): 317–334, April 2009. ISSN 1615-3375. doi: 10.1007/s10208-008-9031-3. URL http://dx.doi.org/10.1007/s10208-008-9031-3.

[52] Wei Dai and Olgica Milenkovic. Subspace pursuit for compressive sensing: Closing the gap between performance and complexity. *CoRR*, abs/0803.0811, 2008.

[53] Y.S. Han, C.R.P. Hartmann, and Chih-Chieh Chen. Efficient priority-first search maximum-likelihood soft-decision decoding of linear block codes. *Information Theory, IEEE Transactions on*, 39(5):1514–1523, 1993. ISSN 0018-9448. doi: 10.1109/18.259636.

[54] Ding Zhu Du and Frank Hwang. *Combinatorial group testing and its applications*. World Scientific, 1993.

[55] Yaniv Erlich, Noam Shental, Amnon Amir, and Or Zuk. Compressed sensing approach for high throughput carrier screen. In *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on*, pages 539–544. IEEE, 2009.

[56] Noam Shental, Amnon Amir, and Or Zuk. Identification of rare alleles and their carriers using compressed se (que) nsing. *Nucleic acids research*, 38(19):e179–e179, 2010.

[57] Raghunandan M Kainkaryam, Angela Bruex, Anna C Gilbert, John Schiefelbein, and Peter J Woolf. poolmc: Smart pooling of mrna samples in microarray experiments. *BMC bioinformatics*, 11(1):299, 2010.

[58] Graham Cormode and Marios Hadjieleftheriou. Finding the frequent items in streams of data. *Communications of the ACM*, 52(10):97–105, 2009.

[59] S Muthukrishnan. Data streams: Algorithms and applications, volume 1 of found. trends in theoretical comput. *Science. Now Publishers, Boston, MA*, 2005.

[60] Saad Qaisar, Rana Muhammad Bilal, Wafa Iqbal, Muqaddas Naureen, and Sungyoung Lee. Compressive sensing: From theory to applications, a survey. *Communications and Networks, Journal of*, 15(5):443–456, 2013.

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

[62] Anna C Gilbert, Martin J Strauss, Joel A Tropp, and Roman Vershynin. Algorithmic linear dimension reduction in the l_1 norm for sparse vectors. *arXiv preprint cs/0608079*, 2006.

[63] Anna C Gilbert, Martin J Strauss, Joel A Tropp, and Roman Vershynin. One sketch for all: fast algorithms for compressed sensing. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 237–246. ACM, 2007.

[64] Radu Berinde, Anna C Gilbert, Piotr Indyk, H Karloff, and Martin J Strauss. Combining geometry and combinatorics: A unified approach to sparse signal recovery. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, pages 798–805. IEEE, 2008.

[65] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[66] Anna C Gilbert, Yi Li, Ely Porat, and Martin J Strauss. Approximate sparse recovery: optimizing time and measurements. *SIAM Journal on Computing*, 41(2):436–453, 2012.

[67] Anna C Gilbert, Martin J Strauss, Joel A Tropp, and Roman Vershynin. One sketch for all: fast algorithms for compressed sensing. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 237–246. ACM, 2007.

[68] Wotao Yin, Stanley Osher, Donald Goldfarb, and Jerome Darbon. Bregman iterative algorithms for \ell_1-minimization with applications to compressed sensing. *SIAM Journal on Imaging Sciences*, 1(1):143–168, 2008.

[69] Joseph F Murray and Kenneth Kreutz-Delgado. An improved focuss-based learning algorithm for solving sparse linear inverse problems. In *Signals, Systems and Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on*, volume 1, pages 347–351. IEEE, 2001.

[70] Hong Jung, Kyunghyun Sung, Krishna S Nayak, Eung Yeop Kim, and Jong Chul Ye. k-t focuss: A general compressed sensing framework for high resolution dynamic mri. *Magnetic Resonance in Medicine*, 61(1):103–116, 2009.

[71] David P Wipf and Bhaskar D Rao. Sparse bayesian learning for basis selection. *Signal Processing, IEEE Transactions on*, 52(8):2153–2164, 2004.

[72] Rick Chartrand and Wotao Yin. Iteratively reweighted algorithms for compressive sensing. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 3869–3872. IEEE, 2008.

[73] Ingrid Daubechies, Ronald DeVore, Massimo Fornasier, and C Sinan Güntürk. Iteratively reweighted least squares minimization for sparse recovery. *Communications on Pure and Applied Mathematics*, 63(1):1–38, 2010.

[74] SJ Godsill, AT Cemgil, C Févotte, and PJ Wolfe. Bayesian computational methods for sparse audio and music processing. *Proc. EUSIPCO, Poznan, Poland*, pages 345–349, 2007.

[75] David L Donoho, Arian Maleki, and Andrea Montanari. Message-passing algorithms for compressed sensing. *Proceedings of the National Academy of Sciences*, 106(45):18914–18919, 2009.

[76] Piotr Indyk and Milan Ruzic. Near-optimal sparse recovery in the l1 norm. In *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 199–207. IEEE, 2008.

[77] Radu Berinde, Piotr Indyk, and Milan Ruzic. Practical near-optimal sparse recovery in the l1 norm. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, pages 198–205. IEEE, 2008.

[78] Radu Berinde and Piotr Indyk. Sequential sparse matching pursuit. In *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on*, pages 36–43. IEEE, 2009.

[79] David L Donoho and Jared Tanner. Counting the faces of randomly-projected hypercubes and orthants, with applications. *Discrete & computational geometry*, 43(3): 522–541, 2010.

[80] Wei Dai and Olgica Milenkovic. Subspace pursuit for compressive sensing: Closing the gap between performance and complexity. *CoRR*, abs/0803.0811, 2008.

[81] Thomas Blumensath, Michael E Davies, et al. How to use the iterative hard thresholding algorithm. In *SPARS'09-Signal Processing with Adaptive Sparse Structured Representations*, 2009.

[82] V. Stodden D. Dononho and Y. Tsaig. About sparselab. pages pp. 1–18, March 2007.

[83] D. Dononho *et al.* 2007. URL http://sparselab.stanford.edu/.

[84] Emmanuel Candes and Justin Romberg. l1-magic: Recovery of sparse signals via convex programming. 4, 2005. URL www.acm.caltech.edu/l1magic/downloads/l1magic.pdf.

[85] Emmanuel Candès and Justin Romberg. l1-magic. 2007. URL www.l1-magic.org.

[86] I. Carron. Compressed sensing codes. URL https://sites.google.com/site/igorcarron2/cscodes.

[87] Arian Maleki and David L. Donoho. Optimally tuned iterative reconstruction algorithms for compressed sensing. *CoRR*, abs/0909.0777, 2009.

[88] T. Blumensath. Sparsify. URL http://www.personal.soton.ac.uk/tb1m08/sparsify/sparsify.html.

[89] M. McCoy. Matlab code. URL http://users.cms.caltech.edu/~mccoy/code/.

[90] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[91] Jeffrey D Blanchard and Jared Tanner. Performance comparisons of greedy algorithms in compressed sensing, 2013.

[92] Jeffrey D Blanchard and Jared Tanner. Gpu accelerated greedy algorithms for compressed sensing. *Mathematical Programming Computation*, 5(3):267–304, 2013.

[93] Jeffrey D Blanchard and Jared Tanner. Gpu accelerate greedy algorithms for compressed sensing. URL http://www.gaga4cs.org/.

[94] Duane Merrill and Andrew Grimshaw. High performance and scalable radix sorting: A case study of implementing dynamic parallelism for gpu computing. *Parallel Processing Letters*, 21(02):245–272, 2011.

[95] Nathan Bell and Jared Hoberock. Thrust: A productivity-oriented library for cuda. *GPU Computing Gems*, 7, 2011.

[96] NVIDIA. Cuda toolkit 6.0 (2014). URL https://developer.nvidia.com/cuda-toolkit.

[97] Donald C. S. Allison and MT Noga. Usort: An efficient hybrid of distributive partitioning sorting. *BIT Numerical Mathematics*, 22(2):135–139, 1982.