ABSTRACT

Image Compression and Recovery Using Compressive Sampling
and Particle Swarm Optimization

Benjamin D. Van Ruitenbeek, M.S.

Mentor: David B. Sturgill, Ph.D.

We present a novel method for sparse signal recovery using Particle Swarm Optimization and demonstrate an application in image compression. Images are compressed with compressive sampling, and then reconstructed with particle swarm techniques. Several enhancements to the basic particle swarm algorithm are shown to improve signal recovery accuracy. We also present techniques specifically for reconstructing sparse image data and evaluate their performance.

Image Compression and Recovery Using Compressive Sampling
and Particle Swarm Optimization

by

Benjamin D. Van Ruitenbeek, B.S.

A Thesis
Approved by the Department of Computer Science

Donald L. Gaitros, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of
Master of Science

Approved by the Thesis Committee

David B. Sturgill, Ph.D., Chairperson

Robert J. Marks II, Ph.D.

Peter M. Maurer, Ph.D.

Accepted by the Graduate School
August 2009

J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

Introduction

It is difficult to determine the amount of information in a signal through observation. Consider the signal shown in Figure 1.1. This signal would be difficult to describe in English using local minima and maxima because it contains a lot of variation. In other words, the signal appears dense, with much more information than a simple sine wave. However, this signal can be described exactly as the sum of three separate sine waves shown in Figure 1.2. The sine waves combine together to produce a signal that appears to contain a lot of information, but actually does not require much information to describe. There are many examples of signals that appear dense but can actually be described with only a small amount of information. Images and audio are two examples of signals that appear to be dense but can be represented in a sparse manner. The underlying latent sparsity of these signals can be exploited to encode the signals in a compressed format.

1.1 Frequency Encoding

Since the sparsity of an input signal $x \in \mathbb{R}^N$ may not be immediately obvious, it must be computed. We denote the elements of x as $x_1 \dots x_N$. The preferred method for determining the sparse signal representation involves a transformation V from a time or spatial domain into a frequency domain. The frequency-domain representation of x is u. Using the transformation V, this can be expressed as $u =$

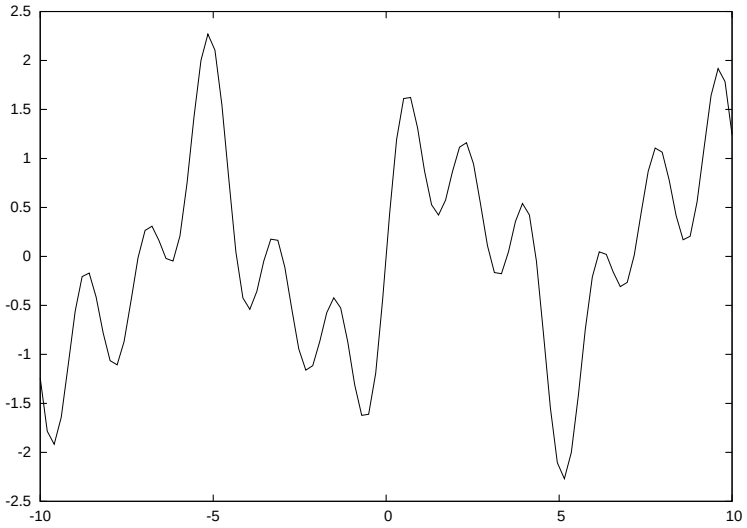

Figure 1.1: Although this signal appears to be complex, it can be easily described as the sum of three sine waves.

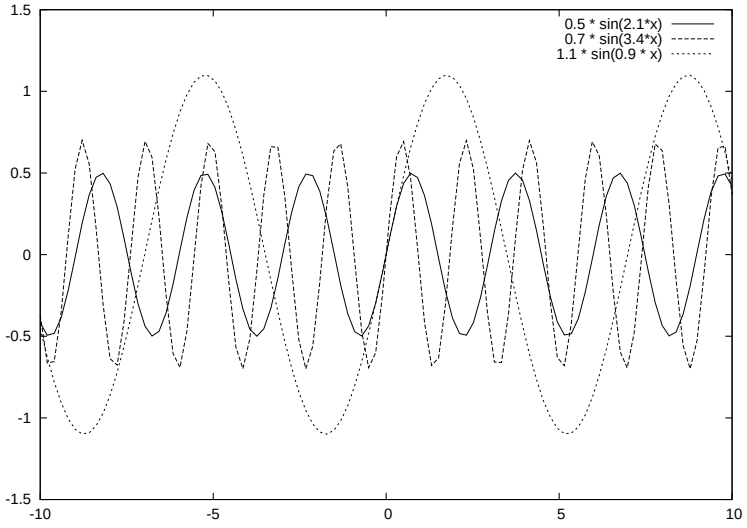

Figure 1.2: The sine waves shown here added together create the signal shown in Figure 1.1.

Vx. One of the most well-known time to frequency-domain transformations is the Discrete Fourier Transform (DFT) (CLW69). Other common transformations include the Discrete Wavelet Transform (DWT) (Mal89) and the Discrete Cosine Transform (DCT) (ANR74).

The DCT is a transformation similar to the DFT that is used for images, among other applications. The DCT creates a frequency-domain view of an input signal by expressing the signal as the sum of cosine functions. For many natural signals, after the DCT is applied, only a few frequencies are needed to capture the information in the signal. Therefore, the unused frequency components can be discarded without losing much information. The resulting signal is sparse, yet contains most of the original data. The sparsity in a signal can be expressed as K, the number of frequencies required to represent the signal in a frequency domain. We say that a signal is K-*sparse* if at most K components of the frequency representation are non-zero. If more frequencies are required, K increases and the level of sparsity decreases. If fewer frequencies are required, K decreases and the level of sparsity increases.

The frequency-domain transformation on a signal does not compress the signal; it only shows that the signal is compressible by revealing its sparsity. In practice, it is not likely that a signal will contain $N-K$ zero-magnitude frequency components, but it is likely that these components will be relatively small in magnitude. Therefore, if a signal can be approximated by a sparse signal then it can be compressed. The signal can be compressed in a variety of different ways. This research will focus on a specific method of compression known as compressive sampling. To understand

compressive sampling, it is important to first understand the traditional method of sampling a signal.

1.2 Sampling

A *sample* is simply the amplitude of a signal at a given point in time. Any continuous signal can be converted into a discrete representation if a series of samples are taken from the signal at regular intervals. According to the Nyquist-Shannon Sampling Theorem (Sha49; Nyq02), a continuous signal can be expressed exactly as a series of samples if the sampling rate is at least twice the maximum frequency in the original signal. If this sampling rate is used, the original signal can be reconstructed from the samples with no information loss. The reconstructed signal is then exactly identical to the original signal.

1.3 Compressive Sampling

Compressive sampling, also known as compressed sensing (Don06; CW08), offers an alternative approach to the signal compression problem for sparse signals. Compressive sampling is a method of compressing a sparse signal by taking a small number of random samples. Whereas traditional sampling methods take samples at different points in time, compressive sampling takes each sample by using a transformation of the entire input signal. The compressed signal is expressed in the form Φx where Φ is a $M \times N$ sampling matrix. The number of samples, M, is dependent on the sparsity, K, of the input signal. When the input signal is sparse, there is little information in the signal, and few samples are required to represent the signal. As

the amount of information in an input signal increases, the signal is less sparse, and more samples are required to capture all of the information in the signal.

The main benefit of compressive sampling is that it requires only a small number of samples to represent a signal. However, the compression transformation is not invertible, so it may be difficult to recover the original signal from its compressed representation. The process of reconstructing the signal given the samples is known as *sparse signal recovery* and can be an NP-hard problem.

1.4 Signal Recovery

Various methods for signal recovery can be used, but they can be generally divided into two groups: convex optimization and greedy algorithms. The convex optimization approach, known as *Basis Pursuit* (CD94; CDS99), requires a relatively small number of samples but is computationally inefficient compared to the greedy algorithms. The basic greedy method for signal recovery is *Orthogonal Matching Pursuit* (OMP) (TG07) which provides a deterministic, fast solution to the signal recovery problem. Other greedy algorithms such as *Regularized OMP* (ROMP) (NV07) and *Compressive Sampling Matching Pursuit* (CoSaMP) (TN08) improve upon OMP.

To achieve a high level of compression, the fewer samples that are used, the fewer bits are needed to encode those samples as the compressed signal. Given the same number of samples and number of bits used for the encoding of the compressed signal, different method of signal recovery may recover the original signal with varying amounts of error. Different algorithms may also require a different number of samples to produce equivalent signal reconstructions. Ultimately, when designing a sparse

signal recovery algorithm, there are many tradeoffs to consider such as resource usage and encoded signal size.

1.5 Particle Swarm Optimization

We develop a new method of signal recovery from a compressively sampled signal using *Particle Swarm Optimization* (PSO). PSO (EK95; KE95) is an evolutionary computing algorithm which uses particles to search different areas of a search space at the same time. The particles attempt to find a location which minimizes or maximizes an optimization criterion. The *fitness* of a particle refers to how well the particle's location meets the optimization criteria. Particles start in a random location within the search space with a random velocity. Each particle keeps track of the best location it has visited so far; this location is referred to as the local best for that particle. All particles also have knowledge of the best location any particle has visited; this is known as the global best. The velocity of each particle is updated in an iterative manner to move the particle in the direction of the local best and the direction of the global best.

We develop a technique that uses PSO to recover a compressively sampled signal given the compressed representation and the sparsity of the signal. The search space for PSO is the space of all signals in $\mathbb{R}^N$. Each particle location represents a candidate signal, and the goal of particle swarm is to find the candidate signal that most closely approximates the original signal. Since the original signal is not available for comparison during the recovery process, a candidate solution is considered optimal if it can create the same compressed representation as the original signal under Φ and

it has the correct level of sparsity. The result of the particle swarm is a recovered input signal that is represented by the global best location in the search space.

1.6 Image Compression

To test the effectiveness of PSO on natural signals, we chose image compression as the target application for this research. Images are a good test for compressive sampling and subsequent recovery methods because image compression is a practical application. Additionally, signal recovery accuracy for image data can be demonstrated visually as well as statistically. Images can be approximated well with sparse signals in the frequency domain; this is a basic assumption for a variety of image compression techniques. We thus consider images to meet the sparse signal requirement of compressive sampling.

To use compressive sampling for image compression, images are divided into square blocks so that each block can be compressed separately. The integer pixel values in each block of an image are converted to real-number values. The real-valued signals for each block are compressively sampled, then recovered using PSO.

1.7 Research Goals

The first major goal of this research is to produce an accurate method of sparse signal recovery using PSO. The straightforward application of PSO to the sparse signal recovery problem may not provide optimal signal reconstruction. Therefore, it may be possible to improve the accuracy of the signal recovery process by modifying the particle swarm algorithm. There are many possible variants to the basic

particle swarm algorithm to investigate, including different search spaces, particle initialization schemes, particle update variations, and fitness criteria.

If successful, this research will yield a class of techniques that are competitive with state-of-the-art methods for sparse signal recovery. The particle swarm solution can be compared to the OMP solution in terms of reconstruction error. The accuracy in signal recovery can be measured by calculating the root mean square (RMS) error between the original signal and the recovered signal. As the accuracy of the signal recovery increases, the RMS error will decrease.

We intend to demonstrate a sparse signal recovery technique that can be successfully used with real-world signals. This requirement adds complexity to the signal recovery process. We show that a PSO-based signal recovery approach can handle real-world signals by reconstructing images that have been compressively sampled.

Finally, the problem domain of image compression lends itself to domain-specific methods for increasing signal reconstruction accuracy. Certain assumptions about image data can be used to decrease error in the particle swarm recovery process.

1.8 Thesis Structure

Chapter Two presents some related work in the areas of compressive sampling, sparse signal recovery, particle swarm, and image compression. The implementation of the compressive sampling process and signal recovery with PSO is described in Chapter Three. Chapter Four expands the PSO-based signal recovery approach to include image-specific techniques that reduce error in signal recovery. Results of

research experiments on synthetic signals and images are presented in Chapter Five. Chapter Six draws conclusions and offers ideas for future research.

CHAPTER TWO

Related Work

There are several existing techniques for the compressive sampling of sparse signals and for recovering compressed signals. Although we are principally concerned with the application of Particle Swarm Optimization to signal recovery, PSO is a general-purpose optimization technique that has been successfully applied to a variety of tasks. We also present an overview of image compression to provide a background for understanding image compression with compressive sampling.

2.1 Compressive Sampling

Compressive sampling (Don06; CW08) is a process whereby a sparse signal $x \in \mathbb{R}^N$ is simultaneously compressed and sampled. Each sample is a random linear combination of all of the elements in the signal. In other words, a sample is the dot product of the signal and a vector of random weights known as a *sampling vector*. The compressed representation of the signal is the collection of samples. Put together, the sampling vectors form the rows of Φ, the sampling matrix. Therefore, the number of rows in Φ is equivalent to the number of samples that are taken. The compressed signal can be expressed mathematically as a *measurement vector* in the form of $y = \Phi x$.

The number of samples required to capture the information in a signal is dependent upon the sparsity of a signal. A frequency-domain transformation V such as the

DCT, DFT, or DWT can be used to reveal the sparsity in a signal. The frequency-domain representation of a signal x is u, so $u = Vx$. With K non-zero frequency components in a signal, the number of samples, M, is chosen as a function of K. If the number of samples is less than the length, N, of the original signal, then Φ is not invertible and recovering the original signal becomes an underdetermined problem. However, the set of samples only needs to contain enough information to recover the K non-zero components in the frequency domain. If the signal is sparse enough, the number of samples, M, will be less than the original dimensionality of the signal, N, so the signal will be compressed.

A sparse input signal is not the only requirement for compressive sampling. Each sampling vector must be spread out in the V domain so that the input signal is compressed evenly; this property is known as *incoherence*. If the sampling matrix exhibits optimal incoherence with the transformation V, then the number of samples can be minimized. This criterion can be met by taking the rows of Φ from a random orthonormal matrix (CR07).

Another important consideration is the mapping of sparse input signals to compressed samples. If two input signals with the same sparsity map to the same set of samples, the signal recovery process may reconstruct either of the two signals. To avoid this problem, each input signal of a given sparsity should map to a unique set of compressed samples. One way to ensure a unique mapping is to place a restriction on the sampling matrix. If each set of M columns from the sampling matrix can form a nonsingular, invertible matrix, the one-to-one mapping of input signals to compressed samples can be preserved. This restriction is known more formally as

the *Restricted Isometry Condition* (RIC). The condition can be met by using random Gaussian, Bernoulli, and partial Fourier matrices (MPTJ06; RV06).

Compressive sampling has been used for a wide variety of applications, including image compression (WLD$^+$06; Rom08), medical imaging (LDP07), biosensing (SMB07), and communications (CR02; BHSN06; TH08).

2.2 Exact and Approximate Sparsity

The sparsity of a signal is determined by a frequency-domain transformation. The sparse representation of a signal in the frequency domain is created from only a few frequencies. When a sparse signal can be represented completely by K frequencies, and all other $N - K$ frequencies are not used at all, we say it has *exact sparsity*.

However, for many natural signals, the frequency-domain representation requires some frequencies to make extremely small contributions. In this case, there are K large-magnitude frequencies, and $N - K$ frequencies which are near-zero but may not be exactly zero. We say that these signals have *approximate sparsity*. Natural signals such as audio and image data can be expected to exhibit approximate rather than exact sparsity, so they present a challenge for compressive sampling. Image compression methods often discard the small-magnitude frequencies of approximately sparse signals (WLD$^+$06; Wal92).

2.3 Signal Recovery

Many approaches to the signal recovery problem have been proposed, although they generally fall within two major categories: convex optimization and greedy al-

gorithms. A function is *convex* on an interval if it lies below a straight line segment connecting two points, for any two points in the interval. Convex optimization attempts to find the minimum value of a convex function given a set of inequality constraints. In this case, convex optimization minimizes a function that helps to produce a signal approximation. Greedy algorithms reconstruct a signal with an iterative process that makes locally optimal decisions during each iteration.

2.3.1 L_1-Minimization

Given a compressed signal Φx, the original signal x can be recovered by finding an approximation z that is as sparse as possible and can produce the same set of samples as x. The sparsity constraint can be satisfied by minimizing the L_0 norm of z. If we define $0^0 = 0$, the L_0 norm of z is simply the number of non-zero components of z:

$$|z|_0 = \Sigma\ z_i^0 \tag{2.1}$$

The sample constraint can be met by only considering signals for which $\Phi z = \Phi x$. However, this L_0-minimization problem is computationally difficult since there are many possible signal approximations to consider, and there is no efficient way to search through them. L_0-minimization is believed to be NP-hard (CRTV05).

Since L_0-minimization is difficult, Donoho et al. (DS89) proposed minimizing the L_1 norm of z instead of the L_0 norm, because minimizing the L_1 norm still promotes sparsity in z. The L_1 norm measures the sum of the magnitudes in a vector, and can be defined as:

$$|z|_1 = \Sigma\ |z_i| \tag{2.2}$$

This approach, L_1-minimization, is also known as Basis Pursuit (CD94; CDS99).

Indirectly, linear programming can be used to solve the L_1-minimization problem. For this application, a linear programming problem can be characterized as the problem of minimizing the sum of elements in u subject to the equality constraint $\Phi V^{-1}u = y$ and with the additional inequality constraint that all $u_i \geq 0$. This characterization cannot be used directly to solve L_1-minimization problems since it constrains all elements of u to be non-negative. However, with a modest increase in the problem size, a L_1-minimization problem can be cast in terms of linear programming. To handle the possibility of positive and negative elements in u, the solution vector is represented as a $2N$-element vector consisting of a q part representing the positive elements in u followed by an r part holding the negative elements in u. As such, signal recovery via L_1-minimization can be solved via a linear programming problem like the following:

$$\min (1^T q + 1^T r) \text{ subject to } \Phi(q - r) = y \text{ with } q, r \geq 0 \tag{2.3}$$

In order to minimize the magnitude of the values of u, we can select basis vectors from either Φ or $-\Phi$ to find non-zero components of y.

The L_1-minimization technique only requires a small number of samples to recover the original signal. The method is stable and robust enough to handle noisy signals, and it will recover any sparse signal if the sampling matrix satisfies the RIC. These guarantees are known formally as *uniform guarantees* for sparse recovery. Unfortunately, linear programming is still computationally inefficient when compared to other recovery techniques because there is no strongly polynomial time linear

programming algorithm. A strongly polynomial time algorithm can run in polynomial time with respect to the size of the input, rather than the numerical value of the input.

2.3.2 Orthogonal Matching Pursuit

Greedy algorithms such as OMP and its refinements improve upon L_1-minimization by providing a much more computationally efficient solution. Orthogonal Matching Pursuit (TG07) uses a simple iterative method to develop an approximation of the signal x. OMP selects K components of the signal one at a time until the approximation has the necessary sparsity. The process of selecting a component is the greedy step: the component that contributes most to the measurement vector is removed from the residual. For this algorithm, a transformation that maps u all the way to y is used; this enhanced matrix is computed as $\Phi_E = \Phi V^{-1}$.

OMP consists of the following steps:

1. Initialize the residual to the measurement vector y. Initialize the index set of columns of Φ_E to the null set.
2. From the columns not in the index set, find the column of Φ_E that contributes the most to the residual and add it to the index set.
3. Orthogonalize the newly selected column with the column vectors selected in previous iterations, then normalize it.
4. Remove the now orthonormal column vector's contribution from the residual.
5. Repeat from step 2 until K columns have been selected.
6. The K selected column indices correspond to the K non-zero dimensions in u. Using Φ_K generated from the K selected columns of Φ_E, the overdetermined system $\Phi_K u = y$ can be solved for a unique solution for u.

With a recovered signal u in the frequency domain, V^{-1} can be used to recover x since $x = V^{-1}u$. While OMP provides a fast solution to sparse signal recovery, it does

not provide the uniform guarantees of the L_1-minimization technique. In particular, OMP cannot guarantee reconstruction of noisy signals. Other greedy algorithms built upon the ideas of OMP in an attempt to produce a fast algorithm with strong signal recovery guarantees.

2.3.3 Regularized Orthogonal Matching Pursuit

Regularized Orthogonal Matching Pursuit (ROMP) (NV07) is a signal recovery approach which combines the speed of OMP with the recovery performance of L_1-minimization. ROMP selects up to K components at each iteration, instead of one component per iteration like OMP.

The ROMP algorithm consists of the following steps:

1. Initialize the residual to the measurement vector y. Initialize the index set of selected component dimensions to the null set.
2. Select the K components that individually contribute the most to the residual, or all of the components that contribute to the residual if there are fewer than K of these components.
3. Find all subsets of the selected components for which the largest-magnitude component is no more than twice the magnitude of the smallest-magnitude component.
4. Select the subset with the largest L_2 norm, and add the component dimensions represented in the subset to the index set.
5. Update the residual by removing the components of the selected subset.
6. Repeat from step 2 until the residual is zero.

At the conclusion of the ROMP algorithm, the index set will list the dimensions of non-zero components in the frequency domain. Selecting the columns of Φ_E from the index set will produce an overdetermined system which can be solved for a signal

approximation u. ROMP is able to provide exact recovery of sparse signals if the sampling matrix meets the RIC.

2.3.4 Compressive Sampling Matching Pursuit

Compressive Sampling Matching Pursuit (CoSaMP), developed by Tropp and Needell (TN08), provides efficient resource usage as an improvement upon regularized OMP. The CoSaMP algorithm finds a signal approximation using the following steps:

1. Initialize the current samples, v, to the measurement vector, y.
2. Calculate a signal proxy, s, from the current samples using the pseudoinverse $s = \Phi^T(\Phi\Phi^T)^{-1}v$ and identify the largest components of the signal proxy.
3. If this is not the first iteration, take the union of the set of largest components specified by the current approximation, a, and the set of the $2K$ largest components of the signal proxy to create the set of components, C. If this is the first iteration, the current approximation will not yet be specified.
4. Using the merged set of components, C, select columns of Φ that correspond to the selected components. Call this new matrix Φ_C. Calculate a new signal approximation using a least-squares solution: $a = \Phi_C^T(\Phi_C\Phi_C^T)^{-1}y$
5. Only retain the K largest components of the new approximation by zeroing out the other components.
6. Update the samples based on the new approximation: $v = y - \Phi a$
7. Repeat from step 2 until a halting criterion is reached.

CoSaMP is a highly configurable algorithm; the number of components selected in step 2 and the number of components retained in step 5 can be adjusted to improve performance. Additionally, the halting criterion can either specify a fixed number of iterations or use a variable measure such as the L_2 norm of the current samples. Like ROMP and L_1-minimization, CoSaMP can guarantee recovery of any sparse signal using a sampling matrix that satisfies the RIC.

2.4 Particle Swarm Optimization

We take a new approach to sparse signal recovery. If it were possible to efficiently search high-dimensional spaces, a compressively sampled signal could be recovered by searching through possible signal approximations. We turn to PSO as a high-dimensional search space technique to let us recover a sparse signal from compressed samples.

Eberhart and Kennedy first developed PSO (EK95; KE95) as a technique for non-linear function optimization and artificial neural network training. PSO was created out of an attempt to model the social behavior of organisms such as a school of fish or flock of birds. However, the benefits of particle swarm as an optimization method quickly became apparent. PSO has been shown to work for various applications including multiobjective optimization problems (CPL04), fractal image compression (FCY07), and lossless data compression (SZZ06).

PSO searches for an optimal position in a search space by moving particles around and testing various locations with a *fitness function*. The fitness function takes a particle location as input, and returns a measure of how well that location meets the criteria under optimization.

PSO is an iterative algorithm. During each iteration, all particles update their velocity, then update their position based on their new velocity. The following two equations describe how a particle's velocity and position are updated:

$$V_i \leftarrow w\, V_i + c_1\, r_1()(P_i - X_i) + c_2\, r_2()(P_g - X_i) \tag{2.4}$$

and

$$X_i \leftarrow X_i + V_i \tag{2.5}$$

where

- w is an inertia weight constant which specifies how much of the previous velocity to retain,
- c_1 and c_2 are positive constants,
- $r_1()$ and $r_2()$ are random variables on $[0, 1]$,
- $X_1, \ldots X_D \in \mathbb{R}^d$ represent the d-dimensional position of each of D particles,
- $V_1, \ldots V_D \in \mathbb{R}^d$ represent the velocity of each particle,
- $P_1, \ldots P_D \in \mathbb{R}^d$ represents the best previous position of each particle as determined by a fitness function, and
- g represents the index of the particle with the best previous fitness.

A higher inertia weight will allow more momentum from the previous velocity to carry over to the new velocity value. A lower inertia weight will cause the overall velocity of the particle to decrease more rapidly. The final global best position is taken as the result of the particle swarm search; it is the best position found by any particle during the search. In terms of sparse signal recovery, the global best position represents the best signal approximation recovered by PSO.

2.5 Image Compression

We apply our PSO-based sparse signal recovery method to image compression to investigate its effectiveness on natural signals. The research area of image compression (Cla95) provides many widely-used compression methods that provide relevant background and a basis of comparison for our application. Image compression schemes can be divided into two categories: lossless compression and lossy compression. Lossless compression allows the exact input signal to be reconstructed after it is compressed.

Lossy compression can achieve a much higher compression by discarding some information in the signal. At low compression levels, the visually insignificant parts of the signal are discarded, so there is not much perceptible change in the compressed image compared to the original. At higher compression levels, the compressed image can loose quality and exhibit compression artifacts as a result of the compression process.

There are several different categories of lossy compression; transform coding is the most analogous to compressive sampling. Transform coding uses a frequency-domain transformation to selectively discard the higher frequency data which only contain a small part of the information in an image. Popular frequency-domain transformations for image compression include the DCT (BMS97) and the DWT (LK92).

2.5.1 JPEG

We base our compressive sampling-based image compression method on an existing transform coding method: the widely used Joint Photographic Experts Group (JPEG) compression format (Wal92). The JPEG standard offers a variable compression ratio to allow for higher quality with less compression or lower quality with more compression. The compression ratio determines how aggressively the quantization step reduces the information stored in the high frequency components of the signal.

The JPEG compression process contains the following steps:

1. Convert the image to the YCbCr color space so that it has one brightness component and two color components.
2. Use chroma subsampling to reduce the amount of information kept for the color components.

3. Split the image into 8-by-8 pixel blocks, then apply a two-dimensional DCT to each block.

4. Quantize the resulting block of frequency information using a quantization matrix which uses more bits to encode low-frequency components and fewer bits to encode high-frequency components.

5. Encode the block with run-length and Huffman coding, moving through the block diagonally to group similar frequencies together. The DC coefficient is encoded separately from the other 63 DCT coefficients.

2.5.2 Image Compression through Compressive Sampling

Like JPEG, we divide an image into 8-by-8 pixel blocks. We compressively sample each block, and quantize the resulting measurement vector so that it can be encoded as a vector of integers rather than floating point values. This technique is a lossy compression technique since the recovery process may yield only an approximation of the original signal. In addition, the quantization process can introduce loss, and characterizing an approximately sparse signal as K-sparse will necessarily ignore some information in the signal.

Compressive sampling has already been successfully applied to image compression. Wakin et al. (WLD$^+$06) use a digital camera that takes random projections of a signal without collecting the pixels. This approach allows them to measure the image fewer times than the number of pixels in the image. Romberg presents another approach to image compression which does not divide an image into blocks (Rom08). This approach considers the pixels in one 256-by-256 pixel image to be a 65,536-dimensional signal. The image is represented by the first 1,000 DCT coefficients as well as random compressive samples taken over the remaining DCT coefficients.

CHAPTER THREE

Signal Recovery via Particle Swarm

We present a PSO-based signal recovery process for the reconstruction of compressively sampled sparse signals. We also develop several refinements to our initial particle swarm technique in an effort to lower reconstruction error. Refinements specifically applicable to natural signals and image compression are covered in the next chapter.

3.1 Direct Method

The *direct method* of sparse signal recovery is a straightforward application of PSO to find a signal $x' \in \mathbb{R}^N$ that best approximates x given samples produced from x and the value of K for that x. We implement both the compressive sampling process and the signal recovery process.

3.1.1 Compressive Sampling

Compressively sampling a sparse signal involves selecting the number of samples to be taken, building the sampling matrix, and then producing the measurement vector. The number of samples used to compress a signal, M, varies based on the sparsity of the signal, K. However, the exact calculation of M involves tradeoffs between size of the measurement vector after compression and the ease of signal recovery. With more samples, the signal recovery process will more easily produce an accurate reconstruction of the original signal but the level of compression will not

be as high. With fewer samples, the signal recovery process becomes more difficult, although the compressed representation of the signal will be more compact.

We build the sampling matrix Φ in three steps. First, we generate an N-by-N matrix of random values uniformly distributed over $[-1, 1]$. Next, we use the Gram-Schmidt process to orthogonalize the rows of the matrix and then normalize each row. Finally, we create Φ from the first M rows of the now orthonormal matrix.

The sampling process consists of a single matrix-vector multiplication: $y = \Phi x$. The measurement vector, y, and the value of K together represent the compressively sampled signal.

3.1.2 Direct Method Overview

We apply PSO to the problem of sparse signal reconstruction to recover a signal from compressed samples. The goal of the signal recovery process is to produce an approximation, x', that is as close as possible to the original signal, x. We use PSO to search $\mathbb{R}^N$ for an x' that has the correct sparsity under V and produces the correct measurement vector under Φ.

If $M = N$, there is no need to search for a solution because Φ is an invertible matrix. The solution x' can be generated directly as $\Phi^{-1}y$. Otherwise, the space of possible solutions must be searched to find the best x' given y and K.

Each particle position, $X_i \in \mathbb{R}^N$, represents a signal approximation in the N-dimensional search space. The search consists of D particles moving through the search space at the same time. During each iteration of the algorithm, each of the D particles is updated to move the particles towards a local best position and a global

best position. Without knowledge of the original signal, it is impossible to know how far any particle's position is from optimal. However, we estimate the optimality of a position using a fitness measure based on the available information: the sparsity of the original signal and the measurement vector. Since fitness estimates error, a lower fitness value is better. As the swarm algorithm progresses through iterations, the global best position is updated when any particle's fitness beats the fitness of the current global best. At the end of the swarm algorithm, the final value of the global best position represents the recovered signal, x'. The swarm runs for a maximum number of iterations before stopping, since it is a computationally intensive task and the rate of additional improvement decreases over time.

3.1.3 Particle Initialization

The number of particles used in the particle swarm can influence both the speed and the accuracy of the solution. If more particles are used with the same number of update iterations, more positions within the search space will be checked, and the error in x' will decrease. However, an increase in the number of particles will also require more computation since every iteration updates the position of all particles.

We assume that the signal is drawn from $[-1, 1]^N$. Each particle is given a uniformly distributed random initial velocity $V_i \in [-0.5, 0.5]^N$, and a uniformly distributed random initial position $X_i \in [-1, 1]^N$. Another uniformly random position $Z_i \in [-1, 1]^N$ is then generated for each particle.

The local bests, $P_1 \ldots P_D$, and global best, P_g, are assigned as follows:

$g \leftarrow 1$
for $i = 1$ to D
    if fitness(Z_i) < fitness (X_i)
        then $P_i \leftarrow Z_i$
        else $P_i \leftarrow X_i$; $X_i \leftarrow Z_i$
    if fitness(P_i) < fitness (P_g)
        then $g \leftarrow i$

If Z_i has a better fitness than X_i, Z_i is used as the local best for that particle. Otherwise, X_i is used as the local best, and Z_i becomes the starting position for the particle. The best local best position out of all the particles becomes the global best.

3.1.4 Particle Update

During each iteration of particle swarm, each of the particles is updated using the velocity and position formulas presented in Chapter 2. We use an inertia weight of 0.99. The constant factor for the vector in the direction of the local best, c_1, is set to 0.1. The constant factor for the vector in the direction of the global best, c_2, is set to 0.005. A larger range of values for the local best vector encourages more local innovation and discourages immediate convergence at the global best.

First, a particle's velocity is updated, and then the particle's position is updated using the new velocity. The new position is then evaluated according to its fitness, and the local best positions and global best position are updated in the following manner:

if fitness(X_i) < fitness(P_i)
    then $P_i \leftarrow X_i$
    if fitness(X_i) < fitness(P_g)
        $g \leftarrow i$

If the particle's new fitness beats the fitness recorded for the particle's saved local best, the local best is updated to be the current position. If the position is a new local best, it is compared to the global best to see if it beats the current global best solution. If so, the global best is updated as well.

3.1.5 Particle Fitness

An effective fitness function for particles is an essential part of the particle swarm algorithm. The ideal fitness function would measure the error between a particle signal approximation X_i and the original x, but since x is not known during the signal recovery process, the fitness of the particle must be determined using the information available: the measurement vector, y, the sparsity of the signal, K, and the transformations used in the compressive sampling process.

There are two parts to the fitness determination for the direct method of signal recovery. The signal approximation should exhibit the correct sparsity in the frequency domain under V and generate the same measurement vector as x when it is sampled. We calculate the first part, the fitness relating to sparsity, or f_1, by the following method:

1. Produce a frequency-domain representation VX_i.
2. Sort the frequencies by magnitude.
3. Square each of the $N - K$ smallest frequencies, then sum them.

If the original signal is exactly sparse, the frequency-domain representation should only contain K non-zero frequencies. Any non-zero value among the $N - K$ smallest frequencies is a good indicator of a recovery error and thus contributes to the

fitness score. Although it may not be the case that the K largest frequencies of VX_i are the same K non-zero frequencies of Vx, by summing the square of the smallest frequencies we can favor signals that exhibit the correct level of sparsity.

The second part of the fitness function, f_2, measures how well the candidate solution produces samples matching those generated from the original signal. If X_i does not generate the same measurement vector as x, the difference in the measurement vectors contributes to the fitness score. More precisely, the L_2 norm of $\Phi X_i - y$ is used for the second part of the fitness score.

We combine the two fitness scores to produce a total fitness score calculated by $f_1 + 0.35f_2$. We give the second part less weight to emphasize the importance of finding a signal approximation with the correct sparsity.

3.1.6 Preliminary Results

If the direct method works well for signal recovery, then no improvements are necessary. We check to see how much room for improvement is available by using the direct method on some randomly generated sample signals. We compressively sample four signals with different sparsity levels, and then use the direct method to recover them. The signals and their reconstructions are shown in Figure 3.1. Each of the signals has 64 dimensions. The X axis of each graph represents the index of each sample in the signal, and the Y axis represents the amplitude. The original signal is red, and the same signal compressively sampled and recovered with the direct method is green. For all four signals, the recovered signal is significantly different from the original signal. These results show that the direct method does not perfectly recover a

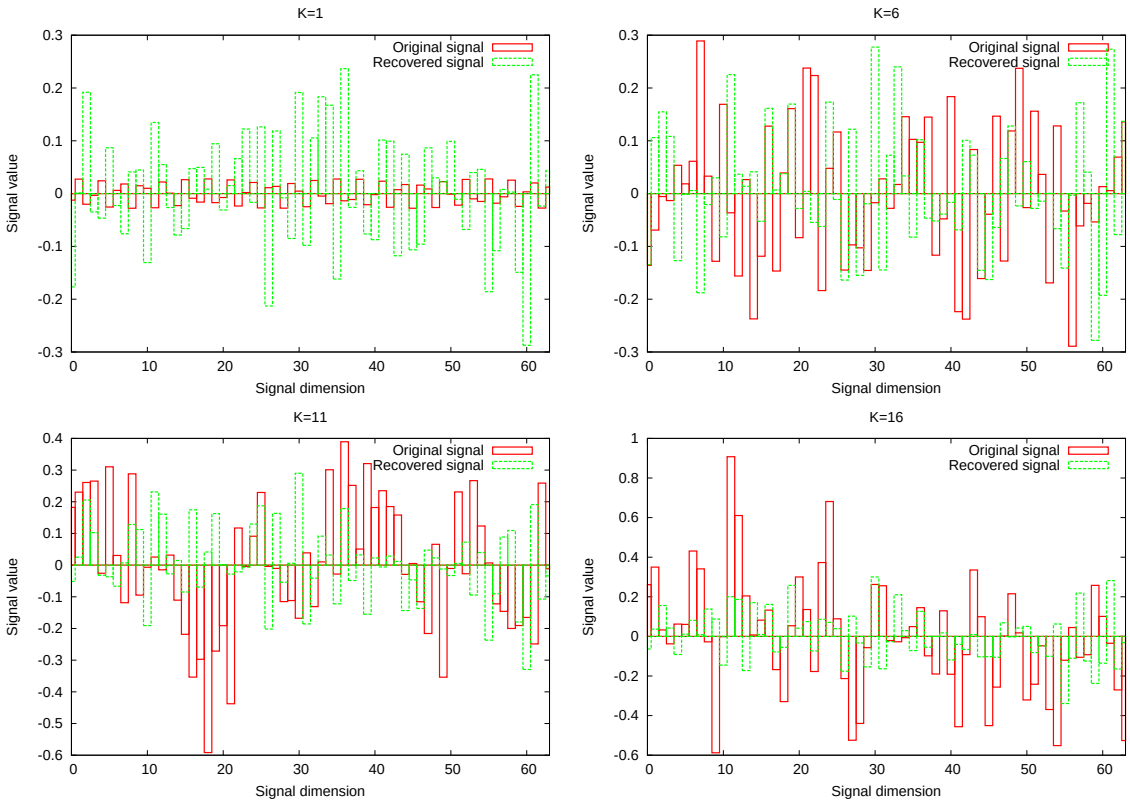

Figure 3.1: Four signals recovered with the direct particle swarm method.

compressively sampled signal, so refinements to the direct method could help improve reconstruction accuracy.

3.2 Refinements

There are several refinements to the direct method that may improve the quality of signals recovered using PSO. These refinements further customize the basic particle swarm algorithm to fit the sparse signal recovery problem.

3.2.1 Constrained Method

By narrowing the search space in which to look for possible signal reconstructions, we can increase the probability of producing an accurately recovered signal.

We can constrain the search space by removing one of the optimization criteria. Instead of searching for a signal in $\mathbb{R}^N$ that has the correct sparsity and can generate the correct measurement vector, we can limit the search to signals that generate the same measurement vector that x generates.

We first generate one of the many vectors that maps to y under Φ in the underdetermined system $\Phi x = y$. This solution is computed as $\hat{x} = \Phi^T(\Phi\Phi^T)^{-1}y$. Although $\hat{x}$ is not likely to approximate x well, it is a solution which is guaranteed to produce the same measurement vector that x produces. We then search for the correct offset for $\hat{x}$ in the null space of Φ; this allows us to maintain a link to the correct measurement vector for any possible solution. Thus, the null space of Φ becomes the search space for the particle swarm. We use N_Φ to represent a transformation from a position in the null space of Φ to an N-dimensional offset for $\hat{x}$. The null space of Φ contains $N - M$ dimensions, so as M increases, the size of the null space decreases. We call this approach the *constrained method* because we constrain the search to signal approximations for which $\Phi x = \Phi x'$.

The constrained method eliminates the need for f_2, or fitness with respect to y, in the fitness function. As a result, the entire fitness measure for the constrained method is the f_1 fitness score, or the fitness with respect to sparsity. The frequency-domain representation of the signal approximation used for calculating f_1 is $V(\hat{x} + N_\Phi X_i)$.

Particle initialization for the constrained method is modified slightly from the direct method. Like the direct method, each particle is given a uniformly distributed random initial velocity $V_i \in [-0.5, 0.5]^N$, and a uniformly distributed random initial

position $X_i \in [-1, 1]^N$. The velocity and position are then converted into a velocity and position in the null space of Φ by solving overdetermined systems:

$$X_i \leftarrow (N_\Phi^T N_\Phi)^{-1} N_\Phi (X_i - \hat{x}) \tag{3.1}$$

$$V_i \leftarrow (N_\Phi^T N_\Phi)^{-1} N_\Phi V_i \tag{3.2}$$

3.2.2 Funnel Refinement

In order to evaluate the fitness with respect to sparsity of a candidate solution X_i, we assume that the dimensions of the K largest-magnitude components of VX_i are the same as the dimensions of the K non-zero components of Vx. The location of the K non-zero components of Vx is known as the *sparsity pattern* of the signal. We can test this assumption to gain additional information about the quality of X_i as a candidate signal approximation. We use the indices of the K largest components of VX_i to create a new signal approximation x'', then measure the difference between $\Phi x''$ and Φx. This new fitness component is designated f_3.

We calculate f_3 using the following process:

1. Select the columns of $V\Phi$ that correspond to the dimensions of the K largest-magnitude components of VX_i; use the columns to form Φ_K.
2. Solve the overdetermined system $\Phi_K z = y$ for z.
3. Convert z to u'' by using the K dimensions of z to fill in the corresponding K non-zero dimensions of u''.
4. Move the new signal approximation back into the original signal domain: $x'' = V^{-1}u''$.
5. Calculate f_3 as the L_2 norm of $\Phi x'' - \Phi x$.

The f_3 fitness score is given ten times the weight of f_1. This fitness calculation is called the *funnel refinement* since it attempts to funnel particles towards their nearest K-sparse representative.

3.2.3 Redrop Refinement

One possible problem with the particle swarm approach to search is that particles can be drawn towards a local minimum. If particles have local minimum positions that are close to the global minimum, then the area of the search space that is actually searched by the particles could be small. Furthermore, whenever a particle succeeds in finding a new local or global best, the dimensionality of its search space is reduced. We attempt to reduce this problem by decreasing the inertia weight to 0.5 and reinitializing particles to new random locations when their velocity drops below a threshold. We call this modification the *redrop refinement* because it randomly redrops particles into the search space in an attempt to test areas that may not otherwise be visited.

During each iteration of particle swarm, the L_2 norm of each particle velocity, V_i, is calculated and compared to a threshold value. If it is below the threshold, the particle is given a new random position and velocity, but it keeps the local best position seen so far. This reinitialization removes particles that are moving too slowly to provide any significant updates to the global best. The threshold must be chosen carefully, since a value too low will decrease the swarm's ability to check new areas of the search space, and a value too high might reinitialize a particle that may be about

to improve upon the global best. For this application, a velocity threshold of 0.001 is used to identify particles for redrop.

3.2.4 OMP Initialization

One of the main advantages of the OMP signal recovery algorithm is its speed. One of the disadvantages of particle swarm is that the global best may not be initialized to a good location, since it is initially only the best position out of $2D$ randomly initialized particles. In order to give the particles a better initial global best, OMP can be used to calculate a solution that becomes the starting global best position for the particle swarm. OMP does not add any significant run time to the algorithm, since particle swarm already takes much more time to run than OMP. With the use of OMP, particles can start the algorithm by moving towards a global best that is likely to be much better than the best of D random locations. This improved starting global best may increase the quality of a recovered signal without increasing the number of particle swarm iterations.

3.2.5 Guided Method

We can indirectly improve the sparsity of a signal approximation using the fitness function, since signals that do not exhibit the correct sparsity will be assigned a worse fitness score than signals that do exhibit the correct sparsity. However, this approach relies on the evaluation of particle locations that have been visited; the particles move towards a local and global best which have already been assigned a fitness score. We can directly improve the sparsity of a signal approximation by

updating the velocity calculation to move particles in the direction of the nearest K-sparse signal. We call this refinement of the constrained method the *guided method* since it guides the particle swarm to search for solutions with the correct sparsity. The velocity update formula for particles now becomes:

$$V_i \leftarrow w\ V_i + c_1\ r_1()(P_i - X_i) + c_2\ r_2()(P_g - X_i) + c_3\ r_3()G_i \tag{3.3}$$

where c_3 is a positive constant, $r_3()$ is a random variable on $[0, 1]$, and G_i is a vector in the direction of the nearest K-sparse signal. For this application, c_3 is set to 0.1.

We compute G_i in two steps:

1. Create S_i from the frequency-domain representation of the signal approximation, VX_i, by negating $N - K$ of the smallest magnitudes. The K largest-magnitude dimensions are set to 0.
2. Solve the overdetermined system $VG_i = S_i$ for G_i.

We can extend this refinement to work with the constrained method as well. For the constrained method, S_i is computed as $V(\hat{x} + N_\Phi X_i)$, and G_i is solved from the overdetermined system $VN_\Phi G_i = S_i$.

By guiding the particles in the direction of their nearest K-sparse representation, the guided method promotes sparsity in candidate solutions. This allows the particle swarm to more easily find and test K-sparse signal approximations.

3.3 Conclusion

The direct method is a straightforward application of PSO for sparse signal recovery. The constrained method improves upon the direct method by constraining the search space, and the guided method adds a new component to the velocity update formula to move each particle in the direction of the nearest K-sparse signal. The

funnel and redrop refinements can be used with any of these methods to enhance the signal recovery process. OMP can also be used to initialize the starting global best position for the particle swarm.

CHAPTER FOUR

Image Compression and Recovery

To apply compressive sampling techniques to images, we convert raw image data into a format that is compatible with the compressive sampling process. Like JPEG, we divide an image into square blocks and compress each block separately. Each compressed block is quantized and encoded using a fixed bit-length encoding for each sample. We examine several refinements to the compression and recovery processes, including thresholding the input to ensure exact sparsity, information sharing between adjacent blocks, and various methods of handling the DC coefficient.

4.1 Image Compression

To compressively sample an image, we first divide it into eight-by-eight pixel blocks. Each block is then sampled separately. Dividing an image into blocks captures local sparsity in various regions of an image. Blocks can also be independently sampled and reconstructed, allowing for greater parallelization.

After the image is divided into blocks, the pixel data are converted from 8-bit grayscale values to floating-point numbers centered at zero in the range [-1,1]. Given a pixel value p, we convert it to a floating point value using the computation $2p/255 - 1$. This calibrates the image intensity information with our compressive sampling and recovery methods.

Since image data is two-dimensional, we use a two-dimensional DCT (BMS97) linearized into a single 64 x 64 matrix. This allows us to treat a block as a linear signal using row-major order within a block. We use the DCT to calculate the sparsity of each block; the compressive sampling process will use more samples for blocks that are less sparse.

4.1.1 Quantization and Encoding

Even if the signal x is integer-valued, the measurement vector y will be a vector of real numbers. To encode the measurement vector, we quantize it using translation and scaling in order to most efficiently use the full range of bits available. Let b be the number of bits available to quantize each element of y. Let y_{min} be the minimum element in y and y_{max} be the maximum element of y. The range of values in y is specified by $y_{max} - y_{min}$.

We use a *translation table* to specify offsets used to move the entire measurement vector into a range of positive numbers. Let T be a translation table implemented as a vector of c floating point values. We use $c = 16$ for our translation table size. The actual translation values in the table are presented in Table 4.1. We select a translation value, t, as follows:

$t \leftarrow T_1$
$d \leftarrow |t - y_{min}|$
for $i = 2$ to c
    if $(|T_i - y_{min}| < d)$
        then $d \leftarrow |T_i - y_{min}|$; $t \leftarrow T_i$

Table 4.1: The table of translation values for quantizing a compressed image block.

Translation Value
-0.0114
-0.1298
-0.2482
-0.3666
-0.485
-0.6034
-0.7218
-0.8402
-0.9586
-1.077
-1.1954
-1.3138
-1.4322
-1.5506
-1.669
-1.7874

The translation value is selected from the table as the value that minimizes $|t - y_{min}|$. We can encode the translation value as an index in the translation table using $\log_2 c$ bits.

We want to be able to use the full range of integer values provided by b bits to encode the elements of y. To achieve this we use a *scaling table* from which we can select a scaling factor that will modify the range of values of y to approximate the range provided by b bits. Let S be a scaling table implemented as a vector of e floating-point values. The scaling table provides scaling values for an encoding using h bits per sample. We use a scaling table with 16 values for an encoding using 4 bits per sample. The scaling table values we use are presented in Table 4.2. If b is not the same as h, we modify the scaling factor so that the scaling factor expands or

Table 4.2: The table of scaling values for quantizing a compressed image block.

Scaling Value
0.0010
0.0100
0.0250
0.0500
0.0625
0.0750
0.0875
0.1000
0.1125
0.1250
0.1375
0.1500
0.1625
0.1750
0.1875
0.2000

contracts to fit an encoding of b bits per sample. We calculate s, the scaling value, as follows:

$j \leftarrow (y_{max} - y_{min})$ / 2^h
$s \leftarrow S_1$
for $i = 2$ to e
    if $(|j - S_i| < |s - S_i|)$
        then $s \leftarrow S_i$
$s \leftarrow s(2^h$ / $2^b)$

We first compute an ideal scaling factor, j, that fits the range of values in y to the h-bit range exactly. We then select the scaling factor from the scaling table that is the closest to the ideal scaling factor. The last step modifies the selected scaling factor so that it scales y to fit a b-bit range.

After the translation and scaling values are calculated, y is quantized:

for $i = 1$ to M
    $y_i \leftarrow \lfloor (y_i - t) \ / \ s \rfloor$
    if $(y_i \geq 2^b)$
        $y_i \leftarrow 2^b - 1$
    if $(y_i < 0)$
        $y_i \leftarrow 0$

After quantization, each element of y is represented by an integer in the range $[0, 2^b - 1]$. Each quantization interval covers the range of values from one integer up to its successor. Under quantization, the integer at the lower end of the range is used to represent anything in the interval. To generate the floating-point values necessary for signal recovery, the translation and scaling steps are reversed:

for $i = 1$ to M
    $y_i \leftarrow y_i + 0.5$
    $y_i \leftarrow y_i s$
    $y_i \leftarrow y_i + t$

During signal recovery, all values in an interval are approximated with the floating point value at the center of the interval.

It is important to note that the quantization process uses a uniform quantization because Φ is an orthonormal transformation. Each sample in y is a measurement of the entire signal, so it is impossible to use a different number of bits to encode different frequency components of an image block when quantizing the measurement vector. Since JPEG quantizes an image block in the frequency domain, it uses a non-uniform quantization to use more bits to encode the low-frequency components.

By using scaling and translation tables, the bit requirement for scaling and translation is only a table index for each operation. The final encoding of an image

block uses $(\log_2 N) + 1$ bits to encode K, $\log_2 c$ bits for the translation table index, $\log_2 e$ bits for the scaling table index, and Mb bits for the quantized elements of y.

4.2 Refinements

Although we can treat an image block just like any other linear signal to compressively sample it, we can use properties of natural signals, and image signals specifically, to improve the signal recovery process. There are several image-specific refinements to consider. Thresholding an image signal ensures exact sparsity rather than approximate sparsity. Simultaneously recovering all blocks of an image allows parallelization of image block signal recovery as well as information sharing among adjacent blocks. The DC coefficient of Vx is a part of the signal that does not behave like the other coefficients, so any effort made to improve the recovery of the DC coefficient will likely help improve the recovery of the image block as a whole.

4.2.1 Approximate Sparsity and Thresholding

If an input signal is synthetically generated for testing it can exhibit exact sparsity, but if the input is a natural signal, it may not exhibit the exact sparsity required for compressive sampling. Thus, there are two options for handling approximately sparse input data: treating the approximately sparse signal as sparse or forcing a certain level of sparsity. The first method is the more straightforward approach because the input signal is not modified in any way. A threshold value is only used to calculate the sparsity of the signal. The second method involves discarding some small coefficients in the frequency-domain representation of a block.

To calculate the sparsity of an approximately sparse signal, all components of u are checked to see if they are above a threshold. We use a threshold value of 0.1. If a component's absolute value is below the threshold, the component is considered to be an insignificant contributor to the signal. Components with an absolute value greater than or equal to the threshold are considered to be significant and are counted towards the signal's sparsity.

The *thresholding method* forces exact sparsity in the image block. All components of u below the threshold are set to zero to create $\hat{u}$, the thresholded frequency-domain representation of x. Given a threshold level l, a signal is thresholded as follows:

$u \leftarrow Vx$
for $i = 1$ to N
    if $(|u_i| < l)$
        then $\hat{u}_i \leftarrow 0$
        else $\hat{u}_i \leftarrow u_i$
$x \leftarrow V^{-1}\hat{u}$

When a signal is reconstructed, the signal recovery process has knowledge about the sparsity of the original signal. However, it is difficult to reconstruct an approximately sparse signal exactly because, even if all K significant frequency components are discovered, the signal recovery process will try to produce a signal with $N - K$ zero-magnitude frequency components. Thresholding an approximately sparse signal discards some information in the signal with the expectation that an exactly sparse signal can be recovered with lower error than a signal which is only approximately sparse.

4.2.2 Simultaneous Particle Swarms and Infusion

Each block in the image requires a particle swarm search to find a solution to reconstruct that image block. Since the image blocks are independent of one another, it is possible to compute the image block reconstructions in parallel. This is an advantage of small image block sizes because, with more total blocks in the image, more processors can be used at the same time. Even without additional processors available, the image blocks can still be recovered simultaneously if the states of all blocks' particle swarms are maintained independently. The simultaneous computation requires more memory, but it also allows the blocks to share information with each other as their respective swarms work to find solutions.

The process of information sharing is called *infusion* because a block sends its best solution to neighboring blocks to try to help them improve upon their locally-developed best solution. Image data can have large regions of high similarity, so it is possible that adjacent blocks have similar patterns of pixels, and the particle swarms for those blocks would be trying to recover similar information from the compressively sampled data.

When the particle swarm for image block i updates its global best value, the solution that the global best represents, x_i', could also provide a better solution than the current global best of a surrounding block. A block could have up to eight surrounding blocks, for blocks in the middle of the image, or as few as three surrounding blocks, for blocks in the corner of an image. The solution x_i' is sent to all surrounding blocks to allow them to check if that solution is better than their own best known solution, x_j'. If so, the receiving swarm uses x_i' to create a new global best and

continues the algorithm normally. Ideally, the process of infusion could increase the accuracy of a swarm solution since the global best has more chances to be updated, both internally and externally, especially early in the search.

4.2.3 Treatment of the DC Coefficient

The DC coefficient is the constant term in the signal produced by the application of the DCT. Since the DC coefficient depends on the average intensity, rather than the frequency components of a block, we can try to improve recovery of the DC coefficient specifically. We call the normal treatment of the DC coefficient the *standard DC method*. If the average intensity of a block is very close to neutral gray, this coefficient is considered one of the N-K zeros in the block's frequency-domain representation. Otherwise, it is counted as one of the K non-zero values. The standard DC method treats the DC coefficient just like any other frequency component. We use two separate techniques that attempt to improve the reconstruction of the DC coefficient.

The *inclusive DC method* interprets the DC coefficient as one of the K non-zero elements in the block, regardless of its actual magnitude. The calculation of K is modified slightly to be the non-zero values in the last $N-1$ elements of u, plus one for the DC coefficient. Compared to the standard method, this technique may result in increasing the value of M for some blocks. If the DC coefficient would not normally be considered one of the K non-zero elements, more samples will be required to compress the block. However, this technique tries to improve the signal recovery of the DC coefficient specifically; this improvement may be worth the slight increase

in compressed signal size. Improvement in recovery of the DC coefficient can improve the intensity level accuracy of the entire block.

The *isolated DC method* is a different approach from the inclusive method. The DC coefficient is encoded separately from the measurement vector as d, allowing us to remove the DC coefficient's contribution to x. We modify x as follows:

$$u \leftarrow Vx$$
$$d \leftarrow u_1$$
$$u_1 \leftarrow 0$$
$$x \leftarrow V^{-1}u$$

The sparsity of a block, K, is computed as the non-zero elements of u, but the the DC coefficient will never contribute to K since it is removed.

Removing the DC coefficient can lower the magnitude of the values in x, which can help the quantization process use the available bits more efficiently. Upon reconstruction, the DC coefficient will always be exactly correct. This method can provide a large increase in reconstruction accuracy if the DC components of image blocks are large contributors to overall error.

4.3 Conclusion

We divide an image into blocks, compressively sample each block, then quantize the blocks to encode the image in a compressed format. To recover the image, we use several image-specific refinements. These refinements include thresholding the image to ensure exact sparsity, infusing particle swarm information from one block into another, and giving special treatment to the DC coefficient in an image block.

CHAPTER FIVE

Evaluation

The evaluation of PSO as a signal recovery technique is considered primarily in terms of reconstruction accuracy. Since existing greedy algorithms are much less computationally intensive, if PSO can show an improvement over existing methods, the improvement will be in signal recovery error or compressed encoding size. The selected error metric for this evaluation is Root Mean Square (RMS) error, which can be calculated for the recovered signal x' as follows:

$$E = \sqrt{\frac{(|x - x'|_2)^2}{N}} \tag{5.1}$$

There are two types of signals to evaluate: synthetic signals which are randomly generated and meet exact sparsity requirements, and natural two-dimensional signals in the form of images, which are only approximately sparse but provide a reasonable real-world application for the PSO-based signal recovery technique.

For all of the experiments, we used a server with two quad-core Intel E5430 processors and two gigabytes of RAM running Red Hat Enterprise Linux 5.3 with the 2.6.18 kernel. We did not have exclusive access to this server, although other use was limited. The PSO-based signal recovery methods and OMP are implemented in C++, and compiled with gcc version 4.1.2.

5.1 Synthetic Signals

Testing with synthetic signals has several advantages. The first advantage is that the synthetic signals are generated with exact sparsity, so the signal recovery process does not have to handle approximate sparsity. All of the error reported for the various PSO-based methods presented here can be attributed to the recovery method, and not to an approximately sparse input signal. Another advantage is that they provide a point of comparison to natural signals such as image data. Although the synthetic signals do not provide the practical test that image signals do, they can help to identify the most effective combination of techniques for signal recovery.

To evaluate our techniques against synthetic signals, we generate 100 random vectors in $\mathbb{R}^{64}$ at each sparsity level from 1 to 16. We call these signals the *synthetic signal test suite*. To produce a random signal, we first generate a random u with sparsity K. We pick K dimensions at random, and assign those dimensions a uniform random number in $[-1, 1]$. All other dimensions are set to zero. We then use u to generate the signal using $x = V^{-1}u$.

For these experiments, we use 10000 iterations of particle swarm unless otherwise specified. We calculate M as $3K + 5$, and we use 20 particles to recover each signal. The reported RMS error measurements for the synthetic signals are averaged over all 100 signals for a given sparsity level.

5.1.1 Direct and Constrained Methods

The first experiment involving synthetic signals compares the direct method to the constrained method of signal recovery. We compressively sample all 1600 signals

in the test suite, then recover the signals using the direct method and the constrained method. Figure 5.1 shows the difference in error between the direct solution which uses the 64-dimensional search space of the original signal x and the constrained solution which uses the null space of Φ with $64-M$ dimensions. The X axis represents the different sparsity levels in the synthetic signal test suite. The Y axis represents the average RMS error over 100 signals at each sparsity level. Lower error values represent more accurate signal reconstructions. One standard deviation error bars are also shown on this graph. We omit error bars from all subsequent graphs since the level of variation in error appears to be consistent.

For all sparsity levels, the constrained method recovers the signals with a lower average RMS error. Since the search space of the constrained solution decreases in dimensionality as K increases, it becomes easier to find solutions for compressively sampled signals at higher sparsity levels.

Table 5.1 presents the results of recovering 100 signals from the synthetic signal test suite for which $K = 1$. The table includes average RMS error for various signal recovery methods and the running time for recovering all 100 signals sequentially. For these 100 signals, the constrained method is able to improve RMS error from an average of 0.121 with the direct method to an average of 0.101 with the constrained method. The runtime also decreases slightly from 213.3 seconds with the direct method to 186.6 seconds using the constrained method.

Table 5.2 presents results for various signal recovery methods on the signals from the synthetic signal test suite for which $K = 16$. This table shows results for average RMS error over these 100 signals as well as the running time for recovering all

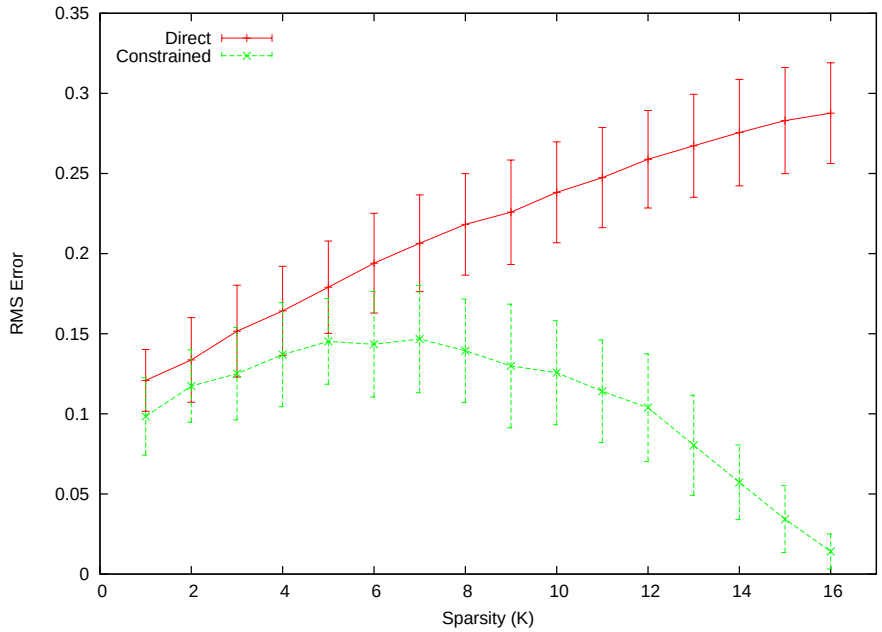

Figure 5.1: Mean RMS error for both the direct and constrained particle swarm reconstructions of the synthetic signal test suite. The direct solution searches in the signal domain, whereas the constrained solution searches the smaller null space of Φ. The X axis indicates the sparsity, K, and the Y axis indicates average root mean squared error across 100 signals at each sparsity level.

100 signals. At $K = 16$, the direct solution searches a 64-dimensional space, whereas the constrained solution searches only 11 dimensions. As a result, the computation time drops from 298.4 seconds to recover all 100 signals at $K = 16$ to 109.9 seconds to recover those same signals with the constrained method. Mean RMS error decreases from 0.288 using the direct method to 0.014 using the constrained method.

5.1.2 Constrained Method with Redrop and Funnel Refinements

Some particle swarm refinements can be used at the same time; it is possible to use both the redrop and funnel refinements together. We evaluate the constrained method of signal recovery with only the redrop refinement, only the funnel refinement, and both refinements together. Figure 5.2 shows average RMS error of each sparsity

Table 5.1: A comparison of signal recovery methods on the 100 signals for which $K = 1$. The RMS error value is an average of RMS error over these 100 signals. The runtime value is a total time in seconds used to recover all 100 signals.

Method	Other Refinements	Mean RMS Error	Runtime (s)
direct	none	0.121	213.3
constrained	none	0.101	186.6
constrained	redrop	0.050	199.0
constrained	funnel	0.104	356.1
constrained	redrop, funnel	0.041	371.1
guided	none	0.006	287.5
guided	redrop	0.001	296.0
guided	funnel	3.16×10^{-16}	458.4
guided	redrop, funnel	1.18×10^{-4}	470.8
OMP	none	4.08×10^{-17}	0.7
constrained	OMP	4.10×10^{-17}	190.2
guided	OMP	4.09×10^{-17}	287.6

Table 5.2: A comparison of signal recovery methods on the 100 signals for which $K = 16$. The RMS error value is an average of RMS error over these 100 signals. The runtime value is a total time in seconds used to recover all 100 signals.

Method	Other Refinements	Mean RMS Error	Runtime (s)
direct	none	0.288	298.4
constrained	none	0.014	109.9
constrained	redrop	0.008	118.4
constrained	funnel	0.026	560.1
constrained	redrop, funnel	0.010	921.4
guided	none	7.19×10^{-16}	140.2
guided	redrop	7.98×10^{-5}	146.4
guided	funnel	7.00×10^{-16}	576.0
guided	redrop, funnel	8.03×10^{-5}	1071.5
OMP	none	3.39×10^{-16}	0.8
constrained	OMP	3.47×10^{-16}	114.2
guided	OMP	3.39×10^{-16}	140.3

level in the synthetic signal test suite using the redrop and funnel refinements added to the constrained method for signal recovery.

The lowest error is achieved using both the redrop and funnel refinements together for $K \leq 11$ and using the only the redrop refinement for $K \geq 12$. Adding the funnel method to the fitness function does not produce a significant change in error when compared to the basic constrained solution. However, redropping particles does decrease RMS error for all 16 sparsity levels. The redrop refinement increases the chance of arriving at a good solution via particle initialization since the particles are reinitialized many times over the course of the particle swarm iterations. Interestingly, for $K < 10$, the funnel fitness method when combined with particle redrop decreases recovery error significantly more than either of the two methods do separately.

5.1.3 Guided Method with Redrop and Funnel Refinements

To promote proper sparsity in the solution, the guided method updates each particle's velocity to move the particle in the direction of the closest sparse vector. We use the guided method as an extension of the constrained method. We evaluate the guided method by itself and with the redrop and funnel refinements. Figure 5.3 shows average RMS error of the guided method with redrop and funnel refinements for 100 signals at each sparsity level in the synthetic signal test suite.

The guided method by itself produces recovered signals which have a much lower error on average than any constrained method-based system. Since the funnel and redrop refinements helped lower error for the basic constrained solution, they are

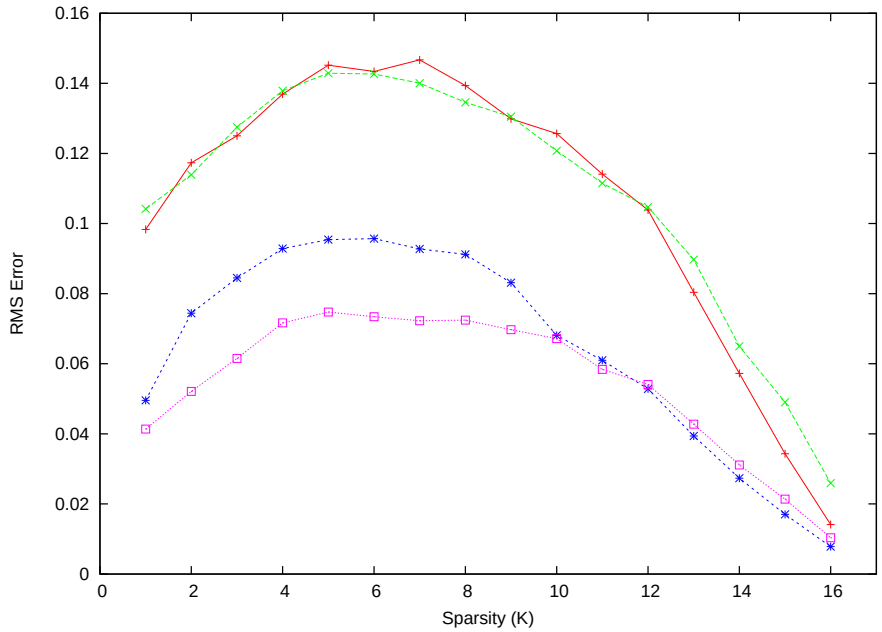

Figure 5.2: Mean RMS error for the basic constrained solution as well as funnel and redrop refinements for particle swarm reconstructions of the synthetic signal test suite.

tested with the guided method to check for similar improvement. Among the tested configurations, the lowest error is achieved using the guided method with the funnel refinement for all K except $K = 2$ and $K = 6$. For $K = 2$, the lowest error is achieved with both the redrop and funnel refinements. For $K = 6$, the lowest error is achieved using the guided method with no other refinements. The redrop refinement does not help much when applied to the guided method; it produces higher error than the basic guided solution for $K > 3$.

The funnel refinement adds considerable computational overhead because the fitness function must solve an additional overdetermined system. As a result, the guided solution by itself is the more efficient solution. For $K = 1$, the guided method is able to recover 100 signals in 287.5 seconds, as shown in Table 5.1. However, with

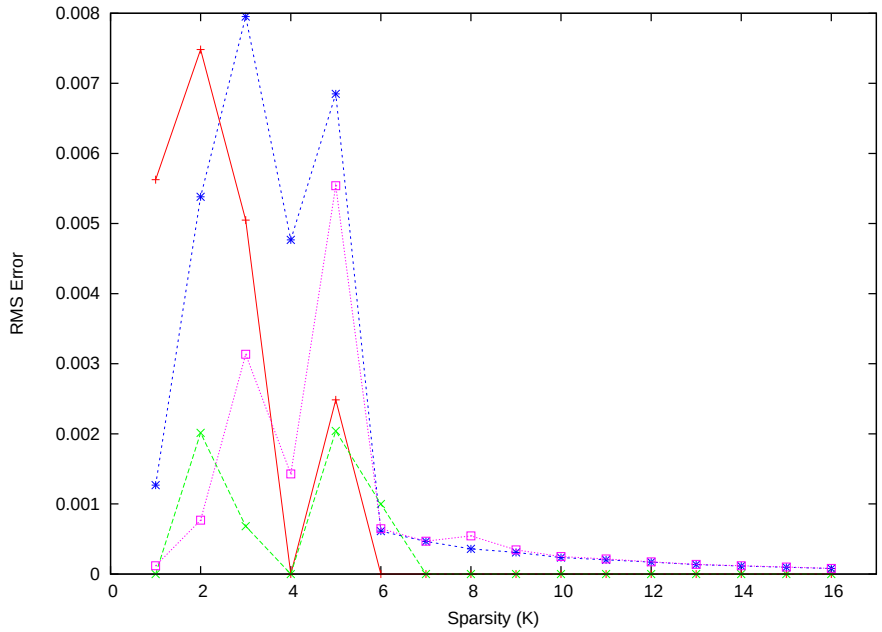

Figure 5.3: Mean RMS error for the guided solution and funnel and redrop refinements for particle swarm reconstructions of the synthetic signal test suite.

the addition of the funnel refinement, the recovery time for the same 100 signals increases to 458.4 seconds. Recovery time increases even more, from 140.2 seconds to 576.0 seconds, with the funnel refinement for the 100 signals for which $K = 16$.

5.1.4 OMP Initialization

We can use OMP to generate a starting global best for the particle swarm. We use OMP to initialize the constrained method and the guided method, and compare the results to OMP by itself and the guided method by itself. Figure 5.4 shows average RMS error for 100 signals recovered using these methods at each sparsity level in the synthetic signal test suite.

By itself, the OMP algorithm recovers input signals with lower error than the basic constrained solution with no refinements. Therefore, when OMP is used to

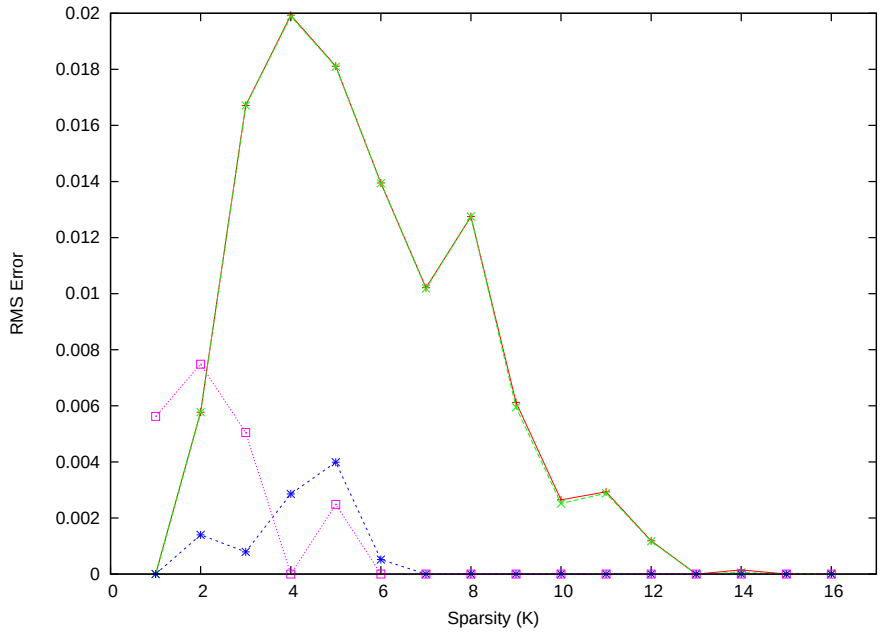

Figure 5.4: Mean RMS error for OMP initialization added to various particle swarm reconstructions of the synthetic signal test suite. The OMP solution with no particle swarm, as well as the basic constrained and guided solutions with no OMP initialization, are also presented for comparison.

initialize the particles for the constrained method, the particle swarm is not expected to do any better than OMP, and the resulting error is almost identical to OMP. When OMP initialization is used for the guided method, however, an error improvement is seen for $K < 4$ compared to the guided method only. Based on these results, OMP initialization should be used for the guided solution, but only for $K < 4$.

Compared to the particle swarm solutions, OMP is fast so it does not add significant processing time to the signal reconstruction. As shown in Table 5.1 and Table 5.2, OMP can recover 100 signals with sparsity $K = 1$ in 0.7 seconds, and 100 signals with $K = 16$ in 0.8 seconds.

5.1.5 Signal Recovery Over Time

Figures 5.5, 5.6, and 5.7 demonstrate the change in recovery error as the particle swarms progress through 20000 iterations of the particle swarm algorithm. We compare three methods: the constrained method, the constrained method with the redrop and funnel refinements, and the guided method. We measure average RMS error over 100 signals at a specific sparsity level recovered using varying numbers of iterations. Every 100 iterations until 20000 iterations, we generate a signal approximation for each of the 100 signals and calculate the average RMS error for the recovered signals. The X axis of these graphs represents the number of iterations of particle swarm that have been completed, and the Y axis represents the average RMS error of signals recovered using a specific number of iterations.

Figure 5.5 shows average RMS error of 100 signals from the synthetic signal test suite for which $K = 1$. Average RMS error values are shown for signals recovered after 100 to 20000 iterations in 100 iteration increments. The basic constrained solution reaches its minimum error after around 3000 iterations. The redrop and funnel refinements contribute to a decrease in error throughout the entire 20000 iterations. This result suggests that further decrease in error is possible with this method if more than 20000 iterations are used. In comparison, the guided solution produces much lower error throughout, but it does not show much error improvement past the first 2000 iterations.

Figure 5.6 shows average RMS error of 100 signals for which $K = 8$ from the synthetic signal test suite recovered using 100 to 20000 iterations. The basic constrained solution reaches its minimum error after 4000 iterations. After 1300

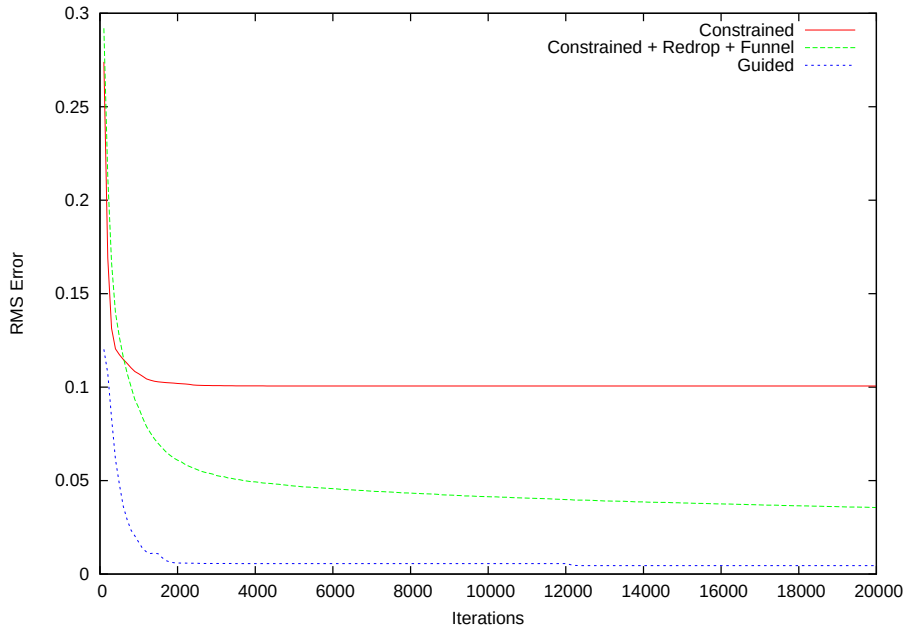

Figure 5.5: Mean RMS error of 100 compressed signals with sparsity $K = 1$. The three methods compared are the basic constrained solution, the constrained solution with redrop and funnel refinements, and the guided solution.

iterations, the guided method is able to decrease error to 6.95×10^{-5}. The redrop and funnel refinements again demonstrate the ability to decrease error through all 20000 iterations of the particle swarm.

Figure 5.7 shows average RMS error of 100 signals for which $K = 16$ from the synthetic signal test suite recovered using 100 to 20000 iterations. Interestingly, the basic constrained system has a lower average error than the redrop and funnel refinements until after 6000 iterations. Since these tests use $M = 3K + 5$, the search space for $K = 16$ contains only 11 dimensions. The main benefit of particle redrop is that over the course of the particle swarm, the particle will test many different random initial locations. The more iterations that are used, the more random locations the particle redrop method can test, and so this method favors larger numbers of itera-

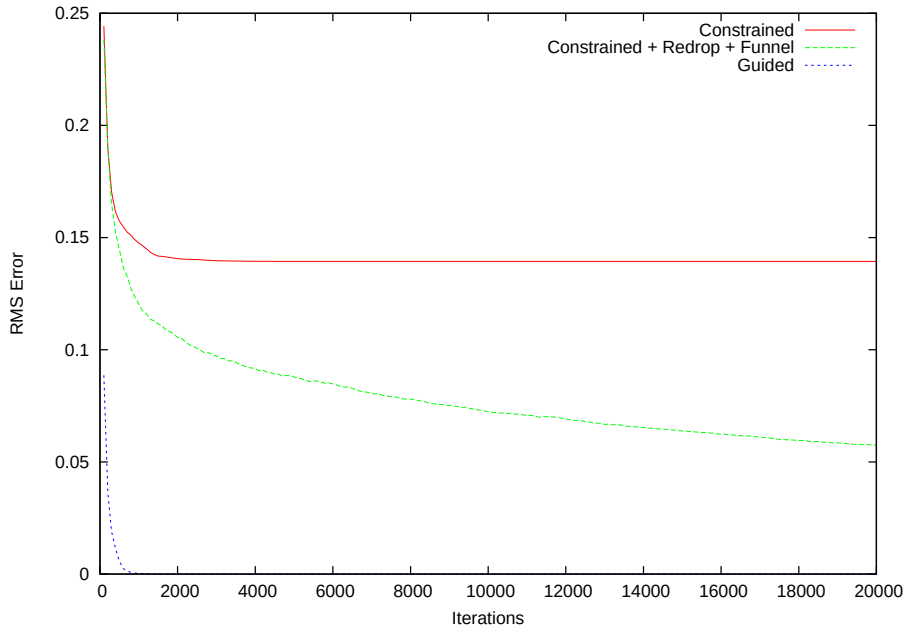

Figure 5.6: Mean RMS error of 100 compressed signals with sparsity $K = 8$. The three methods compared are the basic constrained solution, the constrained solution with redrop and funnel refinements, and the guided solution.

tions. Even after the other methods have focused on a promising region of the search space, the redrop behavior keeps looking for better areas to search. As with $K = 8$ and $K = 1$, the constrained method with redrop and funnel refinements still shows improvement in error throughout all 20000 iterations.

5.2 Image Experiments

We use PSO-based sparse signal recovery methods to reconstruct compressively sampled images so that we can evaluate our techniques on natural signals. These experiments allow us to test image-specific refinements for signal recovery as well as to compare our approach to a standard image compression algorithm such as JPEG.

We use six images from the USC-SIPI database (Web), our *image test suite*, to test the guided method, the best method from the synthetic testing. Figure 5.8 shows

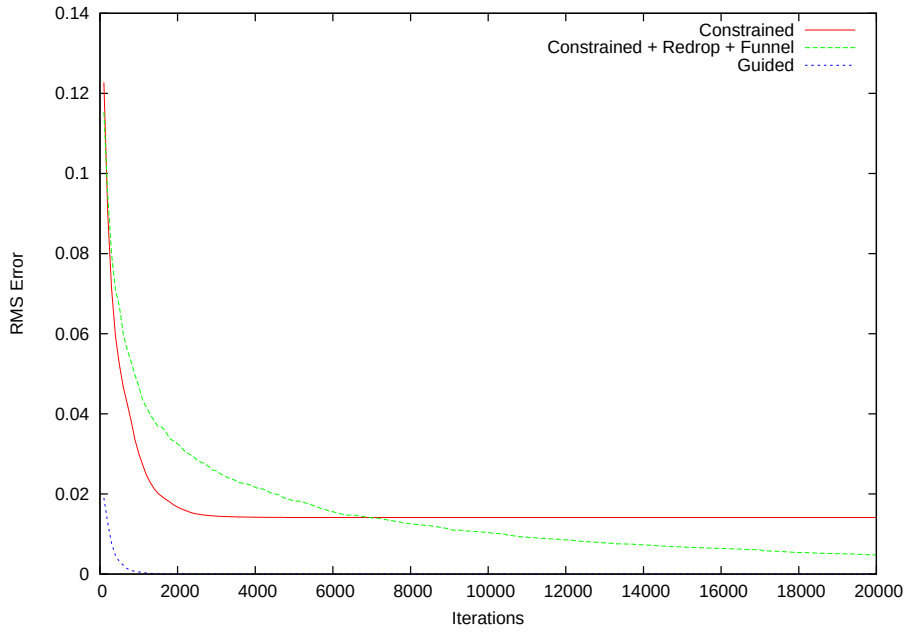

Figure 5.7: Mean RMS error of 100 compressed signals with a sparsity $K = 16$. The three methods compared are the basic constrained solution, the constrained solution with redrop and funnel refinements, and the guided solution.

the images in the test suite. Figure 5.9 shows sparsity histograms for the images in the test suite. These histograms demonstrate the distribution of sparsity levels among blocks in each image. The X axis of a histogram represents the sparsity level, and the Y axis represents the number of blocks in the image at a specific sparsity level.

We use OMP to initialize blocks with $K < 4$. We convert some of the images to grayscale so that all images used 8 bits per pixel to encode grayscale values. We calculate image RMS error from the grayscale pixel values in the image rather than the floating-point representation that was used for the synthetic data testing. We combine all of the recovered image blocks to create a recovered image, then calculate image RMS error between all pixel values of the original and recovered images. Each of the image blocks are quantized at 5 bits per sample, and the particle swarms for

Figure 5.8: The images from the USC-SIPI database that make up our image test suite. Clockwise from the top left, the images are 4.2.04, 4.2.07, 5.2.08, 5.2.10, boat.512, and elaine.512.

each image block recovery use 20 particles and are run for 10000 iterations. Like the synthetic tests, we use $M = 3K + 5$.

5.2.1 DC Coefficient Experiments

We test the various methods of encoding the DC coefficient on all six images in the image test suite. Table 5.3 compares the three DC coefficient methods in terms of encoded size, or bit count, RMS error of the recovered image, and recovery runtime for all six test images.

The inclusive DC method recovers four out the six images with lower error than the standard DC method. On average, the inclusive method increases the size

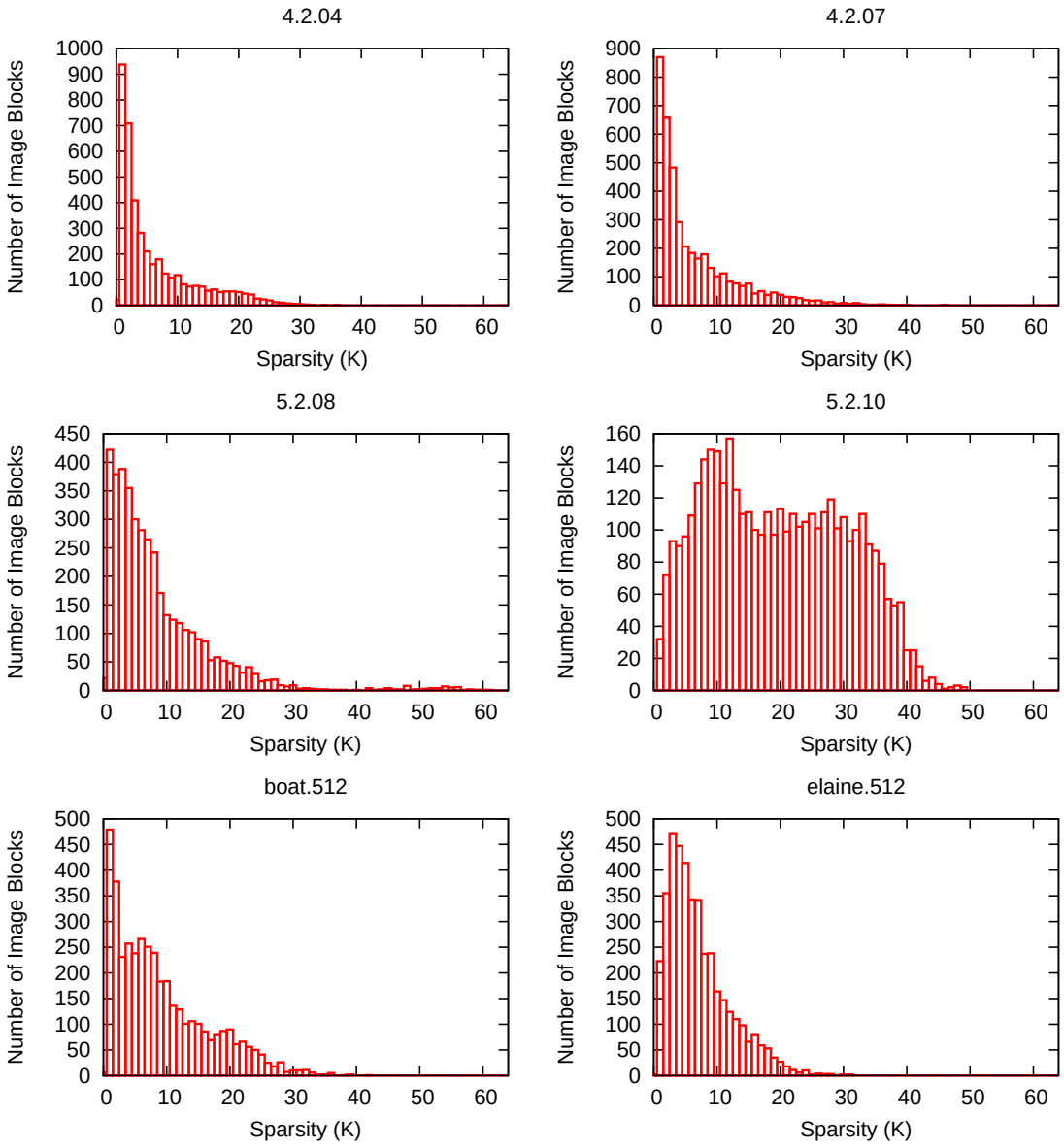

Figure 5.9: The distribution of sparsity levels among blocks for the images in the image test suite. For each histogram, the X axis represents the sparsity level, and the Y axis represents the number of blocks at each sparsity level.

of the encoded image by 0.2% and also increases the error by 0.07% over the standard method. The increase in average error is largely due to the recovery of the 5.2.08 image, which produced more error with the inclusive method than the standard method. When compared to the other methods, the error for the isolated DC method is lower for all images because of the enhanced accuracy of the DC coefficient in the recovered image blocks.

It is possible that the isolated method may require more bits than the standard method to encode some images. Images that have many blocks with near-zero DC coefficients may require more bits to encode all of the DC coefficients separately than they may save by decreasing the sparsity level for only a few blocks. Given an encoding scheme of 5 bits per sample, each large-magnitude DC coefficient saves 7 bits (three samples saved minus 8 bits for encoding the DC coefficient). Each small-magnitude DC coefficient adds 8 more bits to the standard block encoding, since it does not contribute to K and therefore M for that block. In this scenario an image with at least 47% small-magnitude DC coefficient blocks will be more efficient to encode with the standard method. After increasing the encoded size to 8 bits per sample, an image would require more than two-thirds of the blocks to have small-magnitude DC coefficients. Although this situation may happen occasionally, the isolated method will still provide a more compact encoding for most images. For all six images in the test suite, the isolated DC encoding uses fewer bits than either of the other DC coefficient methods.

Table 5.3: A comparison of DC coefficient encoding methods on the image test suite.

Image	Method	Encoded size (bits)	RMS Error	Runtime (s)
4.2.04	standard	529263	5.94	3408.5
4.2.04	inclusive	530443	5.83	3439.3
4.2.04	isolated	473782	5.44	3230.8
4.2.07	standard	530923	6.99	3301.8
4.2.07	inclusive	531463	6.95	3279.5
4.2.07	isolated	477365	6.20	4074.6
5.2.08	standard	635733	6.09	3916.6
5.2.08	inclusive	638098	6.69	4228.1
5.2.08	isolated	600239	5.85	4482.5
5.2.10	standard	1079338	5.72	2570.9
5.2.10	inclusive	1080133	5.76	2530.2
5.2.10	isolated	1079248	5.27	3143.4
boat.512	standard	692528	6.57	3173.0
boat.512	inclusive	693663	6.53	3199.3
boat.512	isolated	655628	6.19	3464.3
elaine.512	standard	592568	7.81	3326.3
elaine.512	inclusive	594488	7.78	3341.7
elaine.512	isolated	559738	7.49	3827.8

5.2.2 *Infusion and Thresholding*

To test the effects of thresholding input and infusing updates of global best particles, we compressively sample all six test images and then recover them with each of the combinations of input thresholding and infusion. To threshold the image blocks, we use a threshold value of 0.1. This is the same value that we use to identify K. Each image is encoded with the isolated DC encoding method. Table 5.4 shows the result of using infusion and thresholding on the six images in the image test suite. Each image is recovered using neither infusion nor thresholding, only thresholding, only infusion, and both infusion and thresholding. For each experiment, we record the RMS error between the original image and the recovered image, and the image recovery runtime in seconds.

The effect of thresholding the image blocks to ensure exact sparsity is largely beneficial, with five out of the six images showing significant decrease in error. One image, 5.2.10, shows an increase in error when the thresholding was applied. As shown in the sparsity histogram for 5.2.10 in Figure 5.9, this image contains many blocks for which $K > 20$. Since we calculate M as $3K+5$, any block with $K > 20$ will yield 64 samples. Thus, the signal recovery process for these blocks is a simple matrix inverse operation, since Φ is invertible for these cases. Thresholding the 5.2.10 image unnecessarily discards information for these large K blocks, since they do not require PSO-based signal recovery. In general, even though some information is discarded by the thresholding process, it produces a compressed signal which is usually recovered with lower error since the input signal exhibits exact rather than approximate sparsity.

The effect of infusion, or sharing global best particles between adjacent blocks, is much less pronounced. For most images, infusion helps slightly or results in no change to the level of error reported. For the 5.2.10 image encoded at 5 bits per sample, infusion actually increases the RMS error of the thresholded version from 5.49 to 5.50.

There are no significant runtime differences with the use of either input thresholding or infusion. This result is not surprising for input thresholding, since the signal recovery process does not change when thresholding is applied. However, the infusion process does not seem to affect the runtime much either. All of the images except the 4.2.07 image show a slight increase in runtime with the use of infusion. This is to be expected because of the additional work required to share a global best solution between two image blocks, especially if the blocks have different sparsity levels. Minor variations in runtime are also expected because of the lengthy nature of image reconstruction and the difficulty involved in controlling the testing environment for that length of time. Adding infusion also decreases memory locality since running particle swarms simultaneously requires switching from block to block to compute one iteration's worth of particle updates.

5.2.3 Comparison with JPEG

The JPEG lossy compression format for images provides a benchmark compression level that can be used to evaluate the compressive sampling method as it is applied to image compression. For this comparison, the lossless versions of all six test images were encoded as JPEGs with 80 quality using the GNU Image Manipulation

Table 5.4: RMS error and runtime performance of infusion and input thresholding on six images.

Image	Size (bits)	Infusion	Thresholding	RMS Error	Runtime (s)
4.2.04	473782	no	no	5.44	2595.5
4.2.04	473782	no	yes	5.44	2811.9
4.2.04	473782	yes	no	4.48	2765.9
4.2.04	473782	yes	yes	4.46	2801.4
4.2.07	477365	no	no	6.20	3506.7
4.2.07	477365	no	yes	6.20	3422.1
4.2.07	477365	yes	no	4.93	3488.0
4.2.07	477365	yes	yes	4.92	3411.2
5.2.08	600239	no	no	5.87	3726.1
5.2.08	600239	no	yes	5.83	3937.6
5.2.08	600239	yes	no	4.97	3459.8
5.2.08	600239	yes	yes	4.96	3647.7
5.2.10	1079248	no	no	5.30	2340.3
5.2.10	1079248	no	yes	5.30	2692.0
5.2.10	1079248	yes	no	5.49	2627.9
5.2.10	1079248	yes	yes	5.50	2686.8
boat.512	655628	no	no	6.17	2977.1
boat.512	655628	no	yes	6.16	3035.2
boat.512	655628	yes	no	5.09	2956.4
boat.512	655628	yes	yes	5.08	3008.6
elaine.512	559738	no	no	7.48	3378.3
elaine.512	559738	no	yes	7.47	3391.9
elaine.512	559738	yes	no	5.74	3339.6
elaine.512	559738	yes	yes	5.72	3405.1

Program (GIMP) version 2.4.4. The size of these image files are recorded in Table 5.5 along with the RMS error between the JPEG encoding and the original version of each image. We compare the JPEG images to compressively sampled images thresholded input encoded at 5 bits per sample and then recovered with particle swarm with infusion. Even though the particle swarm is able to recover image 5.2.10 with the same RMS error as JPEG, it uses almost twice as many bits to do so. For all the other images, JPEG uses fewer bits and has a smaller RMS error value.

There are a few ways that the compressive sampling approach could vary the tradeoff between encoding size and recovery error to get one of the two closer to the JPEG results. If fewer samples are used, the encoded representation could still contain enough information to recover the image, but the size of the encoded image would be reduced. This would likely increase error in the reconstructed image. To improve recovery error, more bits per sample could be used for image encoding.

Figure 5.10 shows a side-by-side comparison of the compressively sampled version and the JPEG version of the 4.2.04 image. The compressively sampled image is visually quite similar to the JPEG image, although the compressively sampled version exhibits block artifacts, most prominently on the photograph subject's shoulder, that the JPEG image does not have.

5.3 Conclusion

We evaluate signal recovery refinements on synthetic signals to identify a good combination of refinements to use. Some refinements, such as the funnel and redrop refinements, decrease error more when used together than either refinement does

Table 5.5: RMS error and encoded size for each of the six test images. CS size and CS RMS error refer to the compressively sampled encoding size and reconstruction error for the thresholded image using infusion, respectively. The JPEG versions of these images were created at a quality level of 80.

Image	CS size (bits)	JPEG size (bits)	CS RMS Error	JPEG RMS Error
4.2.04	473782	310992	4.46	3.17
4.2.07	477365	320400	4.92	3.82
5.2.08	600239	363712	4.96	3.30
5.2.10	1079248	571728	5.50	5.50
boat.512	655628	386288	5.08	3.84
elaine.512	559738	343296	5.72	4.71

Figure 5.10: A comparison of compression schemes for the 4.2.04 image. The image on the left has been compressively sampled from a thresholded version of the 4.2.04 image and encoded at 5 bits per sample, then recovered using particle swarm with infusion. The image on the right has been compressed using the JPEG format at a quality level of 80.

separately. We select the guided method with OMP initialization for image recovery. The isolated DC method provides the smallest encoded size and best recovery error among the DC coefficient methods for the images in our image test suite. While infusion does not significantly lower recovery error, thresholding lowers error for five out of the six tested images. Our image compression method is not yet competitive with JPEG in terms of encoded size or RMS error.

CHAPTER SIX

Conclusion

We show that Particle Swarm Optimization can be applied to the sparse signal recovery problem. Several refinements to the basic particle swarm algorithm decrease error in the recovered signal, and even lower the number of iterations needed to reconstruct a signal from the compressed samples. The guided method provides excellent signal recovery performance on synthetically generated signals, recovering all signals with sparsity $K > 5$ with near-zero error.

As long as the input signal is exactly sparse, the particle swarm approach to sparse signal recovery works well. With image data that is only approximately sparse, thresholding the input signal to ensure exact sparsity usually helps the particle swarm to recover the image with less error. Image-based refinements to the signal recovery process such as separate DC coefficient encoding and infusion also help improve error metrics for compressively sampled images.

Although the JPEG lossy image compression format may currently outperform the image signal recovery ability of particle swarm, the PSO-based approach still achieves lower signal recovery error for exactly sparse input signals than the standard OMP algorithm. However, it is possible that new OMP-derived methods of sparse signal recovery such as CoSaMP may outperform particle swarm.

6.1 Future Work

One area for future research is the encoding method for compressed samples. The current encoding method uses a fixed number of bits per sample, and scales the values in the compressed vector to make the best use of the available bits. However, this scaling is not as effective when there are a large number of samples, since the average range between the minimum and maximum value increases along with an increase in the number of samples. One alternate approach is a variable-length encoding scheme that calculates the number of bits required to encode the samples. This might decrease error by encoding the compressed signal more exactly.

Other frequency-domain transformations could be used in place of the DCT. One possibility is a wavelet transformation. We briefly tested a two-dimensional wavelet transformation on the 4.2.04 image, and noted error results and encoding sizes similar to those achieved using the DCT.

While only 8-by-8 pixel blocks were used for this research, it would be possible to use other image block sizes. Larger image block sizes would increase the dimensionality of the search space for particle swarm, so the difficulty in recovering a compressively sampled image may increase. Larger block sizes would also mean fewer blocks, so the particle swarms could run for more iterations or use more particles to search for solutions. For very sparse images, large blocks would still be as sparse as smaller blocks, so the encoded image size would decrease. However, vector operations would require more computation time since vector sizes would increase.

The particle swarm approach to sparse signal recovery could also be applied to other domains. In particular, audio compression would be an interesting application

of compressive sampling. Like images, audio can be compressed using lossy compression. Just as images show compression errors visually, an audio signal would reveal compression errors aurally.

Further research could also compare particle swarm signal recovery to the best-known greedy method, CoSaMP. This method advertises much lower error than the basic OMP method, so an experimental test may provide greater insight into the relative performance of particle swarm as a sparse signal recovery method. In addition, CoSaMP could serve as a particle initialization scheme if the particle swarm is able to improve upon its solution.

Currently, the infusion process does not significantly improve the recovery of images. More work could be done to enhance infusion in various ways to turn it into an effective technique for lowering error during image recovery. The current infusion method only shares global best updates with immediately surrounding blocks. This could be modified to infuse updates into a larger radius of surrounding blocks. Infusion could also be extended to sharing the local best values of particles between two blocks if either block successfully updated the other block's global best.

More work could also be done on tailoring recovery methods for specific sparsity levels. For example, more particles could be used for swarms recovering signals with higher average error based on sparsity. The number of particles could also be determined by the size of the search space.

BIBLIOGRAPHY

N. Ahmed, T. Natarajan, and K.R. Rao. Discrete cosine transfom. *Computers, IEEE Transactions on*, C-23(1):90–93, Jan. 1974.

Waheed Bajwa, Jarvis Haupt, Akbar Sayeed, and Robert Nowak. Compressive wireless sensing. In *IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks*, pages 134–142, New York, NY, USA, 2006. ACM.

D.M. Bethel, D.M. Monro, and B.G. Sherlock. Optimal quantisation of the discrete cosine transform for image compression. volume 1, pages 69–72 vol.1, Jul 1997.

Shaobing Chen and D. Donoho. Basis pursuit. volume 1, pages 41–44 vol.1, Oct-2 Nov 1994.

S. S. B. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal of Scientific Computing*, 20(1):33–61, 1999.

Roger J. Clarke. *Digital Compression of Still Images and Video*. Academic Press, Inc., Orlando, FL, USA, 1995.

J. Cooley, P. Lewis, and P. Welch. The finite fourier transform. *Audio and Electroacoustics, IEEE Transactions on*, 17(2):77–85, Jun 1969.

C.A.C. Coello, G.T. Pulido, and M.S. Lechuga. Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):256–279, June 2004.

S.F. Cotter and B.D. Rao. Sparse channel estimation via matching pursuit with application to equalization. *Communications, IEEE Transactions on*, 50(3):374–377, Mar 2002.

E. J. Candes and J. Romberg. Sparsity and incoherence in compressive sampling. *Inverse Problems*, 23(3):969–985, June 2007.

Emmanuel Candes, Mark Rudelson, Terence Tao, and Roman Vershynin. Error correction via linear programming. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:295–308, 2005.

E.J. Candes and M.B. Wakin. An introduction to compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):21–30, March 2008.

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

David L. Donoho and Philip B. Stark. Uncertainty principles and signal recovery. *SIAM Journal on Applied Mathematics*, 49(3):906–931, 1989.

R.C. Eberhart and J. Kennedy. Particle swarm optimization: developments, applications and resources. In *A new optimizer using particle swarm theory*, pages 39–43, 1995.

Hsuan-Ming Feng, Ching-Yi Chen, and Fun Ye. Evolutionary fuzzy particle swarm optimization vector quantization learning scheme in image compression. *Expert Syst. Appl.*, 32(1):213–222, 2007.

J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *Proc. of IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

Michael Lustig, David Donoho, and John M. Pauly. Sparse mri: The application of compressed sensing for rapid mr imaging. *Magnetic Resonance in Medicine*, 9999(9999):NA+, 2007.

A.S. Lewis and G. Knowles. Image compression using the 2-d wavelet transform. *Image Processing, IEEE Transactions on*, 1(2):244–250, Apr 1992.

Stephane G. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11:674–693, 1989.

Shahar Mendelson, Alain Pajor, and Nicole Tomczak-Jaegermann. Uniform uncertainty principle for bernoulli and subgaussian ensembles, Aug 2006.

Deanna Needell and Roman Vershynin. Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit, March 15 2007. Comment: This is the final version of the paper, including referee suggestions.

H. Nyquist. Certain topics in telegraph transmission theory. *Proceedings of the IEEE*, 90(2):280–305, Feb 2002.

Justin Romberg. Imaging via compressive sampling [introduction to compressive sampling and recovery via convex programming]. *IEEE Signal Processing Magazine*, 25(2):14–20, March 2008.

Mark Rudelson and Roman Vershynin. On sparse reconstruction from fourier and gaussian measurements. Technical report, Communications on Pure and Applied Mathematics, 2006.

C.E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, Jan. 1949.

M.A. Sheikh, O. Milenkovic, and R.G. Baraniuk. Designing compressive sensing dna microarrays. pages 141–144, Dec. 2007.

Dianxun Shuai, Ping Zhang, and Bin Zhang. Particle algorithm for lossless data compression. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 3766–3771, Oct 2006.

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

G. Taubock and F. Hlawatsch. A compressed sensing technique for ofdm channel estimation in mobile environments: Exploiting channel sparsity for reducing pilots. pages 2885–2888, 31 2008-April 4 2008.

Joel A. Tropp and D. Needell. CosaMP: Iterative signal recovery from incomplete and inaccurate samples. *CoRR*, abs/0803.2392, 2008. informal publication.

G.K. Wallace. The jpeg still picture compression standard. *Consumer Electronics, IEEE Transactions on*, 38(1):xviii–xxxiv, Feb 1992.

Allan Weber. The usc-sipi image database. `http://sipi.usc.edu/database/`.

Michael B. Wakin, Jason N. Laska, Marco F. Duarte, Dror Baron, Shriram Sarvotham, Dharmpal Takhar, Kevin F. Kelly, and Richard G. Baraniuk. An architecture for compressive imaging. In *in IEEE International Conference on Image Processing (ICIP*, pages 1273–1276, 2006.