AARHUS
UNIVERSITET
SCIENCE AND TECHNOLOGY

Master's Thesis

Sparse Approximations and Compressive Sensing in Centralized and Distributed Systems

Kasper C. Klausen
Master of Science in Computer Engineering
Aarhus University
Department of Engineering
4th of June 2019

Title page

Thesis title	Sparse Approximations and Compressive Sensing in Centralized and Distributed Systems
Thesis ECTS	30
University	Aarhus University
Department	Department of Engineering
Section	Section of Electrical and Computer Engineering
Master's program	Computer Engineering
Author	Kasper C. Klausen
Student number	201303660
Date of submission	4^{th} of June 2019
Supervisor	Christian Fischer Pedersen

Preface

The present master's thesis marks the end of the master education in Computer Engineering, at the faculty of Science and Technology at Aarhus University. The project was a study on its own, in investigating matching pursuit algorithms and extensions into the field of wireless sensor networks.

During the project time, three presentations were held for students taking Research and Development courses in Compressive Sensing. These presentation covered; compressive sensing, matching pursuit algorithms and distributed compressive sensing.

I would like to thank my supervisor Christian Fischer Pedersen for introducing me to the fascinating area of Compressed Sensing, as well as supervising me throughout the project

Abstract

The present thesis provides a summary of Matching Pursuit reconstruction algorithms. For comparison, the algorithms have been designed in a uniform manner and analyzed with respect to time and space complexity. New implementation of the algorithms is provided, which have been used in numerical experiments. Based on these experiments it was found that the algorithmic approache of Subspace Pursuit, which initially tries to estimate the complete support set of the desired signal followed by refining iterations, has slightly superior performance compared to iteratively estimating small parts of the support set. This approach provides relatively fast convergence, allowing for a low upper bound for iterations, as well as the use of a stopping criteria considering decreasing residual.
Additionally, a selection of methods and algorithms for using Compressed Sensing in distributed systems have been covered. It was found that using compressed sensing in wireless sensor networks allowed for load balancing comparable to that of data aggregation. Moreover Distributed Compressed Sensing enabled distributed encoding comparable to Distributed Source Coding. In general, it was found that using compressed sensing in wireless sensor networks fulfills dominating requirements in this area, which aim at energy efficient sensors. The reason for this was found to be that compressed sensing, allows for sampling the signal in a compressed form, thus avoiding the computational overhead associated with other compression approaches. In combination with this relative simple combined sampling and compression, compressed sensing moves computational complexity to the reconstruction side, thus giving the opportunity to keep sensor notes simple.

Resume

Dette speciale give en summering a Matching Pursuit rekonstrurerings algoritmer. For sammenligning, er algoritmerne designed på en ensartet måde, såvel som analyseret i forhold til tids og plads kompleksitet. Nye implementering af algoritmerne er skrevet, og disse blevet brugt i numeriske eksperimenter. Baseret på disse eksperimenter, blev det observeret at Subspace Pursuit algoritmen, som starter ud med at prøve at estimere hele support sættet af det komprimerede signal efterfulgt af iterativ fintuning, giver en anelse bedre performance end iterativt at estimere mindre dele af support sættet. Subspace Pursuit giver relativ hurtig konvergerings rater, hvilket gøre det muligt at implementere algoritmen med lav øvre grænse for iterationer, såvel som brugen af et stop kriterie, baseret på faldende residual.
Desuden er et udvalg af metoder og algoritmer som tillad brugen af Compressed Sensing i distribuerede systemer blev gennemgået. Det blev konkluderet af brugen af Compressed Sensing i trådløse sensor netværker kunne introducere load balancing sammenlignelig med data aggregation. Herudover kan brugen af distribueret Compressed Sensing give encoding sammenlignelig med Distributed Source Coding. Generelt blev det observeret at brugen af compressed sensing i trådløse sensor netværker, tilfredsstiller en rækker krav om simpelt sensor design. Grunden til dette er at Compressed Sensing tillader et signal af blive samplet i komprimeret form, og derved undgå at skulle bruger energi på anden form for komprimering. Udover den relativ simple kombinering af sampling og komprimering, flytter Compressed Sensing dele af den overordnede kompleksitet til rekonstruktionen af signaler, hvilket også bridrager til at kunne holde selve sensor enhederne simple.

CONTENTS

1
INTRODUCTION

An ever-increasing amount of data acquisition and processing puts increasing requirements on the involved systems. Data handling systems often have to deal with large quantities of samples, as a result of high Nyquist rates [1] in many applications. In order to sufficiently transmit and store these growing amounts of data, compression has become a necessity [2]. For this compression, transform coding is widely used in relation to data ranging from environmental data to image and audio to medical scanners [3, 4, 5]. A common aspect of these kinds of data is transform sparsity. Transform sparsity implies that there exists a transform domain, in which the signal can be sparsely represented. Though most natural signals are not directly sparse in any transform domain, they are still compressible, as they only have a few large significant coefficients in a transform domain. The rest of the coefficients can often be discarded, without significant signal quality degradation, which is known as lossy compression [4]. However, this sample then compress approach, is rather inefficient, as parts of the data are being discarded right after sampling. Moreover, for applications like medical scanners and radars, increasing the sampling rate, to obtain the Nyquist rate, can be rather expensive [2]. Compressed sensing is an emerging field within signal processing, which deals with the inefficiency of the sample then compress approach, by combining the sampling and compression. This is done by applying a measurement matrix, which performs a dimension reduction during sampling, which result in a smaller quantity of linear measurements, thereby reducing the required sampling rate. This is possible since many natural signals contain relatively limited information in relation to their ambient dimension [2, 6, 4].

The application of compressed sensing raises two problems that must be addressed. First, a measurement matrix must be designed, such that the information in the signal is not damaged by the dimensionality reduction. Second, a reconstruction algorithm must be designed to recover the signal from the resulting underdetermined system [2, 6]. The measurement matrix must satisfy the restricted isometry property, which implies an isometric mapping of compressible signals [2, 6]. Dealing with solutions of underdetermined systems, when interested in compressible signals, is often a convex optimization problem, that reduces to a linear program known as Basis Pursuit [2, 7, 8]. However, while the convex optimization is powerful for computing sparse representations, they are associated with heavy computational burdens [2, 6]. To overcome this computational burden, while still being able to reconstruct signals, a variety of iterative algorithms have been proposed. These algorithms are able to estimate the signal in a greedy fashion by iteratively estimating the sparse support set [9, 10, 11, 12, 13, 14].

An area that stands to highly benefit from compressed sensing is wireless sensor networks. With more and more sensor motes being deployed to monitor, everything from the environment and smart buildings, to people's health and logistics. With these large quantities of sensors being deployed, there is a desire to keep their cost low, which, however, significantly decreases their battery and computational power. A common task in sensor networks is to wirelessly propagate information, however, the wireless communication requires large quantities of the available limited battery power. Therefore, research has been focusing on decreasing the total amount of communication in wireless systems, without compromising the intent of the system. Strategies

used for this purpose have covered aggregation of data, network coding and distributed source coding, where common goals are data compression and load balancing [3].
Another common task for sensor networks is to monitor and detect events, which can require significant amounts of information from multiple sensors [3]. The relative simplicity of compressed sensing on the acquisition side, makes compressed sensing a suitable fit for these tasks. Additionally, when applying compressed sensing, the workload is shifted to the reconstruction, which in wireless sensor networks often is performed on a more powerful device. Moreover, compressed sensing is able to introduce load balancing, as the sensors are able to combine their measurements, which limits the workload of bottleneck notes close to the sink, as these then have to propagate less information [6].

1.1 Problem Statement

The focus of the present master's thesis is to design, implement, test, compare, and document a selection of Matching Pursuit inspired algorithms for sparse approximations and compressive sensing with applications to both centralized and distributed systems. The algorithms must be optimized with regard to time and space complexities. Moreover, the algorithms must be proven functional via concrete experiments. To quantify the quality of the algorithms, comparative evaluations should be carried out via objective metrics; in the comparisons, time and space complexity aspects should also be considered. Further, the algorithms should be assessed against the state of the art.

1.2 Organisation

The rest of this thesis is structured as follows:

In chapter 2, required background knowledge is covered, this includes prerequisites for compressed sensing as well as other forms of compression for comparison.
In chapter 3, a selection of Matching Pursuit algorithms will be summarized and analyzed, with the purpose of understanding their working and being able to implement them. Moreover, the analysis will be performed for the worst-case scenarios of arbitrary signals, in order to clarify requirements for stable implementations. An extension to handle multiple correlate signals will be given, to ease the transaction to distributed setups.
In chapter 4, different ways of utilizing compressed sensing in distributed systems will be covered, in order to show the power of compressed sensing.
In chapter 5, numerical experiments will be performed. The purpose behind these experiments is to; implement more effective algorithms, give an example of the use of matching pursuit algorithms with a given basis. As well as show and verify the workings of Distributed Compressed Sensing. Additionally, the results for each algorithm will be discussed.
In chapter 6, the overall result of the project will be drawn together.
In chapter 7, the work will be concluded.
In chapters 8, 9 and 10, a list of the algorithms will be presented as well as an explanation of the notations used thought out the thessi, and a list of references used will be given.

2
BACKGROUND

2.1 Sparse Signals

Signal processing is a field dealing with acquisition, processing, and extraction of information from different kinds of signals. In order to work with signals, much of signal processing is based on modeling signals as vectors, in an appropriate vector space. While this may be reasonable, in a lot of cases not all possible vectors model a valid signal. Based on this a model of the signal of interest is often useful to apply a priori knowledge, to efficiently handle the signal. Such models are known as a basis or dictionary, well known examples are Fourier and Wavelet [6].
In a wide variety of settings, valid signals can often be approximated as a linear combination of elements from a known basis. If this approximation is exact and only consists of a few elements of the basis, the signal is said to be sparse, in that basis or transform domain. This is more formally stated as; a signal $\mathbf{x}$ is s-sparse, when its represention in a basis has at most s nonzero coefficients: $||\mathbf{x}||_0 \leq s$. Though many real-world signals are not completely sparse, often their representation in a certain basis has only a few large and many small coefficients. Such signals are known as compressible signals, which can be sufficiently approximated by the few elements of the basis, associated with the few large coefficients [6, 2].
Moreover, a basis can be formed as an orthonormal quadratic matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$. For such a matrix the following holds: $\mathbf{A}^T\mathbf{A} = \mathbf{I}$, which implies that for two vectors $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{z} \in \mathbb{R}^n$ the following holds

$$\mathbf{z} = \mathbf{A}\mathbf{x} \tag{2.1.1}$$

and

$$\mathbf{x} = \mathbf{A}^T\mathbf{z} \tag{2.1.2}$$

which allows for relative simple transition between the two domains [7].
Sparsity has long been utilized in signal processing for compression using transform coding [6].

2.2 Compression

The Shannon-Nyquist sampling theorem states that in order to exactly reconstruct a signal from its samples, the samples must be acquired at a rate higher than twice the highest present frequency, known as the Nyquist rate [1, 6]. In many application, it can be either very expensive or time consuming to comply to the Nyquist rate, or it can result in a quantity of samples making compression a necessity prior to storage or transmission [2, 6, 5]. Since many natural signals are sparse, or at least compressible, sparse signal models suggest that many natural high-dimensional signals contain relatively limited information in relation to their ambient dimensions. Based on this a widely used method of compression is known as transform cording [6].

2.2.1 Transform Coding

Transform coding is a subcategory of data compression that utilized the compressibility of signalsin a well known basis [4]. The overall process that transform coding and decoding follows is illustrated in figure 2.2.1.

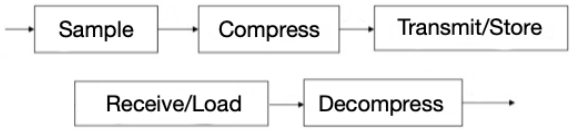


Figure 2.2.1. Original sampling and compression. Initially, the entire signal is sampled, with regard to the Nyquist rate. Then compression is performed using a selected approach, before the compressed signal is transmitted or stored. When the compressed signal is being received or loaded decompression is performed to reconstruct the signal.

The process of transform coding requires the signal to be fully sampled, with regard to the Nyquist rate. Then the signal is transformed into a well known basis, as explained in section 2.1, in which it is sparse or compressible. The significant coefficients of this transformation are then used as a compressed representation of the signal, which can then be decompressed. More specifically transform coding consists of three steps: Transform, quantization, and encoding. There exists a variety of different strategies to this, of which JPEG is one of the best known. JPEG uses the discrete cosine transform for transformation, a uniform midtread quantization to quantize, and prefix coding for the encoding [4].
More generally transform coding can be subdivided into two categories; lossless and lossy compression. Lossy compression is more effective than the lossless counterpart, at the cost of reconstruction quality, by discarding the small transform coefficients. Though since lossy compression is able to obtain significant data size reduction before significant information loss occurs, these are widely used [4].

2.3 Compressed Sensing

In many applications, it can be either very expensive or time consuming to comply to the Nyquist rate, or it can result in a number of samples making compression a necessity prior to storage or transmission [2, 6, 5]. However, by applying techniques from Compressed Sensing, sparse and compressible signals can be sampled at a lower rate. Consider the signal vector $\mathbf{x} \in \mathbb{R}^n$. Any such signal can be represented as a linear combination of columns, taken from an orthogonal basis $\mathbf{\Psi} \in \mathbb{R}^{n \times p}$. This takes the following form

$$\mathbf{x} = \mathbf{\Psi}\mathbf{z} \tag{2.3.1}$$

with $\mathbf{z} \in \mathbb{R}^p$, being an equal representation of $\mathbf{x}$ in the basis $\mathbf{\Psi}$. This implies that $\mathbf{x}$ and $\mathbf{z}$ are equivalent representations of the same signal, though in different domains. In order for Compressed Sensing to be applicable there must exist a known pre-determined basis in which the signal is either sparse, i.e. having only a few nonzero coefficients, or compressible, i.e. having only a few large coefficients and many small coefficients [2].

When applying Compressed Sensing the signal $\mathbf{x}$ is measured by sampling it in relation to a measurement matrix $\mathbf{\Phi} \in \mathbb{R}^{m \times n}$ with $m < n$, resulting in a compressed signal representation $\mathbf{y} \in \mathbb{R}^m$. This is equivalent to

$$\mathbf{y} = \mathbf{\Phi}\mathbf{x} \tag{2.3.2}$$

Combining eq. (2.3.2) with eq. (2.3.1) give

$$\mathbf{y} = \mathbf{\Phi x} = \mathbf{\Phi \Psi z} \tag{2.3.3}$$

This equation is illustrated in figure 2.3.1. From here the task is to reconstruct $\mathbf{x}$ (or $\mathbf{z}$) from $\mathbf{y}$. Though due to the dimensions of $\mathbf{\Phi}$ eq. (2.3.3) is under-determined, making this an ill-posed problem. However, due to the sparsity requirement stated earlier and some restriction that must be imposed on $\mathbf{\Phi}$, this is feasible. The restriction on $\mathbf{\Phi}$ will be covered in section 2.3.1.
For simplicity, when dealing with reconstruction algorithms, $\mathbf{x}$ is often assumed to be sparse in its own domain. This introduces simplicity as it allows for focusing on the $\mathbf{y} = \mathbf{\Phi x}$ part of eq. (2.3.3). This simplification is reliable, assuming the restriction on the measurement matrix are complied.

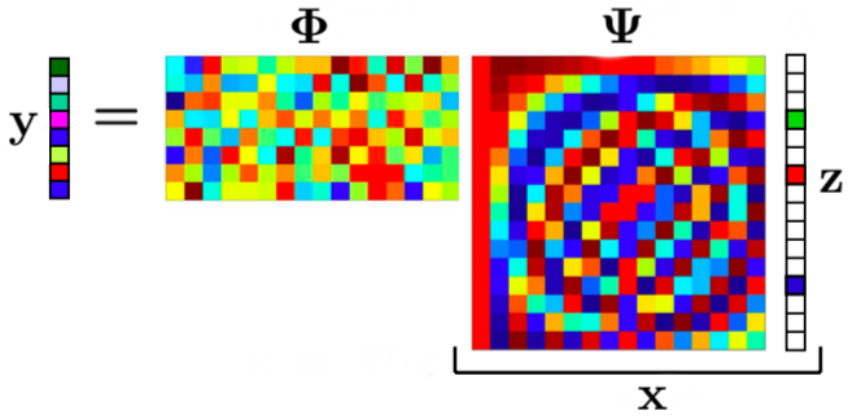


Figure 2.3.1. Relation between vectors and matrices. With $\mathbf{y} \in \mathbb{R}^m$ the measurement vector, $\mathbf{\Phi} \in \mathbb{R}^{m \times n}$ the measurement matrix, $\mathbf{\Psi} \in \mathbb{R}^{n \times n}$ a known basis, $\mathbf{z} \in \mathbb{R}^n$ a sparse representation of $\mathbf{x}$ in the $\mathbf{\Psi}$ domain, and $\mathbf{x} \in \mathbb{R}^n$ being the signal of interest. Based on figure 1 in [2]

The process of sampling while applying a measurement matrix, can be seen as combining the sampling and compression steps of figure 2.2.1. This combination is the reason behind the name Compressed Sensing. Compressed Sensing changes figure 2.2.1 into figure 2.3.2. The Compressed Sensing step is often much more efficient than the sample then compress approach, this being the main advantage of Compressed Sensing. Though this simplifies the acquistion aspect, it increases the complexity of the reconstruction, for which reconstruction algorithms are required. However, this tradeoff is an advantages in a wide range of application [6, 2, 5].

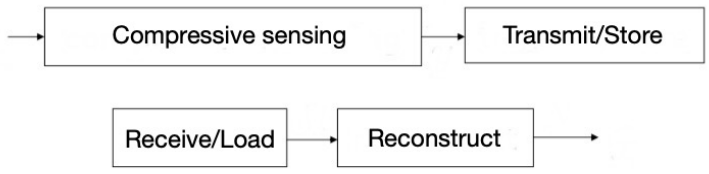


Figure 2.3.2. Compressive sensing, in relation to standard way figure 2.2.1. Initially a compressed representation of the signal is being sampled using compressed sensing. Then the compressed representation of the signal is being transmitted or stored. When the compressed signal is being received or loaded a reconstruction algorithm is used to recreate the originl signal.

2.3.1 Measurement Matrices

In order to be able to reconstruct all s sparse signals from their measurement $\mathbf{y}$, then $\mathbf{\Phi}$ must not map two distinct s sparse signals $\mathbf{x}$ and $\mathbf{x}'$ to the same $\mathbf{y}$, i.e. $\mathbf{\Phi x} \neq \mathbf{\Phi x}'$, otherwise, it would not be possible to distinguish them based on the measurements. From the fact that if $\mathbf{\Phi x} = \mathbf{\Phi x}'$ it follows that $\mathbf{\Phi}(\mathbf{x} - \mathbf{x}') = \mathbf{0}$, with vector $\mathbf{x} - \mathbf{x}'$ having a sparsity of at most $2s$. Thus $\mathbf{\Phi}$ uniquely maps all s sparse signals if and only if the null space of $\mathbf{\Phi}$ contains no vector with sparsity of $2s$. Meaning that any collection of $2s$ columns taken from $\mathbf{\Phi}$ must be linearly independent [6]. This can also be seen as a parallel to the Nyquist rate, as it also gives anti-aliasing.
The above only states when reconstruction is possible for exactly sparse signals. Though when signals are compressible, or approximately sparse, more restrictive conditions is required, related to the null space of $\mathbf{\Phi}$. This is called the Null Space Property, and it stats that the vectors in the null space of $\mathbf{\Phi}$ should not be concentrated on a small subset of indices[6].

2.3.1.1 Restricted Isometry Property (RIP)

When the measurements are subject to noise, an even stronger condition is needed, than the Null Space Property. This is called the Restricted Isometry Property (RIP), and is defined as [15, 6]

Definition 1 *A matrix $\mathbf{\Phi}$ satisfies the Restricted Isometry Property (RIP) of order s if there exists a $\delta_s \in (0, 1)$ such that*

$$1 - \delta_s \leq \frac{||\mathbf{\Phi x}||_2^2}{||\mathbf{x}||_2^2} \leq 1 + \delta_s \tag{2.3.4}$$

This can be interpreted as, if a measurement matrix $\mathbf{\Phi}$ satisfy the RIP of order $2s$, then $\mathbf{\Phi}$ approximately preserves the distance between any pair of s sparse vectors, thus making it robust to noise, in other words the dimensionality reduction of

$$\mathbf{\Phi}$$

of s sparse signals, must be an isometric mapping [6]. Another required and related condition is incoherence between the measurement matrix and sparsity basis. This implies that the rows of the measurement matrix cannot sparsely represent the the columns of the basis and vice versa [2].

Verifying the RIP requires test of all the $\binom{n}{s}$ combinations. However there exist certain matrices known to satisfy the RIP and incoherence with high probability, examples are [2]:

1. Random matrices constructed from the discrete Fourier transform (DFT), by selecting m rows, uniformly at random [6, 13, 14, 15].
2. Random matrices with entries i.i.d. sampled from a Gaussian distribution with mean 0 and variance $1/m$, $\mathcal{N}(0, \frac{1}{m})$ [6, 13, 14, 15].

The latter is used for all of the performed experiments.

2.3.2 ℓ_1 Minimization Reconstruction

With sparsity and the measurement matrix in place the final thing that is need is a reconstruction algorithm that can reconstruct the signal, or at least find its sparse approximation, based on the compressed measurement $\mathbf{y}$ and the measurement matrix $\mathbf{\Phi}$. As state above, in section 2.3, reconstruction is concerned with solving an underdetermined system $\mathbf{y} = \mathbf{\Phi x}$.

Solving an underdetermined system of the form $\mathbf{y} = \mathbf{\Phi x}$ is a general problem of linear algebra. Linear algebra states that the best solution is the one that minimizes the norm residual. Put another way the goal is to find an $\hat{\mathbf{x}}$ that minimizes $||\mathbf{y} - \mathbf{\Phi}\hat{\mathbf{x}}||_p$, using a given ℓ_p norm [16].

ℓ_2 Minimization
The classical approach to solving this inverse problem, is to use the ℓ_2 norm, also known as euclidean norm, having the form $||\mathbf{x}||_2 = \sqrt{\sum_{i=1}^{n} x_i}$. This sets up the following problem to be solved

$$\hat{\mathbf{x}} = \text{argmin}||\mathbf{x}||_2 \text{ s. t. } \mathbf{\Phi x} = \mathbf{y} \tag{2.3.5}$$

This optimization problem has the convenient closed form $\hat{\mathbf{x}} = \mathbf{\Phi}^{\dagger}\mathbf{y}$, using the Moore-Penrose Pseudo inverse for when $m \leq n$, which has the form $\mathbf{\Phi}^{\dagger} = \mathbf{\Phi}^{*}\left(\mathbf{\Phi\Phi}^{*}\right)^{-1}$ [7, 16]. However this will almost never result in a sparse solution, and is thus not a suitable option [2]. This is illustrated in figure 2.3.3.

ℓ_0 Minimization
Since the ℓ_2 norm does not give the sparse result, the ℓ_0 norm could be used, as it counts the number of non-zero elements of $\mathbf{x}$, as $||\mathbf{x}||_0 = |\text{supp}(\mathbf{x})|$. This changes the problem to be solved to the form:

$$\hat{\mathbf{x}} = \text{argmin}||\mathbf{x}||_0 \text{ s. t. } \mathbf{\Phi x} = \mathbf{y} \tag{2.3.6}$$

Which can recover the sparse signal exactly, with high probability. Unfortunately solving these kind of systems is often computationally burdens some, as it generally requires a run though of all possible solutions [6].

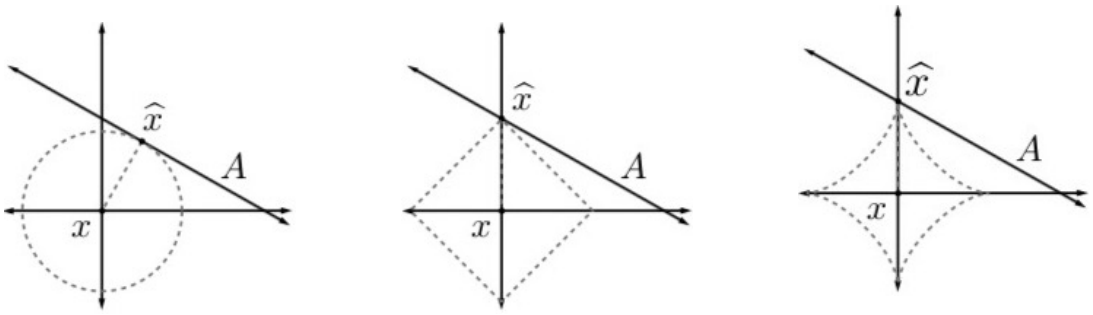
ℓ_1 Minimization
Using the ℓ_1 norm, also enables, with high probability, the finding of the sparse $\mathbf{x}$. This has the form:

$$\hat{\mathbf{x}} = \text{argmin}||\mathbf{x}||_1 \text{ s. t. } \mathbf{\Phi x} = \mathbf{y} \tag{2.3.7}$$

This is a convex optimization problem, that reduces to a linear program known as Basis Pursuit [8]. Basis Pursuit has a complexity of about $\mathcal{O}(n^3)$

Graphical representation

The above optimizations problems, can be illustrated graphically for the $\mathbb{R}^2$ case.



(a) Using ℓ_2 norm, also known as the euclidian norm.

(b) Using ℓ_1 norm, also known as the Manhatten norm

(c) Using $\ell_{0<p<1}$ norm, when p is zero the unit circle reduces to the axises

Figure 2.3.3. Graphical representation of the minimization for different norms. The solution set for the optimization problems lies in the subspace A. The solution using a given ℓ_p is the point for which the associated unitsphere, first touches the subspace A, when the size is increased. Based [17] figure 2.2

As figure 2.3.3 shows, the ℓ_0 and ℓ_1 norms, will find the sparse solution, which is lying on an axis, as the other axis is then zero, where as the ℓ_2 norm rarely will find this [17].

2.3.3 Matching Pursuit Reconstruction

Though the convex optimization techniques provides strong guarantees, they are computationally time consuming [13, 10, 17]. An alternative is greedy methods, which take an iterative approach to approximating the signal support and coefficients. This results in a faster algorithms, tough they might not offer as strong guarantees as the convex case [10, 6].
Matching Pursuit is a class of iterative greedy algorithms, that have received much attention in compressed sensing [6].

2.3.3.1 General Framework

Matching Pursuit can be considered as a family of algorithms that share some common steps. The fundamentals being iteratively estimation of the support set and coefficients, but can be seen as a four step process[6, 13]

1. Support set estimation
2. Support merge
3. Coefficient estimation
4. Residual update

These steps can be formed as a general framework (algorithm 2.3.1). The algorithms are typically initialized with a zero-vector as the initial estimate, an empty initial support set, and initial residual equal to the provided measurement vector. Then over a number of iterations, the support set is estimated, by selecting indices based on a matched filter. Doing each iteration the signal can be estimated by solving the resulting well-determined system, of using only the columns of the measurement matrix that are indexed by the current support set estimation. Based on this estimation the the contribution of the currently estimated support set can be removed from the

measurement vector, by compressing the estimated signal using the measurement matrix, and subtracting the result from the initial measurement. residual [6].

Algorithm 2.3.1 Matching Pursuit framework

Input:

- Measurement matrix $\mathbf{\Phi} \in \mathbb{R}^{m \times n}$
- Observation vector $\mathbf{y} \in \mathbb{R}^m$
- Sparsity level s of the ideal signal $\mathbf{x} \in \mathbb{R}^n$

Output:

- An estimate $\hat{\mathbf{x}} \in \mathbb{R}^n$ of the signal $\mathbf{x}$

1: Initialize iteration counter k, support set $\boldsymbol{\lambda}$, signal estimate $\hat{\mathbf{x}}$ and residual $\mathbf{r}$
2: **while** stopping criterion is not met **do**
3: Calculate correlation between current residual and the measurement matrix ($\mathbf{\Phi}^T\mathbf{r}$)
4: Select and add new indexes based on this correlation
5: Update estimator for x and current residual using $||\mathbf{y} - \mathbf{\Phi}_I\hat{\mathbf{x}}||_2$
6: **end while**

The Matching Pursuit algorithms primarily differ on two points: 1) Determining when to stop the iteration, and 2) how many indices to select each iteration [6].

2.3.3.2 Indices Selection Approaches

In general there are two overall approaches Which for this thesis has been named forward and hybrid selection. These approaches shares commonalities with forward-, backward-, and hybrid stepwise selection from the field of regression, from where the names are take [18].

Forward selection This approach selects indices in a forward manner and adds them to the support set. The goal of the strategies following this approach, is to estimate as many indices each iteration, without included wrong indices, i.e. indices not in the actual support set.

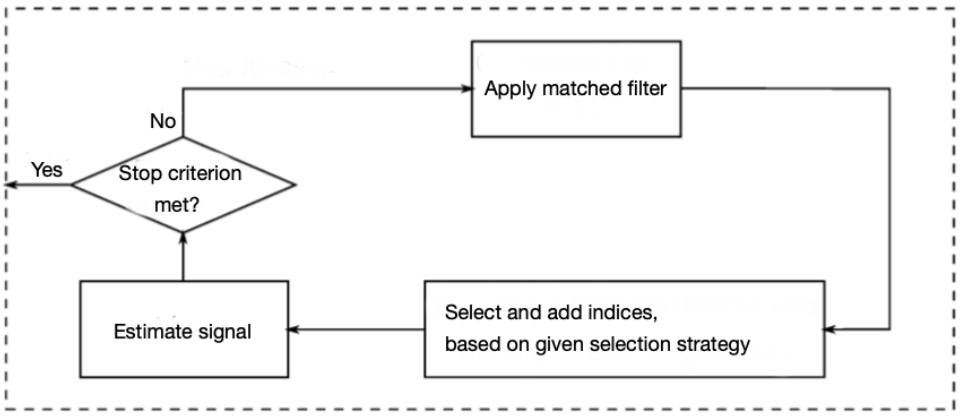


Figure 2.3.4. General activity diagram for the iterations of the forward selection approach. Iterations are performed until a give stopping criterion is met. Then for each iteration a mathced filter is applied. Based on this mathced filter a number of indices is selected for the support set using a given selection strategy. Then using the currently estimated support set, the original signal is estimated, before the stopping criterion is tjecked.

Hybrid selection This approach aims at speeding up the process of support set estimation, by overestimating the support set each iteration. This introduces the need for at bardward step, that can remove wrongly estimated indices.

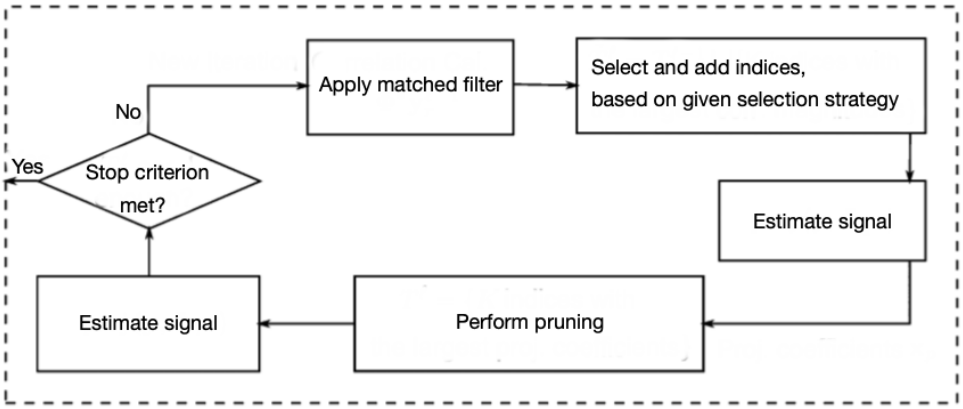


Figure 2.3.5. General activity diagram for the iterations of the hybrid selection approach. Iterations are performed until a give stopping criterion is met. Then for each iteration a mathced filter is applied. Based on this mathced filter a number of indices is selected for the support set using a given selection strategy. Then using the currently estimated support set, the original signal is estimated. Since the general selection approach for hybrid selection over estimates the signal a pruning step is introduces, reducing the support set to the desired size. Then using the pruned estimated support set, the original signal is estimated, before the stopping criterion is tjecked.

Different strategies within the two approaches will be discussed more in-depth in chapter 3.

2.4 Distributed Data Gathering

As increasing amounts of sensor networks are being deployed and based on the desire to keep these sensor devices as cheap as possible. Increasing demands are put on their software. In

order to keep these networks efficient and give them as long a life time as possible, certain aspect has to be optimized. Especially the amount of communication most be kept as low as possible, as transmitting and receiving are highly battery draining. Moreover, notes close to the sink, that have to forward a lot of data, as illustrated in figure 2.4.1, is highly loaded, making load balancing desirable. Additionally, the notes have limiting processing power, so advanced compression techniques might not be possible [3].

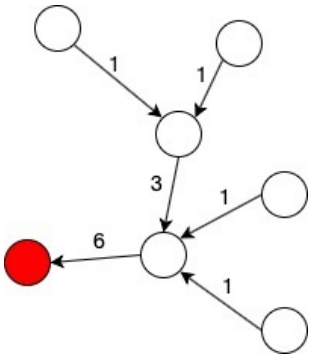


Figure 2.4.1. Distributed network standard data delivery. Colorless nodes are data points, red node is the sink. Numbers on the edges indicate data transmissions on the given link, to forward all data readings to the sink.

2.4.1 Aggregation

When the goal is to limit the communication needed in networks, a simple approach which introduces loadbalancing is data aggregation. Data aggregation generally consists of computing a compressed representation of the data, or at least an approximately representation [3].
An example of data aggregation is computation of the mean temperature, and only forwarding this, this can decrease the amount of data transmission, of that seen in figure 2.4.1 to that of figure 2.4.2. This reduces the communication load of the nodes close to the sink, which can increase the network life time, as these are important for keeping the connectivity. This strategies is however quite limiting, as individual sensor reading can not be propogate this way [3].

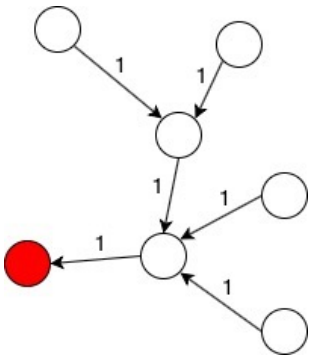


Figure 2.4.2. Distributed network packet delivery with aggregation. Colorless nodes are data points, red node is the sink. Numbers on the edges indicate packet transmissions, to forward all packets to the sink, when aggregation is applied to the same network as in figure 2.4.1.

2.4.2 Network Coding

If a source needs to send the information XY to a sink, and the network structure is known, a path can be calculated, which can be used to minimize the amount of data transmission on each link, like the one in figure 2.4.3. Here only half the message (or the XOR of the two halves) is transmitted on each link. The sink(s) can then reconstruct the message, using simple logic operations [3].

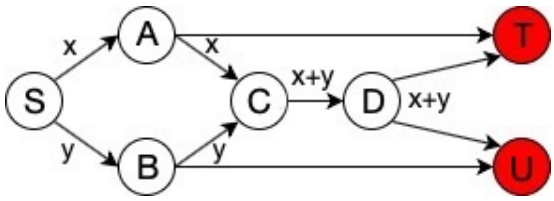
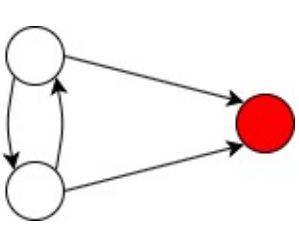


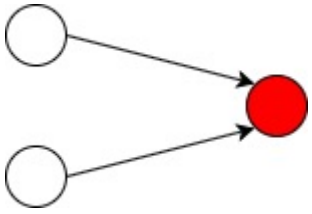
Figure 2.4.3. Network coding. Notes are colorless and sinks are red. When node S needs to send data XY to the sinks and the network layout is known, a route can be calulated that allows for only transmitting half the data on each link. S spilts the the data into the parts X and Y and send on part to A and the other part to B. Then A and B forwards the received data, which either reaches sink T or U. The transmission from both A and B reaches D, which calcuates the XOR'ed representation of the two data halfs, and forwards this to both sinks, which can then reconstruct the original data.

2.4.3 Distributed Source Coding

Where aggregation is a simple way of introducing load balancing, it is limited to scenarios of the kind described in section 2.4.1. When specific data values of different notes are needed at the sink, more advanced methods are required [3].
Compression techniques covered in section 2.2 can be applied, however, they might require communication among notes (as in figure 2.4.4a), which is counter intuitive, as the goal is to decrease the communication [3].
Distributed Source Coding utilizes separate encoding and joint decoding of correlated data, only requiring the communication as in figure 2.4.4b. The rate at which the two sources independently can compress their data, while the sink being able to reconstruct, is bound by the Slepian-Wolf Theorem [3].



(a) The two sources, exchanges information inorder to compress and send to the sink.



(b) The correlation between the data of the two sources is know, so no information exhange is needed

Figure 2.4.4. Two sources(colorless) gather correlated data, to be send to the sink(red)

2.4.3.1 Slepian-Wolf Theorem

The most straight forward approach for the two sources of figure 2.4.4b to send their data to the sink, without communicating with each, is to send the full uncompressed data. Though if their data is correlated, distributed source coding can be applied, which can achieve the same overall communication reduction as if both signals, were available to each source, this is known as the Slepian-Wolf Theorem. This is based on that if one of the sources, transmits its entire signal, the other source can send a compressed representation that can be reconstructed at the sink, as the correlation can be used to reconstruct the second signal, using the first one. This will be more clearly demonstrated below [3].

DISCUS Distributed Source Coding Using Syndromes (DISCUS) presented by Pradhan and Ramchandran is the first approach for constructing codes, achieving the Slepian-Wolf bound. It is often illustrated by the following example.
Given two sources X and Y, both wanting to send a three bit long data stream to the source. Assuming that source Y sends all three bits. Then source X can send fewer than three bit from which the sink can reconstruct the original three bits, assuming a correlation between the sources. The correlation structure assumed, is that the three bits from X only can differ in one position, from the bits of Y. Then when Y sends the message 000, the message from X can only be 000, 001, 010 or 100. Thus by dividing all possible messages in to pairs, that differ on more than one bit (e.g. { 001, 110}, { 010, 101}, { 100, 011} and { 000, 111}), X only need to send enough information, for the source to distinguish between the four pairs, which would only require two bits. The sources then select the three bit message, in the indicated pair, that differ the least from the message received from Y [3].

2.5 Distributed Compressed Sensing

The increasing interested in and deployment of wireless sensor networks, as well as the requirement to decrease the expensive communication in these networks, has created an interest in utilizing Compressed Sensing. However, while Compressed Sensing is an efficient tool, it is mainly designed to deal with intra-signal structures, which makes it less suitable for dealing with multiple signals [19, 20, 3]. Due to this Duarte, Sanvotham, Baron, Wakin, and Baraniuk [30] proposed the Distributed Compressed Sensing theory, which expands the theory of compressed sensing to also cover inter-signal correlation structures, as well as enabling distributed encoding. In their work they have presented a Joint Sparsity Model, that expands the notation of a single signal being sparse in some basis, to the notation that an ensemble of signals being jointly sparse [19]. The Joint Sparsity Model covers three different models, and has later been generalized to a more generic model known as the Mixed Support-Set Model [21]. Both of these models have been argued to cover a wide variety of signals encounter by wireless sensor networks [19, 21].

2.5.1 Signal Models

2.5.1.1 Joint Sparsity Model

The three different data models of the Joint Sparsity Model is termed JSM-1, JSM-2 and JSM-3, and are defined as follows.

JSM-1 Builds on the assumption that the signals share a common support set, in a given basis. Additionally each signal is assumed to have a unique innovation support set. Thus the individual signals can mathematically be split in to two part.

$$\mathbf{x}_j = \mathbf{z} + \mathbf{z}_j, \quad j \in \{1, 2, ..., J\} \tag{2.5.1}$$

With J being the number of correlated signals. Thus the j^{th} signal $\mathbf{x}_j$ is the combination of $\mathbf{z}$ which represent the common support set, and $\mathbf{z}_j$ representing the innovation support set of the j^{th} signal [19].
An examples of this is outdoor temperature measurements taken over a given period of time. Here the common support set, is influences by global effects of sun, while the innovation support set is influenced by more local effects such as shade [19].

JSM-2 Builds on the assumption that the signals have a shared support set, though with different coefficients. Thus a mathematical representation is.

$$\mathbf{x}_j = \mathbf{\Psi}\mathbf{z}_j, \quad j \in \{1, 2, ..., J\} \tag{2.5.2}$$

Here $\mathbf{\Psi}$ is the basis for which the signal is sparse. Each $\mathbf{z}_j$ have common support set, but different coefficients [19].
An example of this is where multiple sensors measure a common sound source, though at different locations introducing phase shift as well as different attenuation [19].

JSM-3 Is a later extension of JSM-1, where only the innovation support set are assumed sparse [22]. Though no further interesting explanation and use have been found

2.5.1.2 Mixed Support-Set Model (MSM)

The more generalized signal model, Mixed Signal Model [21], is defined as

$$\mathbf{x}_j = \mathbf{z}_j^{(c)} + \mathbf{z}_j^{(p)}, \quad j \in \{1, 2, ..., J\} \tag{2.5.3}$$

Here each $\mathbf{z}_j^{(c)}$ has a common support set, though with different coefficients. Each $\mathbf{z}_j^{(p)}$ contains the individual innovation support sets and coefficients [21].
The Mixed Support-Set Model is equivalent to JSM-1, when the coefficients of each $\mathbf{z}_j^{(c)}$ are the same. The Model is equivalent to JSM-2 when $\mathbf{z}_j^{(p)}$ is the null vectors [21].

3

MATCHING PURSUIT FOR SPARSE APPROXIMATION AND COMPRESSIVE SENSING

A wide selection of Matching Pursuit based algorithms have been selected and these will be presented in the following. Additionally, the algorithms will be analyzed in regards to computational and storage complexities.
The selected algorithms cover both forward and hybrid selection approaches, as well as an extension for handling multiple correlated signals.
In general the algorithms will be analyzed based on worst case scenarios, with the only assumption on the inputs being $s < n < m$ as well as the restricted isometry property being obeyed for the measurement matrix. Additionally the analysis will be performed based on using the Moore-Penrose pseudo inverse, for the signal estimation steps, though more computationally effective approaches exists, such as maintaining a QR factorization, though which increases to storage complexity [23].
The analysis will be performed with extra focus on implementation, as the idea behind the summary and analysis, is to understand the mechanics of the algorithm as well as being able to implement them.

3.1 Forward Selection

3.1.1 Orthogonal Matching Pursuit (OMP)

Orthogonal Matching Pursuit was introduced by Pati, Rezaiifar, and Krishnaprasad in 1993 [9]. It is a slight improvement of the original Matching Pursuit algorithm, which was introduced by Mallet and Zhang in 1993 [24]. Due to this Matching Pursuit will only be included in relation to how Orthogonal Matching Pursuit improves upon it.
Orthogonal Matching Pursuit follows the basic structure for Matching Pursuit based algorithms, described in section 2.3.3. Algorithm 3.1.1 lists Orthogonal Matching Pursuit and will be described and analyzed below.

As inputs Orthogonal Matching Pursuit takes; the measurement matrix $\mathbf{\Phi}$, measurement vector $\mathbf{y}$ and the sparsity level s of the signal $\mathbf{x}$, which is to be estimated. Initially, the algorithm sets up an iteration counter k, a support set vector $\boldsymbol{\lambda}$ with a size equal to the sparsity s, an estimation null vector $\hat{\mathbf{x}}$, of the original signal $\mathbf{x}$, with length deduced from the row length of the measurement matrix. Lastly, a residual vector $\mathbf{r}$ is created and set equal to $\mathbf{y}$. The work of the algorithm is performed inside a while loop, that runs for a number of iterations, equal to the provided sparsity level s of the desired signal $\mathbf{x}$. Initially, the iteration counter is incremented (line 6). Then Orthogonal Matching Pursuit selects the single element of the measurement matrix, which has the highest correlation with the current residual (line 7), based on a matched filter $\mathbf{\Phi}^T(r)^{(k-1)}$. This has computational complexity $\mathcal{O}_t(nm)$ as a matrix-vector multiplication, and a space complexity of $\mathcal{O}_s(1)$, when calculated for a column of $\mathbf{\Phi}^T$ at a time, as it then only requires the index of the highest coefficient to be stored. This column index is then added to the support set (line 8). This has both computational and space complexity $\mathcal{O}(1)$, as it requires insertion in the support set, at the index equal to the iteration counter. Then the signal is

estimated by solving a least squares problem of the form $||\mathbf{y} - \mathbf{\Phi}_{\boldsymbol{\lambda}}\hat{\mathbf{x}}||_2$, using only the k columns of $\mathbf{\Phi}$ pointed out by the current support set (line 9). This has complexities $\mathcal{O}_t(mk + mk^2 + k^3)$ from the matrix multiplication with mk^2 being the dominating part and $\mathcal{O}_s(1)$, as it assigns to the already initialized $\hat{\mathbf{x}}$ vector. Lastly the residual is updated by subtracting the estimation from the measured signal, with complexity $\mathcal{O}_t(nm)$ from the matrix-vector multiplication and $\mathcal{O}_s(1)$, as the residual vector is already allocated.
Summing the above analysis, Orthogonal Matching Pursuit has computation complexity $\mathcal{O}_t(snm + smk^2)$. The storage complexity is only influenced by the initialization, though if implemented using QR, this will increase.
With regards to reconstruction guarantees, the authors of [25] demonstrate theoretical that Orthogonal Matching Pursuit is able to reliably recover a s sparse signal $m \geq s \log n$ random linear measurements. Empirically they demonstrate Orthogonal Matching Pursuit can reconstruct signals from fever measurements for the noiseless case.

Algorithm 3.1.1 Orthogonal Matching Pursuit (OMP)

Input:

- Measurement matrix $\mathbf{\Phi} \in \mathbb{R}^{m \times n}$
- Observation vector $\mathbf{y} \in \mathbb{R}^m$
- Sparsity level s of the ideal signal $\mathbf{x} \in \mathbb{R}^n$

Output:

- An estimate $\hat{\mathbf{x}} \in \mathbb{R}^n$ of the signal $\mathbf{x}$

1:	$k \leftarrow 0$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
2:	$\boldsymbol{\lambda}^{(0)} \leftarrow \mathbf{0}_{s \times 1}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(s)$
3:	$\hat{\mathbf{x}}^{(0)} \leftarrow \mathbf{0}_{n \times 1}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(n)$
4:	$\mathbf{r}^{(0)} \leftarrow \mathbf{y}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(m)$
5:	**while** $k \leq s$ **do**	$\mathcal{O}_t(s)$	$\mathcal{O}_s(1)$
6:	$\quad k \leftarrow k + 1$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
7:	$\quad \omega^{(k)} \leftarrow \text{supp}(\mathbf{\Phi}^T \mathbf{r}^{(k-1)}, 1)$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(1)$
8:	$\quad \boldsymbol{\lambda}^{(k)} \leftarrow \boldsymbol{\lambda}^{(k-1)} \cup \omega^{(k)}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
9:	$\quad \hat{\mathbf{x}}^{(k)} \leftarrow \text{argmin}_{\mathbf{u}:\text{supp}(\mathbf{u}) = \boldsymbol{\lambda}^{(k)}} ||\mathbf{y} - \mathbf{\Phi}\mathbf{u}||_2$	$\mathcal{O}_t(mk^2)$	$\mathcal{O}_s(1)$
10:	$\quad \mathbf{r}^{(k)} \leftarrow \mathbf{y} - \Phi\hat{\mathbf{x}}^{(k)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(1)$
11:	**end while**		
12:	**return** $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}}^{(k)}$		
		$\mathcal{O}_t(snm + smk^2)$	$\mathcal{O}_s(n)$

3.1.2 Stagewise Orthogonal Matching Pursuit (StOMP)

Stagewise Orthogonal Matching Pursuit was introduced by Donoho, Tsaig, Drori, and Starck in 2007 [10]. It is designed to be significantly faster than Orthogonal Matching Pursuit, by increasing the number of selected indices each iteration and by running a fixed number of iterations, much lower than required for Orthogonal Matching Pursuit.
Stagewise Orthogonal Matching Pursuit follows the basic structure for Greedy Pursuit algo-

rithms, described in section 2.3.3. Algorithm 3.1.2 lists Stagewise Orthogonal Matching Pursuit and it will be described and analyzed below.

Stagewise Orthogonal Matching Pursuit takes some different input parameters than Orthogonal Matching Pursuit. It no longer takes the sparsity level s, making it more appealing for the case of this being unknown. Instead, it takes a max iteration count S and a thresholding parameter t_s.
For the initialization, the final size of the support set is not as well known as for Orthogonal Matching Pursuit, having a worst case of size n as all the indices could be selected.
As for Orthogonal Matching Pursuit, the work of the algorithm is performed within a loop, which now runs for a predetermined number of iterations. The authors of [10], state that 10 should be a decent choice, for estimating the real support set, for sufficiently sparse signals.
For the iterative work, Stagewise Orthogonal Matching Pursuit only utilizes a different strategy for selecting indices. Stagewise Orthogonal Matching Pursuit still performs the selection based on the matched filter. However, the indices of all the coefficients that are above a given threshold are selected (line 7). This threshold is calculated based on the formal noise level $\sigma^2 = ||\mathbf{r}||_2/\sqrt{(n)}$ which is multiplied by the threshold parameter t_s, for which the authors [10] states that a value between $2 \leq t_s \leq 3$, would in combination with 10 iterations, result in the true support set for sufficiently sparse signals, with high probability. However, this could end up including all n indices for arbitrary signals. This selecting strategy is inspired by Gaussian noise removal, in digital communication known as Multiple-Access Interference, and is motivated by the desire to reduce the false alarm rate and false discovery rate [10]. The selected indices are then merged with the current estimate, which would require a merge of the two sets, with complexity $\mathcal{O}_t(n)$.
The signal estimation (line 9) is similar to that of Orthogonal Matching Pursuit, though due to indeterminacy of the support set size, the computational complexity changes to $\mathcal{O}_t(n^3)$, in the worst case, where all indices are selected.
Summing the above analysis, Stagewise Orthogonal Matching Pursuit has computational complexity $\mathcal{O}_t(Sn^3)$. All of the memory allocation can be performed at initialization with a bound of $\mathcal{O}_s(n)$.
With regard to guaranteesn the authors state that for sufficiently sparse signals no more than m indices would be chosen, with high probability. This reduces the complexity to $\mathcal{O}_t(Sm^3)$. Should only s indices be chosen the complexity becomes $\mathcal{O}_t(Smn + Smk^2)$, which is less than for Orthogonal Matching Pursuit given that $S < s$.

Algorithm 3.1.2 Stagewise Orthogonal Matching Pursuit (StOMP)

Input:

- Measurement matrix $\mathbf{\Phi} \in \mathbb{R}^{m \times n}$
- Observation vector $\mathbf{y} \in \mathbb{R}^m$
- Number of iterations S to perform
- Threshold value t_s

Output:

- An estimate $\hat{\mathbf{x}} \in \mathbb{R}^n$ of the signal $\mathbf{x}$

		$\mathcal{O}_t$	$\mathcal{O}_s$
1:	$k \leftarrow 0$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
2:	$\boldsymbol{\lambda}^{(0)} \leftarrow \varnothing$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
3:	$\hat{\mathbf{x}}^{(0)} \leftarrow \mathbf{0}_{n \times 1}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(n)$
4:	$\mathbf{r}^{(0)} \leftarrow \mathbf{y}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(m)$
5:	**while** $k < S$ and $\left\|\left\|\mathbf{r}^{(k)}\right\|\right\|_2 > \epsilon$ **do**	$\mathcal{O}_t(S)$	$\mathcal{O}_s(1)$
6:	$\quad k \leftarrow k + 1$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
7:	$\quad \boldsymbol{\omega}^{(k)} \leftarrow \left\{ j : \left\| \left[\mathbf{\Phi}^T \mathbf{r}^{(k-1)} \right]_j \right\| \leq t_s \sigma_s \right\}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(n)$
8:	$\quad \boldsymbol{\lambda}^{(k)} \leftarrow \boldsymbol{\lambda}^{(k-1)} \cup \boldsymbol{\omega}^{(k)}$	$\mathcal{O}_t(n)$	$\mathcal{O}_s(n)$
9:	$\quad \hat{\mathbf{x}}^{(k)} \leftarrow \operatorname{argmin}_{\mathbf{u}:\operatorname{supp}(\mathbf{u})=\boldsymbol{\lambda}^{(k)}} \|\mathbf{y} - \mathbf{\Phi u}\|_2$	$\mathcal{O}_t(n^3)$	$\mathcal{O}_s(1)$
10:	$\quad \mathbf{r}^{(k)} \leftarrow \mathbf{y} - \Phi \hat{\mathbf{x}}^{(k)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(1)$
11:	**end while**		
12:	**return** $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}}^{(k)}$		
		$\mathcal{O}_t(Sn^3)$	$\mathcal{O}_s(n)$

3.1.3 Regularized Orthogonal Matching Pursuit (ROMP)

Regularized Orthogonal Matching Pursuit was introduced by Needell and Vershynin in 2009 [11, 26, 27]. It is designed to be faster and give stronger guarantees than Orthogonal Matching Pursuit.
Regularized Orthogonal Matching Pursuit follows the basic structure for Greedy Pursuit algorithms, described in section 2.3.3. Algorithm 3.1.3 lists Regularized Orthogonal Matching Pursuit and it will be described and analyzed below.

Regularized Orthogonal Matching Pursuit takes the same inputs as Orthogonal Matching Pursuit.
For the initialization, the end size is of the support set is unknown, and as for Stagewise Orthogonal Matching Pursuit would have a worst-case size n.
As for Orthogonal Matching Pursuit, the work of the algorithm is performed within a loop, which runs for s iterations or until the residual is sufficiently low.
For the iterative work, Regularized Orthogonal Matching Pursuit changes the strategy for selecting indices. Regularized Orthogonal Matching Pursuit also performed the selection based on the matched filter. However, the indices are selected based on a regularization step (line 9). This can be done by calculating and storing the matched filter output (line 7). These coefficients are

then sorted and the s highest in magnitude are selected (line 8). This selection is then split into subsets, where the coefficients of a subset are within a factor two of each other (line 9). The original indices of the subset with the highest energy are then selected. Though the article states that it suffices to only consider consecutive subsets, making the complexity $\mathcal{O}_t(s)$. This could select all indices for an arbitrary signal. These indices are then merged with previously found ones (line 10). If the support set er kept sorted, this can be done with complexity $\mathcal{O}_t(n)$. The signal estimation (line 11) is similar to Orthogonal Matching Pursuit, though due to worst-case size of the support set the computational complexity increases to $\mathcal{O}_t(n^3)$ Summing the above analysis, Regularized Orthogonal Matching Pursuit has computational complexity $\mathcal{O}_t(sn^3)$. All of the memory allocations can be performed at initialization, though doe to regularization, it requires more than Orthogonal Matching Pursuit.
With regards to guarantees, the article states the following theorems of interest.

Theorem 1 *(Theorem 1.3 of [11]) (Exact sparse recovery via ROMP) Assume a measurement matrix* $\mathbf{\Phi}$ *satisfies the restricted isometry condition with parameters (2s,* ϵ*) for* $\epsilon = 0.03/\sqrt{\log s}$ *Let* $\mathbf{x}$ *be an s-sparse vector in* $\mathbb{R}^n$ *with measurements* $\mathbf{y} = \mathbf{\Phi x}$*. Then ROMP in at most s iterations outputs a set I such that*

$$supp(\mathbf{x}) \subset I \quad and \quad |I| \leq 2s \tag{3.1.1}$$

Theorem 2 *(Theorem 3.1 of [11]) (Iteration invariant of ROMP) Assume* $\mathbf{\Phi}$ *satisfies the Restricted Isometry Condition with parameters (2s,* ϵ*) for* $\epsilon = 0.03/\sqrt{\log s}$*. Let* $\mathbf{x} \neq 0$ *be an s-sparse vector with measurements* $\mathbf{y} = \mathbf{\Phi x}$*. Then at any iteration of ROMP, after the regularization step, we have* $J_0 \neq \varnothing$*,* $J_0 \cap I = \varnothing$ *and*

$$|J_0 \cap supp(\boldsymbol{x})| \geq 1/2\,|J_0| \tag{3.1.2}$$

Which states that if the signal is in fact s sparse, then at least half of the indices selected after regularization are in the true support set. As well as the resulting support set will be at most $2s$ in size and contain the true support set. Assuming this holds the overall complexity changes to $\mathcal{O}_t(smn + ms^3)$.
The article[11] also states that Regularized Orthogonal Matching Pursuit should, with high probability, be able to reconstruct signals when $m \geq s\ \log n$.

Algorithm 3.1.3 Regularized Orthogonal Matching Pursuit (ROMP)

Input:

- Measurement matrix $\mathbf{\Phi} \in \mathbb{R}^{m \times n}$
- Observation vector $\mathbf{y} \in \mathbb{R}^m$
- Sparsity level s of the ideal signal $\mathbf{x} \in \mathbb{R}^n$

Output:

- An estimate $\hat{\mathbf{x}} \in \mathbb{R}^n$ of the signal $\mathbf{x}$

1:	$k \leftarrow 0$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
2:	$\boldsymbol{\lambda}^{(0)} \leftarrow \varnothing$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
3:	$\hat{\mathbf{x}}^{(0)} \leftarrow \mathbf{0}_{n \times 1}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(n)$
4:	$\mathbf{r}^{(0)} \leftarrow \mathbf{y}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(m)$
5:	**while** $\mathbf{r} \geq \epsilon$ and $k < s$ **do**	$\mathcal{O}_t(s)$	$\mathcal{O}_s(1)$
6:	$k \leftarrow k + 1$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
7:	$\mathbf{v}^{(k)} \leftarrow \mathbf{\Phi}^T \mathbf{r}^{(k-1)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(n)$
8:	$\boldsymbol{\omega}^{(k)} \leftarrow \text{supp}(\mathbf{v}^{(k)}, s)$	$\mathcal{O}_t(n \ \log n)$	$\mathcal{O}_s(s)$
9:	$\boldsymbol{\omega}_0^{(k)} \leftarrow \text{argmax}_{\forall i,j \in \boldsymbol{\omega}_0 : \lvert\mathbf{v}(i)\rvert \leq 2\lvert\mathbf{v}(j)\rvert} \lvert\lvert u_{\boldsymbol{\omega}_0} \rvert\rvert_2$	$\mathcal{O}_t(s)$	$\mathcal{O}_s(s)$
10:	$\boldsymbol{\lambda}^{(k)} \leftarrow \boldsymbol{\lambda}^{(k-1)} \cup \boldsymbol{\omega_0}^{(k)}$	$\mathcal{O}_t(n)$	$\mathcal{O}_s(n)$
11:	$\hat{\mathbf{x}}^{(k)} \leftarrow \text{argmin}_{\mathbf{u}:\text{supp}(\mathbf{u}) = \boldsymbol{\lambda}^{(k)}} \lvert\lvert \mathbf{y} - \mathbf{\Phi u} \rvert\rvert_2$	$\mathcal{O}_t(n^3)$	$\mathcal{O}_s(1)$
12:	$\mathbf{r}^{(k)} \leftarrow \mathbf{y} - \mathbf{\Phi} \hat{\mathbf{x}}^{(k)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(1)$
13:	**end while**		
		$\mathcal{O}_t(sn^3)$	$\mathcal{O}_s(n)$

3.1.4 Generalized Orthogonal Matching Pursuit (gOMP)

Generalized Orthogonal Matching Pursuit was introduced by Wang and Shim in 2011 [12, 28]. It is designed to improve the recovery performance of Orthogonal Matching Pursuit, by increase the indices selection rate, in a more straightforward way than Stagewise Orthogonal Matching Pursuit or Regularized Orthogonal Matching Pursuit.
Generalized Orthogonal Matching Pursuit follows the basic structure for Greedy Pursuit algorithms, described in section 2.3.3. Algorithm 3.1.4 lists Generalized Orthogonal Matching Pursuit and it will be described and analyzed below.

Generalized Orthogonal Matching Pursuit takes the same inputs as Orthogonal Matching Pursuit, as well as a number S of indices to select each iteration.
For the initialization, Generalized Orthogonal Matching Pursuit follows the same structures as Orthogonal Matching Pursuit, whereas in contrast to Stagewise Orthogonal Matching Pursuit and Regularized Orthogonal Matching Pursuit, no more than s indices can be selected.
As for Orthogonal Matching Pursuit, the work of the algorithm is performed within a loop, which now runs for a maximum of m/S iteration, to accommodate for the increase in index selection and as long as the residual norm is above a certain limit.
For the iterative, work Generalized Orthogonal Matching Pursuit uses the matched filter to select indices, just like Orthogonal Matching Pursuit. Only now S indices are selected. Which

can be done by sorting of the coefficient of the matched filter. The merging with previously selected indices can be done with complexity $\mathcal{O}_t(s)$, given the indices are kept sorted. The signal estimation () is similar to Orthogonal Matching Pursuit, though the complexity increases as m columns can be selected

With regards to reconstruction guarantees, the authors [12, 28], states that Regularized Orthogonal Matching Pursuit provide stable reconstruction for all s sparse signal, given that $\mathbf{\Phi}$ satisfy the RIP with $\delta_{\max\{9,S+1\}} \leq \frac{1}{8}$, within $\max\{s, \frac{8s}{S}\}$ iterations. Their results then indicate that the number of required measurements is $m = \mathcal{O}(s \log \frac{n}{s})$.

Algorithm 3.1.4 Generalized Orthogonal Matching Pursuit (gOMP)

Input:

- Measurement matrix $\mathbf{\Phi} \in \mathbb{R}^{m \times n}$
- Observation vector $\mathbf{y} \in \mathbb{R}^m$
- Sparsity level s of the ideal signal $\mathbf{x} \in \mathbb{R}^n$
- Number of indices for each selection $S \leq s$

Output:

- An estimate $\hat{\mathbf{x}} \in \mathbb{R}^n$ of the signal $\mathbf{x}$

1:	$k \leftarrow 0$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
2:	$\boldsymbol{\lambda}^{(0)} \leftarrow \varnothing$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
3:	$\hat{\mathbf{x}}^{(0)} \leftarrow \mathbf{0}_{n \times 1}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(n)$
4:	$\mathbf{r}^{(0)} \leftarrow \mathbf{y}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(m)$
5:	**while** $\left\|\left\|\mathbf{r}^{(k)}\right\|\right\|_2 > \epsilon$ and $k < \frac{m}{S}$ **do**	$\mathcal{O}_t(m/S)$	$\mathcal{O}_s(1)$
6:	$\quad k \leftarrow k + 1$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
7:	$\quad \mathbf{v}^{(k)} \leftarrow \mathbf{\Phi}^T \mathbf{r}^{(k-1)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(n)$
8:	$\quad \boldsymbol{\omega}^{(k)} \leftarrow \text{supp}(\mathbf{v}^{(k)}, S)$	$\mathcal{O}_t(n \log n)$	$\mathcal{O}_s(S)$
9:	$\quad \boldsymbol{\lambda}^{(k)} \leftarrow \boldsymbol{\lambda}^{(k-1)} \cup \boldsymbol{\omega}^{(k)}$	$\mathcal{O}_t(S)$	$\mathcal{O}_s(m)$
10:	$\quad \hat{\mathbf{x}}^{(k)} \leftarrow \text{argmin}_{\mathbf{u}:\text{supp}(\mathbf{u})=\boldsymbol{\lambda}^{(k)}} \|\mathbf{y} - \mathbf{\Phi}\mathbf{u}\|_2$	$\mathcal{O}_t(m^3)$	$\mathcal{O}_s(1)$
11:	$\quad \mathbf{r}^{(k)} \leftarrow \mathbf{y} - \mathbf{\Phi}\hat{\mathbf{x}}^{(k)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(1)$
12:	**end while**		
13:	**return** $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}}^{(k)}$		
		$\mathcal{O}_t(nm^2/S + m^4/S)$	$\mathcal{O}_s(n)$

3.2 Hybrid Selection

3.2.1 Compressive Sampling Matching Pursuit (CoSaMP)

Compressive Sampling Matching Pursuit was introduced by Needell and Tropp in 2008 [13, 27]. It is designed to deliver the same guarantees, as the convex optimization-based approaches, though with more rigorous bound on computation and storage requirements.

Compressive Sampling Matching Pursuit follows the basic structure for Greedy Pursuit algorithms, described in section 2.3.3, with pruning. Algorithm 3.2.1 lists Compressive Sampling Matching Pursuit and it will be described and analyzed below.

Compressive Sampling Matching Pursuit takes the same inputs as Orthogonal Matching Pursuit. For the initialization, it is similar to Orthogonal Matching Pursuit, though the size of the support set will have a maximum size of $3s$, due to the selection strategy. As for Orthogonal Matching Pursuit, the work of the algorithm is performed within a loop, which runs until a halting condition is met. For this work a halting condition that allows for s iterations or until the residual is negligible is used.
For the iterative work, the selection of indices is based on a matched filter (line 7). The $2s$ highest coefficients are then selected (line 8). These indices are merged with the previous estimate (line 9), which gives a set of at most $3s$ indices. Based on these $3s$ indices the signal is estimated by least squre, as for Orthogonal Matching Pursuit (line 10). After the estimation pruning is used to set all but the s highest coefficients to 0. Lastly the residual is updated (line 12).
Summing the above this gives complexities of $\mathcal{O}_t(snm + ms^3)$ and $\mathcal{O}_s(n)$

With regards to guarantees, the authors state that Compressive Sampling Matching Pursuit can reconstruct compressible signals when $m = \mathcal{O}(s \log(n/s))$. Moreover, they state that the iterations required for reconstructing compressible signals is $\mathcal{O}(\log s)$.

Algorithm 3.2.1 Compressive Sampling Matching Pursuit (CoSaMP)

Input:

- Measurement matrix $\mathbf{\Phi} \in \mathbb{R}^{m \times n}$
- Observation vector $\mathbf{y} \in \mathbb{R}^m$
- Sparsity level s of the ideal signal $\mathbf{x} \in \mathbb{R}^n$

Output:

- An estimate $\hat{\mathbf{x}} \in \mathbb{R}^n$ of the signal $\mathbf{x}$

1:	$k \leftarrow 0$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
2:	$\boldsymbol{\lambda}^{(0)} \leftarrow \mathbf{0}_{3s \times 1}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(s)$
3:	$\hat{\mathbf{x}}^{(0)} \leftarrow \mathbf{0}_{n \times 1}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(n)$
4:	$\mathbf{r}^{(0)} \leftarrow \mathbf{y}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(m)$
5:	**while** Halting condition is false **do**	$\mathcal{O}_t(s)$	$\mathcal{O}_s(1)$
6:	$\quad k \leftarrow k + 1$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
7:	$\quad \mathbf{v}^{(k)} \leftarrow \mathbf{\Phi}^T \mathbf{r}^{(k-1)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(n)$
8:	$\quad \boldsymbol{\omega}^{(k)} \leftarrow \text{supp}(\mathbf{v}^{(k)},\ 2s)$	$\mathcal{O}_t(n \log n)$	$\mathcal{O}_s(s)$
9:	$\quad \boldsymbol{\lambda}^{(k)} \leftarrow \text{supp}(\hat{\mathbf{x}}^{k-1}) \cup \boldsymbol{\omega}^{(k)}$	$\mathcal{O}_t(s)$	$\mathcal{O}_s(s)$
10:	$\quad \bar{\mathbf{x}} \leftarrow \text{argmin}_{\mathbf{u}:\text{supp}(\mathbf{u})=\boldsymbol{\lambda}^{(k)}} \lvert\lvert \mathbf{y} - \mathbf{\Phi}\mathbf{u} \rvert\rvert_2$	$\mathcal{O}_t(ms^2)$	$\mathcal{O}_s(1)$
11:	$\quad \hat{\mathbf{x}}^{(k)} \leftarrow \bar{\mathbf{x}}^S$	$\mathcal{O}_t(n)$	$\mathcal{O}_s(1)$
12:	$\quad \mathbf{r}^{(k)} \leftarrow \mathbf{y} - \mathbf{\Phi}\hat{\mathbf{x}}^{(k)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(m)$
13:	**end while**		
14:	**return** $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}}^{(k)}$		
		$\mathcal{O}_t(snm + ms^3)$	$\mathcal{O}_s(n)$

3.2.2 Subspace Pursuit (SP)

Subspace Pursuit was introduced by Wei Dai and Olgica Milenkovic in 2009 [14]. It is designed to have low computational complexity, compared to Orthogonal Matching Pursuit, and provide reconstruction guarantees of the same order as the convex optimization-based approaches.
Subspace Pursuit follows the basic structure for Greedy Pursuit algorithms, described in section 2.3.3, with pruning. Algorithm 3.2.2 lists Subspace Pursuit and it will be described and analyzed below.

Subspace Pursuit takes the same inputs as Orthogonal Matching Pursuit. For the initialization, it is similar to Orthogonal Matching Pursuit, though the size of the support set will have a maximum size of $2s$, due to the selection strategy As for Orthogonal Matching Pursuit, the work of the algorithm is performed within a loop, which runs for s iterations or until the residual stop decreasing. For the iterative work, the selection of indices is based on a matched filter (line 8). The s highest coefficients are selected (line 9). These indices are merged with the previous estimate (line 10), which gives a set of at most $2s$ indices. Based on these $2s$ indices the signal is estimated by least squares (line 11). After the estimation pruning is used to set all but the s highest coefficients to 0, followed by again another least square problem with the new support set of size s (line 13). Lastly the residual is updated (line 14).
Summing the above this gives complexities of $\mathcal{O}_t(snm + ms^3)$ and $\mathcal{O}_s(n)$.

With regards to guarantees, the authors state that Subspace Pursuit is able to reconstruct signals with a worst-case iteration bound of $\mathcal{O}(s)$, which reduces to $\mathcal{O}(\log s)$ when the coefficients of $\mathbf{x}$ decay according to a power law. Additionally, the authors state that reconstruction be be performed when $m = 2s$.

Algorithm 3.2.2 Subspace Pursuit (SP)

Input:

- Measurement matrix $\Phi \in \mathbb{R}^{m \times n}$
- Observation vector $\mathbf{y} \in \mathbb{R}^m$
- Sparsity level s of the ideal signal $\mathbf{x} \in \mathbb{R}^n$

Output:

- An estimate $\hat{\mathbf{x}} \in \mathbb{R}^n$ of the signal $\mathbf{x}$

1:	$k \leftarrow 0$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
2:	$\boldsymbol{\lambda}^{(0)} \leftarrow \text{supp}(\boldsymbol{\Phi}^T y, s)$	$\mathcal{O}_t(mn)$	$\mathcal{O}_s(s)$
3:	$\hat{\mathbf{x}}^{(0)} \leftarrow \text{argmin}_{\mathbf{u}:\text{supp}(\mathbf{u})=\boldsymbol{\lambda}^{(0)}} \|\|\mathbf{y} - \boldsymbol{\Phi}\mathbf{u}\|\|_2$	$\mathcal{O}_t(ms^2)$	$\mathcal{O}_s(n)$
4:	$\mathbf{r}^{(-1)} \leftarrow \mathbf{y}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(m)$
5:	$\mathbf{r}^{(0)} \leftarrow \mathbf{y} - \Phi\hat{\mathbf{x}}^{(0)}$	$\mathcal{O}_t(mn)$	$\mathcal{O}_s(m)$
6:	**while** $\mathbf{r}^{(k)} < \mathbf{r}^{(k-1)}$ and $k < s$ **do**	$\mathcal{O}_t(s)$	$\mathcal{O}_s(1)$
7:	$\quad k \leftarrow k + 1$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
8:	$\quad \mathbf{v}^{(k)} \leftarrow \boldsymbol{\Phi}^T \mathbf{r}^{(k-1)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(n)$
9:	$\quad \boldsymbol{\omega}^{(k)} \leftarrow \text{supp}(\mathbf{v}^{(k)},\ s)$	$\mathcal{O}_t(n \log n)$	$\mathcal{O}_s(s)$
10:	$\quad \boldsymbol{\lambda}^{(k)} \leftarrow \boldsymbol{\lambda}^{(k-1)} \cup \boldsymbol{\omega}^{(k)}$	$\mathcal{O}_t(s)$	$\mathcal{O}_s(1)$
11:	$\quad \hat{\mathbf{x}}^{(k)} \leftarrow \text{argmin}_{\mathbf{u}:\text{supp}(\mathbf{u})=\boldsymbol{\lambda}^{(k)}} \|\|\mathbf{y} - \boldsymbol{\Phi}\mathbf{u}\|\|_2$	$\mathcal{O}_t(ms^2)$	$\mathcal{O}_s(1)$
12:	$\quad \boldsymbol{\lambda}^{(k)} \leftarrow \text{supp}(\hat{\mathbf{x}}^{(k)}, s)$	$\mathcal{O}_t(n)$	$\mathcal{O}_s(1)$
13:	$\quad \hat{\mathbf{x}}^{(k)} \leftarrow \text{argmin}_{\mathbf{u}:\text{supp}(\mathbf{u})=\boldsymbol{\lambda}^{(k)}} \|\|\mathbf{y} - \boldsymbol{\Phi}\mathbf{u}\|\|_2$	$\mathcal{O}_t(ms^2)$	$\mathcal{O}_s(1)$
14:	$\quad \mathbf{r}^{(k)} \leftarrow \mathbf{y} - \Phi\hat{\mathbf{x}}^{(k)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(1)$
15:	**end while**		
16:	**return** $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}}^{(k)}$		
		$\mathcal{O}_t(snm + ms^3)$	$\mathcal{O}_s(n)$

3.3 Joint Compressed Sensing Problems

From the interest of utilizing compressed sensing for joint sparse signals, extensions of the reconstruction algorithms have been designed. Dennis Sundman, Saikat Chatterjee and Mikael Skoglund have modified Orthogonal Matching Pursuit and Subspace Pursuit [21] to tackle these problems. The modifications are very similar, for which only the Subspace Pursuit version will be included. This modification is relevant for distributed compressed sensing case.

3.3.1 Joint Subspace Pursuit (JSP)

Joint Subspace Pursuit was introduced by Sundman, Chatterjee and Skoglund in 2011 [21]. It is designed to expand Subspace Pursuit, for the more general multiple sensor system setups.
JSP consists of two primary changes; a new outer layer is added to the algorithm, and the actual algorithm now takes an initial support set.

The new outer layer, is iteratively calling a modified version of Subspace Pursuit, that can take side information. Based on the individual results of the modified Subspace Pursuit, voting is

used to deduct the common sparsity, which is assumed to be indices, pointed out by more than one modified Subspace Pursuit run. The iteration stop criterion of Subspace Pursuit is again used, however, it know considers the decrease of the sum of all the individual residuals.

Algorithm 3.3.1 JSP

Input:

- Measurement matrices $\{\Phi_l\}_{l=1}^{L} \in \mathbb{R}^{m \times n}$
- Observation vectors $\{\mathbf{y}_l\}_{l=1}^{L} \in \mathbb{R}^{m}$
- Sparsity level $s^{(c)}$ of the common part
- Sparsity levels $\{s_l^{(p)}\}_{l=1}^{L}$ of the private parts

Output:

- Estimates $\{\hat{\mathbf{x}}_l\}_{l=1}^{L} \in \mathbb{R}^{n}$ of the signals $\{\mathbf{x}\}_{l=1}^{L}$

1:	$k \leftarrow 0$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
2:	$\mathbf{r}^{(-1)} \leftarrow \infty$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(m)$
3:	$\mathbf{r}^{(0)} \leftarrow \sum_{l=1}^{L} \mathbf{y}_l$	$\mathcal{O}_t(mL)$	$\mathcal{O}_s(m)$
4:	$\boldsymbol{\lambda}^{(0)} \leftarrow \varnothing$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
5:	**while** $\mathbf{r}^{(k)} < \mathbf{r}^{(k-1)}$ and $k < s$ **do**	$\mathcal{O}_t(s)$	$\mathcal{O}_s(1)$
6:	$k \leftarrow k + 1$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
7:	$\mathbf{s}^{(k)} \leftarrow \mathbf{0}_{n \times 1}$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(n)$
8:	**for** $\forall l \in \{1, 2, ..., L\}$ **do**	$\mathcal{O}_t(L)$	$\mathcal{O}_s(1)$
9:	$(\hat{\mathbf{x}}_l^{(k)},\ \mathbf{r}_l^{(k)}) \leftarrow \text{modifiedSP}(\Phi_l,\ \mathbf{y}_l,\ s^{(c)} + s_l^{(p)},\ \boldsymbol{\lambda}^{(c)})$	$\mathcal{O}_t(snm + ms^3)$	$\mathcal{O}_s(n)$
10:	$\mathbf{s}^{(k)} \leftarrow \text{vote}(\mathbf{s}^{(k)},\ \text{supp}(\hat{\mathbf{x}}_l^{(k)}))$	$\mathcal{O}_t(s)$	$\mathcal{O}_s(1)$
11:	**end for**		
12:	$\mathbf{r}^{(k)} \leftarrow \sum_{l=1}^{L} \mathbf{r}_l^{(k)}$	$\mathcal{O}_t(mL)$	$\mathcal{O}_s(1)$
13:	$\boldsymbol{\lambda}^{(c)} \leftarrow \text{supp}(\mathbf{s}^{(k)},\ s^{(c)})$	$\mathcal{O}_t(n)$	$\mathcal{O}_s(1)$
14:	**end while**		
		$\mathcal{O}_t(s^2nm + ms^4)$	$\mathcal{O}_s(n)$

The modified Subspace Pursuit, only differs from Subspace Pursuit on two points. It take as input a side information support set, assumed to be the common support set. The algorithm then merge this support with its own initial estimate, before performing the initial signal estimate. After this it runs like Subspace Pursuit until the residual stops decreasing.

Algorithm 3.3.2 modified SP

Input:

- Measurement matrix $\Phi \in \mathbb{R}^{m \times n}$
- Observation vector $\mathbf{y} \in \mathbb{R}^m$
- Sparsity level s of the ideal signal $\mathbf{x} \in \mathbb{R}^n$
- Initial support set $\boldsymbol{\lambda}_{ini}$

Output:

- An estimate $\hat{\mathbf{x}} \in \mathbb{R}^n$ of the signal $\mathbf{x}$
- Remaining residual $\mathbf{r}$

1:	$l \leftarrow 0$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
2:	$\mathbf{v}^{(0)} \leftarrow \boldsymbol{\Phi}^T \mathbf{r}^{(l-1)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(n)$
3:	$\boldsymbol{\lambda}^{(0)} \leftarrow \text{supp}(\mathbf{v}^{(0)},\ s)$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(s)$
4:	$\boldsymbol{\lambda}^{(0)} \leftarrow \boldsymbol{\lambda}^{(0)} \cup \boldsymbol{\lambda}_{ini}$	$\mathcal{O}_t(s)$	$\mathcal{O}_s(1)$
5:	$\hat{\mathbf{x}}^{(0)} \leftarrow \text{argmin}_{\mathbf{u}:\text{supp}(\mathbf{u})=\boldsymbol{\lambda}^{(0)}} \|\|\mathbf{y} - \boldsymbol{\Phi}\mathbf{u}\|\|_2$	$\mathcal{O}_t(ms^2)$	$\mathcal{O}_s(n)$
6:	$\mathbf{r}^{(0)} \leftarrow \mathbf{y} - \Phi\hat{\mathbf{x}}^{(0)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(m)$
7:	**while** $\mathbf{r}^{(l)} < \mathbf{r}^{(l-1)}$ and $k < s$ **do**	$\mathcal{O}_t(s)$	$\mathcal{O}_s(1)$
8:	$\quad l \leftarrow l + 1$	$\mathcal{O}_t(1)$	$\mathcal{O}_s(1)$
9:	$\quad \mathbf{v}^{(l)} \leftarrow \boldsymbol{\Phi}^T \mathbf{r}^{(l-1)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(n)$
10:	$\quad \boldsymbol{\omega}^{(l)} \leftarrow \text{supp}(\mathbf{v}^{(l)},\ s)$	$\mathcal{O}_t(n \log n)$	$\mathcal{O}_s(s)$
11:	$\quad \boldsymbol{\lambda}^{(l)} \leftarrow \boldsymbol{\lambda}^{(l-1)} \cup \boldsymbol{\omega}^{(l)}$	$\mathcal{O}_t(s)$	$\mathcal{O}_s(1)$
12:	$\quad \hat{\mathbf{x}}^{(l)} \leftarrow \text{argmin}_{\mathbf{u}:\text{supp}(\mathbf{u})=\boldsymbol{\lambda}^{(l)}} \|\|\mathbf{y} - \boldsymbol{\Phi}\mathbf{u}\|\|_2$	$\mathcal{O}_t(ms^2)$	$\mathcal{O}_s(1)$
13:	$\quad \boldsymbol{\lambda}^{(l)} \leftarrow \text{supp}(\hat{\mathbf{x}}^{(l)}, s)$	$\mathcal{O}_t(n)$	$\mathcal{O}_s(s)$
14:	$\quad \hat{\mathbf{x}}^{(l)} \leftarrow \text{argmin}_{\mathbf{u}:\text{supp}(\mathbf{u})=\boldsymbol{\lambda}^{(l)}} \|\|\mathbf{y} - \boldsymbol{\Phi}\mathbf{u}\|\|_2$	$\mathcal{O}_t(ms^2)$	$\mathcal{O}_s(1)$
15:	$\quad \mathbf{r}^{(l)} \leftarrow \mathbf{y} - \Phi\hat{\mathbf{x}}^{(l)}$	$\mathcal{O}_t(nm)$	$\mathcal{O}_s(m)$
16:	**end while**		
		$\mathcal{O}_t(snm + ms^3)$	$\mathcal{O}_s(n)$

4
DISTRIBUTED COMPRESSED SENSING

In this section, some expansions to Compressed Sensing for use in distributed systems will be ...

4.1 Compressive Data Gathering (CDG)

Compressive Data Gathering is a method of applying compressed sensing to data gathering in large-scale wireless sensor networks, introduced by Luo, Wu, Sun and Chen in 2009 [29].

The method is rooted in seeing the signal as distributed in space, rather than in time. Thus multiple sensors are used in acquiring the signal and relaying it to a sink. This can be combined with the use of a measurement matrix, by letting each sensor know a given column. A sensor then multiplies its reading with each entry. The resulting vector must be added to a possible incoming vector, and then forwarded. Then the sink receives the measurement vector, as if the signal, had been multiplied by the measurement vector locally. [29]

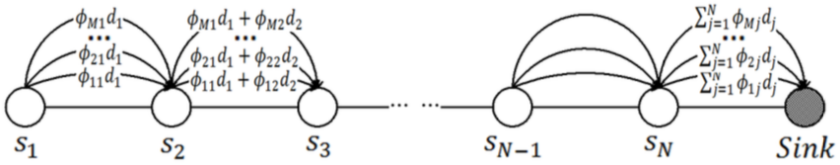


Figure 4.1.1. Compressive Data Gathering way of compressing while forwarding through a chain of nodes. Each note multiplies its own data with an assigned column of the measurement matrix, then this is added to the possible incomming data from child notes and forward. When the data reaches the sink this would have the same result as multiplying the entire data collection with the measurement matrix, thus allowing the sink to reconstruct the signal. (Source: [29])

4.1.1 Application

4.1.1.1 Topology fit

From the description of CDG, it is well designed to fit onto a chain topology. From figure 4.1.1 this clearly reduces the overall traffic, as long as $m \ll n$, as well as load balancing as all nodes have to transmit equal amounts of data.
It can be extended to fit onto a tree-topology, by letting intermediate nodes, know of their child notes, and accumulate all their signals and add its own, before forwarding.
Generally, CDG requires the nodes to know of their neighbors, so some kind of subscribe/unsubscribe protocol would be needed.

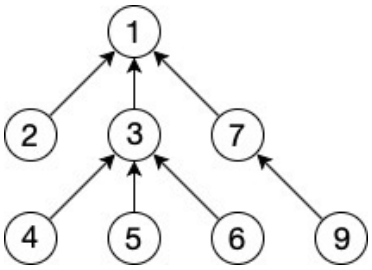
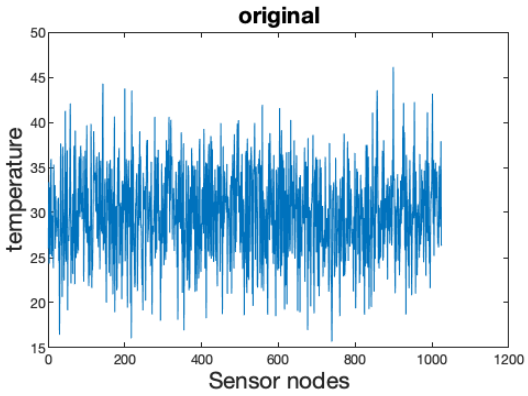


Figure 4.1.2. Simple tree structure

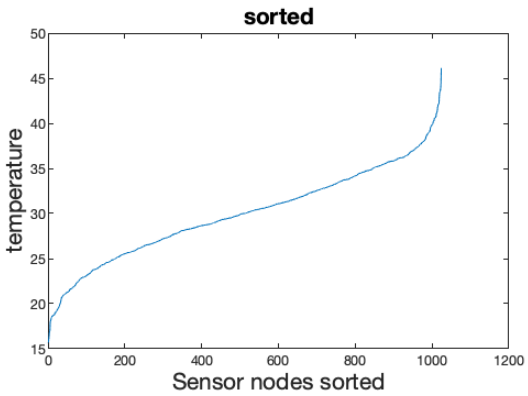
4.1.1.2 Examples

Ocean Temperature Data
The authors of [29] include a real-life example, involving temperature readings at different levels of the ocean. They then show how this data is nearly sparse in the wavelet domain and how well it can be reconstructed after compressed sensing.

Data center temperature
A more interesting example is looking at real-life temperature data from a data center. In this case, when the data is plotted, it does not seem sparse in any intuitive basis. But by sorting the data points, in ascending order, it becomes piece-wise smooth and thus sparse in the wavelet domain. As the authors point out, this kind of temperature readings does not change violently, which implies that new sensor reading, taken within a given time limit, will also be sparse in the wavelet domain, when sorted in the same order.
This makes CDG applicable, as the sensors only do multiplication and addition in a "blind" way, and when the reordering scheme as determined for a given interval, CDG can be applied to all following measurements, as the sink only has to order the respective measurement matrix columns in the same way. As the authors point out, this kind of reshuffling to obtain sparsity is not possible in conventional ways.
This applies to a wide range of wireless sensor networks, where the detection of fluctuations in data is the goal. A given system only needs to reestimate the recording with some intervals, as a calibration.



(a) Temperature readings sorted according to sensor deployment structure. This inherently seems extremely noise, thus not being sparse in any well known basis. This implies that genral transform coding strategies and distributed source coding cannot be applied



(b) Temperature readings sorted according to temperature values. This makes the readings piecewise smooth, thus compressed sensing can be applied, assuming that the sorting order is known. The data can then be reconstructed if the columns of the measurement matrix is sorted, according to the determined sorting order of the temperature readings

Figure 4.1.3. Simulation of temperature readings at distributed notes, similar to the example of [29]

4.2 Distributed Parallel Pursuit (DIPP)

Distributed Parallel Pursuit was introduced by Dennis Sundman, Saikat Chatterjee and Mikael Skoglund in 2013 [30, 31]. DIPP is based on Subspace pursuit, with added side-information. Overall it consists of four parts, performed in iterations; Parallel Pursuit with Side Information, Consensus and Expansion, which will all be discussed and analyzed more in depth later, and a Transmit/Receive part, were currently estimated support sets are communicated between nodes, as a rendezvous point.

DIPP starts out by evaluating its own measurement and estimate its support set (2), by using the SIPP algorithm, without side information, which reduces it to Subspace pursuit. Then iteratively improves upon the estimate. The first thing in the loop is an exchange of estimated support sets, with neighboring nodes (5 and 6), as a rendezvous point. Then the consensus algorithm is used on all the acquired support sets, to estimate the common support set (7). The estimated common support set is then used along with the estimated signal to expand the support set to include the innovation part (8). Lastly, SIPP is called, now with side information, consisting of the estimated common support set.

Algorithm 4.2.1 Distributed Parallel Pursuit (DIPP)

Input:

- Measurement matrix $\mathbf{\Phi}_p \in \mathbb{R}^{m \times n}$ for the node
- Observation vector $\mathbf{y}_p \in \mathbb{R}^m$ for the node
- Sparsity level s of the ideal signal $\mathbf{x}_p \in \mathbb{R}^n$
- Set $\mathcal{L}_p^{in}$ of incoming connections from other nodes
- Set $\mathcal{L}_p^{out}$ of outgoing connections to other nodes

Output:

- An estimate $\hat{\mathbf{x}}_p \in \mathbb{R}^n$ of the signal $\mathbf{x}$

1: $k \leftarrow 0$
2: $(\hat{\mathbf{x}}_p^{(k)},\ \hat{\boldsymbol{\lambda}}_p^{(k)},\ \mathbf{r}_p^{(k)}) \leftarrow \mathsf{SIPP}(\mathbf{y}_p,\ \mathbf{\Phi}_p,\ s,\ \varnothing)$
3: **while** Stopping criterion not met **do**
4: $\quad k \leftarrow k + 1$
5: $\quad$ **Transmit:** $\hat{\boldsymbol{\lambda}}_p^{(k-1)}$ to all nodes $p \in \mathcal{L}_p^{out}$
6: $\quad$ **Receive:** $\hat{\boldsymbol{\lambda}}_q^{(k)}$ from all nodes $q \in \mathcal{L}_p^{in}$
7: $\quad \hat{\boldsymbol{\lambda}}_{p,c}^{(k)} \leftarrow \mathsf{consensus}(\{\hat{\boldsymbol{\lambda}}_q^{(k)}\}_{q \in \mathcal{L}_p^{in}},\ \hat{\boldsymbol{\lambda}}_p^{(k-1)}, s)$
8: $\quad \hat{\boldsymbol{\lambda}}_{p,si}^{(k)} \leftarrow \mathsf{expansion}(\hat{\boldsymbol{\lambda}}_{p,c}^{(k)},\ \hat{\mathbf{x}}_p^{(k-1)},\ s)$
9: $\quad (\hat{\mathbf{x}}_p^{(k)},\ \hat{\boldsymbol{\lambda}}_p^{(k)},\ \mathbf{r}_p^{(k)}) \leftarrow \mathsf{SIPP}(\mathbf{y}_p,\ \mathbf{\Phi}_p,\ s,\ \hat{\boldsymbol{\lambda}}_{p,si}^{(k)})$
10: **end while**
11: **return** $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}}_{p,k}$

4.2.1 Parallel Pursuit with Side Information (SIPP)

The SIPP algorithm is an extension of Subspace pursuit, able to also include side information, as the common support set estimate. The biggest difference between SIPP and Joint Subspace Pursuit is that SIPP incorporates the side information in the loop, whereas Joint Subspace pursuit only considers it at the initialization.

The algorithms execute much like Subspace Pursuit but add lines 11 and 12. The side information support set is merged with the estimated support set each iteration (line 11). Then an extra least squares step is taking (line 12) before pruning the signal to have the given support set size.

Algorithm 4.2.2 Parallel Pursuit with Side Information (SIPP)

Input:

- Measurement matrix $\mathbf{\Phi} \in \mathbb{R}^{m \times n}$ for the node
- Observation vector $\mathbf{y} \in \mathbb{R}^m$ for the node
- Sparsity level s of the ideal signal $\mathbf{x} \in \mathbb{R}^n$
- Side information $\hat{\boldsymbol{\lambda}}_{p,si}$, containing indexes from previous iterations

Output:

- An estimate $\hat{\mathbf{x}} \in \mathbb{R}^n$ of the signal $\mathbf{x}$
- $\hat{\boldsymbol{\lambda}}$ non-zeros indexes of $\hat{\mathbf{x}}$
- Remaining residual $\mathbf{r}$

$l \leftarrow 0$
$\mathbf{r}^{(0)} \leftarrow \mathbf{y}$
$\hat{\boldsymbol{\lambda}}^{(0)} \leftarrow \mathbf{0}_{2s \times 1}$
$\hat{\mathbf{x}}^{(0)} \leftarrow \varnothing$
while Stopping criterion not met **do**
    $l \leftarrow l + 1$
    $\boldsymbol{\omega}^{(l)} \leftarrow \text{supp}(\mathbf{\Phi}^T \mathbf{r}^{(l-1)},\ s)$
    $\hat{\boldsymbol{\lambda}}^{(l)} \leftarrow \hat{\boldsymbol{\lambda}}^{(l-1)} \cup \boldsymbol{\omega}^{(l)}$
    $\bar{\mathbf{x}}^{(l)} \leftarrow \text{argmin}_{\mathbf{u}:\text{supp}(\mathbf{u})=\hat{\boldsymbol{\lambda}}^{(l)}} ||\mathbf{y} - \mathbf{\Phi}\mathbf{u}||_2$
    $\boldsymbol{\omega}^{(l)} \leftarrow \text{supp}(\bar{\mathbf{x}}^{(l)},\ s)$
    $\hat{\boldsymbol{\lambda}}^{(l)} \leftarrow \hat{\boldsymbol{\lambda}}_{p,si} \cup \boldsymbol{\omega}^{(l)}$
    $\bar{\mathbf{x}}^{(l)} \leftarrow \text{argmin}_{\mathbf{u}:\text{supp}(\mathbf{u})=\hat{\boldsymbol{\lambda}}^{(l)}} ||\mathbf{y} - \mathbf{\Phi}\mathbf{u}||_2$
    $\hat{\boldsymbol{\lambda}}^{(l)} \leftarrow \text{supp}(\bar{\mathbf{x}}^{(l)},\ s)\ 1$
    $\hat{\mathbf{x}}^{(l)} \leftarrow \text{argmin}_{\mathbf{u}:\text{supp}(\mathbf{u})=\hat{\boldsymbol{\lambda}}^{(l)}} ||\mathbf{y} - \mathbf{\Phi}\mathbf{u}||_2$
    $\mathbf{r}^{(l)} \leftarrow \mathbf{y} - \mathbf{\Phi}\bar{\mathbf{x}}^{(l)}$
end while
return

4.2.2 Consensus

The consensus algorithm is responsible for taking the estimated support of the present node $\hat{\boldsymbol{\lambda}}_p$ and neighboring nodes $\{\hat{\boldsymbol{\lambda}}_q\}_{q \in \mathcal{L}_p^{in}}$, and estimate the common support set, with an upper limited support set size s. This is done by voting. Initially, a zero vector, with length equal to the signal is created (1), as the vote vector. First, the nodes own support set estimate is considered (2), the voting is done by taking the voting vector and a support set, and increment the coefficients at all the indices, indicated by the support set. Then the algorithm iterates over all the L incoming nodes support set (3), and perform the same voting for them (4). Finally, the common support set is formed, by selecting all the indices from the voting vector that is higher than or equal to two, which indicates the index might be in the true support set. Should the number of selected indices exceed the commin support set sizes, then it only selects the indices of the s highest coefficients, as indices with more votes, have a higher probability of being in the true support set.

Algorithm 4.2.3 consensus

Input:

- Nodes own support set $\hat{\boldsymbol{\lambda}}_p$
- Neighbour nodes support sets $\{\hat{\boldsymbol{\lambda}}_q\}_{q\in\mathcal{L}_p^{in}}$
- Support set size s

Output:

- Estimated common support set $\hat{\boldsymbol{\lambda}}_{p,c}$

1: $\mathbf{z} \leftarrow \mathbf{0}_{n\times 1}$
2: $\mathbf{z} \leftarrow \text{vote}(\mathbf{z},\ \hat{\boldsymbol{\lambda}}_p)$
3: **for** each $q \in \mathcal{L}_p^{in}$ **do**
4: $\quad \mathbf{z} \leftarrow \text{vote}(\mathbf{z},\ \hat{\boldsymbol{\lambda}}_q)$
5: **end for**
6: **return** $\hat{\boldsymbol{\lambda}}_{p,c}$ s.t. $(z(i) \geq 2)\forall i \in \hat{\boldsymbol{\lambda}}_{p,c}$ and $|\hat{\boldsymbol{\lambda}}_{p,c}| \leq s_c$

4.2.3 Expansion

The expansion algorithm is responsible for taking the common support set estimate $\hat{\boldsymbol{\lambda}}_{p,c}$ from consensus, as well as the nodes, own signal estimate $\hat{\mathbf{x}}_p$, and expend the common support set to also cover the innovation part. This is done by initially setting the indices of the estimated signal, pointed out by the common support set, equal to 0 (1). Then a number of indices, equal to the difference between the estimated common support set size and total support set size, is selected, based on the highest coefficients. Lastlyn the common and innovation support sets are merged, and due to line 1, they would not have any shared elements.

Algorithm 4.2.4 expansion

Input:

- Estimated common support set $\hat{\boldsymbol{\lambda}}_{p,c}$
- Estimated own signal $\hat{\mathbf{x}}_p$
- Sparsity s

Output:

- $\hat{\boldsymbol{\lambda}}_{p,si}$

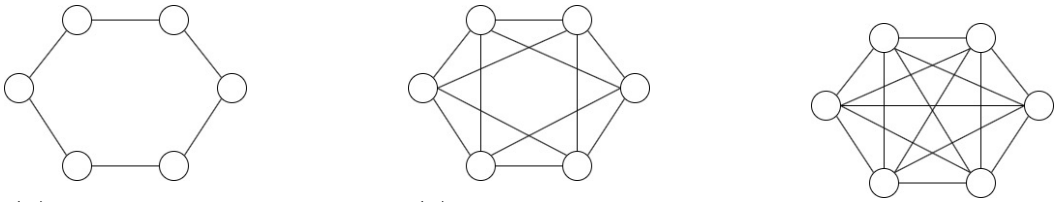
1: $(\hat{\mathbf{x}}_p)_{\hat{\boldsymbol{\lambda}}_{p,c}} \leftarrow 0$
2: $\hat{\boldsymbol{\lambda}}_{p,i} \leftarrow \text{supp}(\hat{\mathbf{x}}_p,\ s - |\hat{\boldsymbol{\lambda}}_{p,c}|)$
3: $\hat{\boldsymbol{\lambda}}_{p,si} \leftarrow \hat{\boldsymbol{\lambda}}_{p,c} \cup \hat{\boldsymbol{\lambda}}_{p,i}$
4: **return**

4.2.4 Application

The usability of DIPP will be discussed below. First, the overall topology fit of the algorithm will be discussed followed by application examples.

4.2.4.1 Topology fit

Distributed Parallel Pursuit fits onto ring and mesh network topologies, in structured and unstructured ways. The structured networks can be seen in figure 4.2.1, and have different degrees of connectivity, with connectivity one being a ring. The unstructed networks are general mesh networks as shown in figure 4.2.2
The most important part related to the topology is the connectivity, which might give a tradeoff between convergence speed and communication need for each iteration. When increasing the connectivity, more data will be sent between the nodes each iteration, thus more information is available for the consensus step, causing the algorithm to converge faster. Thus an important question in designing a network running DIPP would be what degree of connectivity would give the least amount of total communication $MessagesPrIteration * Iterations$.



(a) Connectivity of one

(b) Connectivity of two

(c) Full connectivity

Figure 4.2.1. Ring topologies, with different connectivity

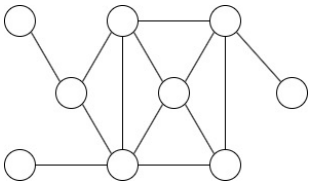


Figure 4.2.2. General mesh topology

4.2.4.2 Examples

DIPP can be applied to any dsitributed setup, where individual notes has aquire data that is correlated with the data of other notes. The true benefit of using DIPP is when the connection between notes is inexpensive compared to communication to a sink. As the notes in collaboration can estimate the support set, and then only send these to the far away sink. This is rather similar to using transform coding on the notes, as the notes in collaboration performs a form of compression on the data, and can achieve compression down to s rather than m.

4.3 Common-Innovation Subspace Pursuit (CISP)

Common-Innovation Subspace Pursuit was introduced by Liu, Huang and Yao in 2019 [32].
At the basis CISP utilizes a Matching Pursuit algorithm to estimate the innovation supports signals, however, the access to multiple joint signals is used to rapidly estimate the common support set, without iteration. The contribution to the signals from the common support set is then removed by projections, creating a smaller scale problem to be solved by the underlying matching pursuit algorithm.

CISP takes all the signals $\{\{\mathbf{y}_j^k\}_{k=1}^{K_j}\}_{j=1}^{J}$ and the related measurement matrices $\{\{\mathbf{\Phi}_j^k\}_{k=1}^{K_j}\}_{j=1}^{J}$, of all the J clusters with sizes of K_j, as well as knowledge of the size of the common support set s_c and innovation support set s_i.
In CISP the measurements are seen as coming from different clusters, though this not strictly necessary as it can be simulated. The algorithm starts out by averaging the measurement of the different clusters and combining these averages as columns in a new matrix(1), this both reduces noise and suppresses the innovation parts for the individual cluster. Then the a average measurement matrix is formed by averaging over all the measurement matrices. The algorithm then performs Eigen-value decomposition of the matrix formed by the averaged measurement and select the Eigen-vectors related to the s_c largest eigenvalues (3). These eigenvectors are then used with the columns of the averaged measurement matrix, in order to obtain the common support set, as the vectors in the averaged measurement matrix with the smallest cosine distance to the eigenspace spanned by the selected eigenvectors. An iteration is then performed for each individual measurement. Here the individual measurements and measurement matrices are projected into a subspace, to remove the influence of the common subspace. A matching pursuit algorithm is then performed on each of these projections in order to estimate the innovation part of each measurement. This is followed by an estimation of the coefficient related to the common support set part using the Moore-Penrose Pseudo inverse.

Algorithm 4.3.1 Common-Innovation Subspace Pursuit (CISP)

Input:

- Measurment vectors $\{\{\mathbf{y}_j^k\}_{k=1}^{K_j}\}_{j=1}^{J}$, from all the nodes, of the J cluster with sizes K_J
- Measurement matrices $\{\{\mathbf{\Phi}_j^k\}_{k=1}^{K_j}\}_{j=1}^{J}$, from all the nodes, of the J cluster with sizes K_J
- Common sparsity size s_c
- Innovation sparsity size s_i

Output:

- $\{\{\hat{\mathbf{x}}_j^k\}_{k=1}^{K_j}\}_{j=1}^{J}$

$\mathbf{Y} \leftarrow [\bar{\mathbf{y}}_1, \bar{\mathbf{y}}_2, ..., \bar{\mathbf{y}}_J]$, where $\bar{\mathbf{y}}_j = \frac{1}{K_j}\sum_{k=1}^{K_j}\mathbf{y}_j^k$
$\bar{\mathbf{\Phi}} \leftarrow \frac{1}{N_{node}}\sum_{j=1}^{J}\sum_{k=1}^{K_j}\mathbf{\Phi}_j^k$
$\mathbf{V} \leftarrow \mathrm{EVD}(\mathbf{Y}\mathbf{Y}^T)$
$\mathbf{C} \leftarrow 1 - \frac{||\mathbf{V}^T\phi_j||}{||\phi_j||}$
for $j = 1, 2, ..., J$ **do**
    for $k = 1, 2, ..., K_j$ **do**
        $\mathbf{P}_j^k \leftarrow \mathbf{I} - \left(\mathbf{\Phi}_j^k\right)_C \left[\left(\mathbf{\Phi}_j^k\right)_C^T \left(\mathbf{\Phi}_j^k\right)_C\right]^{-1} \left(\mathbf{\Phi}_j^k\right)_C^T$
        $\bar{\mathbf{y}}_j^k \leftarrow \mathbf{P}_j^k \mathbf{y}_j^k$
        $\bar{\mathbf{\Phi}}_j^k \leftarrow \mathbf{P}_j^k \left(\mathbf{\Phi}_j^k\right)_{C^c}$
        $\left(\hat{\mathbf{x}}_j^k\right)_{C^c} \leftarrow \mathrm{CoSaMP}(\bar{\mathbf{\Phi}}_j^k,\ \bar{\mathbf{y}}_j^k,\ s_i)$
        $\left(\hat{\mathbf{x}}_j^k\right)_C \leftarrow \left(\mathbf{\Phi}_j^k\right)_C^{\dagger} \left[\mathbf{y}_j^k - \left(\mathbf{\Phi}_j^k\right)_{C^c} \left(\hat{\mathbf{x}}_j^k\right)_{C^c}\right]$
    end for
end for
return $\{\{\hat{\mathbf{x}}_j^k\}_{k=1}^{K_j}\}_{j=1}^{J}$

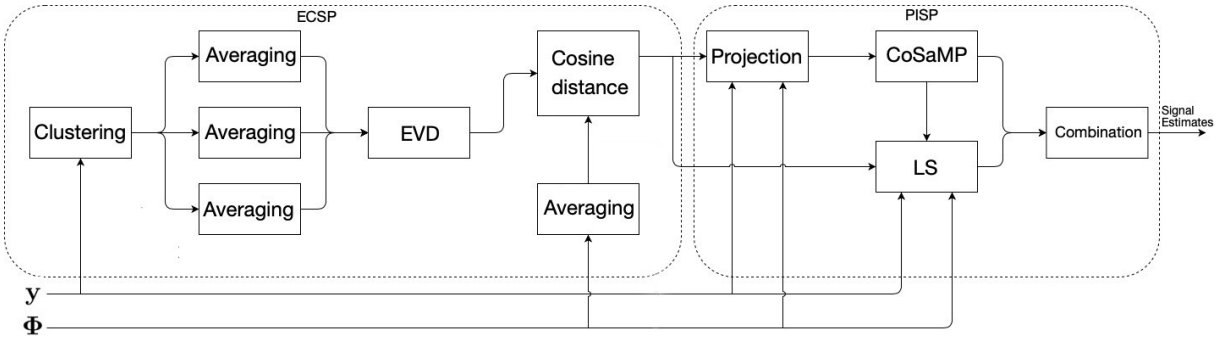


Figure 4.3.1. Activity diagram for the CISP algorithm. The ECSP section is concerned with estimation of the common support set, and the PISP section is concerned with estimating the innovation support sets. In the ECSP part, the measurements are divided into clusters, and within the clusters the average measurement is calculated. ECSP also average all the measurement matrices. These averages is then used in an eigen values decomposition. The common support set is then estimated based on the cosine distances of the columns of the averaged measurement matrix and the eigen vectors assoicated with the with the highest eigenvalues. Based on the common support set estimate of ECSP, PISP performs projection of the measurements in order to remove the contributions of the common support set. These projection are then used for estimating the innovation support set uisng CoSaMP. Finally the estiamted innovation support sets are combined with the common support set, as the final result.

As can be seen in figure 4.3.1, the algorithm can be seen as two part.

4.3.1 EVD based Common Subspace Pursuit (ECSP)

The eigenvalue decomposition based Common Subspace Pursuit is the part that is concerned with estimating the common support set, based on eigenanalysis of the covariance matrix of the averaged measurements. Line 1-4 of algorithm 4.3.1, the part before the for loops.

4.3.2 Projection based Innovation Subspace Pursuit (PISP)

The Projection based Innovation Subspace Pursuit is the part that is concerned with estimating the innovation support set, based on projections of the measurement and measurement matrix, which aims at removing the influence of the common part. Line 5-13 of the algorithm 4.3.1, the part within the for loops.

4.3.3 Application

The usability of CISP will be discussed below. First, the overall topology fit of the algorithm will be discussed followed by application examples.
An important factor in the usage of CISP is that two measurement matrices are needed at each node. A common part, that all nodes know of, and an innovation part that is unique to each node. The nodes then add these two matrices together and use this as the measurement matrix.

4.3.3.1 Topology fit

CISP is designed to be applicable to a star-of-stars topology. For these kinds of topologies that different parts of the algorithms can be distributed to different parts of the networks, as will be discussed below. The fact that CISP is applicable to a star-of-stars topologies, makes it highly

scaleable compared to DIPP, where propagation of support set can increase the convergence time, as propagating the support set estimation between notes that cannot reach each is other might increase iterations and thus make the algorithm less effective.

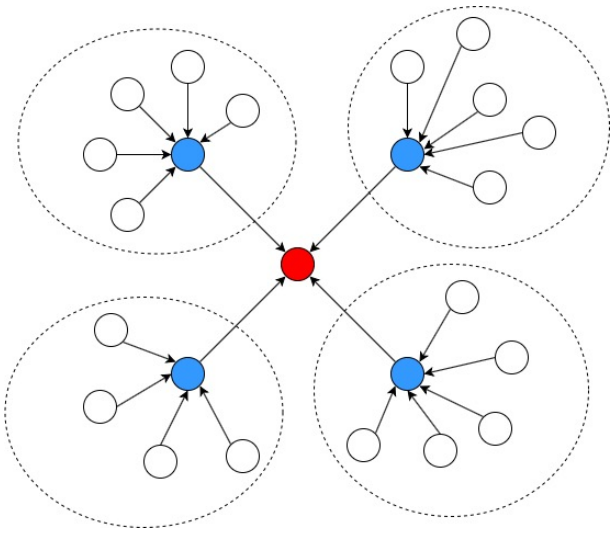


Figure 4.3.2. Star-of-stars topology. Colorless notes are sensors, blue notes are cluster headers and the red note is the sink

The structure of the algorithm allows for different distribution of the different parts, as stated above. The methods mentioned in the article [32], puts the cluster measurements averaging on the cluster heads, which sends these averages to the sink, which then perform the eigenanalysis to estimate the common support set. The common support set estimate is then sent back to each node, which then individually performs the PISP part to estimate their own innovation support set, which is then send back to the sink. Other alternative setups could be to send all the measurements to the sink to then perform the entire algorithm, or the PISP parts could be performed on the cluster headers.

Due to the flexibility of distributing parts of the algorithm, CISP is applicable to a wide range of networks. The network does not necessarily have to be following a star-of-stars topology, as the sink is free to take incoming measurements and set them up as if they were coming from different clusters, in order to use the algorithm.

5
NUMERICAL EXPERIMENTS AND RESULTS

In this section the performed experiments will explained, as well as their purpose. Additionally the results will be shown and explained.
The experiments will be using new implementation of the algorithms, which have been writen and tested based on the knowledge gained through the analysis in chapter 3 and 4.

As a general way of illustrating the results phase diagrams will be used [10, 32]. Phase diagrams are 2D graphics that allows the results to be discussed as regions based on selected coordinate measurements. This allows for a more overall result discussion as well as providing an efficient way of comparing different results. As the mostly used example phase diagrams will be presented, with coordinates measuring relative sparsity and compression rate. These graphics will be used for plotting both estimated reconstruction accuracy as well as iterations used for reconstruction. Thus reconstruction can be discussed across the algorithms in terms of region of reconstruction. As well as determined how many iteration a given algorithm requires, when in the region where reconstruction is possible.

5.1 Matching Pursuit Experiments

The following tests are ment to verify the workings of the written implementation of the Matching Pursuit algorithms. Additionally, the results will be used to argument further limitations on the algorithms, than those of chapter 3 in order to improve upon the complexities.
The implementations used can be found in appendix A, and are all using the Moore-Penrose pseudo-inverse, which was found sufficient as the results of interest will be correct reconstruction and iteration taken.

5.1.1 Data Dimensions and Reconstruction

Protocol

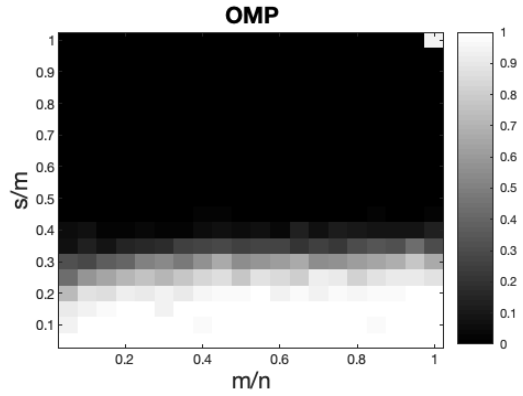
Purpose	Verify the working of the implemented algorithms, as well as evaluate the reconstruction performance, with respect to different relation between m and n and between m and s.
Data	Three test data setups, all with i.i.d. sampled support set from $\mathcal{U}(1, n)$. Setup 1) Coefficients i.i.d. sampled from $\mathcal{N}(0, 1)$. Setup 2) Flat signals, i.e. coefficients set to one. Setup 3) like 1, but with added white gaussian noise, with an SNR of 80.
Success criterion	Exact reconstruction. Support set of estimate is the same as that of the generated signal
Algorithms	OMP, StOMP, ROMP, CoSaMP, SP and gOMP.
Notes	In order to evaluate and compare the algorithms phase diagrams will be used, showing regions in which the algorithms can reconstruct correctly with high probability, with coordinates measure relative sparsity and compression rate. For all of the performed testes $n = 400$, and for every value combination of s and m 100 trials was performed and averaged.

Table 5.1.1. Test protocol: Data dimensions and reconstruction

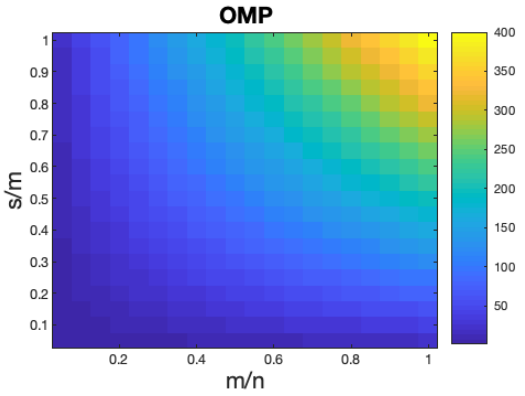
5.1.1.1 Results

In the following the results for the indivdual algorithms will be shown and described.

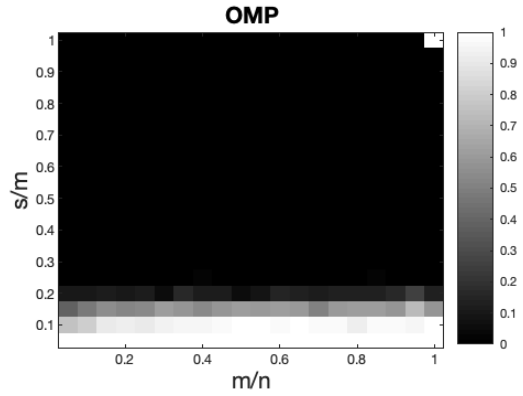
Orthogonal Matching Pursuit



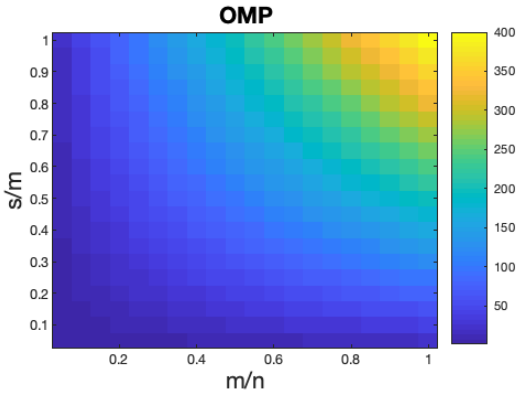
(a) Reconstruction accuracy for case 1



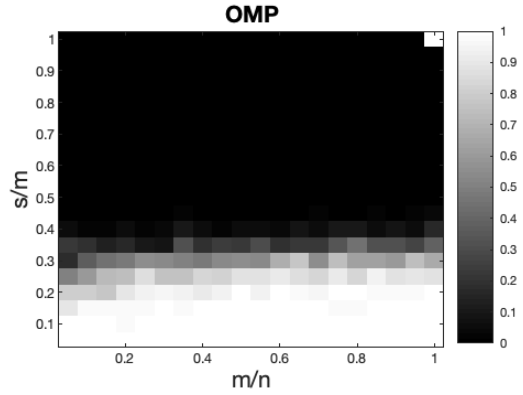
(b) Iterations for case 1



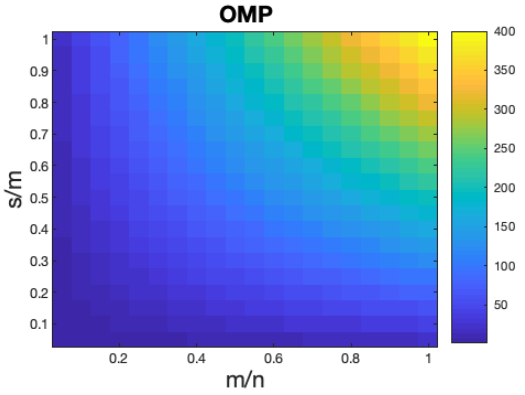
(c) Reconstruction accuracy for case 2



(d) Iterations for case 2



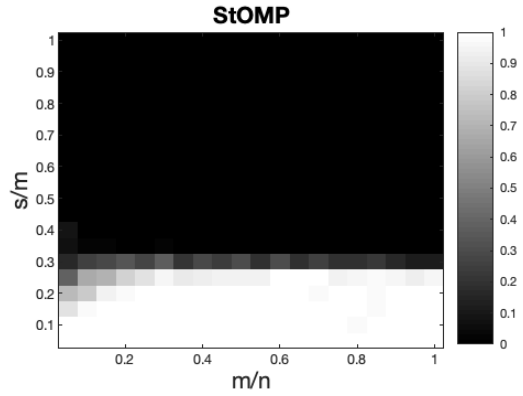
(e) Reconstruction accuracy for case 3



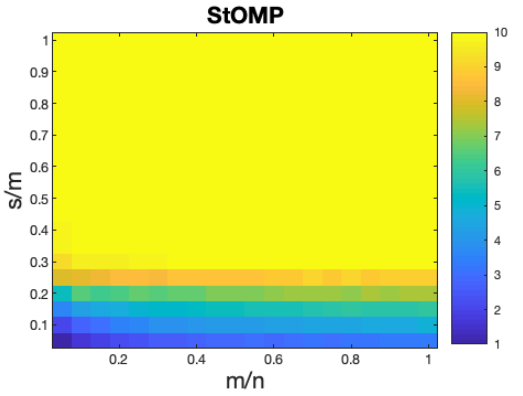
(f) Iterations for case 3

Figure 5.1.1. Performance of OMP, illustrated through phase diagrams with coordinates of relative sparsity and compression rate, the plots uses either estimated reconstruction accuracy or iterations taken to form different regions.. Setup 1 (a and b) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients i.i.d. sampled from $\mathcal{N}(0,1)$. Setup 2 (c and d) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients set to one. Setup 3 (e and f) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients i.i.d. sampled from $\mathcal{N}(0,1)$ and with added white noise gaussian noise with an SNR of 80.

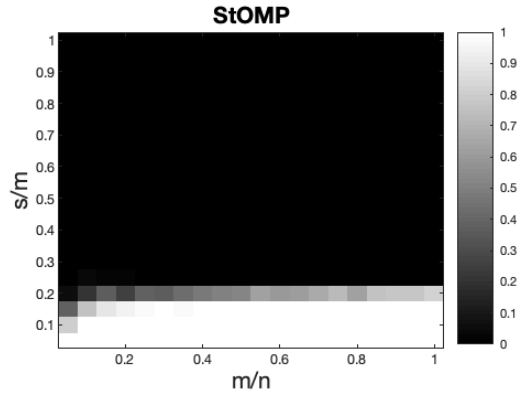
The results of figure 5.1.1 indicates that for signals with gaussian coefficients Orthogonal Matching Pursuit can reconstruct the signals when the relation between s and m is below $0,3$, and below $0,15$ for flat sigals. When the gaussian signal is contaminated with noise, the phase diagram shows equal regions as the noiseless case. This indicates a bound of $m = \mathcal{O}(s \log n)$, which would be a horizontal line in the phase diagrams.
Iteration wise the results follow the expected trend of s iterations required.



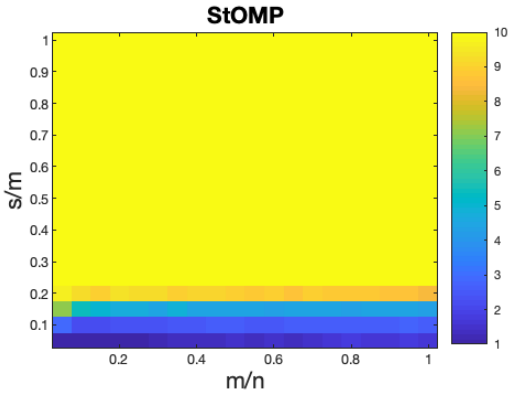
(a) Reconstruction accuracy for case 1



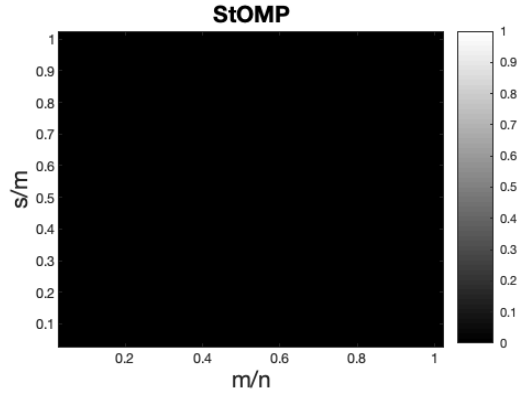
(b) Iterations for case 1



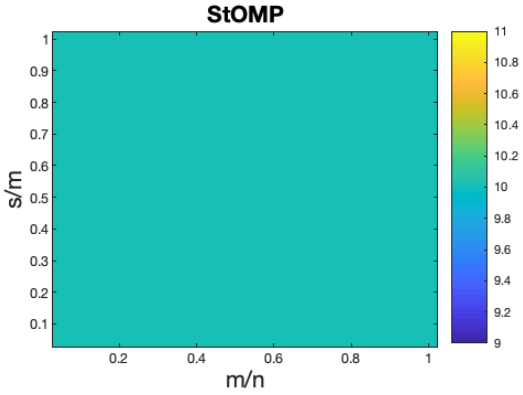
(c) Reconstruction accuracy for case 2



(d) Iterations for case 2



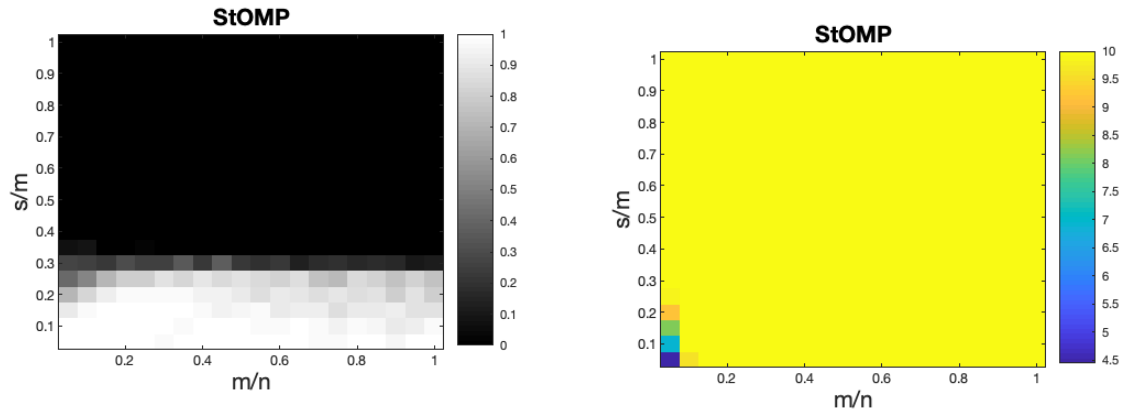
(e) Reconstruction accuracy for case 3



(f) Iterations for case 3

Figure 5.1.2. Performance of StOMP, illustrated through phase diagrams with coordinates of relative sparsity and compression rate, the plots uses either estimated reconstruction accuracy or iterations taken to form different regions.. Setup 1 (a and b) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients i.i.d. sampled from $\mathcal{N}(0,1)$. Setup 2 (c and d) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients set to one. Setup 3 (e and f) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients i.i.d. sampled from $\mathcal{N}(0,1)$ and with added white noise gaussian noise with an SNR of 80.

The results of figure 5.1.2 indicates that for signals with gaussian coefficients Stagewise Orthogonal Matching Pursuit can reconstruct the signals when the relation between s and m is below $0,3$, and below $0,15$ for flat sigals. This is very similar to Orthogonal Matching Pursuit, though with much sharper cutoff regions. When the gaussian signal is contaminated with noise, the phase diagram shows that Stagewise Orthogonal Matching Pursuit is completely unable to reconstruct the signal. However as can be seen on the results in figure 5.1.3, this can be greatly improved by adding a pruning step at the end, like is used for Compressive Sampling Matching Pursuit and Subspace Pursuit. As for Orthogonal Matching Pursuit, this is similar to the result of the noiseless case, though with some degradation is accuracy, though especially when the relation between m and n is above $0,5$. This change in accuracy, when adding pruning, is found to be due to overestimation in the noisy case.
Iteration wise the results shows that Stagewise Orthogonal Matching Pursuit is upper bound by the 10 allowed iterations. For the two cases without noise, Stagewise Orthogonal Matching Pursuit is able to reconstruction the result in less iteration.
The phase diagram in figure 5.1.4, shows the number of selected indices for different s and m value pairs, in the noisy case(3). It shows how the number of selected indices, in the region of which reconstruction was possible when adding a pruning step, is still very close to s. Additionally it shows that for the region where reconstruction was not possible, it does not get close to the real support set size.



(a) Reconstruction accuracy for case 3, with added pruning step

(b) Iterations for case 3, with added pruning step

Figure 5.1.3. StOMP, measured by reconstruction accuracy and iterations taken. For case 3, noisy measurements, and with adding pruning step at the end.

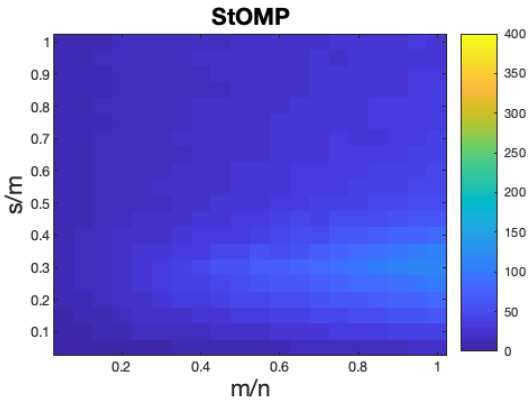
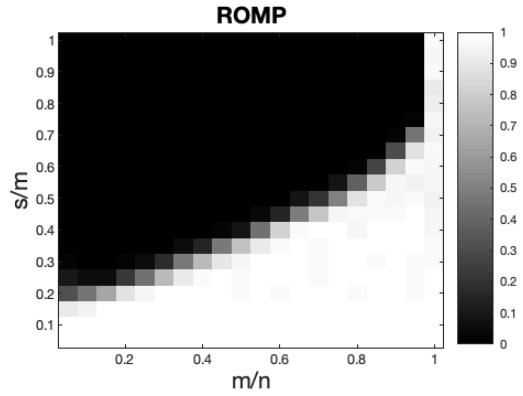
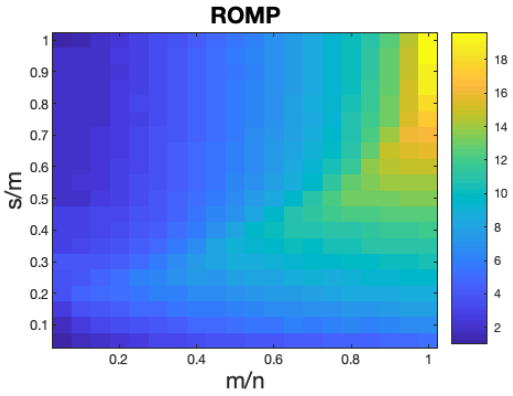


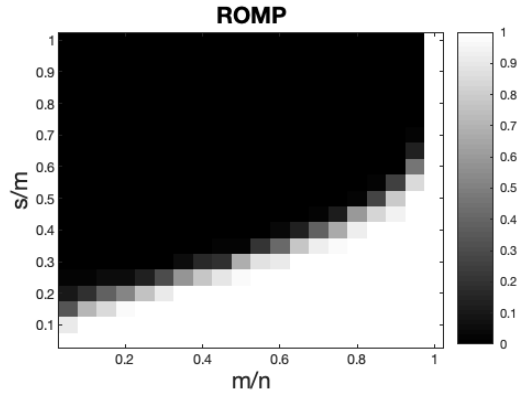
Figure 5.1.4. Number of indices selected in total at each s and m value pair.



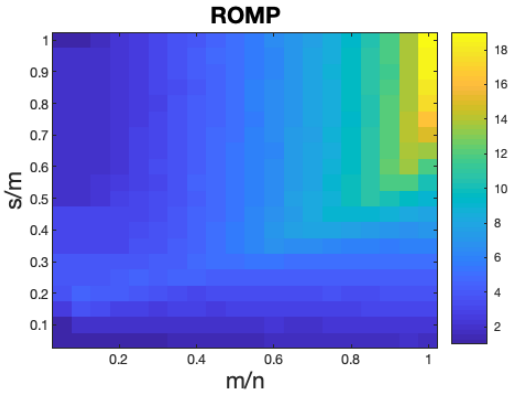
(a) Reconstruction accuracy for case 1



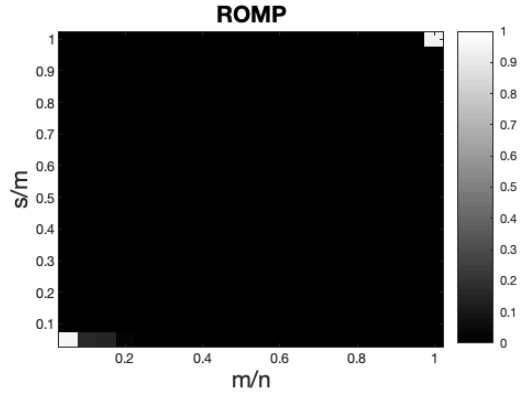
(b) Iterations for case 1



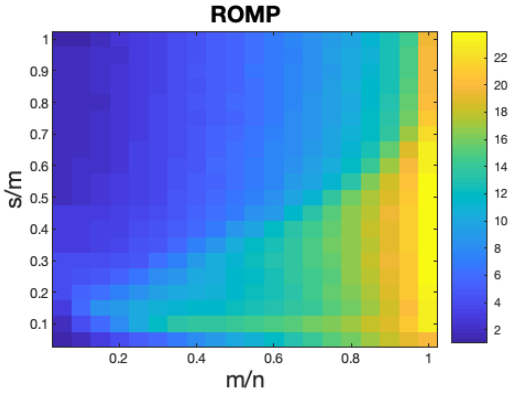
(c) Reconstruction accuracy for case 2



(d) Iterations for case 2



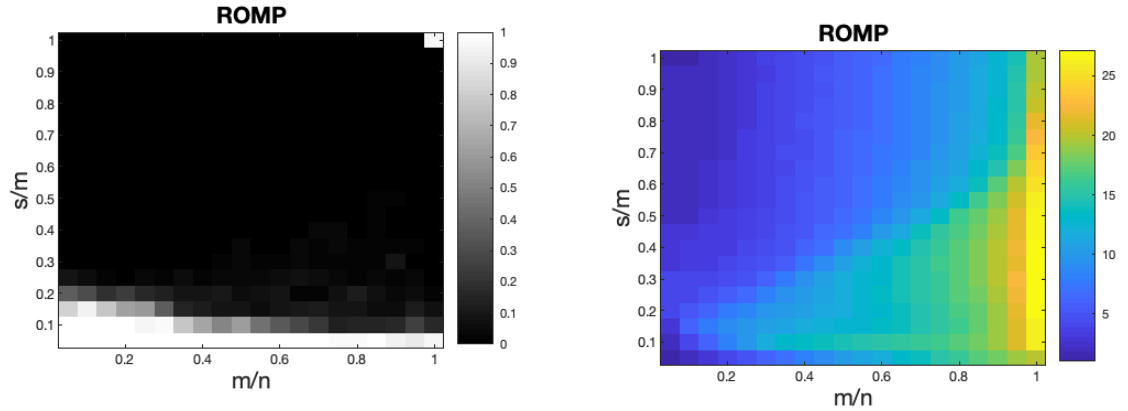
(e) Reconstruction accuracy for case 3



(f) Iterations for case 3

Figure 5.1.5. Performance of ROMP, illustrated through phase diagrams with coordinates of relative sparsity and compression rate, the plots uses either estimated reconstruction accuracy or iterations taken to form different regions.. Setup 1 (a and b) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients i.i.d. sampled from $\mathcal{N}(0,1)$. Setup 2 (c and d) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients set to one. Setup 3 (e and f) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients i.i.d. sampled from $\mathcal{N}(0,1)$ and with added white noise gaussian noise with an SNR of 80.

The results in figure 5.1.5 shows that when the relation between m and n decreses, so does the relation between s and m for which reconstruction is possible. This indicates that Regularized Orthogonal Matching Pursuit is can deal with much higher sparsity levels, when little compression is used. When dealing with flat signals, rather than gaussian, the cutt-off region is slightly lower for all values of m. As for Stagewise Orthogonal Matching Pursuit the result shows that when the gaussian signal is contaminated with noise, Regularized Orthogonal Matching Pursuit is unable to perform reconstruction, despite when the sparsity level is 1, and when no compression is used, and the signal is not sparse at all. Again as for Stagewise Orthogonal Matching Pursuit, the addition of a pruning step improves upon the result (figure 5.1.6), however not even nearly as much as for Stagewise Orthogonal Matching Pursuit. Figure figure 5.1.7 shows the ammount of selected indices for the noisy case. As for Stagewise Orthogonal Matching Pursuit, when in the region where reconstruction is possible, when using a pruning step, the overestimate size is limited. However, when not in the region of reconstruction the result seems to contain m significant coefficients, instead of only s.
When looking a the amount of selected indices, it is bound by $\mathcal{O}(s)$ in the regions where reconstruction is possible, thus justifying the stated computational complexity of $\mathcal{O}_t(smn + ms^3)$, atleast for reconstructable signals.



(a) Reconstruction accuracy for case 3, with added pruning step

(b) Iterations for case 3, with added pruning step

Figure 5.1.6. ROMP, measured by reconstruction accuracy and iterations taken. For case 3, noisy measurements, and with adding pruning step at the end.

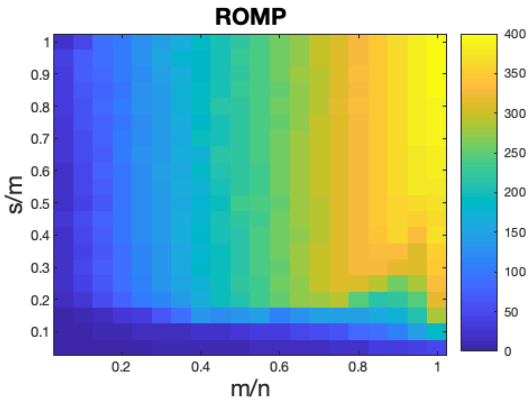
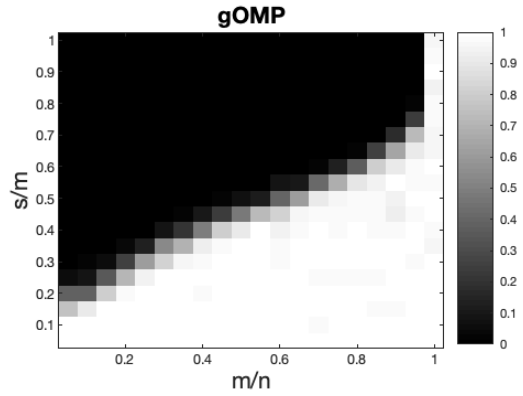
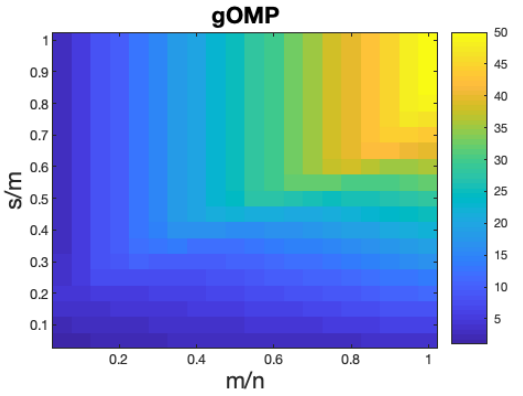


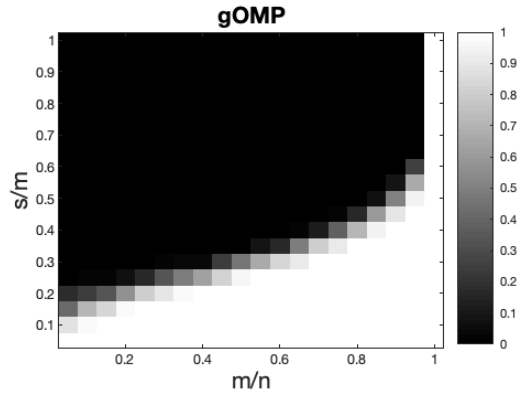
Figure 5.1.7. Number of indices selected in total at each s and m value pair.



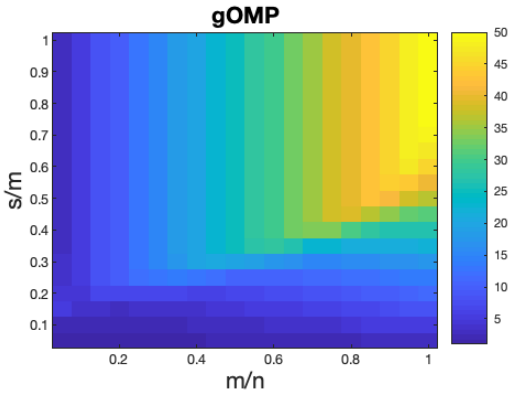
(a) Reconstruction accuracy for case 1



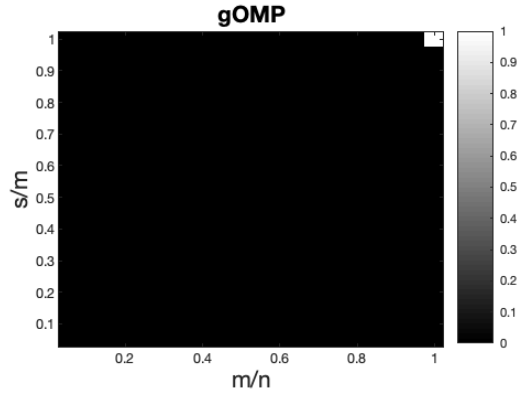
(b) Iterations for case 1



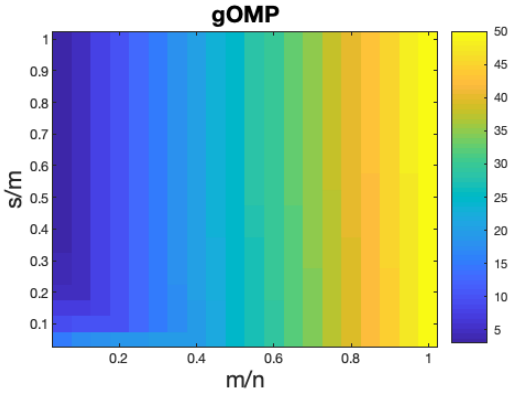
(c) Reconstruction accuracy for case 2



(d) Iterations for case 2



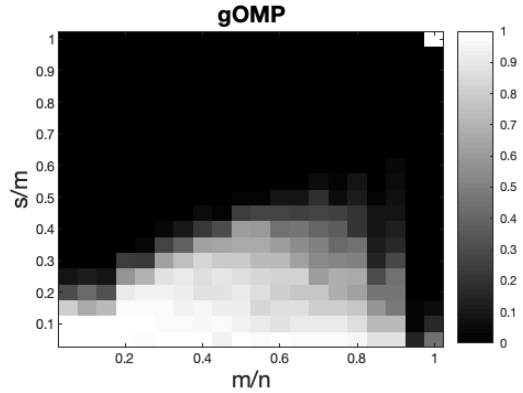
(e) Reconstruction accuracy for case 3



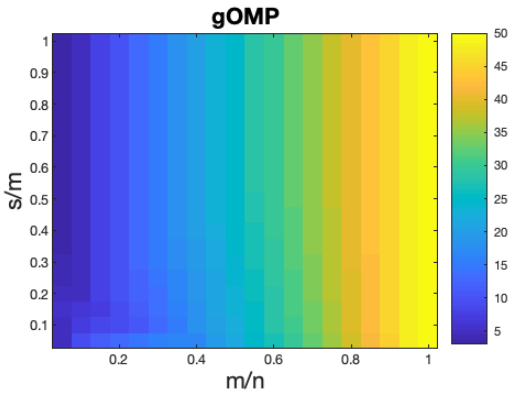
(f) Iterations for case 3

Figure 5.1.8. Performance of gOMP, illustrated through phase diagrams with coordinates of relative sparsity and compression rate, the plots uses either estimated reconstruction accuracy or iterations taken to form different regions.. Setup 1 (a and b) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients i.i.d. sampled from $\mathcal{N}(0,1)$. Setup 2 (c and d) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients set to one. Setup 3 (e and f) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients i.i.d. sampled from $\mathcal{N}(0,1)$ and with added white noise gaussian noise with an SNR of 80.

The results in figure 5.1.8 shows performance equal to the of Regularized Orthogonal Matching Pursuit, with higher relation between m and n allowing for reconstruction of higher relations between s and m. As for both Regularized Orthogonal Matching Pursuit and Stagewise Orthogonal Matching Pursuit, Generalized Orthogonal Matching Pursuit seems unable to reconstruct gaussian signals contaminated with noise. Adding a pruning step again improves upon this (figure 5.1.9), allowing for almost similar reconstruction region, as for the noiseless case, when the relation between m and n is less than $0,5$. Figure 5.1.10 shows number of estimated indecies for different levels of sparsity and compression when dealing with noisy signals. The number over wrongly estimated indices in the region for which reconstruction was possible seems to be bound $\mathcal{O}(s)$ thus reducing the upper limit of iterations to a bound of $\mathcal{O}(s/S)$ should suffice. As well as assuming a computation complexity of $\mathcal{O}(ms^2)$ for the signal estimation step.



(a) Reconstruction accuracy for case 3, with added pruning step



(b) Iterations for case 3, with added pruning step

Figure 5.1.9. gOMP, measured by reconstruction accuracy and iterations taken. For case 3, noisy measurements, and with adding pruning step at the end.

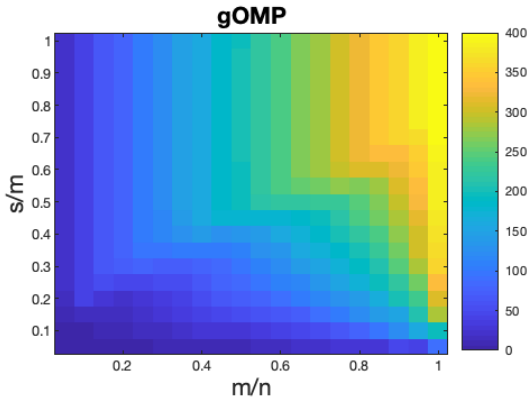
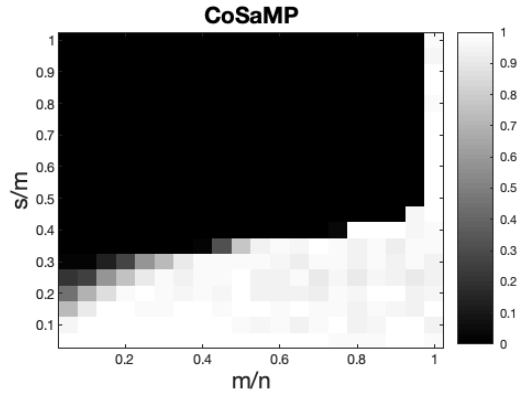
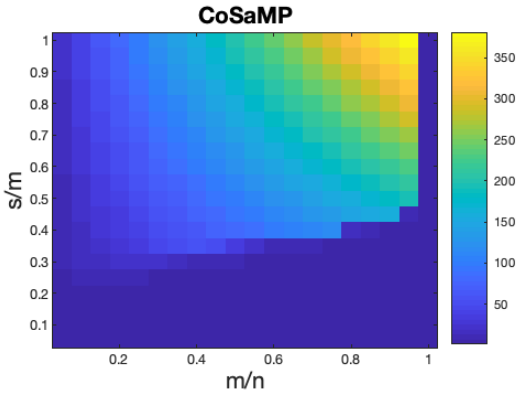


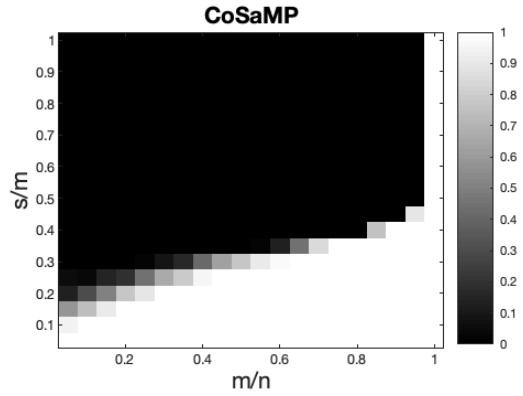
Figure 5.1.10. Number of indices selected in total at each s and m value pair.



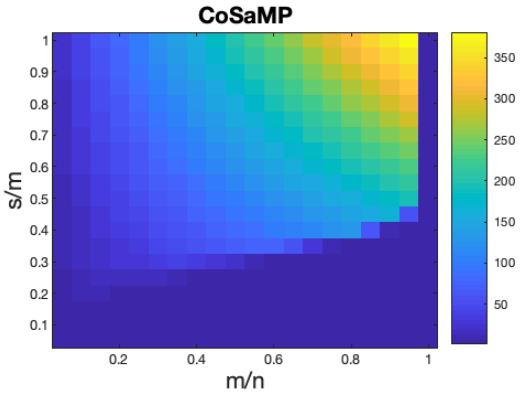
(a) Reconstruction accuracy for case 1



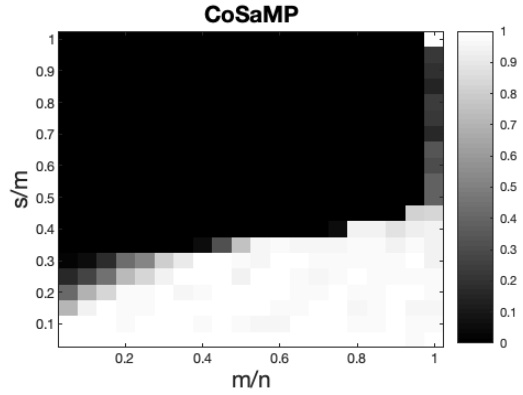
(b) Iterations for case 1



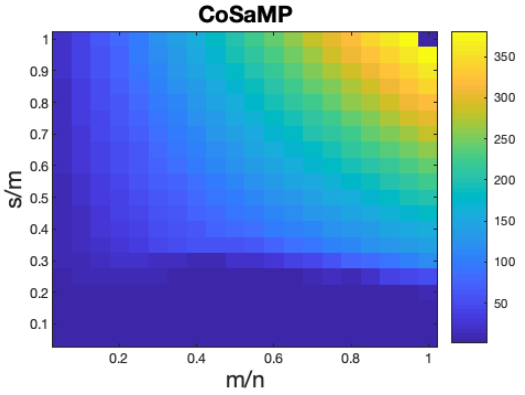
(c) Reconstruction accuracy for case 2



(d) Iterations for case 2



(e) Reconstruction accuracy for case 3

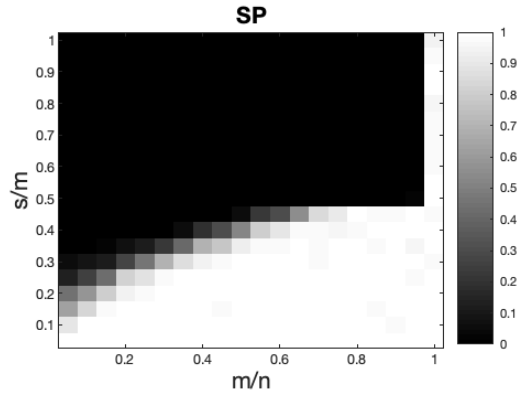


(f) Iterations for case 3

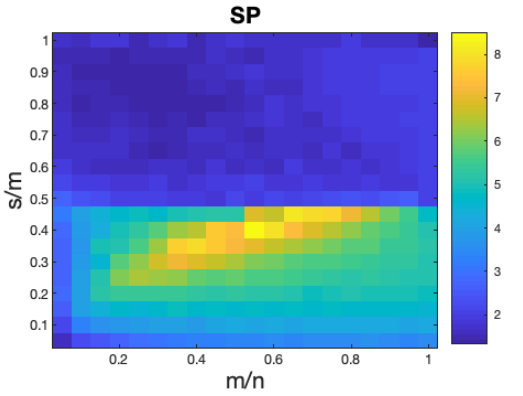
Figure 5.1.11. Performance of CoSaMP, illustrated through phase diagrams with coordinates of relative sparsity and compression rate, the plots uses either estimated reconstruction accuracy or iterations taken to form different regions.. Setup 1 (a and b) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients i.i.d. sampled from $\mathcal{N}(0,1)$. Setup 2 (c and d) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients set to one. Setup 3 (e and f) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients i.i.d. sampled from $\mathcal{N}(0,1)$ and with added white noise gaussian noise with an SNR of 80.

The results of figure 5.1.11 shows how Compressive Sampling Matching Pursuit performes reconstruction of the stated relation between s and m of $0,5$ when little compression is used, however, when compression is increased the relation between s and m for which reconstruction is possible, also decreases. When working on flat signals, the region of which Compressive Sampling Matching Pursuit can perform reconstruction, decreses much less, than what was the case for the forward selection algorithms. The region of reconstruction for Compressive Sampling Matching Pursuit seems to be equal for the noisy and noiseless guassian case, with even better accuracy for higher relation between m and n when the signal is noisy.
Iteration wise, Compressive Sampling Matching Pursuit takes very few iterations when in the region where compression is possible, however it uses all the allowed iterations when in the region where reconstruction is not possible. This indicates that a lower bound on allowable iteration would be wise. Moreover, the results indicate that an upper limit of iteration bound by $\mathcal{O}(\log s)$ would still have the same region of reconstruction, while greately reducing the iteration used when reconstruction is not possible. This would change the overall computational complexity of the algorithm to $\mathcal{O}_t(nm \log s + ms^2 \log s)$.

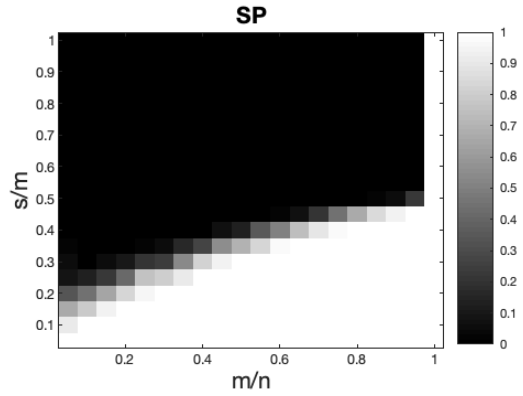
Subspace Pursuit



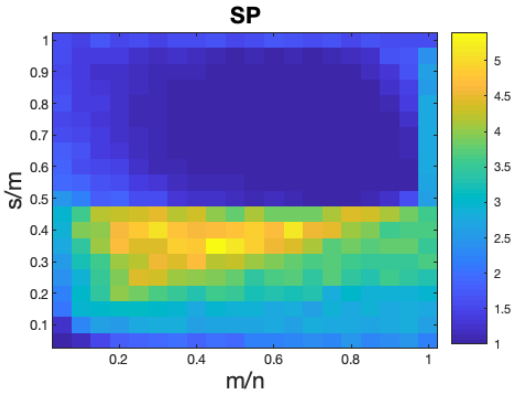
(a) Reconstruction accuracy for case 1



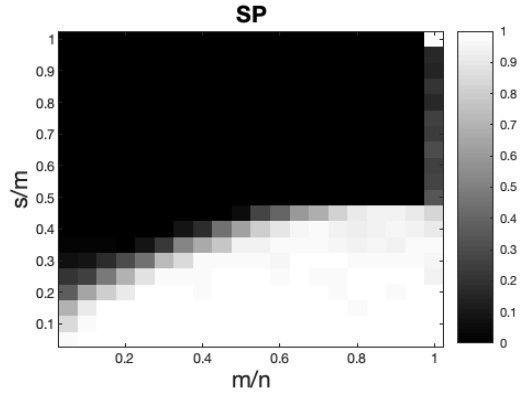
(b) Iterations for case 1



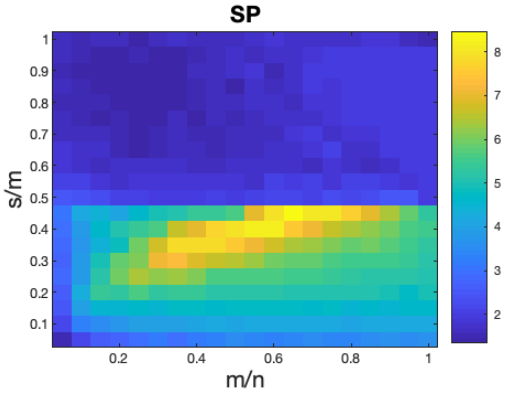
(c) Reconstruction accuracy for case 2



(d) Iterations for case 2



(e) Reconstruction accuracy for case 3



(f) Iterations for case 3

Figure 5.1.12. Performance of SP, illustrated through phase diagrams with coordinates of relative sparsity and compression rate, the plots uses either estimated reconstruction accuracy or iterations taken to form different regions.. Setup 1 (a and b) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients i.i.d. sampled from $\mathcal{N}(0,1)$. Setup 2 (c and d) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients set to one. Setup 3 (e and f) is signals with i.i.d. sampled support set from $\mathcal{U}(1,n)$ and coefficients i.i.d. sampled from $\mathcal{N}(0,1)$ and with added white noise gaussian noise with an SNR of 80.

The results of figure figure 5.1.12 indicates that Subspace Pursuit is able to perform reconstruction of signals with a relation between s and m of up to the stated $0,5$ when little compression is used. However, as the compression rate is increased, the relation between s and m for which reconstruction is possible decreases. Subspace Pursuit shows very similar reconstruction regions for all of the three signals.
Iteration wise Subspace Pursuit uses a few iteration when in the region where reconstruction is possible, indicating that a lower bound on iteration could be used. The iteration plots also shows that when in the region where reconstruction is not possible, Subspace Pursuit returns with even fewer iteration. The results are generally very similar to those of Compressive Sampling Matchign Pursuit, when in the region where reconstruction is possible, thus also alowing an upper iteration bound of $\mathcal{O}(\log s)$, which inturn also changes the computation complexity to $\mathcal{O}_t(nm \log s + ms^2 \log s)$.

5.1.2 Reconstruction Error

Protocol

Purpose	Investigate reconstruction accuracy for different degree of compression and sparsity level, using a less strict metric than in section 5.1.1. This will indicate if the algorithms could give fair approximation above the region of reconstruction found in section 5.1.1
Data	Test data, with i.i.d. sampled index from $\mathcal{U}(1, N)$, coefficients i.i.d. sampled from $\mathcal{N}(0, 1)$, and with white gaussian noise added to the measurement, with an SNR of 80
Success criterion	Relative error of reconstruction below 1
Algorithms	SP
Notes	As a performance metric relative error of reconstruction (RER) will be used, inspired from [32]. RER is defined as: $RER = \frac{\|\mathbf{x}-\hat{\mathbf{x}}\|_2}{\|\mathbf{x}\|_2}$, which for perfect reconstruction would be 0, and for fair approximation, be below 1. For the test $n = 400$, and for every value combination of s and m 100 trials was performed and averaged

Table 5.1.2. Test protocol: Reconstruction Error

5.1.2.1 Results

When comparing the result shown in figure 5.1.13 with that of figure 5.1.12e, which is for equivalent signals, it seems like the cutoff between the regions of success and failure, is as sharp as was found in section 5.1.1. Trials using the other matching pursuit algorithms, seems to give similar results.

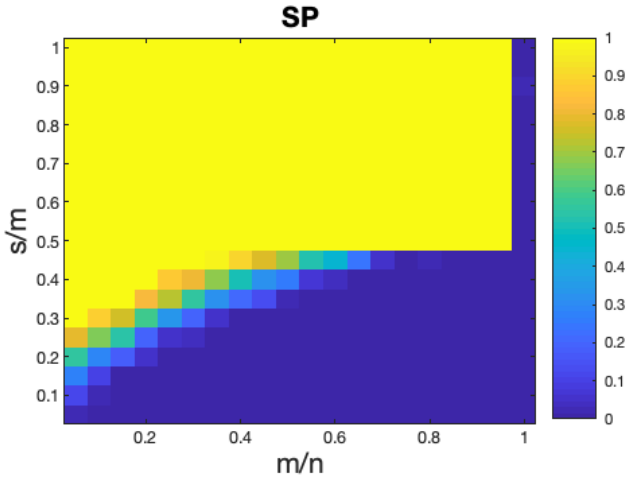


Figure 5.1.13. Phase plot, showing relative error of reconstruct: $RER = \frac{\|\mathbf{x}-\hat{\mathbf{x}}\|_2}{\|\mathbf{x}\|_2}$, for different degree of compression and sparsity levels, using SP. Simulated measurements have a fixed size of $n = 400$

5.2 Distributed Compressed Sensing Experiments

The following tests focus on ...

5.2.1 Obscure Sparsity

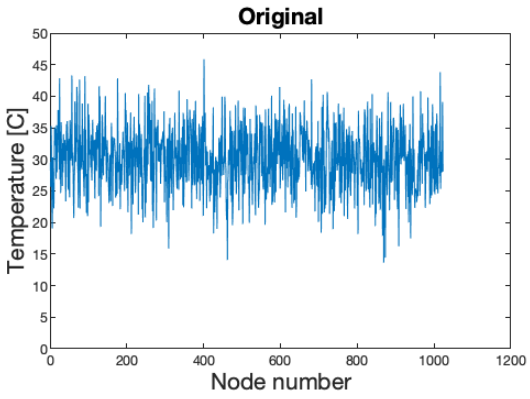
Protocol

Purpose	Illustrate the usability of Compressed Sensing in distributed setup, when the signals are extremely noise like. In this case transform coding and distributed source coding are not applicable. Moreover, the test illustrates how the use of compressed sensing, allows for event detection, which would not be possible using simple data aggregation. Additionally, this test illustrates how reconstruction is performed when the signal used is sparse in a transform domain, thus introducing the need for a transform basis..
Data	Signals formed from i.i.d values from $\mathcal{N}(30, 10)$, to simulate temperature readings similar to those of [29].
Success criterion	Reconstruction and events detected
Method	Compressive Data Gathering
Notes	

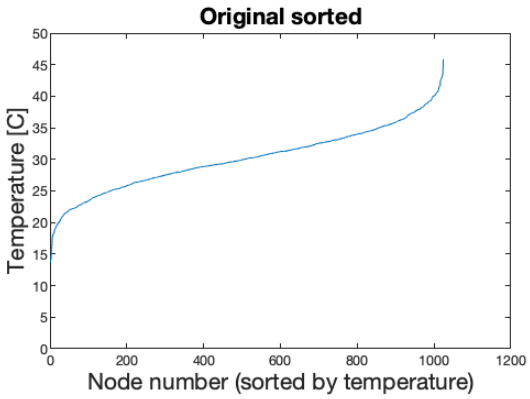
Table 5.2.1. Test protocol: Obscure Sparsity

5.2.1.1 Results

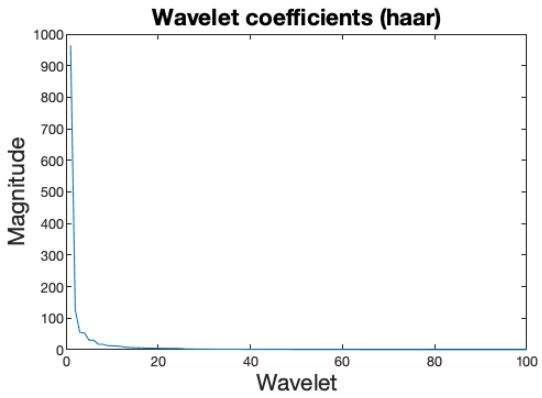
Figure 5.2.1a shows the simulated temperature readings, which have a much noiselike structure, and is thus not sparse in any known transfer domain. However when the readings are sorted by magnitude in figure 5.2.1b, they form a piecewise smooth plot, which is then shown to be compressible in the wavelet domain figure 5.2.1c, by having only a few large coefficients.



(a) Simulated temperature readings, sorted simmilar to the deployment structure of the sensor in [20]



(b) Simulated temperature readings, sorted with respect temperature magnitudes



(c) Haar wavelet transform of the simulated temperature readings sorted with respect to temperature magnitudes. The transform, clearly only consist of a few large coefficients, making compression possible

Figure 5.2.1. Original signal, when sorted differently, as well as representation in a wavelet domain

To illustrate the compressed sensing is applicable to the original unsorted signal in figure 5.2.1a, the following test is performed. The original signal is compressed using a measurement matrix, with i.i.d. entries from $\mathcal{N}(0,1)$ and with a size 256×1024. The resulting measurement vector is then reconstructed using Subspace Pursuit, with sparsity of 30. For the reconstruction the measurement matrix initially has its columns sorted in the same way the original signal was sorted to make it piecewise smooth, it is then combined with a transposed Haar basis matrix. The result is then multiplied by the Haar basis matrix, to transform it back into the space domain. Lastly the columns are resorted, using the same sorting pattern as originally used, to recreate the signal. The result is shown in figure 5.2.2 which have the very similar noise structure as the original signal.

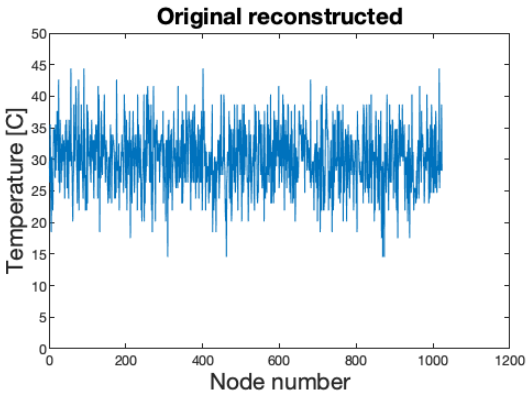
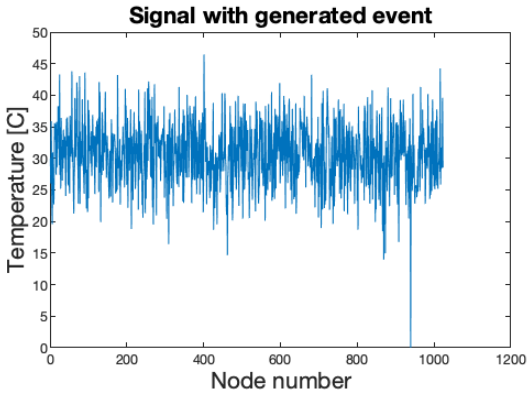
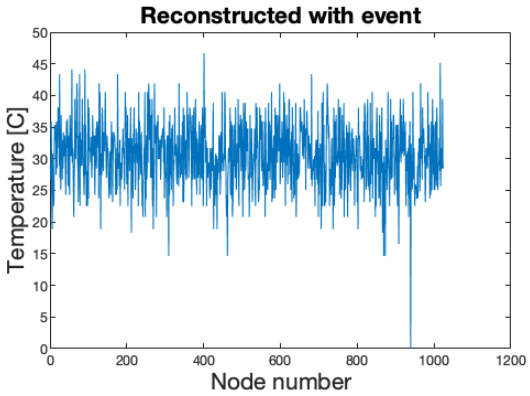


Figure 5.2.2. Reconstruction result, after having simulated compressed sensing on the signal of figure 5.2.1a to obtain a measurement. The reconstruction has been performed by rearranging the columns of the measurement matrix, in the same way the readings was sorted in figure 5.2.1b, this rearranged measurement matrix is then multiplied by a Haar basis, and the product is used in Subspace Pursuit with the measurement. After the reconstruction the signal as been rearranged again, in reverse order of the sorting used in figure 5.2.1b. When comparing this result with figure 5.2.1a, similar structes are clearly seen in the apperently noise signal.

To illustrate the usefulness of the above followed approach, new readings are simulated by adding a vector for i.i.d values from $\mathcal{N}(1/2, 1/10)$ to the original signal to simulate temperature increases. Additionally a node is randomly selected and has it reading set to zero, in order to simulate a node loss. As a node loss will amount to the node not giving any contribution to the measurement vector. These new readings are then compressed and reconstructed, just like in the example above. The new signals and its reconstructed counter part is shown in figure 5.2.3. Which again has very similar structures, and more importantly the location of zero reading is perserved.



(a) Originally modified signal



(b) Result of simulating compressed sensing on the originally modifed signal, and reconstruction it using the same approach for figure 5.2.2

Figure 5.2.3. The signal of figure 5.2.1a, modified by adding a vector i.i.d drawn from $\mathcal{N}(1/2, 1/10)$ to simulate an over all temperature increase, as well as simulating the event of a node loss, by setting a reading to 0, thus it want have an impact on the compressed sensing operation.

This shows how the location of the node loss is preserved, as a reading of 0. In figure 5.2.4 the reconstructed signal in figure 5.2.3b is subtracted from the original readings 5.2.1a, and it shows

how the selected node, has a much larger difference magnitude then the readings from the rest of the nodes.

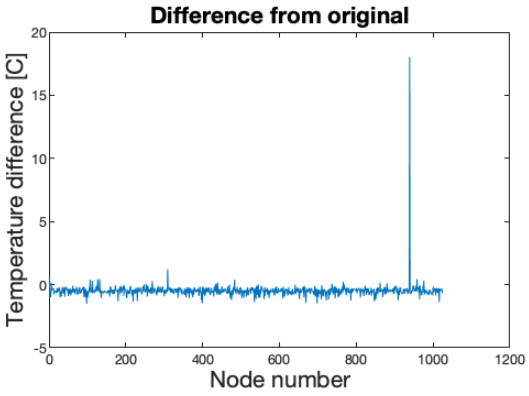


Figure 5.2.4. Difference of the signal in figure 5.2.1a and figure 5.2.3b. The spike indicates the location simulated note failure

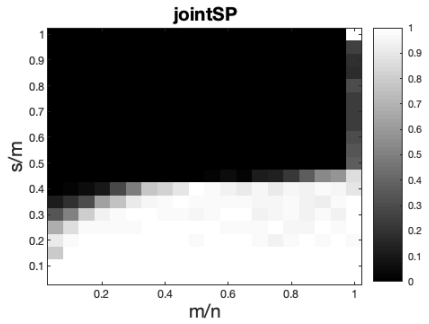
5.2.2 Multiple Correlated Signals

Protocol

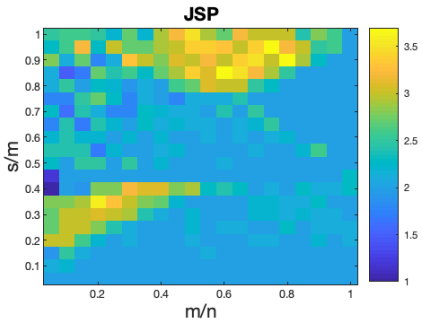
Purpose	Investigate the performance of reconstruction algorithms considering multiple correlated signals.
Data	Test data, with i.i.d. sampled support set from $\mathcal{U}(1, n)$ and coefficients drawn from $\mathcal{N}(0, 1)$ and white gaussian noise added to the measurement, with an SNR of 80.
Success criterion	The amount of individual signals, being reconstructed with the exact support set of det original signals.
Algorithms	JointSP and DIPP with different degree of connectivity
Notes	For all of the performed testes $n = 400$, and for every value combination of s and m 10 trials was performed and averaged, for each run 10 correlated signals was used.

Table 5.2.2. Test protocol: Multiple Correlated Signals

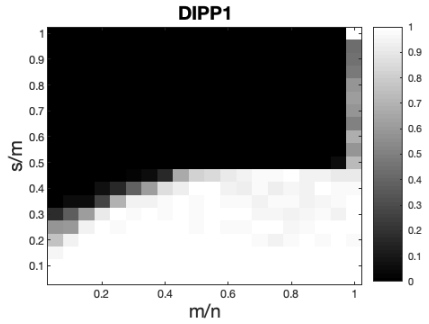
5.2.2.1 Results



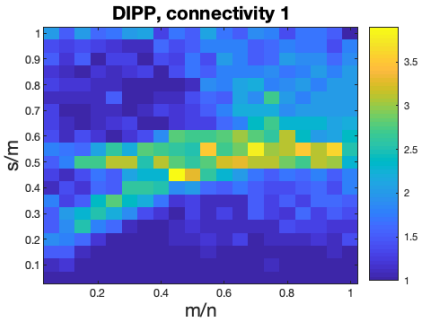
(a) Reconstruction accuracy for jointSP



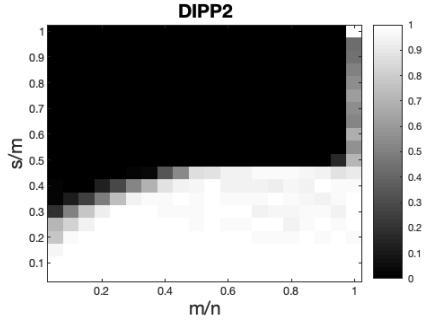
(b) Iterations for jointSP



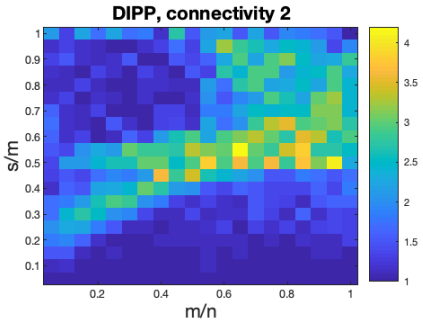
(c) Reconstruction accuracy DIPP, concetivity 1



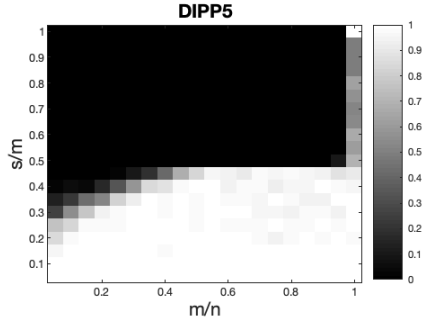
(d) Iterations for DIPP, connectivity 1



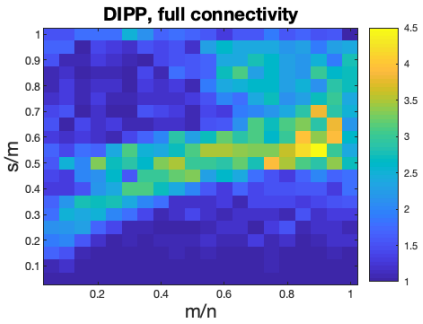
(e) Reconstruction accuracy for DIPP connectivity 2



(f) Iterations for DIPP, connectivity 2



(g) Reconstruction accuracy for DIPP full connectivity



(h) Iterations for DIPP, full connectivity

Figure 5.2.5. Result for test of JSP and DIPP on Test data, with i.i.d. sampled support set from U(1,n) and coefficients drawn from $\mathcal{N}(0,1)$ and white gaussian noise added to the measurement, with an SNR of 80.

The phase plots for reconstruction shown in figure 5.2.5, show very similar results. This indicates that when concerned with reconstruction, whether Joint Subspace Pursuit or Distributed Parallel Pursuit is used does not make a difference. Even running Distributed Parallel Pursuit with different degree of connectivity does not make a clear difference in reconstruction. When looking at the iteration plots in figure 5.2.5, the only noticeable difference, when in the region where reconstruction is possible is that Joint Subspace Pursuit seems to use slightly more iterations than Distributed Parallel Pursuit. Moreover, the regions where construction is possible, is similar to those found by using Subspace Pursuit, which is also the underlying Matching Pursuit algorithm for both Joint Subspace Pursuit and Distributed Parallel Pursuit. Thus no significant improvement is found when considering multiple correlated signals.

5.2.3 CISP Performance Verification

Protocol

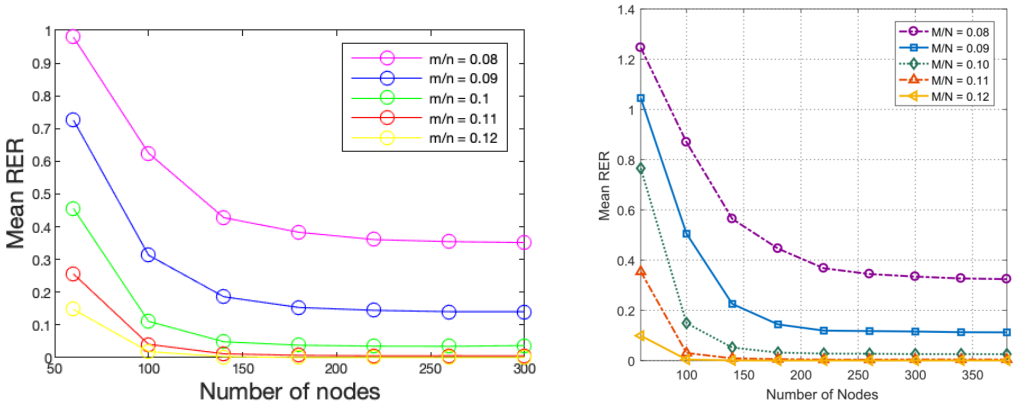
Purpose	Verify the performance of the CISP algorithms is as stated in [32]
Data	Simulated data, following the mixed support set model
Success criterion	Results comparable to those of [32].
Algorithms	CISP
Notes	As a performance metric the article [32], uses relative error of reconstruction (RER) defined as $RER = \frac{\|\|\mathbf{x}-\hat{\mathbf{x}}\|\|_2}{\|\|\mathbf{x}\|\|_2}$, which for perfect reconstruction would be 0, and for fair approximation, be below 1

Table 5.2.3. Test protocol: CISP Performance Verification

5.2.3.1 Results

The results of figure 5.2.6, figure 5.2.7 and figure 5.2.8 are all nearly identical to those of those found in [32]. Though the results in figure 5.2.6 and figure 5.2.7, shows slightly better performence than those from [32], where as the result in figure 5.2.8 seems slightly worse.

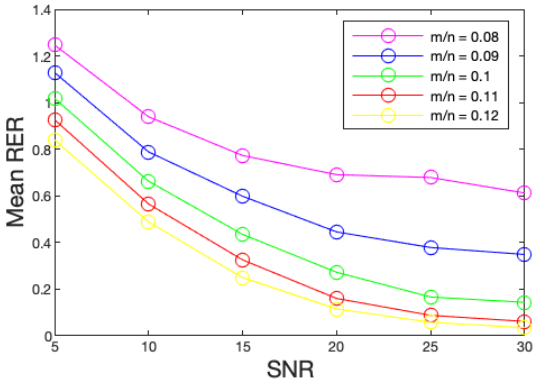
Performance results for reconstruction (noiseless)



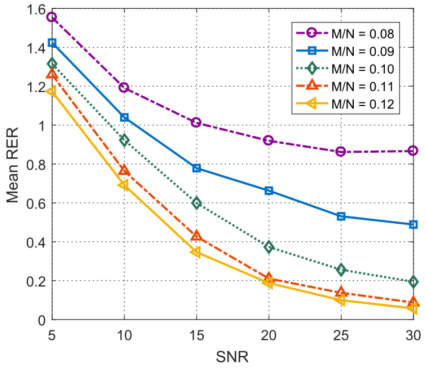
(a) Test results, averaged over 100 trials ***(b)*** Article results ([32] figure 6a)

Figure 5.2.6. Verification of the result in [32] for reconstruction of noiseless signals. The test sets $n = 1000$, $s_c = 30$, $s_i = 10$ and the private part of measurement matrix having variance of only $0,1$

Performance results for reconstruction (noisy)



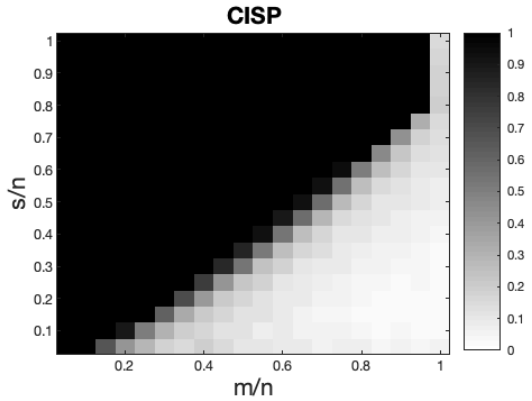
(a) Test results, averaged over 100 trials



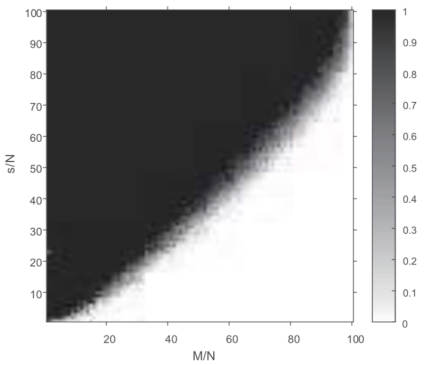
(b) Article results ([32] figure 8a)

Figure 5.2.7. Verification of the result in [32] for reconstruction of noisy signals. The test sets $n = 1000$, $s_c = 30$, $s_i = 10$ and the private part of measurement matrix having variance of only $0,1$

Performance results for different sparsity



(a) Test results, averaged over 100 trials



(b) Article results ([32] figure 4)

Figure 5.2.8. Verification of the phase diagram result in [32]. The test sets $n = 200$, $s_c = 0,8s$, $s_i = 0,2s$, with number of clusters equal s_c and the private part of measurement matrix having variance of only $0,1$

5.2.4 Unknown Common Support Set Size

Protocol

Purpose	Investigate the claim in [32], that CISP can estimate the common support set, when the size is unknown, based on the eigenvalues.
Data	Simulated data, following the mixed support set model.
Success criterion	Size of the common support set being clear from eigenvalues of the covariance matrix used in CISP.
Algorithms	CISP
Notes	A general cluster size of 4 will be used, based on the analysis [32]. For each test 50 trials is run, the the averaged eigenvalues are plotted at well as the max and min values.

Table 5.2.4. Test protocol: Unknowm Common Support Set Size

5.2.4.1 Results

From figure 5.2.9 it seems like perfect estimation of the common support set size is possible using Common-Innovation Subspace Pursuit, by looking at the number of eigenvalues that are significantly different from zero. However the number of clusters simulated for this result is equal to the size of the total support set. Though when the number of clusters is below the size of the total support set, only a number of eigevalues equal the number of cluster is clearly above zero, as shown in figure 5.2.10. On the other hand when the number of clusters is higher than the size of the support set size, a number of eigenvalues equal to the number of clusters is clearly above zero, as shown in figure 5.2.11.
Thus in order to estimate the size of the common support set, based on the eigenvalue decomposition performed by the Common-Innovation Subspace Pursuit algorithm, the number of cluster must be at least equal to the total support set size, though more thorough analysis would be required

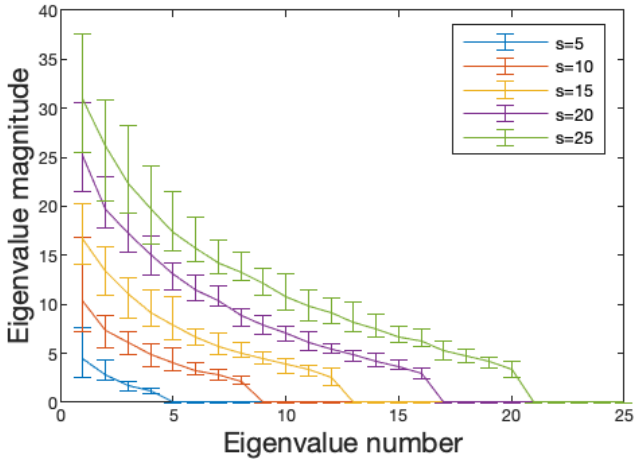


Figure 5.2.9. Eigenvalue decay, when $s_c = 0,8s$. Number of clusters equal s with 4 nodes in each cluster, $n = 1000$ and $m = 100$

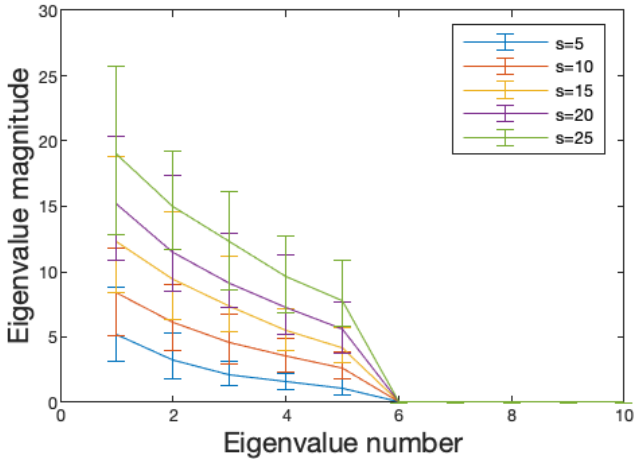


Figure 5.2.10. Eigenvalue decay, when $s_c = 0,8s$. Number of clusters is 5 with 4 nodes in each cluster, $n = 1000$ and $m = 100$

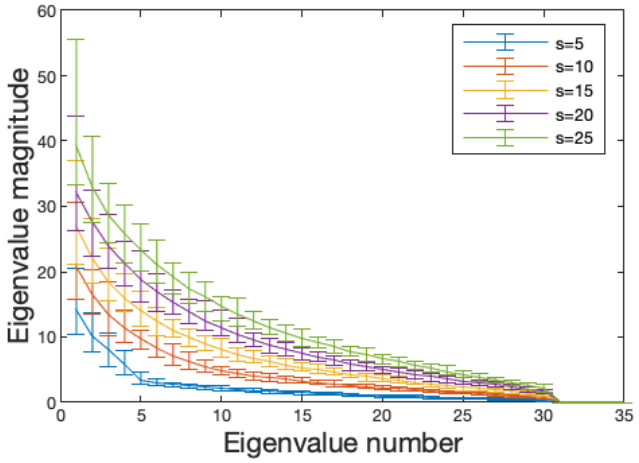


Figure 5.2.11. Eigenvalue decay, when $s_c = 0,8s$. Number of clusters is 30 with 4 nodes in each cluster, $n = 1000$ and $m = 100$

6
DISCUSSION

6.1 Measurement Matrix

The measurement matrix is responsible for performing a linear dimensionality reduction during the sampling, resulting in a smaller number of samples. In order for later reconstruction to be possible, the measurement matrix must preserve the information of the signal in the compressed representation. The signal information is preserved when the measurement matrix obeys the restricted isometry property, which ensures an isometric dimensionality mapping of sparse and compressible signals. Besides this, the measurement matrix must also be incoherent with the sparsity basis. Verifying whether the restricted isometric property is obeyed for a given matrix, requires that all $\binom{n}{s}$ possible combinations of the matrix are linearly independent [2]. However, both the restricted isometry property and the incoherence are fulfilled with high probability, by matrices formed with i.i.d drawn coefficients from a normal distribution of $\mathcal{N}(0, \frac{1}{m})$ [2]. More specifically such randomly drawn sensing matrix satisfy the restricted isometry property, in relation to s sparse signals when $m > cs\ \log \frac{n}{s}$, with c being a small constant. Thus a reconstruction of s sparse signal with length n would be possible from only $m > cs\ \log \frac{n}{s}$ measurements [2, 6].

6.2 Matching Pursuit

Based on the analysis of chapter 3 and experiments results in section 5.1.1 the following (table 6.2.1), contains the current complexity estimates.

Algorithm	**Time complexity**	**Space Complexity**
OMP	$snm + smk^2$	n
StOMP	$Snm + Sms^2$	n
ROMP	$snm + ms^3$	n
gOMP	$snm/S + ms^3/S$	n
CoSaMP	$nm \log s + ms^2 \log s$	n
SP	$nm \log s + ms^2 \log s$	n

Table 6.2.1. Current complexity results of matching pursuit algorithms, based on the analysis of chapter 3 and numerical results of section 5.1.1

For the work of this thesis the Matching Pursuit based algorithms have been subdivided into the categories forward and hybrid selection, due to the selection strategies highly resembling the forward and hybrid subset selection in linear regression [18].
Moreover, as a general performance metric used to test the algorithm implementations, phase diagrams were found effective for the experiments. The phase diagrams are 2D graphics with coordinates measuring the relative sparsity level as well as the compression rate [10]. This allows for discussing regions within which reconstruction is possible, with relative sparsity and compression rate more generally determined.

6.2.1 Forward Selection

The forward selection algorithms try to estimate one or as many correct indices of the support set in each iteration.

Orthogonal Matching Pursuit selects a single index in each iteration, this index is found by applying a matched filter. Then in each iteration, the signal of interest is estimated using the selected indices. Based on this estimation a residual is updated by recompressing the estimate using the measurement matrix and subtracting the result from the measurement vector. This fairly removes most of the contribution of the currently estimated support set from the measurement, allowing the following iteration to better estimate the next index. This single step strategy forces the algorithm to run for a number of iterations equal to sparsity.
The rest of the forward selection algorithms considered, aims at decreasing the required number of iteration by estimating more of the support set in each iteration.

Stagewise Orthogonal Matching Pursuit select multiple indices of the support set each iteration by taking the indices of the matched filter coefficient that exceeds a given threshold. This threshold is calculated based on formal noise levels. This selection strategy seems to give a slightly better reconstruction result than the single step strategy of Orthogonal Matching Pursuit while running for significantly fewer iterations. Though, when the signal is contaminated with noise, a pruning step was found a necessity, as Stagewise Orthogonal Matching Pursuit, otherwise, overestimate the signal, resulting in too many indices selected for the support set.

Regularized Orthogonal Matching Pursuit takes a different approach to the indices selection based on regularization. Initially Regularized Orthogonal Matching Pursuit selects the indices of the s highest coefficients of the matched filter. These indices are then split into consecutive subsets, where the related coefficients from the matched filter, are all within a factor two of each other. The subset with the maximal energy, based on the ℓ_2 norm is selected as the new indices. It was observed that this operation could be reduced to picking the first subset, which would increase the region of reconstruction, at the cost of a few more iterations. Just as for Stagewise Orthogonal Matching Pursuit, a pruning step at the end was found necessary when the signal was contaminated with noise, as Regularized Orthogonal Matching Pursuit, also will overestimate the signal.

Generalized Orthogonal Matching Pursuit takes a simpler approach to element selection. It selects a predetermined number of indices in each iteration, for which 8 is advertised as a decent choice, though it should be less if the sparsity level is below 8. Based on this it would be fair to assume that the relation between the selected number of indices and the actual sparsity, would suffice as an upper iteration limit. Though as for Stagewise Orthogonal Matching Pursuit, Regularized Orthogonal Matching Pursuit a pruning step was found necessary when the signal was contaminated with noise. This overestimation, however, was of a limited quantity, thus running for just a few more iteration than the relation between the indices selection number and actual sparsity, should suffice.

6.2.2 Hybrid Selection

The hybrid selection algorithms take a more aggressive approach and try to estimate the entire support set each iteration. This, however, makes it inevitable to avoid indices that are not in the true support set, thus introducing the need for a pruning step in each iteration.

Compressive Sampling Matching Pursuit selects a number of indices, based on the matched filter, which is twice the targeted support set size. This introduces the need for a pruning step, after the signal estimation, which removes all but the indices, from the support set, that have the s highest coefficients in the estimate. An iteration limit set to the sparsity level was initially used, though through the experiments it was found that, when reconstruction was possible far fewer iterations were required. This limit would be assumed to be in the magnitude of $c \log s$ with c a small constant. This would also greatly reduce the amount of iteration taken, when reconstruction was not possible, without making the algorithm perform worse.

Subspace Pursuit takes the same approach as Compressive Sampling Matching Pursuit, though only select a number of indices, each iteration, as indicated by the desired sparsity. As for Compressive Sampling Matching Pursuit, an upper iteration limit was set equal to the desired sparsity, though the algorithm is originally presented to also stop when the residual stops decreasing. Through the experiments, it was found, as for Compressive Sampling Matching Pursuit, that an upper iteration limit of $c \log s$, would be sufficient with degrading the performance of the algorithm. Moreover, the strategy of stopping when the residual stop decreasing was found highly effective, as Subspace Pursuit in relation to Compressive Sampling Matching Pursuit, did not spend a significant amount of time, trying to reconstruct signals, outside the region of reconstruction.

6.2.3 Different Signal Models

Through the experiments it was observed that the ability of the algorithms to reconstruct signals, also depends on the signal model, thus further testing the algorithms on signals, corresponding those in a given application scenario, is advised.

6.3 Distributed Compressed Sensing

Though compressive sensing is a powerful tool, it is designed to primarily deal with intra-signal structures, from a single sensor. This makes it inefficient for multiple sensor readings, which has been widespread through the interest of wireless sensor networks. Due to this Duarte, Sanvotham, Baron, Wakin and Baraniuk [19] proposed Distributed Compressed Sensing, which expands the theory to also cover inter-signal correlation structures, as well as enabling distributed encoding [20].
Lately, multiple methods and algorithms for applying compressed sensing to distributed systems have been presented. In this thesis, three different alternatives have been covered. This be Compressive Data Gathering, Distributed Parallel Pursuit and Common-Innocation Subspace Pursuit. In the following subsection, these will be individually discussed.

6.3.1 Compressive Data Gathering

Compressive Data Gathering is an initial method, for applying compressed sensing to the data acquisition of large scale wireless sensor networks [29]. The basic principle of Compressive Data Gathering is that the signal of interest is distributed in space, thus multiple sensors are required to acquire it. These sensors must be set up using either a chain or three topology. Then each sensor multiply its own reading with an assigned column of the measurement matrix. In order to avoid the overhead of distributing these columns, each sensor can generate it self by drawing i.i.d gaussian coefficients, using a predetermined seed, thus the sink is able to construct the entire measurement matrix using the same seeds for the individual columns. After having multiplied its own signal with its measurement matrix column, added the resulting measurement, with possible incoming measurements of child notes, and forwards this to its parents. This result in the space distributed signal, having been multiplied with the entire measurement matrix. By doing this simpler compression is obtained, as adding and multiplying scalar, than what might have been need if distributed source coding as been used. Additionally load balancing is achieved as each note only needs to transmit m values, though this m in addition to being less than the total signal length, must also be smaller than the routes from leaf notes to the sink, in order to be effective. In addition to having load balancing comparable to data aggregation, more data will be available at the sink, than simple max or averages.
Moreover, as has been shown trough experiments, this compression is also applicable to data that is not directly sparse in any basis, i.e. not piecewise smooth. This compression is though made achievable through occasionally gather the uncompressed data, which can then be used to sort the measurements, be rearranging the columns of the measurement matrix, such that the data becomes sparse in a wavelet domain. This however requires that the reading does not change rapidly, in relation to each other.

6.3.2 Distributed Parallel Pursuit

Distributed Parallel Pursuit is an algorithm for cooperatively, sparsely approximate signals between nodes. This is based on local use of a Matching Pursuit algorithm, to estimate the support set of a signal, that have been locally acquired through compressed sensing. The estimated support set is then exchanged between neighboring nodes, in order to iteratively perform more accurate estimation of the support sets, using neighboring support sets, as side information.
This algorithm is suitable for system where the nodes are relatively far away for the sink, and thus being able to estimate support set, and transmit these to a far away sink.

6.3.3 Common-Innovation Subspace Pursuit

Common-Innovation Subspace Pursuit is a recently presented algorithms , for dealing with reconstruction of a large quantities of of correlated signals, that have been individually acquired through compressed sensing. Moreover, Common-Innovation Subspace Pursuit is designed to estimate the common and innovation support set separately, aiming at reducing reconstruction error as well computation time.
Common-Innovation Subspace Pursuit focuses on large scale sensor networks, where the sensor notes can be divided into clusters, making it an appropriate fit for a star-of-stars topology. A sequential description of the algorithm starts out by averaging the measurement vectors in the

different clusters, and putting these averages together as columns of a matrix, which is later used in the estimation of the common support set. This averaging can be performed on cluster headers, which then send these average to a sink, which combines them. The sink also performs an averaging of all the measurement matrices, which is also used to estimate the common support set. The measurement matrices can be separately build at each location, if known seeds are used for the drawing of the gaussian coefficients. Then eigenvalue decomposition is performed on the covariance matrix of the matrix formed by the averaged measurements. The eigenvectors associated with the s_c highest eigenvalues are then used as columns in a new matrix, with s_c being the size of the common support set. The eigen-subspace spanned by these eigenvectors should be relatively close to the subspace spanned by the columns of the averaged measurement matrix indexed by the common support set. Thus the common support set is estimated by calculating the cosine distance between every column of the averaged measurement matrix and the eigen-subspace, and selecting the indices of the averaged measurement matrix closest the eigen-subspace. This common support set can then be communicated back to the nodes. Each node can then calculate their own innovation support set. This is done by eliminating the contribution of the common support set, by projecting the notes measurement vector and measurement matrix onto the subspace that is perpendicular to the span of the measurement vector columns indexed by the common support set estimation. A Matching Pursuit algorithms is then used to these projection, in order to separately estimate the innovation support set of the given node.

Though the algorithm design fit onto a star-of-stars topology, it can be used for any setup. If all the individual measurements are collected at a sink, the sink can perform the entire algorithm. It only needs a strategy for dividing the measurements into cluster.

An important aspect for the estimation of the common support set to work, is that every measurement matrix is the sum of two measurement matrices. One of these matrices must be common to all the nodes, while the other is unique. Otherwise the eigen-value decomposition step won't work.

The article presenting the algorithm estimates that only a few nodes is needed in each cluster, and their results based on this assumption as been verified. Further the article briefly states that since the common support set estimation is based on eigenvalue decomposition, the algorithm should be able to deal with cases where the size of the common support set is unknown, by considering the eigenvalues. In general the experimental results in the article, deal with scenarios where the number of cluster is equal to the size of the common support set. In the experiments performed in this thesis, it has been shown that, when the number of clusters is equal to the the size of the common support set, the eigenvalues do indicate the size of the common support set, as in this case the number og eigenvalues, that are significantly different from zero, is equal the size of the common support set. Though it was also shown that, when the number og clusters is less than the size of the common support set, this approach cannot be used, as only a number of eigenvalues equal to the number of clusters is significantly different from zero, though the magnitude of these eigenvalues, might hint somewhat at the common support set size. On the other hand, when the number cluster is greater than the size of the common support set, the number of eigenvalues, significantly different from zero was numerically shown to equal to the number of clusters. Thus in order to deal with an unknown size of the common support set, the number of cluster might have to be larger than what the size of the common support set might be, and then a more complex method would have to used to estimate the common support set

size, based on the eigenvalues.

7
CONCLUSION

7.1 Contributions and Conclusion

The term compressed sensing originate from the fact, that if prior knowledge is available regarding sparsity in a well-known transform domain of a signal, the process of sampling and compressing the signal can be combined.
Two aspects need to be addressed in the system design, in order to utilize compressed sensing. First, a measurement matrix has to be applied during the sampling. This measurement matrix must satisfy the restricted isometry property, which ensures an isometric mapping of compressible signals into a smaller vector space. Moreover, the measurement matrix must be incoherent with the basis, in which the signal is sparse, in order to preserve the information in the signal. Second, a reconstruction algorithm is required, which can solve the resulting underdetermined system. Linear programming techniques are capable of solving this task, however, they have proven insufficient, due to high computational complexity. For practical use, a family of iterative greedy algorithms, known as matching pursuit, has been advocated. These algorithms have been proven to be able to reconstruct signals while being more computationally efficient than linear programming.
The matching pursuit algorithms primarily differ on two points; stopping criteria and element selection strategy. For stopping criteria, upper limits for iterations, related to the sparsity, has been used, as well as sufficiently low residual and decreasing residual. Upper limits have been shown to be as low as a logarithmic order of the sparsity, which makes for very efficient algorithms. With regards to element selection strategies, this thesis has subdivided the algorithms into the two categories termed forward and hybrid selection, as they highly resemble these kinds of subset selection in linear regression. The forward selection algorithms carefully select and add indices to the support set in each iteration. The most simple approach is that of orthogonal matching pursuit, which selects a single element in each iteration. Three approaches have been considered, each trying to increase the convergence rate by selecting multiple indices in each iteration. These have been based on selecting indices with comparable correlation, using a regularization step or simply selecting a larger number of coefficients each iteration. The hybrid selection strategies deliberately overestimate the signal in each iteration in order to find as many indices of the support as possible. They then introduce a pruning step, which removes wrongly estimated indices. In general, based on the performed experiments, the hybrid approach was found most effective, when considering both reconstruct accuracy and iterations required. Moreover the stopping criteria used in Subspace Pursuit, that terminates when an iteration is no longer able to reduce to residual, was found suitable.
It has been argued that compressed sensing is suitable for wireless sensor networks, which have been shown to be true. The utilization of compressed sensing, and thereby combining the sensing and compression, simultaneously reduces the energy used for sampling signals as well as removes the need for spending time and energy on compressing the signal afterward. This is highly desirable in wireless sensor networks, where sensor notes often must be kept relatively simple, in order to keep their cost low. Additionally, compressed sensing enables data aggregation, as the reading of different sensors can be combined during propagation through networks. This is highly desirable in sensor networks, where raw propagation of every single reading puts high workloads

of notes close to a sink. Thus, the fact that compressed sensing simplifies the acquisition and compression of signals, and increases the complexity of reconstruction, makes it a suitable fit for wireless sensor networks, where the sensors are kept simple while the sink has significantly more processing power.

The contributions of the present thesis covers.

- Summary of Matching Pursuit algorithms, written in a comparable manner
- New comparable Matlab implementations of Matching Pursuit algorithms
- Analysis based on computational and storage complexity of Matching Pursuit algorithms
- Analysis of selected methods and algorithms for applying compressed sensing in distributed systems
- Sequential test implementations of distributed algorithms for evaluating reconstruction using matching pursuit algorithms on data
- Proof of concept for applying compressed sensing and reconstruction to a signal that is sparse in wavelet a domain
- Proof of concept for applying compressed sensing to distributed signals that are not apparently sparse, whereas other methods are not applicable
- Verifying results of a new algorithm for distributed compressed sensing reconstruction, known as Common-Innovation Subspace Pursuit
- Testing applicability of new distributed compressed sensing reconstruction algorithm, known as Common-Innovation Subspace Pursuit, on signals with unknown common sparsity level

7.2 Perspectives and Further Work

The results obtained from the study of Matching Pursuit algorithms, can be used for further development of efficient implementations. As well as the study of Distrbuted Compressed Sensing forming a foundation of the knowledge needed to applying Compressed Sensing to wireless sensor networks.
Further test of the Matching Pursuit algorithm would be advised, using even more different data models in order to conlcude further on their performance. Additionally, tests using real-life data, should be performed.
Further investigation of the Common-Innovation Subspace Pursuit algorithms, ability to estimate common support set of unknown side, are required, to draw conlcusion.
Tests using the Distributed Compressed Sensing implementation should be peformed using hugh quantities of correlated signals, in order determined their true performance.

8

REFERENCES

[1] A. J. Jerri, "The shannon sampling theorem – its various extensions and applications: A tutorial review," *Proceedings of the IEEE*, vol. 65, no. 11, p. 1565–1596, 1977.

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

[3] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks.* John Wiley & Sons Inc, 2007.

[4] K. Sayood, *Introduction to data compression.* Elsevier Science & Technology, 2006.

[5] M. Lustig, D. L. Donoho, J. M. Santos, and J. M. Pauly, "Compressed sensing MRI," *IEEE Signal Processing Magazine*, vol. 25, no. 2, p. 72–82, 2008.

[6] Y. C. Eldar and G. Kutyniok, *Compressed Sensing: Theory and Applications.* Cambridge University Press, 2012.

[7] D. C. Lay, S. R. Lay, and J. J. McDonald, *Linear Algebra and its Applications.* Pearson, 2016.

[8] S. Chen and D. Donoho, "Basis pursuit," *IEEE, Asilomar Conference on Signals, Systems, and Computers*, pp. 41–44, 1995.

[9] Y. C. Pati, R. Rezaiifar, and P. Krishnaprasad, "Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition," *IEEE, Asilomar Conference on Signals, Systems, and Computers*, p. 40–44, 1993.

[10] D. L. Donoho, Y. Tsaig, I. Drori, , and J.-L. Starck, "Sparse solution of underdetermined systems of linear equations by stagewise orthogonal matching pursuit," *IEEE Transactions on Information Theory*, vol. 58, p. 1094–1121, February 2012.

[11] D. Needell and R. Vershynin, "Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit," *Springer Journal of Foundations of Computational Mathematics*, vol. 9, no. 3, p. 317–334, 2009.

[12] J. Wang and B. Shim, "Exact reconstruction of sparse signals via generalized orthogonal matching pursuit," *Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers*, pp. 1139 – 1142, 2011.

[13] D. Needell and J. A. Tropp, "Cosamp: Iterative signal recovery from incomplete and inaccurate samples," *Elsevier Applied and Computational Harmonic Analysis*, vol. 26, p. 301–321, 2009.

[14] W. Dai and O. Milenkovic, "Subspace pursuit for compressive sensing signal reconstruction," *IEEE Transactions on Information Theory*, vol. 55, pp. 2230–2249, May 2009.

[15] E. J. Candés and T. Tao, "Decoding by linear programming," *IEEE Transactions on Information Theory*, vol. 51, pp. 4203 – 4215, December 2005.

[16] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization.* John Wiley & Sons Inc, 2013.

[17] R. Baraniuk, M. A. Davenport, M. F. Duarte, and C. Hegde, *An Introduction to Compressive Sensing.* OpenStax-CNX, 2014.

[18] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning.* Springer, 2013.

[19] M. F. Duarte, S. Sarvotham, D. Baron, M. B. Wakin, and R. G. Baraniuk, "Distributed compressed sensing of jointly sparse signals," *Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers*, pp. 1537 – 1541, 2005.

[20] Q. Wang and Z. Liu, "A novel distributed compressed sensing algorithm for multichannel electrocardiography signal," *International Conference on Biomedical Engineering and Informatics (BMEI)*, vol. 4, no. 2, pp. 607 – 611, 2011.

[21] D. Sundman, S. Chatterjee, and M. Skoglund, "Greedy pursuits for compressed sensing of jointly sparse signals," *19th European Signal Processing Conference*, pp. 368 – 372, August 2011.

[22] K. Huang and J. Liu, "Inner–outer support set pursuit for distributed compressed sensing," *Ieee Transactions on Signal Processing*, vol. 66, pp. 3024 – 3039, June 2018.

[23] B. L. Sturm and M. G. Christensen, "Comparison of orthogonal matching pursuit implementations," *20th European Signal Processing Conference*, pp. 220 –224, August 2012.

[24] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, p. 3397–3415, 1993.

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

[26] D. Needell and R. Vershynin, "Signal recovery from incomplete and inaccurate measurements via regularized orthogonal matching pursuit," *IEEE Journal of Selected Topics in Signal Processing*, vol. 4, p. 310–316, April 2010.

[27] D. Needell, J. A. Tropp, and R. Vershynin, "Greedy signal recovery review," *IEEE, Asilomar Conference on Signals, Systems, and Computers*, pp. 1048–1050, 2008.

[28] J. Wang, S. Kwon, P. Li, and B. Shim, "Recovery of sparse signals via generalized orthogonal matching pursuit: A new analysis," *IEEE Transactions on Signal Processing*, vol. 64, pp. 1076–1089, February 2016.

[29] C. Luo, F. Wu, J. Sun, and C. W. Chen, "Compressive data gathering for large-scale wireless sensor networks," *Proceedingsof the 15th annual international conference on mobile computing and networking*, pp. 145 – 156, 2009.

[30] D. Sundman, S. Chatterjee, and M. Skoglund, "Parallel pursuit for distributed compressed sensing," *IEEE Global Conference on Signal and Information Processing*, pp. 783 – 786, 2013.

[31] D. Sundman, S. Chatterjee, and M. Skoglund, "Design and analysis of a greedy pursuit for distributed compressed sensing," *IEEE Transactions on Signal Processing*, vol. 64, pp. 2803–2818, June 2016.

[32] J. Liu, K. Huang, and X. Yao, "Common-innovation subspace pursuit for distributed compressed sensing in wireless sensor networks," *IEEE SENSORS JOURNAL*, vol. 19, pp. 1091–1103, February 2019.

A
ALGORITHM IMPLEMENTATIONS

In this section the implementations of the algorithms will be listed.

A.1 Orthogonal Matching Pursuit (OMP)

Listing A.11. Orthogonal Matching Pursuit (OMP)

```
% OMP.m
% Orthogonal Matching Pursuit
% ----------------------------------------
% Inputs:
%   s - Sparsity of signal
%   y - Measurement vector
%   A - Measurement Matrix
% ----------------------------------------
% Outputs:
%   x - Reconstructed signal
%   k - Iterations taken
% ----------------------------------------
function [x, k] = OMP(s,y,A)
    k = 0;
    lambda = zeros(s, 1);
    x = zeros(s, 1);
    r = y;
    while k < s
        k = k + 1;
        [~ ,omega] = max(abs(A.' * r));
        lambda(k) = omega;
        x(1:1:k) = (A(:,lambda(1:1:k)).' * A(:,lambda(1:1:k))) \
            A(:,lambda(1:1:k)).' *y;
        r = y - A(:,lambda(1:1:k))*x(1:1:k);
    end
    xh = zeros(size(A,2),1);
    xh(lambda) = x;
    x = xh;
end
```

A.2 Stagewise Orthogonal Matching Pursuit (StOMP)

Listing A.21. Stagewise Orthogonal Matching Pursuit (StOMP)

```
% StOMP.m
% Stagewise Orthogonal Matching Pursuit
% ------------------------------------------
% Inputs:
%   S - Max number of iterations to perform
%   t - Thresholding value
%   y - Measurement vector
%   A - Measurement Matrix
% ------------------------------------------
% Outputs:
%   x - Reconstructed signal
%   k - Iterations taken
% ------------------------------------------
function [x, k] = StOMP(S, t, y, A)
    k = 0;
    lambda = [];
    x = zeros(size(A,2),1);
    r = y;
    while k < S && norm(r) > 0.0001
        k = k + 1;
        v = A.' * r;
        omega = Select(v, r, A, t);
        if size(A,1) < size(omega,1)
            [size(A,1) size(omega,1)]
        end
        lambda = union(lambda, omega);
        x(lambda) = (A(:,lambda).' * A(:,lambda)) \ A(:,lambda).'
            *y;;
        r = y - A*x;
    end
end
% Helper function for indices selection
function j = Select(v, r, A, t)
    sig = norm(r) / sqrt(size(A,1));
    [B,I] = sort(abs(v),'descend');
    ind = 1;
    while B(ind+1) > sig*t && ind < max(size(B))
        ind = ind+1;
    end
    j = I(1:1:ind);
end
```

A.3 Regularized Orthogonal Matching Pursuit (ROMP)

Listing A.31. Regularized Orthogonal Matching Pursuit (ROMP)

```
% ROMP.m
% Regularized Orthogonal Matching Pursuit
% ------------------------------------------
% Inputs:
%   s - Sparsity of signal
%   y - Measurement vector
%   A - Measurement Matrix
% ------------------------------------------
% Outputs:
%   x - Reconstructed signal
%   k - Iterations taken
% ------------------------------------------
function [x, k] = ROMP(s, y, A)
    k = 0;
    lambda = [];
    x = zeros(size(A,2), 1);
    r = y;
    while norm(r) > 0.0001 && k < s
        k = k + 1;
        v = A.' * r;
        [B,I] = sort(abs(v),'descend');
        omega = subset(I(1:1:s,:), B(1:1:s,:));
        lambda = union(lambda, omega);
        x(lambda) = (A(:,lambda).' * A(:,lambda)) \ A(:,lambda).'
            *y;
        r = y - A*x;
    end
end
% Helper function for subet split and selection
function [j0] = subset(j, jb)
    ib = [1];
    ie = [1];
    while ie(length(ie)) ~= length(j)
        if jb(ib(length(ib))) <= 2*jb(ie(length(ie))+1)
            ie(length(ie)) = ie(length(ie)) + 1;
        else
            ib = [ib ie(length(ie))+1];
            ie = [ie ie(length(ie))+1];
        end
    end
    x = zeros(length(ib),1);
```

```
        for i = 1:length(ib)
            x(i) = norm( jb(ib(i):1:ie(i)) );
        end
        [~,I] = sort(x,'descend');
        ind = I(1);
        j0 = j(ib(ind):1:ie(ind)); % Select best interval
        %j0 = j(ib(1):1:ie(1)); % Select first interval
    end
    function [j0] = subsetOld(j, jb)
        j0b = abs(jb);
        j0 = j;
        while max(j0b) > 2*min(j0b)
            j0(length(j0b)) = [];
            j0b(length(j0b)) = [];
        end
        dec = 2;
        j0bt = j0b;
        j0t = j0;
        while j0bt(length(j0bt)) ~= jb(length(jb))
            j0bt = abs(jb(dec:1:length(jb)));
            j0t = j(dec:1:length(j));
            while max(j0bt) > 2*min(j0bt)
                j0t(length(j0bt)) = [];
                j0bt(length(j0bt)) = [];
            end
            if norm(j0b) < norm(j0bt)
                j0b = j0bt;
                j0 = j0t;
            end
            dec = dec+1;
        end
    end
```

A.4 Generalized Orthogonal Matching Pursuit (gOMP)

Listing A.41. Generalized Orthogonal Matching Pursuit (gOMP)

```
% gOMP.m
% Generalized Orthogonal Matching Pursuit
% ------------------------------------------
% Inputs:
%   s - Sparsity of signal
%   S - Indices to select each iteration
%   y - Measurement vector
%   A - Measurement Matrix
% ------------------------------------------
% Outputs:
%   x - Reconstructed signal
%   k - Iterations taken
% ------------------------------------------
function [x, k] = gOMP(K, S, y, A)
    k = 0;
    lambda = [];
    x = zeros(size(A,2), 1);
    r = y;
    [m,~] = size (A);
    while (k < (m/S)) && (norm(r) > 0.0001)
        k = k + 1;
        v = A.' * r;
        [~,I] = sort(abs(v),'descend');
        omega = I( 1:1:S );
        lambda = union(lambda,omega);
        x(lambda) = (A(:,lambda).' * A(:,lambda)) \ A(:,lambda).'
            *y;
        r = y - A*x;
    end
end
```

A.5 Compressive Sampling Matching Pursuit (CoSaMP)

Listing A.51. Compressive Sampling Matching Pursuit (CoSaMP)

```
% CoSaMP.m
% Compressive Sampling Matching Pursuit
% ------------------------------------------
% Inputs:
%   s - Sparsity of signal
%   y - Measurement vector
%   A - Measurement Matrix
% ------------------------------------------
% Outputs:
%   x - Reconstructed signal
%   k - Iterations taken
% ------------------------------------------
function [x, k] = CoSaMP(s, y, A)
    k = 0;
    lambda = [];
    x = zeros(size(A,2), 1);
    r = y;
    while ((k <= s) && (norm(r) > 0.001))
        k = k + 1;
        v = A.' * r;
        [~,I] = sort(abs(v),'descend');
        omega = I(1 : 1 : min(s*2, size(A,2)));
        [lambda,~] = find(x);
        lambda = union(lambda, omega);
        x(lambda) = (A(:,lambda).' * A(:,lambda)) \ A(:,lambda).'
            *y;
        B = sort(abs(x), 'descend');
        x(abs(x) < B(s)) = 0;
        r = y - A*x;
    end
end
```

A.6 Subspace Pursuit (SP)

Listing A.61. Subspace Pursuit (SP)

```
% SP.m
% Subspace Pursuit
% ------------------------------------------
% Inputs:
%   s - Sparsity of signal
%   y - Measurement vector
%   A - Measurement Matrix
% ------------------------------------------
% Outputs:
%   x - Reconstructed signal
%   k - Iterations taken
% ------------------------------------------
function [x, k] = SP(s,y,A)
    x = zeros(size(A,2), 1);
    r = y;
    k = 0;
    [~,I] = sort(A.'*r, 'descend');
    lambda = I(1:1:s,:);
    x(lambda) = (A(:,lambda).' * A(:,lambda)) \ A(:,lambda).' *y;
    ro = r;
    r = y - A*x;
    while norm(ro) > norm(r) & k <= s
        k = k+1;
        [~,I] = sort(abs(A.'*r), 'descend');
        lambda = union(lambda, I(1:1:s,:));
        x = zeros(size(A,2), 1);
        x(lambda) = (A(:,lambda).' * A(:,lambda)) \ A(:,lambda).'
            *y;
        [~,I] = sort(abs(x), 'descend');
        lambda = I(1:1:s,:);
        x = zeros(size(A,2), 1);
        x(lambda) = (A(:,lambda).' * A(:,lambda)) \ A(:,lambda).'
            *y;
        ro = r;
        r = y - A*x;
    end
```

A.7 Joint Subspace Pursuit (JSP)

Listing A.71. Joint Subspace Pursuit (JSP)

```
% JSP.m
% Joint Subspace Pursuit
% ------------------------------------------
% Inputs:
%   y - Correlated measurement vectors
%   A - Measurement Matrixs
%   sc - Common support set size
%   si - Innovation support set size
% ------------------------------------------
% Outputs:
%   x - Reconstructed signal
%   k - Iterations taken
% ------------------------------------------
function [x, k] = JSP(y, A, sc, si)
    k = 0;
    %y2 = reshape(y, size(y,1), size(y,3));
    r = sum(y.').';
    ro = r+1;
    lambda = [];
    x = zeros(size(A,2), size(A,3));
    while norm(ro) > norm(r)
        k=k+1;
        sup = zeros(size(A,2),1);
        ro = r;
        r = zeros(size(r));
        for l = 1:size(y,2)
            [x(:,l), rl] = modifiedSP(y(:,l), A(:,:,l), sc+si,
                lambda);
            r = r + rl;
            sup = vote(sup, find(x(:,l)));
        end
        [~,I] = sort(sup, 'descend');
        lambda = I(1:1:sc);
    end
end
function [x, r] = modifiedSP(y, A, s, lini)
    x = zeros(size(A,2), 1);
    r = y;
    lambda = [];
    l = 0;
    [~,I] = sort(abs(A.'*r), 'descend');
```

```
        lambda = union(I(1:1:s,:), lini);
        x(lambda) = LSESolve(A(:,lambda), y);
        ro = r;
        r = y - A*x;
        while norm(ro) > norm(r)
            l = l+1;
            [~,I] = sort(abs(A.'*r), 'descend');
            lambda = union(lambda, I(1:1:s,:));
            x = zeros(size(A,2), 1);
            x(lambda) = LSESolve(A(:,lambda), y);
            [~,I] = sort(abs(x), 'descend');
            lambda = I(1:1:s,:);
            x = zeros(size(A,2), 1);
            x(lambda) = LSESolve(A(:,lambda), y);
            ro = r;
            r = y - A*x;
        end
    end
    function [sup] = vote(sup, new)
        sup(new) = sup(new) + 1;
    end
```

A.8 Sequential test - Distributed Parallel Pursuit (DIPP)

Listing A.81. Sequential test - Distributed Parallel Pursuit (DIPP)

```
% DIPP.m
% Distributed Paralle Pursuit
% ------------------------------------------
% Inputs:
%   y - Correlated measurement vectors
%   A - Measurement Matrixs
%   in - incomming connection
%   out - outgoing connection
%   nodes - Number of measurements
% ------------------------------------------
% Outputs:
%   x - Reconstructed signal
%   k - Iterations taken
% ------------------------------------------
function [x, k] = DIPP(y, A, s, in, out, nodes)
    x = zeros(size(A,2), nodes);
    tp = zeros(s,nodes);
    r = zeros(size(y));
    for i = 1:nodes
        [x(:,i), tp(:,i), r(:,i)] = SIPP(s, y(:,i), A(:,:,i), []);
    end
    k = 0;
    ro = y;
    while norm(sum(r,2)) < norm(sum(ro,2))
        k = k+1;
        ro = r;
        % -- Transmit
        % -- Receive
        J = zeros(s, nodes);
        for i = 1:nodes
            j = consensus(tp(:,in(:,i)), tp(:,i), s, x(:,i));
            J(:,i) = [j ; zeros(s-size(j,1),1)];
        end
        tpsi = zeros(s, nodes);
        for i = 1:nodes
            tpsi(:,i) = expansion(J(:,i), x(:,i), s);
        end
        %[x, tp, r] = SIPP(s, y, A, tpsi);
        for i = 1:nodes
            [x(:,i), tp(:,i), r(:,i)] = SIPP(s, y(:,i), A(:,:,i),
                tpsi(:,1));
```

```
            end
            z = [tp(:,1) tp(:,2) tp(:,3) tp(:,4) tp(:,5) tp(:,6) tp
                (:,7) tp(:,8)];
        end
    end
    %% Main helpers
    function [x, lambda, r] = SIPP(s, y, A, tsi)
        x = zeros(size(A,2), 1);
        r = y;
        ro = inf;
        lambda = [];
        l = 0;
        while (norm(ro) > norm(r)) && (l < size(A,2) )
            [~,I] = sort(abs(A.'*r), 'descend');
            lambda = union(lambda, I(1:1:s,:));
            x = zeros(size(A,2), 1);
            x(lambda) = LSESolve(A(:,lambda), y);
            [~,I] = sort(abs(x), 'descend');
            lambda = union(tsi, I(1:1:s,:));
            x = zeros(size(A,2), 1);
            x(lambda) = LSESolve(A(:,lambda), y);
            [~,I] = sort(abs(x), 'descend');
            lambda = I(1:1:s,:);
            x = zeros(size(A,2), 1);
            x(lambda) = LSESolve(A(:,lambda), y);
            ro = r;
            r = y - A*x;
        end
    end
    function J = consensus(tq, tp, T, x)
        z = zeros(size(x));
        z(tp) = 1;
        for i = 1:size(tq, 2)
            z(tq(:,i)) = z(tq(:,i)) + 1;
        end
        s = 2;
        while sum(z(:) >= s) > T
            s=s+1;
        end
        J = find(abs(z) >= s);
    end
    function Tpsi = expansion(J, x, T)
        J(find(J == 0)) = [];
        x(J) = 0;
        [B,I] = sort(abs(x), 'descend');
```

```
        Tpsi = union(J, I(1:1:T-length(J)));
        hold = 1;
    end
```

A.9 Common-Innovation Subspace Pursuit (CISP)

Listing A.91. Common-Innovation Subspace Pursuit (CISP)

```
% CISP.m
% Common-Innovation Subspace Pursuit
% ------------------------------------------
% Inputs:
%   y - Correlated measurement vectors
%   A - Measurement Matrixs
%   cluster - Number of clusters
%   clustersize - Size of the clusters
%   sc - Common support set size
%   si - Innovation support set size
% ------------------------------------------
% Outputs:
%   x - Reconstructed signals
%   k - Iterations taken
% ------------------------------------------
function [x] = CISP(y, A, cluster, clustersize, sc, si)
    x= zeros(size(A,2), size(A,3));
    Y = zeros(size(y,1),cluster);
    for i = 1:cluster
        int = ((i-1)*clustersize+1):1:(i*clustersize);
        Y(:,i) = (1/clustersize) * sum(y(:,int),2);
    end
    Ac = (1/(cluster*clustersize)) * sum(A,3);
    [V,D] = eig(Y*Y.');
    V2 = V(:, max(size(V,2)-sc+1,1):1:size(V,2));
    dist = zeros(1,size(Ac,2));
    for i = 1:size(Ac,2)
        dist(i) = 1 - ( norm(V2.' * Ac(:,i))/norm(Ac(:,i)) );
    end
    [B,I] = sort(dist, 'ascend');
    C = sort(I(1:1:sc));
    Acc = Ac(:,C);
    inter = 1:1:size(A,2);
    inter(C) = [];
    test = zeros(1,400); i=1;
    for c = 1:cluster
        for cs = 1:clustersize
            ind = (c-1)*clustersize+cs;
            test(ind) = i;
            i=i+1;
            Acc = A(:,C,ind);
```

```
                P = eye(size(A,1)) - ((Acc / (Acc.' * Acc)) * Acc.');
                yp = P*y(:,ind);
                At = A(:,:,ind);
                At(:,C) = [];
                Ap = P*At;
                xk = zeros(size(A,2),1);
                xk(inter) = SP(si, yp, Ap);
                xk(C) = 1;
                xk(C) = LSESolve(Acc,y(:,ind)-At*xk(inter));
                x(:,ind) = xk;
            end
        end
    end
```

LIST OF ALGORITHMS

C
NOTATION

In this thesis, the following notations are used. Scalars are denoted as lower case letters e.g. c. Vectors are denoted as lower case bold letters e.g. $\mathbf{x}$. Matrices are denoted as capital bold letter e.g. $\mathbf{A}$. Capital calligraphy letter will be used to indicate sets of processing notes.

A few general help function are:

$\text{supp}(\mathbf{x})$	The support of vector $\mathbf{x}$, i.e. indexes associated with nonzero coefficients.
$\text{supp}(\mathbf{x}, c)$	The index of the c highest coefficient values in $\mathbf{x}$