2015

One-Bit Compressive Sensing with Partial Support Information

Phillip North
Claremont McKenna College

Claremont McKenna College

One-Bit Compressive Sensing with Partial Support Information

submitted to
Professor Deanna Needell
and
Dean Nicholas Warner

by
Phillip North

for
Senior Thesis
Spring 2015
April 24, 2015

Abstract

This work develops novel algorithms for incorporating prior-support information into the field of One-Bit Compressed Sensing. Traditionally, Compressed Sensing is used for acquiring high-dimensional signals from few linear measurements. In applications, it is often the case that we have some knowledge of the structure of our signal(s) beforehand, and thus we would like to leverage it to attain more accurate and efficient recovery. Additionally, the Compressive Sensing framework maintains relevance even when the available measurements are subject to extreme quantization. Indeed, the field of One-Bit Compressive Sensing aims to recover a signal from measurements reduced to only their sign-bit. This work explores avenues for incorporating partial-support information into existing One-Bit Compressive Sensing algorithms. We provide both a rich background to the field of compressed sensing and in particular the one-bit framework, while also developing and testing new algorithms for this setting. Experimental results demonstrate that newly proposed methods of this work yield improved signal recovery even for varying levels of accuracy in the prior information. This work is thus the first to provide recovery mechanisms that efficiently use prior signal information in the one-bit reconstruction setting.

Contents

Chapter 1

Compressed Sensing

1.1 Motivation

Compressed Sensing is a method for accurately acquiring high dimensional signals from a set of relatively few linear measurements [CT06] [Don04] [CRT06]. Typical approaches to signal acquisition exert serious computational efforts to acquire the entirety of the signal, only to then discard much of it in favor of a compressed representation. Certainly it would be more efficient to acquire only the information necessary to define the signal in its compressed form. This idea of Compressed Sensing has applications across many domains including statistics, machine learning, imaging, and the natural sciences.

1.2 Background Information

The problem posed by Compressed Sensing (CS) may be viewed as solving an under-determined system of linear equations. Rudimentary Linear Algebra provides that the reconstruction of general signals from an incomplete set of measurements is not possible. Therefore we must enforce a particular structure on the desired signal. Indeed, we require that our desired signal have relatively few non-zero entries, i.e. that it is sparse. This may seem to be an extremely debilitating requirement. However, even signals that do not initially

appear to be sparse have a sparse representation with respect to a different basis. Furthermore, non-sparse signals encountered in practice are often compressible, which implies that they may be well-approximated by some sparse signal. We define our desired signal, x, to be k-sparse if $\|x\|_{\ell_0} \leq k$, where the ℓ_0 norm of x is simply a count of the number of non-zero elements.

A measurement of x is seen as the result of applying a linear function to x. The process of collecting multiple measurements can be represented by the multiplication of x by a measurement matrix, denoted Φ. If x has length n and we take m measurements then Φ is an $m \times n$ matrix. The point of CS is to accurately recover x from $m \ll n$ measurements. If we denote our set of m measurements as y then the problem becomes solving the highly underdetermined linear system $\Phi x = y$ for the sparsest solution.

Intuitively, it is clear that we would like the measurements generated from Φ to preserve and be indicative of the geometry of x. In [CT05] Candès and Tao rigorously require this by imposing the *Restricted Isometry Property (RIP)* on Φ, which requires that:

$$(1 - \delta_k)\|x\|_2^2 \leq \|\Phi x\|_2^2 \leq (1 + \delta_k)\|x\|_2^2$$

for all k-sparse vectors x, where δ_k, referred to as the *restricted isometry constant*, is the smallest number such that the above expression is satisfied. If Φ satisfies the RIP then in order to recover a k-sparse signal we must take at least $2k$ measurements. Constructing a measurement matrix that will satisfy the RIP with a satisfiable constant is a nearly impossible task. Amazingly enough, randomly generated matrices, such as Gaussian matrices or Partial Fourier matrices, satisfy the RIP with $\delta_{2k} < 0.1$ with exceptionally high probability.

When reconstructing x, we would like to find the sparsest signal that is consistent with the measurements that we are initially given. A large portion of the work in the reconstruction of x is finding its support: the location of the non-zero entries. This may be represented as

a solution to:

$$min_x \|x\|_{\ell_0} : \Phi x = y.$$

Unfortunately, the above program is NP-Complete. However, [CRT06] [CT05] [Don06] show that we may relax the program to instead minimize the ℓ_1 norm of the signal. The following program produces the desired result and may be solved using convex optimization methods:

$$min_x \|x\|_{\ell_1} : \Phi x = y.$$

This framework is also relevant when our signal is contaminated with noise, the above program then requires that $\|\Phi x - y\|_2 \leq \epsilon$. Figure 1.1 shows the recovery of x resulting from minimizing the above ℓ_1 program. The mean squared error (MSE) between the resulting approximation and the exact signal is plotted as a function of m, the number of measurements. Here and throughout, unless stated otherwise: entries of our measurement matrix Φ are drawn from the standard Gaussian distribution, the support of our signal x is drawn from a discrete uniform distribution of integers 1 through n, and the non-zero entries of x are also generated from the standard Gaussian distribution.

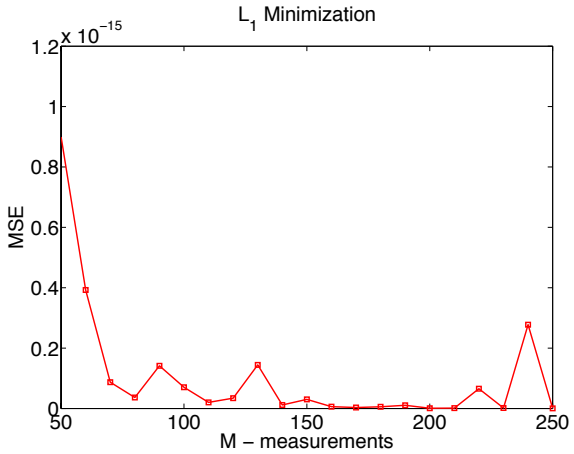


Figure 1.1: For $n = 256$ and $k = 8$, this graph plots the average MSE over 100 trials, for different values of m, when using ℓ_1 minimization.

Notice that only taking $m = 50$ measurements when the signal length is $n = 256$ results in an approximation with a MSE on the order of machine precision; this is indicative of near

perfect recovery. Though techniques used for minimizing this ℓ_1 program are accurate, they are often times slow and computationally expensive. In fact, equally accurate recovery may be achieved from faster greedy algorithms.

In [NT09] Needell and Tropp present Compressive Sampling Matching Pursuit (CoSaMP) as a greedy algorithm that, in the absence of noise, can recover a k-sparse signal to arbitrarily high precision. Furthermore, the runtime of this algorithm is proportional to the cost of simple matrix-vector multiplication. CoSaMP recovers x from the given measurements y, by iteratively determining the support of x. At any given iteration, $\Phi^*\Phi x = \Phi^* y$ is used as a proxy for x, then a residual $r = y - \Phi\hat{x}$ is computed. The support of this residual updates our approximation $\hat{x}$ by solving a simple least squares problem. Algorithm 1 describes the steps of CoSaMP in greater detail.

Algorithm 1 Compressive Sampling Matching Pursuit. Given: measurement matrix Φ, measurements of desired signal y, assumed sparsity level k

procedure CoSaMP(Φ, y, k)
  $x_0 = 0$ ▷ Initialize trivial approximation of x
  $v = y$ ▷ Initialize current measurements as the given measurements
  $i = 0$ ▷ Let i denote our iteration count
  repeat
    $i = i + 1$ ▷ Increment iteration count

    $p = \Phi^* v$ ▷ Compute proxy
    $\Omega = support(p_{2k})$ ▷ Determine support of proxy
    $\Gamma = support(x_0)$ ▷ Determine support of current approximation
    $T = \Omega \cup \Gamma$ ▷ Merge the two supports

    $\beta_T = \Phi_T^{\dagger} y$ ▷ Use the p. inverse of Φ restricted to T to solve least squares
    $\beta_T^c = 0$ ▷ Assign elements off the estimated support to zero

    $x_i = \beta_k$ ▷ Prune β to greatest k elements and update approximation of x
    $v = y - \Phi x_i$ ▷ Update remaining measurements

  until no change in approximation or iteration count is sufficient
  return x_i
end procedure

Figure 1.2 shows the recovery results of CoSaMP for different sparsity levels and measure-

ments taken. We see that ℓ_1 minimization outperforms CoSaMP when less measurements are taken. Otherwise, their recovery accuracy is roughly the same, with CoSaMP using less resources.

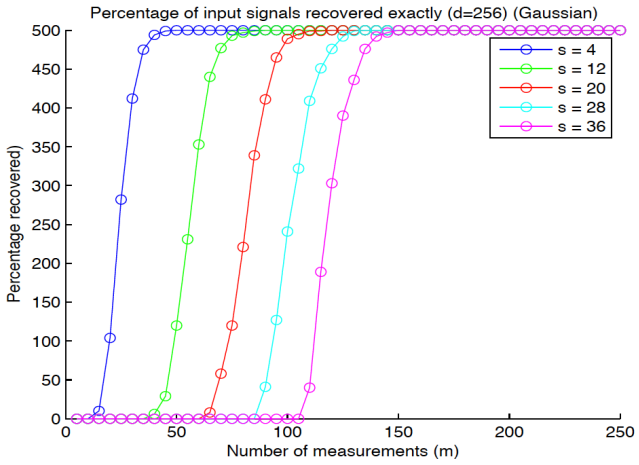


Figure 1.2: With the dimension of our signal 256, this graph from [Nee09] plots the percentage of trials resulting in perfect recovery for different values of m and various sparsity levels s.

We have seen that there exist efficient algorithms for recovering signals that are sparse with respect to an orthonormal basis. Unfortunately, in practice most signals of interest have few entries that are zero. However, it is possible to construct a sparse representation of these signals by using a *redundant dictionary,* D: usually a poorly conditioned, highly redundant matrix.

When this is the case our desired signal x is re-expressed as $x = D\alpha$ where α is a sparse coefficient vector. Thus, our linear measurements are now of the form $y = \Phi x = \Phi D\alpha$. There are two distinct approaches to solving this problem, the first is a *coefficient* focused approach while the second is a *signal* based approach. A coefficient based strategy will aim to recover α, while signal based approaches are concerned with recovering $D\alpha$. A major difficulty in trying to recover α is that in order to apply classical Compressive Sensing techniques one must require that D satisfy the RIP. This is simply not a practical expectation. Davenport, Needell, Wakin in [DNW12] take a signal-focused approach and show that we need only require Φ to satisfy the *Dictionary Restricted Isometry Property (D-RIP)*. Explicitly, this

requires that our measurement matrix Φ satisfies:

$$(1 - \delta_k)\|D\alpha\|_2^2 \leq \|\Phi D\alpha\|_2^2 \leq (1 + \delta_k)\|D\alpha\|_2^2$$

for all k-sparse α, where again δ_k is the smallest number such that this expression is satisfied. This is a much more feasible demand than requiring ΦD to satisfy the *RIP*.

In addition, [DNW12] presents Signal Space Compressive Sampling Matching Pursuit (SS-CoSaMP), one signal-focused algorithm designed to recover signals sparse with respect to a redundant basis. This algorithm iteratively updates support information relevant to the signal approximation $\hat{x}$. Similar to CoSaMP, we start with a residual r and some arbitrary initial guess for $\hat{x}$ and its support Γ. We construct a proxy $p = \Phi^* r$ and use a projection method to find the support information corresponding to an approximation that is $2k$-sparse with respect to ΦD. Here too, a linear least-squares problem is solved and projected to the nearest k-sparse approximation. The signal approximation is then updated with this support information, the residual is computed, and the process continues. Presented in Algorithm 2 is a more complete description of SS-CoSaMP.

This algorithm requires the existence of a near optimal method for projecting our estimate onto a vector that is sparse with respect to D. As an approximation for such a projection method, we elect to use standard $CoSaMP$. However, this is not a provably optimal choice; ℓ_1-Minimization and Orthogonal Matching Pursuit (OMP), as presented in [Zha11], are also suitable choices. Setting $n = 256$ and our sparsity level to $k = 8$, Figure 1.3 plots the MSE for different values of m. It has been empirically shown that SSCoSaMP, when CoSaMP is chosen as the method for projection, works particularly well when the k non-zero entries of the coefficient vector α are placed in a single randomly positioned block.

Algorithm 2 Signal Space Compressive Sampling Matching Pursuit. Given: measurement matrix Φ, dictionary D, measurements of desired signal y, assumed sparsity level k

procedure SS-CoSaMP(Φ, D, y, k)
  $x_0 = 0$ ▷ Initialize trivial approximation of x
  $r = y$ ▷ Initialize trivial residual
  $\Gamma = \emptyset$ ▷ Initialize trivial support estimate
  $i = 0$ ▷ Let i denote our iteration count
  repeat
    $i = i + 1$ ▷ Increment iteration count

    $p = \Phi^* r$ ▷ Compute proxy p
    $\Omega = support(CoSaMP(D, p, 2s))$ ▷ Use CoSaMP to find support information of an approximation 2s-sparse with respect to D

    $T = \Omega \cup \Gamma$ ▷ Merge support estimates
    $\tilde{x} = D_T (AD_T)^\dagger y$ ▷ Solve least squares approximation
    $\Gamma = support(CoSaMP(D, \tilde{x}, s))$ ▷ Similar use of CoSaMP

    $x^i = D_\Gamma D_\Gamma^\dagger \tilde{x}$ ▷ Update signal approximation
    $r = y - \Phi x^i$ ▷ Update residual

  until no change in approximation or iteration count is sufficient
  return x_i
end procedure

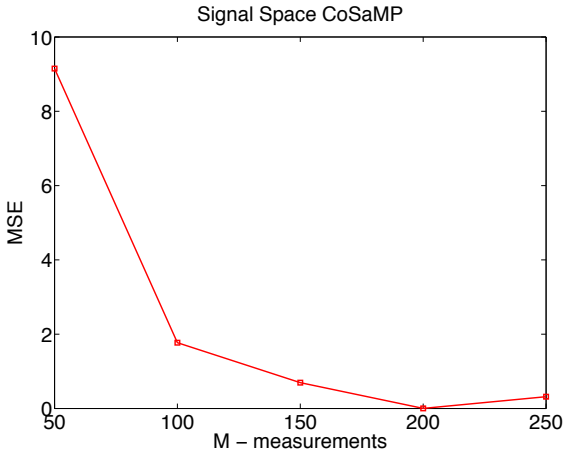


Figure 1.3: For $n = 256$ and $k = 8$, this graph plots the average MSE over 10 trials, for different values of m, when using the SS-CoSaMP algorithm to recover signals that are not inherently sparse.

Chapter 2

Partial Support Information & One-Bit Compressed Sensing

2.1 Partial Support Information Motivation

In the field of high-dimensional signal acquisition it may be the case that there are many signals of interest, and it could be that they are correlated to one another in some fashion. Perhaps, domain-specific knowledge provides us with a good idea of the structure of our desired signal. In either case, it is only prudent that computational efforts are tuned to leverage the given prior knowledge. More explicitly, prior knowledge sometimes comes in the form of a support estimate which may be used in accordance with a traditional CS algorithm. Compressive Sensing techniques that make use of such support estimates can be particularly useful for the task of acquiring multiple signals or for the acquisition of signals in wavelet or frequency domains.

2.2 Partial Support Background Information

We have discussed the compressive sensing technique of ℓ_1-minimization before, now let us consider the instance where some structure of the desired signal, x, is known. Say that we

have some estimate $\tilde{T} \subset \{1, 2, \ldots, n\}$ of the true support T of x. [MS14] [FMSY10] extend the conventional ℓ_1-minimization approach to a *weighted* ℓ_1 approach that effectively incorporates such a support estimate. Recall that the standard linear program for ℓ_1 minimization is:

$$min_x \|x\|_{\ell_1} : \Phi x = y.$$

As an incorporation of our estimate $\tilde{T}$, now consider the following program:

$$min_x \sum_{i=1}^{n} w_i |x_i| : y = \Phi x \text{ where } w_i = \begin{cases} c \in [0,1] \text{ if } w_i \in \tilde{T} \\ 0 \text{ if } w_i \notin \tilde{T} \end{cases}.$$

The idea here is the same, we are still minimizing the ℓ_1 norm of the recovered signal, however we now impose a penalty for placing non-zero entries in locations not specified in the support estimate $\tilde{T}$. Entries of the weight vector w are determined by the confidence that we put in each element of $\tilde{T}$. Locations that are thought to be more likely to be non-zero are given a value closer to zero, and thus they are penalized less for taking on more significant values. In [FMSY10], Friedlander et al. show that if the accuracy of $\tilde{T}$ is greater than 50%, then the weighted ℓ_1 program recovers the signal when the traditional ℓ_1 approach also succeeds; in instances when the accuracy exceeds 50% and the weights are small enough, the weighted ℓ_1 approach will succeed when ℓ_1 minimization fails.

Furthermore, [GMY13] presents a very similar approach where an ℓ_p-norm, for $p \in (0, 1)$, is instead minimized. The authors show that using such an ℓ_p-norm yields stronger recovery guarantees than the traditional ℓ_1-norm. In addition they show that a weighted ℓ_p approach, when support information is available, has stronger guarantees than the aforementioned weighted-ℓ_1 approach. However, by using the ℓ_p norm the minimization program becomes convex and thus requires expensive numerical methods to attain only a rough approximation.

2.3 One-Bit Compressed Sensing Motivation

The goal of Compressed Sensing is to acquire a high-dimensional signal using as little memory as possible, hence why we attempt to recover x with as few measurements as possible. We have, up to this point, assumed each of our measurements y_i to have an infinite bit-depth, i.e. $y_i \in \mathbb{R}$. In applications, however, this is never the case, any measurement is quantized to a finite bit-depth when it takes on a physical representation; for one example, most modern computers represent real numbers with 64-bits. When constrained by the physical world, there exists an inverse relationship between the number of measurements that may be taken and each measurement's bit-depth. The most extreme form of quantization involves representing a measurement by only one-bit: its sign bit. One-Bit Compressed Sensing aims to recover sparse signals from a set of measurements subjected to such quantization [BB08]. The reduced cost of one-bit measurements offers sincere hardware advantages and allows a larger number of measurements to be taken at a lesser cost.

2.4 One-Bit Compressed Sensing Background Information

Extending our framework from traditional CS, our available measurements $y = \Phi x$ are now quantized to only their sign bit. We now have access to $\bar{y} := sign(y)$ where $\bar{y_i} = 1$ if $y_i > \lambda$ and $\bar{y_i} = 0$ if $y_i < \lambda$. We define λ to be our threshold for comparison, without a loss of generality we will assume $\lambda = 0$. As a result of such extreme quantization we sacrifice the amplitude of our original signal x and thus aim to recover an approximation $\tilde{x}$ that is within a positive scalar multiple of x (see [KSW14] [BFN$^+$14] for recent results which overcome this and other obstacles). However, we are now afforded the ability to take $m > n$ measurements; under the traditional CS framework taking this many measurements would completely defeat the initial motivation.

In [PV11], Plan and Vershynin present a convex program that is a simple extension of the traditional ℓ_1-minimization discussed above. The authors show that a solution to the following program yields robust recovery of x:

$$min_z \|z\|_{\ell_1} \text{ such that: } sign(\Phi z) = \bar{y} \text{ and } \|\Phi z\|_{\ell_1} = c.$$

The first constraint of the above program ensures that we receive a solution who's quantized measurements are consistent with the ones that we are initially given. The second constraint ensures that we do not attain the zero vector as a solution. We measure the accuracy of this approach by solving the above program for a fixed number of measurements and measuring the average MSE between the returned approximation and $\frac{x}{\|x\|_2}$ over 100 trials. These results are displayed in Figure 2.1. Similar to the traditional CS setting, the minimization of the above program is expensive and similar recovery may be attained by instead using a greedy algorithm.

One greedy algorithm for the reconstruction of sparse signals from one-bit measurements is the Binary Iterative Hard Thresholding Algorithm (BIHT) [JLBB11], an extension of the traditional Iterative Hard Thresholding Algorithm (IHT) [BD09]. BIHT is an iterative and very intuitive algorithm. Assuming that our desired signal is k-sparse, the objective of BIHT is to return a solution that is k-sparse and consistent with the given measurements. At each iteration, BIHT computes and takes a step in the direction of the gradient to attain a new approximation. This approximation is then thresholded to retain only the k largest in magnitude entries. Finally, after we have a consistent approximation or enough iterations have elapsed, we normalize and return our solution. Algorithm 3 presents a more detailed explanation of BIHT.

Algorithm 3 Binary Iterative Hard Thresholding (BIHT). Given: measurement matrix Φ, one-bit measurements $\bar{y}$, assumed sparsity level k

procedure BIHT(Φ, $\bar{y}$, k)
    $x_0 = 0$ ▷ Initialize trivial approximation of x
    $i = 0$ ▷ Let i denote our iteration count
    $\tau = c$ ▷ Set gradient-descent step-size to desired constant c
    repeat
        $i = i + 1$ ▷ Increment iteration count

        $\Gamma = x_i + \frac{\tau}{2}\Phi'(\bar{y} - sign(\Phi x_i))$ ▷ Compute and move approximation in the direction of the gradient

        $x_{i+1} = prune(\Gamma, k)$ ▷ Hard threshold all but the greatest k in magnitude entries of Γ to 0

        $x_i = x_{i+1}$ ▷ Continue algorithm with new approximation
    until approximation is consistent with $\bar{y}$ or iteration count is sufficient
    return $\frac{x_i}{\|x_i\|_2}$ ▷ Project approximation onto unit ℓ_2 sphere
end procedure

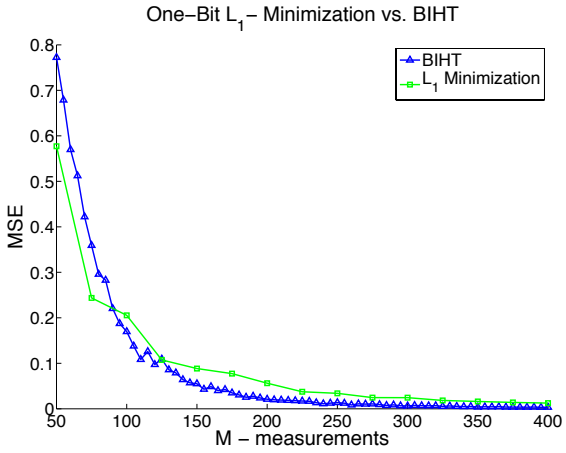


Figure 2.1: This graph compares the performance of BIHT and ℓ_1 minimization as One-Bit Compressive Sensing algorithms. Fixing the length of our signal $n = 256$ and sparsity-level $k = 8$, the recovery of our normalized desired signal as measured by MSE is plotted as a function of m, the number of measurements. Notice that we consider $m > 256$ in the one-bit context because of the significantly decreased cost per measurement.

Chapter 3

Bridging the Gap: One-Bit Compressed Sensing with Partial Support Information

3.1 Motivation

We have seen how partial support information can be used in accordance with traditional Compressive Sensing algorithms to attain more accurate and effective signal recovery techniques. Additionally, we have seen that even in the presence of the most severe form of quantization the Compressive Sensing framework is still extremely relevant via one-bit recovery algorithms. The motivation for incorporating prior support estimates is certainly not lost in the one-bit setting. In this section we explore novel methods for extending BIHT to make use of a partial support estimate. This work builds upon previous techniques in the one-bit and prior information setting, developing novel methods for the important setting of extreme quantization with partial knowledge of the signal structure. Such a framework gives much needed results for many applications such as imaging under wavelet signal structures, distributed sensing, and many other statistical and engineering applications.

3.2 Findings & Results

Recall that Binary Iterative Hard Thresholding (BIHT) is a greedy, iterative algorithm meant for the recovery of sparse signals from one-bit measurements. At each iteration the algorithm takes a step in the direction of the gradient, hopefully honing in on an approximation that is consistent with the initial binary measurements. More explicitly, at the $(i+1)^{th}$ iteration, $\Gamma := x_i + \frac{\tau}{2}\Phi'(\bar{y} - sign(\Phi x_i))$ is computed. Then Γ is hard thresholded to retain only the k-greatest in magnitude entries; this pruning step ensures that a k-sparse solution is returned. In the interest of incorporating a prior support estimate $\tilde{T}$ of unknown accuracy, it is intuitive to modify this pruning step.

3.2.1 Hard and Soft Thresholding

As a first, most basic approach, let us assume that our support estimate $\tilde{T}$ is completely accurate, i.e. $\tilde{T} \equiv T$. At the pruning step, no matter our result, we could simply set all entries of Γ not in $\tilde{T}$ to zero. Instead of locating the k largest entries of Γ and setting all others to zero, we would naively only retain the entries of $\tilde{T}$. The entries of our new estimate would be determined as follows:

$$\tilde{x}_i = \begin{cases} \Gamma_i \text{ if } i \in \tilde{T} \\ 0 \text{ if } i \in \tilde{T}^C \end{cases}$$

Figure 3.1 shows the result of this approach; observe how powerful a perfect support estimate can be via this bold hard-thresholding strategy. As a similar approach, we could instead try soft-thresholding the entries of Γ not in $\tilde{T}$. This would involve multiplying the entries of Γ not in $\tilde{T}$ by some constant $0 < c < 1$. Figure 3.1 shows that soft-thresholding with our perfect support estimate performs identically to when we were hard-thresholding. This result is not hard to understand: after j iterations the elements not in $\tilde{T}$ have been scaled down by c^j, as j increases $c^j \approx 0$ and we are basically hard-thresholding. Of course, knowing the full

support beforehand is not likely, but these examples show that this information can seriously expedite the recovery of x.

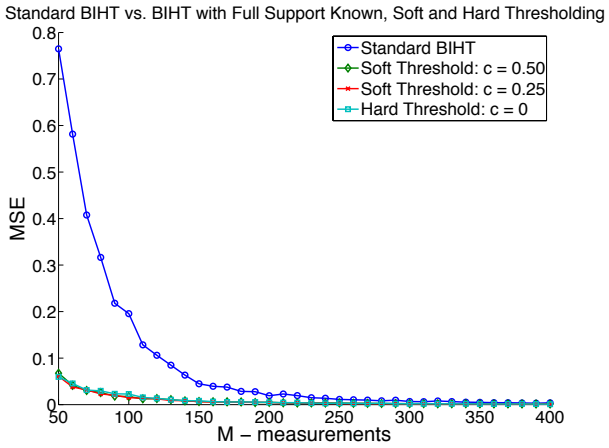


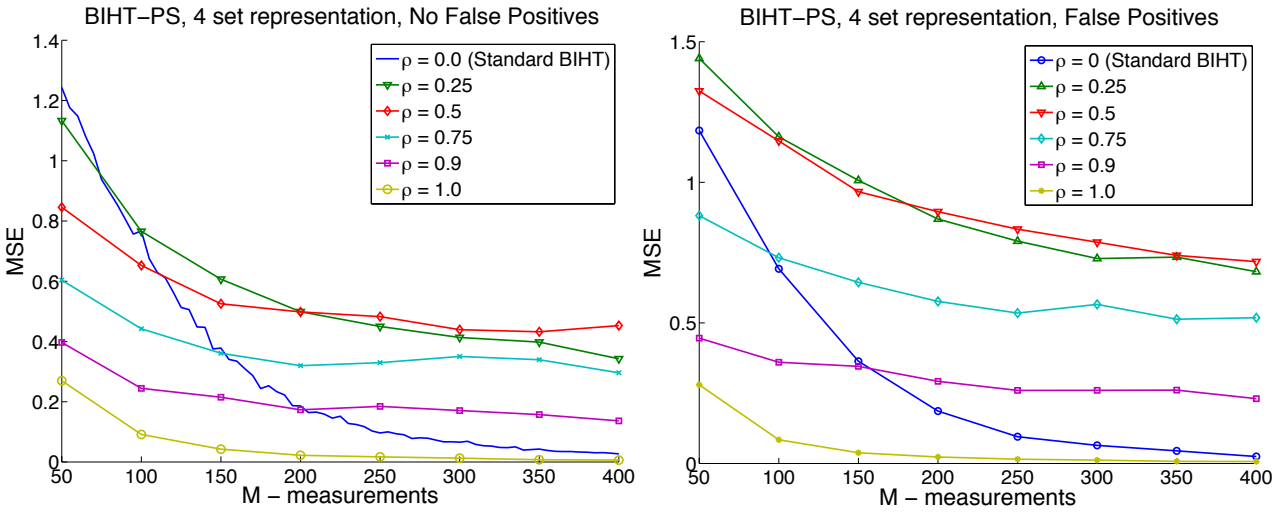
Figure 3.1: This graph shows the performance of BIHT when our support estimate is the exact support and we employ soft and hard thresholding. Here our signal length is $n = 256$, sparsity level $k = 8$, and the MSE averaged over 100 trials is plotted for various value of m, the number of measurements. At each iteration, entries of Γ that are not on our support estimate are scaled down by c, when $c = 0$ the result is hard-thresholding.

3.2.2 4-Set Representation

In practice our support estimate will seldom equal the true support, and thus at a given iteration we should consider both $\tilde{T}$ and the locations of the k greatest in magnitude entries of Γ, denoted $\dot{T}$, when updating our estimate. Let us assume that we know our support estimate to be $(\rho \times 100)\%$ accurate. Now, in the pruning step of BIHT the entries of Γ may be thresholded according to 4 distinct possibilities:

$$\tilde{x}_i = \Gamma_i w_i \text{ where } w_i = \begin{cases} 1 \text{ if } \Gamma_i \in \tilde{T} \cap \dot{T} \\ 1 \text{ if } \Gamma_i \in \tilde{T} \cap \dot{T}^C \\ 1 - \rho \text{ if } \Gamma_i \in \tilde{T}^C \cap \dot{T} \\ 0 \text{ if } \Gamma_i \in \tilde{T}^C \cap \dot{T}^C \end{cases}$$

Additionally, we can use this 4-set framework when our support estimate includes erroneous elements. Say that we have a support estimate that contains $(\rho \times k)$ correct elements but also includes $(1-\rho) \times k$ incorrect elements. Figure 3.2 shows BIHT's performance when using this 4-set representation to incorporate prior support information. In the case of no false positives, when no elements of our support estimate are incorrect, we see that performance of BIHT is not bad: as ρ increases the MSE decays. Similarly, with the inclusion of false positives, when some elements of our support estimate are incorrect, performance is intuitive: improvements are seen when there are more correct estimates than incorrect estimates, i.e. $\rho \geq 0.5$; for lesser values of ρ the support estimate consists mostly of incorrect elements and performance is worse than standard BIHT. Do notice that under this 4-set representation a k-sparse approximation is not necessarily returned. If $\tilde{T} \cap \dot{T} = \emptyset$, then in fact a $2k$-sparse solution is returned, certainly this will result in a less accurate solution. Perhaps this is the reason for the slower rate of error decay in comparison with standard BIHT, as seen in Figure 3.2.



(a) No False Positives

(b) Includes False Positives

Figure 3.2: This figure shows the performance of BIHT when the 4 set representation of a partial support estimate is used. In (a) we know our estimate to contain ρk correct elements, in (b) there are an additional $(1-\rho)k$ incorrect elements. Here MSE is averaged over 100 trials and plotted as a function of m, the number of measurements. The desired signal's length is set to $n = 256$ and our assumed sparsity level is $k = 8$.

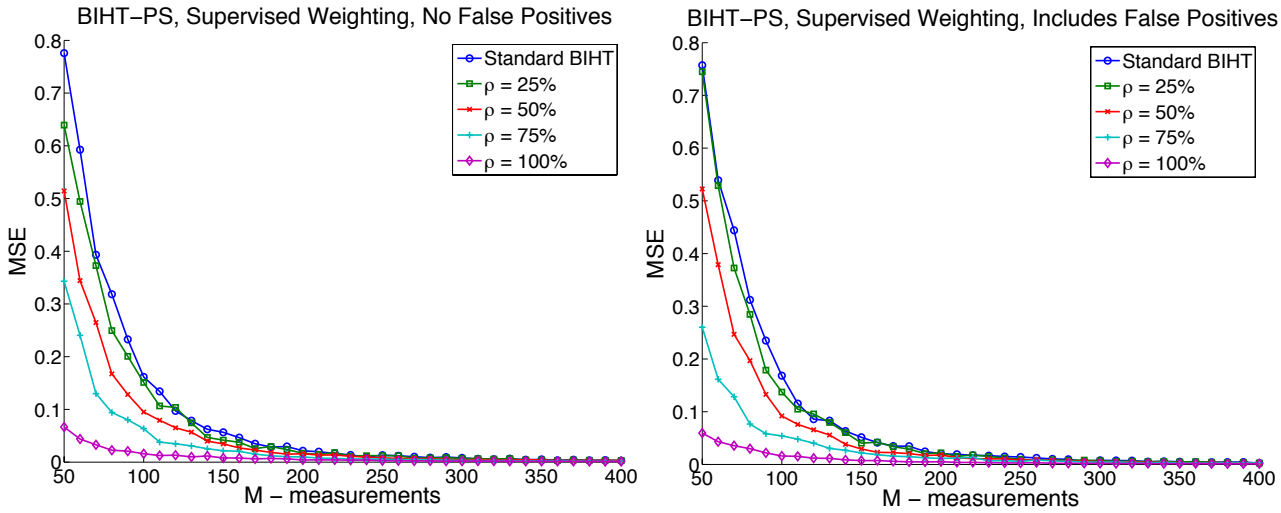
3.2.3 Supervised Weighting

As a means for incorporating a partial support estimate and returning a k-sparse solution, we use the weighting framework for traditional ℓ_1-minimization as presented in [MS14]. Again, say that we believe our estimate $\tilde{T}$ to be $(\rho \times 100)\%$ accurate. Let us create a weight vector w where

$$w_i = \begin{cases} 1 \text{ if } i \in \tilde{T} \\ 1 - \rho \text{ if } i \in \tilde{T}^C \end{cases} .$$

Then, at some iteration of BIHT, once we compute Γ let us multiply component-wise Γ and w to attain ψ. Now we locate the support of ψ, denoted $T_\psi := \mathit{support}_k(\psi)$, and hard threshold the elements of $\Gamma \in T_\psi^C$, i.e. set them equal to zero. This approach is very similar to BIHT when there is no prior support estimate, except that here Γ is instead pruned to retain the k greatest in magnitude elements of $\Gamma \odot w$. A more thorough break down of this procedure is presented in Algorithm 4. We see that this supervised weighting approach outperforms the 4-set formulation when false positives are and are not included. *Figure 3.3 shows that for every value of m, when some prior information exists, the weighting approach performs the same as or better than standard BIHT.*

This weighting framework for incorporating partial support information into BIHT (BIHT-PSW) performs well in the current context. However, we are assuming that the value for ρ is correct. This is a very bold assumption that, in practice, is not likely to hold. In the weighting step, the value of ρ determines how diminished the magnitude of an entry off the support estimate will be. If we are more confident in certain entries being non-zero, then other entries will be scaled by a constant closer to zero. Empirical results, as displayed in Figure 3.4, show that using the correct value of ρ is crucial to the incorporation of a partial support estimate.



(a) No False Positives (b) Includes False Positives

Figure 3.3: This figure shows the performance of BIHT when using a weight vector as means for incorporating the partial support estimate. In (a) we know our estimate to contain ρk correct elements, in (b) there are an additional $(1-\rho)k$ incorrect elements. Here MSE is averaged over 100 trials and plotted as a function of m the number of measurements. The desired signal's length is set to $n = 256$ and our assumed sparsity level is $k = 8$.

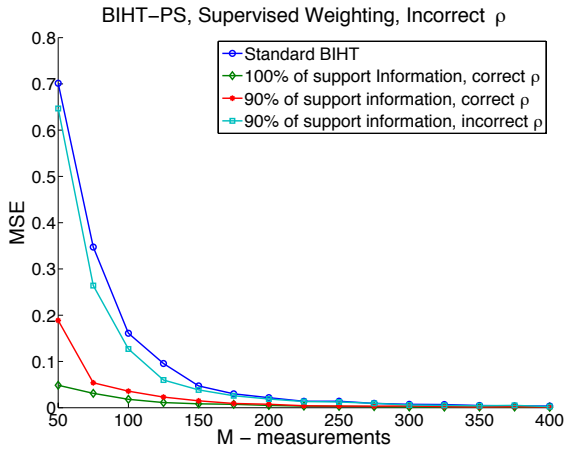


Figure 3.4: Notice the performance gap between the red (•) line and the cyan (□) line, for each the same partial support estimate with 90% accuracy was used. However, instead of setting $\rho = 0.9$ the cyan line has a value of $\rho = 0.1$. The signal length here is fixed to $n = 256$, the sparsity level $k = 8$, and the average MSE over 100 trials is plotted as a function of m, the number of measurements.

3.2.4 Unsupervised Re-weighting

The performance of BIHT-PSW is promising, it appears to be a valid incorporation of prior support information: when a portion of the support is known improvement in recovery is

Algorithm 4 Binary Iterative Hard Thresholding with Partial Support Estimate Weighting (BIHT-PSW). Given: measurement matrix Φ, one-bit measurements $\bar{y}$, assumed sparsity level k, support estimate $\tilde{T}$, accuracy of support estimate ρ

procedure BIHT-PSW(Φ, $\bar{y}$, k, $\tilde{T}$, ρ)

$w_i = \begin{cases} (1-\rho) \text{ if } i \in \tilde{T}^C \\ 1 \text{ if } i \in \tilde{T} \end{cases}$ ▷ Construct weight vector from $\tilde{T}$ and ρ

$x^0 = 0$ ▷ Initialize trivial approximation of x

$i = 0$ ▷ Let i denote our iteration count

$\tau = c$ ▷ Set gradient-descent step-size to desired constant c

repeat

$i = i + 1$ ▷ Increment iteration count

$\Gamma = x^i + \frac{\tau}{2}\Phi'(\bar{y} - sign(\Phi x^i))$ ▷ Compute and move approximation in the direction of the gradient

$\Omega = support_k(\Gamma \odot w)$ ▷ Locate support of Γ multiplied component-wise with w

$x_j^{i+1} = \begin{cases} \Gamma_j \text{ if } j \in \Omega \\ 0 \text{ if } j \in \Omega^C \end{cases}$ ▷ Update approximation

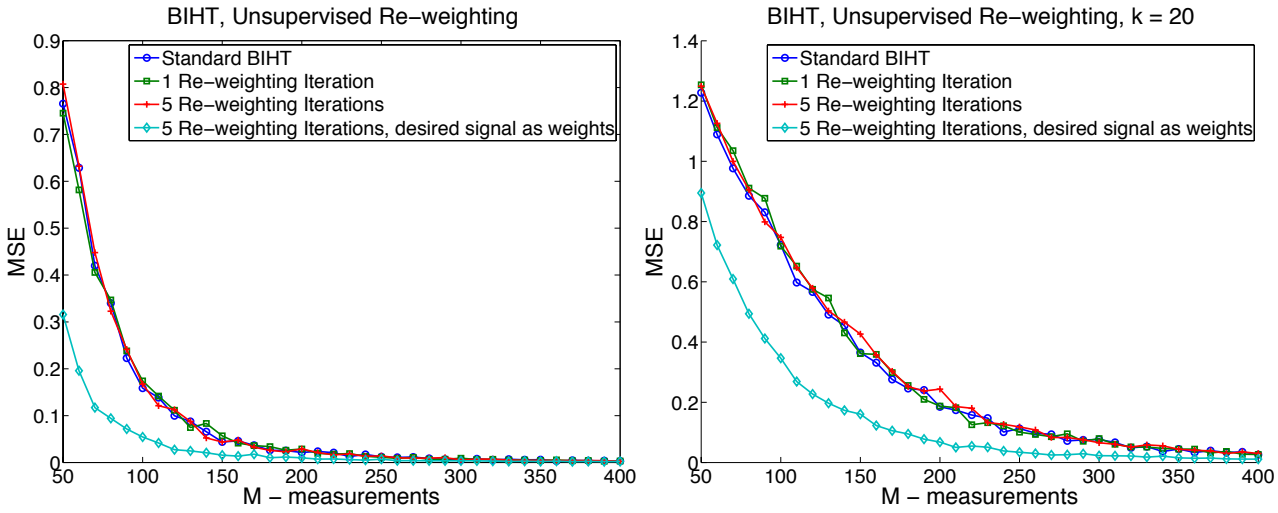
$x^i = x^{i+1}$ ▷ Continue algorithm with new approximation

until approximation is consistent with $\bar{y}$ or iteration count is sufficient

return $\frac{x^i}{\|x^i\|_2}$ ▷ Project approximation onto unit ℓ_2 sphere

end procedure

observed for all values of m. We hypothesized that this result could be leveraged even when no support estimate is available. If we take M measurements of x and use BIHT to attain some estimate $\tilde{x}$, then we may use the support of $\tilde{x}$, denoted $\ddot{T}$, as an estimate for T. Then, we could use $\ddot{T}$ to run BIHT-PSW, iteratively, and get a more accurate approximation. This is the general idea behind the BIHT Unsupervised Re-weighting (BIHT-URW) algorithm, presented in Algorithm 5. The performance of BIHT-URW is displayed in Figure 3.5, we observe no improvement from standard BIHT, perhaps due to the arbitrary selection of ρ. As a benchmark for optimal performance of BIHT-URW we create a weight vector from the desired signal x, and run the algorithm as normal. This results in a significant improvement in performance, although in comparison with the performance of BIHT-PSW when the full support is known the improvement is less than expected. Again, we believe this to be a result of an incorrect value of ρ.



(a) $k = 8$ (b) $k = 20$

Figure 3.5: This figure displays the performance of BIHT-URW for various re-weighting iterations. It seems that extra iterations do nothing to find a better approximation. Figure (a) has a sparsity level of $k = 8$, while figure (b) has a sparsity level of $k = 20$. In both figures, the cyan ($\Diamond$) line is a result of creating a weight vector out of the desired signal x. For both, $n = 256$ and the average MSE over 100 trials is plotted as a function of m, the number of measurements.

Algorithm 5 Binary Iterative Hard Thresholding with Unsupervised Re-weighting. Given: measurement matrix Φ, one-bit measurements $\bar{y}$, assumed sparsity level k, number of re-weighting iterations n

procedure BIHT-URW(Φ, $\bar{y}$, k, n)

$\ddot{T} = support_k(BIHT(\Phi, \bar{y}, k))$ ▷ Use BIHT to generate an initial support estimate

$W_i = \begin{cases} 0 < \lambda < 1 \text{ if } i \in \ddot{T}^C \\ 1 \text{ if } i \in \ddot{T} \end{cases}$ ▷ Construct weight vector from $\ddot{T}$

$i = 0$ ▷ Initialize iteration count

while $i < n$ **do**

$i = i + 1$ ▷ Increment iteration count

$\tilde{x} = \text{BIHT-PSW}(\Phi, \bar{y}, k, \ddot{T}, \lambda)$ ▷ Use BIHT-PSW with $\ddot{T}$ to attain approximation

$\ddot{T} = support_k(\tilde{x})$ ▷ Update support estimate

$W_i = \begin{cases} \lambda \text{ if } i \in \ddot{T}^C \\ 1 \text{ if } i \in \ddot{T} \end{cases}$ ▷ Update weight vector from updated $\ddot{T}$

end while

return $\frac{x}{\|x\|_2}$ ▷ Project approximation onto unit ℓ_2 sphere

end procedure

Chapter 4

Conclusion & Future Work

4.1 Conclusion

This paper began by exploring traditional Compressive Sensing algorithms for high-dimensional signal acquisition. We discussed the extension of these algorithms to incorporate prior information about the signal, given as a prior support estimate. A framework for signal recovery in the presence of extreme quantization was developed and one-bit Compressive Sensing algorithms were introduced. Finally, novel algorithms were developed and studied to incorporate support estimates into one-bit CS algorithms. Explicitly, the BIHT algorithm was extended in a number of ways to leverage such an estimate. First, we assumed the support estimate was fully accurate and experimented with hard and soft thresholding the other entries. Second, we accounted for the supports identified by BIHT, and those given in our prior estimate, through thresholding based on the 4-set framework. This 4-set framework did not return a k-sparse solution and was improved upon via BIHT-PSW, where the support estimate is converted into a weight vector that is then used in the pruning process. We attempted to extend the success of BIHT-PSW into the unsupervised setting through BIHT-URW, which aims to use consecutive iterations of BIHT and BIHT-PSW to generate support estimates and attain a more accurate approximation than standard BIHT. The lack of improvement

seen with BIHT-URW is believed to be a result of the sensitivity of BIHT-PSW to the choice of ρ, though further investigation is certainly required.

4.2 Future Work

The field of One-Bit Compressive Sensing is rapidly emerging, hence the need for a strong framework for incorporating partial support estimates. Continuing with the ideas explored in this paper, it is only natural that we further test these approaches empirically and strive to understand them theoretically. Specifically, rigorous guarantees for the recovery of x from one-bit measurements when there exists prior information have yet to be established. Techniques used for incorporating partial support information need to be further developed and investigated in conjunction with one-bit algorithms other than BIHT. We suspect that in the unsupervised setting, the performance of BIHT-URW may be improved by using heuristics to estimate and assign the value of ρ dynamically. Additionally, considering probabilistic incorporations of prior support estimates would certainly be worth while.

Bibliography

[BB08] P. T. Boufounos and R. G. Baraniuk. 1-bit compressive sensing. In *42nd Annual Conf. Info. Sciences and Systems (CISS)*, pages 16–21. IEEE, 2008.

[BD09] T. Blumensath and M. E. Davies. Iterative hard thresholding for compressed sensing. *Appl. Comput. Harmon. A.*, 27(3):265–274, 2009.

[BFN+14] R. Baraniuk, S. Foucart, D. Needell, Y. Plan, and M. Wootters. Exponential decay of reconstruction error from binary measurements of sparse signals. *arXiv preprint arXiv:1407.8246*, 2014.

[CRT06] E. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete fourier information. *IEEE T. Inform. Theory*, 52(2):489–509, Feb. 2006.

[CT05] E. J. Candès and T. Tao. Decoding by linear programming. *IEEE T. Inform. Theory*, 51:4203–4215, 2005.

[CT06] E. J. Candès and T. Tao. Near optimal signal recovery from random projections: Universal encoding strategies? *IEEE T. Inform. Theory*, 52(12):5406–5425, Dec. 2006.

[DNW12] M. Davenport, D. Needell, and M. B. Wakin. Signal space CoSaMP for sparse recovery with redundant dictionaries. *IEEE T. Inform. Theory*, 59(10):6820, 2012.

[Don04] D. L. Donoho. Compressed sensing, Oct. 2004. Unpublished manuscript.

[Don06] D. L. Donoho. For most large underdetermined systems of linear equations the minimal l_1-norm solution is also the sparsest solution. *Comm. Pure Appl. Math.*, 59(6):797–829, 2006.

[FMSY10] M. P. Friedlander, H. Mansour, R. Saab, and Ö. Yilmaz. Recovering compressively sampled signals using partial support information. *CoRR*, abs/1010.4612, 2010.

[GMY13] N. Ghadermarzy, H. Mansour, and Ö. Yilmaz. Non-convex compressed sensing using partial support information. *CoRR*, abs/1311.3773, 2013.

[JLBB11] L. Jacques, J. N. Laska, P. T. Boufounos, and R. G. Baraniuk. Robust 1-bit compressive sensing via binary stable embeddings of sparse vectors. *IEEE T. Inform. Theory*, 59(4):2082–2102, 2011.

[KSW14] K. Knudson, R. Saab, and R. Ward. One-bit compressive sensing with norm estimation. *arXiv preprint arXiv:1404.6853*, 2014.

[MS14] H. Mansour and R. Saab. Recovery analysis for weighted $\ell 1$-minimization using a null space property. *CoRR*, abs/1412.1565, 2014.

[Nee09] D. Needell. Topics in compressed sensing, 2009.

[NT09] D. Needell and J. Tropp. CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. *Appl. Comput. Harmon. A.*, 26(3):301–321, 2009.

[PV11] Y. Plan and R. Vershynin. One-bit compressed sensing by linear programming. *Commun. Pur. Appl. Math.*, 2011. To appear.

[Zha11] T. Zhang. Sparse recovery with orthogonal matching pursuit under RIP. *IEEE T. Inform.Theory,*, 57(9):6215–6221, 2011.

Appendix A

MATLAB Code

A.1 ℓ_1 Minimization

```
function MSE = L1(m)
%This function uses CVX to perform ell-1 minimization. As a function of m,
%the norm squared error is returned

% dimension parameters
n = 256;
k = 8;

% generate our measurement matrix
A = randn(m,n);

% generate our signal: x
x = zeros(n,1);
tmp = randperm(n);
x(tmp(1:k)) = randn(k,1);

% gather our measurements
Y = A*x;

% use CVX to perform the minimization
    cvx_begin
variable approx(n)
minimize(norm(approx,1))
subject to
norm(Y - A*approx,2) <= 10^(-10);
    cvx_end

MSE = norm(abs(approx - x))^2;
approx

end
```

A.2 CoSaMP

```
function MSE = coSamp(m)
% This function recovers a high dimensional signal x using the CoSaMP
% algorithm

% recovers the signal x from its measurements y using CoSaMP algorithm
n = 256;
s = 8;

% Construct measurement matrix A and desired s-sparse signal x
A = randn(m,n);
x = zeros(n,1);
tmp = randperm(n);
x(tmp(1:s)) = randn(s,1);

% initial measurements of x
y = A * x;

% initialize trivial approximation of x
a = zeros(n,1);

% Initially, set the samples to be updated to be our given measurements
v = y;

for i = 1:250

    % update proxy signal
    p = A' * v;

    % locate and restrict our proxy to its 2s largest elements
    [~, it] = sort(abs(p),'descend');
    Omega = it(1:2*s);

    % determine support of a, i.e. the location of the s largest elements of a
    [~, it] = sort(abs(a), 'descend');
    suppA = it(1:s);

    % merge supports of proxy signal and current approximation
    T = union(Omega, suppA);

    % restrict measurement matrix to columns indicated by our set of supports
    AT = A(:,T);

    % now we find the pseudo inverse of these columns (existence provided by
    % RIP) and apply this to the initial measurement
    temp = pinv(AT) * y;

    % construct our next approximation
    b = zeros(n,1);
    b(T) = temp;

    % now we prune b to only its greatest s elements and store it as our
    % current approximation
    [~, it] = sort(abs(b), 'descend');
    b(it(s+1:length(b))) = 0;
```

```
    % update approximation and current measurements
    a = b;
    v = y - A*a;

end

% return mean squared error
MSE = norm(abs(a-x))^2;
end
```

A.3 Signal-Space CoSaMP

```
function MSE = sscoSamp(m)
% This function supports the SSCoSaMP: signal space compressive sampling
% matching pursuit algorithm, where CoSaMP is used as the projection
% approximation

% define sparsity level and length of signal
s = 8;
n = 256;

% generate our dictionary
d = 4*n;
D = ifft(eye(d))*d/sqrt(n);
D = D(1:n,:);

% generate our measurement matrix
A = randn(m,n);

% generate our s sparse coefficient vector alpha
alpha = zeros(d,1);
tmp = randi(n-s);
alpha(tmp:(tmp+s-1)) = randn(s,1);

% generate x = D * alpha
X = D*alpha;

% compute noiseless measurements of the form y = Ax, where x = D*alpha
y = A*X;

% initialize trivial residual, signal approximation, and support
r = y;
x = zeros(n,1);
t = [];

for i=1:50

    % create proxy
    v = A'*r;

    % first identification step
```

```
    temp = coSamp(v, D, 2*s);
    [~,it] = sort(abs(temp),'descend');
    omega = it(1:2*s);

    % merging
    T = union(omega, t);

    % least squares step
    DT = D(:,T);
    ex = DT*pinv(A*DT)*y;

    % second coSamp step
    temp = coSamp(ex, D, s);
    [~, it] = sort(abs(temp),'descend');
    t = it(1:s);

    % update approximation
    Dt = D(:,t);
    x = (Dt*pinv(Dt))*ex;

    % update residual
    r = y - A*x;

end

% return absolute error
MSE = norm(abs(X-x))^2;
end
```

A.4 One-Bit ℓ_1 Minimization

```
function MSE = onebit_L1(m)
%This function uses CVX to perform ell-1 minimization in the one-bit
%setting. As a function of m, the mean squared error is returned

% dimension parameters
n = 256;
k = 8;

% generate our measurment matrix
A = randn(m,n);

% generate our normalized signal: x
x = zeros(n,1);
tmp = randperm(n);
x(tmp(1:k)) = randn(k,1);
x = x/norm(x);

% gather our measurements and convert them to one-bit representation
y = sign(A*x);

% use CVX to perform the minimization
```

```
signs = sign(y);
cvx_begin quiet
variable approx(n)
minimize(norm(approx,1))
subject to
A*approx.*signs >= 0
signs'*A*approx >= m
cvx_end

% normalize approximation and return MSE
approx = approx/norm(approx);
MSE = norm(abs(approx - x))^2;
end
```

A.5 BIHT

```
function MSE = biht(m)
% This function performs one-bit compressive sensing via the Binary
% Iterative Hard Thresholding algorithm

% dimension parameters
n = 256;
k = 8;

% generate our measurement matrix
A = randn(m,n);

% generate our normalized signal: x
x = zeros(n,1);
tmp = randperm(n);
x(tmp(1:k)) = randn(k,1);
x = x/norm(x);

% gather our measurements, convert to one-bit representation
y = A*x;
y = sign(y);

% keep iteration count, initialize trivial approx., set gradient-descent step-size
i = 0;
x0 = zeros(n,1);
tau = .001;
while (i < 1000)
    i = i + 1;
    temp = x0 + (tau/2)*A'*(y - sign(A*x0));
    x1 = prune(temp,k);
    if (norm(abs(x1-x0))) < 10^(-16)
        break
    end
    x0 = x1;
end

% normalize approximation and return MSE
```

```
x0 = x0/norm(x0);
MSE = norm(x0 - x)^2;
end

function xk = prune(x, k)
% This function takes a signal x and returns a vector xk with the k largest
% in magnitude entries of x not set to zero

[~, indices] = sort(abs(x),'descend');
largest = indices(1:k);
xk = zeros(length(x),1);
xk(largest) = x(largest);

end
```

A.6 BIHT-PS, Full Support Known, Hard and Soft Thresholding

```
function mse = biht_fullSupportInfo(m)
% This function perfomrs one-bit compressive sensing via the BIHT algorithm,
% this considers the instance when the full support is known, during the
% pruning step entries that are not in the support are scaled do4n by a
% constant factor (.5, .25,...)

% dimension parameters
n = 256;
k = 8;
% m = 100; %optional

% generate our measurment matrix
A = randn(m,n);

% generate our normalized signal: x
x = zeros(n,1);
tmp = randperm(n);
support = tmp(1:k);
x(support) = randn(k,1);
x = x/norm(x);

% gather our measurements, convert to one-bit representation
y = A*x;
y = sign(y);

i = 0;
x0 = zeros(n,1);
tau = .001;
while (i < 1000)
    i = i + 1;
    temp = x0 + (tau/2)*A'*(y - sign(A*x0));
    x1 = prune_priorInfo2(temp, support);
    if (norm(abs(x1-x0))) < 10^(-16)
```

```
        break
    end
    x0 = x1;
end
x0 = x0/norm(x0);
mse = norm(x0 - x)^2;
end

function  xk = prune(x, support)
% This function sets all entries of x that are not in the support to c times
% their initial magnitude. Note that if c = 0, then this is hard-thresholding
% In this instance the support information is perfectly accurate.
c = 0.50;
xk = zeros(length(x),1);
for i = 1:length(x)
    if ~any(i == support)
        xk(i) = c * x(i);
    else
        xk(i) = x(i);
    end
end

end
```

A.7 BIHT-PS, 4-Set Representation

```
function mse = biht_4Set(m)
% This function performs one-bit compressive sensing via the Binary
% Iterative Hard Thresholding algorithm when a patial support estimate
% is incorporated via the 4set representation

% dimension parameters
n = 256;
k = 8;

% generate our measurment matrix
A = randn(m,n);

% generate our normalized signal: x
x = zeros(n,1);
tmp = randperm(n);
Support = tmp(1:k);
x(Support) = randn(k,1);
x = x/norm(x);

% gather our measurements, convert to one-bit representation
y = A*x;
y = sign(y);

% define partial support estimate:

% add correct estimates
```

```
rho = 0.9;
numElem = floor(k*rho);
tmp = randperm(k);
suppEstimate = Support(tmp(1:numElem));

% add incorrect estimates
suppComp = setdiff(1:n, Support);
numElem = k - numElem;
tmp = randperm(length(suppComp));
suppEstimate = [suppEstimate suppComp(tmp(1:numElem))];

i = 0;
x0 = zeros(n,1);
tau = .001;
while (i < 1000)
    i = i + 1;
    residual = x0 + (tau/2)*A'*(y - sign(A*x0));
    x1 = prune(residual, suppEstimate, k, rho);
    if (norm(abs(x1-x0))) < 10^(-16)
        break
    end
    x0 = x1;
end
x0 = x0/norm(x0);
mse = norm(x0 - x)^2;
end

function  newX = prune(x, S, k, rho)
% This function handles the pruning step for BIHT-PS where a partial
% support estimate is incorporated via the 4 Set Representation

% Sort residual to determine k largest in magnitude entries, this set of supports is T
[~, indices] = sort(abs(x),'descend');
T = indices(1:k);

% handles the intersection T^C and S^C, multiplies by zero
newX = zeros(length(x),1);

% handles the intersection of T and S, multiplies by 1
TnS = intersect(T,S);
newX(TnS) = x(TnS);

% compute Sc and Tc
Sc = setdiff(1:length(x), S);
Tc = setdiff(1:length(x), x);

% handles T intersect Sc, multiplies by 1 - rho
TnSc = intersect(T,Sc);
newX(TnSc) = x(TnSc).*(1-rho);

% handles Tc intersect S, multiplies by one
SnTc = intersect(S, Tc);
newX(SnTc) = x(SnTc).*rho;
```

```
end
```

A.8 BIHT-PS, Supervised Weighting

```
function mse = bihtPS_Weighting(m)
% This function performs one-bit compressive sensing via the Binary
% Iterative Hard Thresholding algorithm, a prior support estimate is
% incorporated as a weight vector

% dimension parameters
n = 256;
k = 8;

% generate our measurment matrix
A = randn(m,n);

% generate our normalized signal: x
x = zeros(n,1);
tmp = randperm(n);
Support = tmp(1:k);
x(Support) = randn(k,1);
x = x/norm(x);

% gather our measurements, convert to one-bit representation
y = A*x;
y = sign(y);

% define partial support estimate:
% (i) include correct estimates
rho = 0.5;
W = repmat(1-rho, n,1);
numElem = floor(k*rho);
tmp = randperm(k);
correctEstimates = Support(tmp(1:numElem));
W(correctEstimates) = ones(numElem,1);

% (ii) include incorrect estimates
suppComp = setdiff(1:n,Support);
numElem = k - numElem;
tmp = randperm(length(suppComp));
wrongEstimates = suppComp(tmp(1:numElem));
W(wrongEstimates) = ones(numElem,1);

% BIHT:
i = 0;
x0 = zeros(n,1);
tau = .001;
while (i < 1000)
    i = i + 1;
    residual = x0 + (tau/2)*A'*(y - sign(A*x0));
    x1 = prune(residual, W, k);
    if (norm(abs(x1-x0))) < 10^(-16)
```

```
        break
    end
    x0 = x1;
end

% normalize approximation and return MSE
x0 = x0/norm(x0);
mse = norm(x0 - x)^2;
end

function  newX = prune(x, W, k)
% This function handles the prune step of BIHT-PS with Weightings. The
% vector x is multiplied componentwise with W, the resulting greatest k in
% magnitude entries are then not set to zero in the initial vector x

X = W.*x;
[~, indices] = sort(abs(X), 'descend');
largest = indices(1:k);
newX = zeros(length(x),1);
newX(largest) = x(largest);

end
```

A.9 BIHT, Unsupervised Re-weighting

```
function mse = BIHT_URW(m)
% This function perfoms one-bit compressive sensing via the Binary
% Iterative Hard Thresholding Unsupervised Re-weighting algorithm.
% Standard BIHT is used to attain a support estimate which is then
% used in conjunction with BIHT-PSW

% dimension parameters
n = 256;
k = 8;

% generate our measurment matrix
A = randn(m,n);

% generate our normalized signal: x
x = zeros(n,1);
tmp = randperm(n);
Support = tmp(1:k);
x(Support) = randn(k,1);
x = x/norm(x);

% gather our measurements, convert to one-bit representation
y = A*x;
y = sign(y);

% let the number of iterations of the reweighting process be denoted by j
j = 5;
if j > 0
```

```
    % Initial run of biht, used to generate first set of partial support weights
    [~, firstApprox] = biht(A, y, k, x);
    W = createWeights(firstApprox);

    for q = 1:j
        x0 = Weighted_BIHT(A, y, k, W, x);
        W = createWeights(x0);
    end
else
    [~, x0] = biht(A, y, k, x);
end

% normalize approximation and return MSE
x0 = x0/norm(x0);
mse = norm(x0 - x)^2;
end

% create weight vector out of a given signal
function W = createWeights(signal)

n = length(signal);
rho = 0.5;
W = repmat(1 - rho, n, 1);
W(find(signal ~= 0)) = 1;
end

% Weighted BIHT
function x0 = Weighted_BIHT(A, y, k, W)

n = size(A,2);
x0 = zeros(n,1);
i = 0;
tau = .001;
while (i < 1000)
    i = i + 1;
    residual = x0 + (tau/2)*A'*(y - sign(A*x0));
    x1 = Weighted_BIHT_prune(residual, W, k);
    if (norm(abs(x1-x0))) < 10^(-16)
        break
    end
    x0 = x1;
end
x0 = x0/norm(x0);
end

% Weighted BIHT pruning
function  newX = Weighted_BIHT_prune(x, W, k)

X = W.*x;
[~, indices] = sort(abs(X), 'descend');
largest = indices(1:k);
newX = zeros(length(X),1);
newX(largest) = x(largest);
```

```
end

% Standard BIHT
function x0 = biht(A, y, k)
n = size(A, 2);
x0 = zeros(n,1);
i = 0;
tau = .001;
while (i < 1000)
    i = i + 1;
    residual = x0 + (tau/2)*A'*(y - sign(A*x0));
    x1 = prune(residual,k);
    if (norm(abs(x1-x0))) < 10^(-16)
        break
    end
    x0 = x1;
end
x0 = x0/norm(x0);
end

% Standard BIHT pruning
function  xk = prune(x, k)

[~, indices] = sort(abs(x),'descend');
largest = indices(1:k);
xk = zeros(length(x),1);
xk(largest) = x(largest);
end
```