Fast Algorithms for Structured Sparsity (ICALP 2015 Invited Tutorial)

Chinmay Hegde
Iowa State University
chinmay@iastate.edu

Piotr Indyk
MIT
indyk@mit.edu

Ludwig Schmidt
MIT
ludwigs@mit.edu

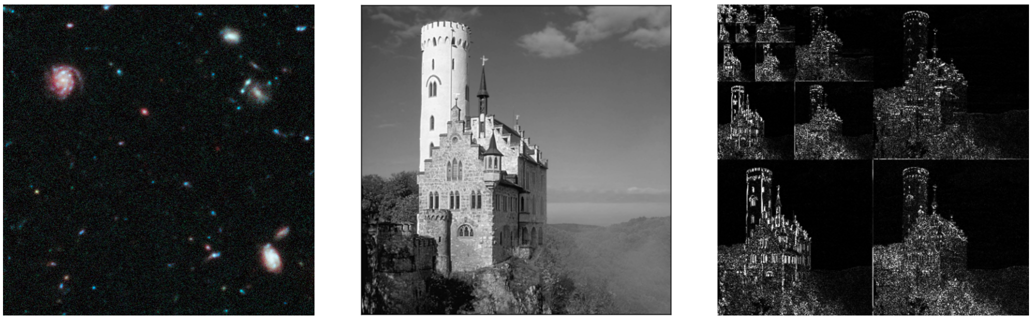
Abstract

Sparsity has become an important tool in many mathematical sciences such as statistics, machine learning, and signal processing. While sparsity is a good model for data in many applications, data often has additional structure that goes beyond the notion of "standard" sparsity. In many cases, we can represent this additional information in a *structured sparsity model*. Recent research has shown that structured sparsity can improve the sample complexity in several applications such as compressive sensing and sparse linear regression.

However, these improvements come at a computational cost, as the data needs to be "fitted" so it satisfies the constraints specified by the sparsity model. In this survey, we introduce the concept of structured sparsity, explain the relevant algorithmic challenges, and briefly describe the best known algorithms for two sparsity models. On the way, we demonstrate that structured sparsity models are inherently combinatorial structures, and employing structured sparsity often leads to interesting algorithmic problems with strong connections to combinatorial optimization and discrete algorithms. We also state several algorithmic open problems related to structured sparsity.

1 Introduction

Over the past two decades, *sparsity* has emerged as an important model for data in several fields including signal processing, statistics, and machine learning. In a nutshell, sparsity allows us to encode structure present in many relevant types of



(a) Sparse image (b) Dense image (c) Wavelet coefficients

Figure 1: Examples of sparsity. Subfigure (a) contains an image from the Hubble space telescope [34]. The image is sparse because it contains a small number of bright pixels. The castle in Subfigure (b) [12] is not sparse, but its wavelet coefficients in Subfigure (c) give a much sparser representation.

data but still remains a simple concept that is amenable to mathematical analysis and efficient algorithms. Informally, we say that a vector or a matrix are (approximately) sparse if they contain only a small number of non-zero entries or are well approximated by a small subset of their coefficients.

Examples of sparsity. As a simple illustration of sparsity, consider the image of space in Figure 1 (a). Clearly, the image contains only a small number of bright pixels while the majority of pixels is very dark and hence approximately zero. Therefore, the image can be modeled as a sparse matrix or a sparse vector (for the latter, we simply stack the image columns to form a large vector).

While common in some scientific domains, "natural" images are rarely the combination of a black background with a small number of bright point sources. For instance, the picture of a castle in Figure 1 (b) contains many pixels in different shades of grey and only a small number of very dark (and hence approximately zero) pixels. Although this prevents us from modeling the canonical representation of Figure 1 (b) as sparse, research in signal processing has shown that many classes of images are often significantly more sparse when represented in a suitable basis. In particular, the wavelet basis often gives a sparse representation of natural images, e.g., see Figure 1 (c) for the wavelet coefficients of the castle image.[1] A wide range of algorithms for sparse processing can also be applied when the data is sparse in some transform domain.

[1] The main technical content of this survey requires no prior knowledge of wavelets or other concepts from signal processing.

Applications of sparsity. As we have seen above, sparsity allows us to encode structure in several types of data. In many applications, utilizing this structure leads to significant performance improvements over worst-case assumptions, and hence there is a wide range of algorithms building on the concept of sparsity. Some of these applications are the following:

Image compression. The popular JPEG compression standard is engineered for the common case that the input image is sparse when represented in the Fourier basis. By storing only the small number of large coefficients, JPEG can save space compared to a dense representation of all coefficients in the image. At the same time, JPEG incurs only a small loss in image quality from discarding the small coefficients.

Denoising. One approach to denoising signal and image data is to assume that the large coefficients contain the salient information about our data while the small coefficients contain mostly noise. Thus, removing the small coefficients can reduce the amount of noise in the data.

Compressive sensing. The goal in compressive sensing is to recover a signal from a small number of measurements. Over the past decade, a large body of research has established that this is possible in a variety of settings when the signal is sparse.

Sparse linear regression. High-dimensional statistics studies problems in which the number of unknown parameters can be significantly larger than the number of data samples. While these cases are out of reach for traditional approaches such as linear least squares, exploiting sparsity of the parameters can lead to problems that are both information-theoretically and computationally tractable.

Sparse coding. Sparsity is also employed in popular machine learning algorithms. One instance is sparse coding (also known as dictionary learning), which is an unsupervised learning algorithm that automatically finds a sparse representation for a given data set. The sparse coefficients and the corresponding basis elements then often correspond to relevant features of the data.

1.1 Structured sparsity

While basic sparsity usually captures some structure in our problem, we often possess additional prior information about the particular type of data we are working with. For instance, consider Figure 1 (a). In addition to the image being sparse, the bright pixels of the image also form a small number of *clusters* instead of being

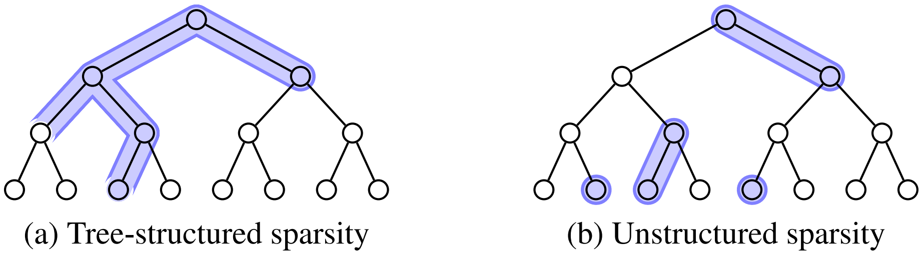


Figure 2: Structured vs unstructured sparsity. Since wavelet coefficients are defined at multiple scales, they can naturally be arranged as a balanced tree. In some cases (Subfigure (a)), we know that the large coefficients tend to form a rooted subtree as opposed to being spread over the entire tree of wavelet coefficients (Subfigure (b)). Using tree-structured sparsity can lead to a better compression ratio in image encoding and better sample complexity in sparse recovery.

randomly spread over the image as individual bright pixels. Since this structure is shared by similar astronomical images, we therefore expect that the large coefficients form a small number of connected components in this type of data.

Similarly, the wavelet coefficients in Figure 1 (c) also contain additional structure beyond sparsity. Due to the hierarchical nature of the wavelet decomposition, the wavelet coefficients can be arranged as a balanced *tree*, and the large coefficients tend to form a rooted subtree. See Figure 2 for an illustration and Section 3 for a more detailed explanation.

Another common form of structure is *group sparsity*. For instance, consider a linear regression setting where each covariate indicates the expression level of a certain gene and we want to predict the outcome of a certain disease. It is a natural assumption that the disease outcome depends only on a small number of genes. Moreover, we sometimes know from biology that certain genes belong to functional groups that are usually active or inactive together. In those cases, we expect that the sparsity pattern of our regression parameters can be explained as the union of a small number of such groups.

All three examples share a similar theme: sparsity captures important “primary” structure in the data, but there is also rich “secondary” structure beyond the fact that the data is sparse. Since sparsity has turned out to be a useful ingredient in many fields, this poses a natural question: How can we encode such secondary structure, and how can we utilize it in our applications of interest? In the rest of this survey, we will address these questions and show that ideas from discrete algorithms and combinatorial optimization can lead to novel algorithms for the resulting problems.

As an illustration of how structured sparsity can be beneficial, consider image compression with wavelet coefficients. In this case, we can think of our nonzero coefficients (i.e., the coefficients we wish to encode) as a subtree in a larger tree of all available wavelet coefficients. To give a quantitative comparison, let s be the number of nonzero coefficients and d be the total number of wavelet coefficients (most of which are zero). Without structure in our sparsity pattern, we would require $O(s \log d)$ bits to store the locations of the nonzeros. With the tree assumption, we only need to store a constant number of bits per node, which requires only $O(s)$ bits in total. Specifically, consider a traversal of the tree that starts and finishes at the root and visits each edge twice. Such a traversal can be described by labeling each node with two bits that specify whether the left child (or the right child, respectively) is non-empty.

As a result, structured sparsity leads to an improved compression performance. In fact, this insight is at the core of the influential Embedded Zerotrees of Wavelet transforms (EZW) algorithm [49].

1.2 Structured sparsity in sparse recovery

A large part of this survey is focused on structured sparsity in the context of sparse recovery problems such as compressive sensing and sparse linear regression. Here, structured sparsity is often a desirable ingredient because it allows us to reduce the sample complexity: a *structured* sparse vector can usually be recovered from asymptotically fewer observations than an arbitrarily sparse vector. Due to the wide applicability of sparse recovery problems, there is also a wide range of sparsity structures that have been studied, such as block sparsity, tree sparsity, cluster sparsity, and others (see later sections).

One way of encoding these sparsity structures in a common framework is via *structured sparsity models*. The idea behind this concept is that the essential features of structured sparse data can often be captured by restricting the support sets of the corresponding vectors (the support of a vector is the set of indices corresponding to nonzero coefficients). While "standard" sparsity makes no assumptions about the supports besides a bound on their cardinality, a structured sparsity model permits only supports sets with a certain structure. Continuing the wavelet tree example from Figure 2 (a), the corresponding structured sparsity model contains only supports that have both a bounded cardinality and form a subtree of the wavelet coefficient tree.

In addition to formulating sparsity structures in a common language, structured sparsity models also allow us to clearly separate sparse recovery algorithms from the sparsity model we want to employ. In particular, there are several sparse recovery algorithms that can be adapted to an *arbitrary* structured sparsity model as long as we supply a *model-projection* oracle as a subroutine for the recovery algorithm. Intuitively, a model-projection oracle finds the *best* approximation of an arbitrary input vector with a vector in our sparsity model of interest. More precisely, let $\mathcal{M}$ be the set of vectors in our sparsity model and let $x \in \mathbb{R}^d$ be an arbitrary real vector. Then a model projection oracle $P_{\mathcal{M}}(x) : \mathbb{R}^d \rightarrow \mathbb{R}^d$ for the model $\mathcal{M}$ satisfies

$$P_{\mathcal{M}}(x) = \arg\min_{x' \in \mathcal{M}} \|x - x'\|_2^2 \,. \qquad (1)$$

Given a model projection oracle satisfying guarantee (1), we can directly instantiate known results in order to construct an efficient algorithm for the sparse recovery problem. Hence the main algorithmic challenge is to design efficient model projection oracles for a given sparsity model. In the rest of this survey, we will see a variety of model projection algorithms for Problem (1) and suitable relaxations of this guarantee. It turns out that many sparsity structures give rise to model projection problems that are closely related to questions studied in discrete algorithms and combinatorial optimization.

1.3 Related work

There is a wide range of work on utilizing structured sparsity in sparse recovery, e.g., [53, 37, 22, 28, 39, 40, 35, 11, 46, 43, 48, 20]. The different approaches employ a wide range of techniques, including convex programming, Bayesian methods, and iterative algorithms. We refer the reader to the surveys [17, 10, 52] for an overview of these approaches.

In this survey, we focus on the *model-based compressive sensing* framework introduced in [5], which employs an iterative approach to sparse recovery. This framework enables a clear separation between the "outer" sparse recovery algorithms (described in Section 5) and the "inner" projection oracles specific to particular sparsity models (described in Sections 3 and 4).

1.4 Outline of the paper

In Section 2, we give a formal definition of structured sparsity models and introduce the main algorithmic problems. We illustrate these definitions with block sparsity, which is a simple sparsity model. Sections 3 and 4 then give algorithms for two sparsity models that require more sophisticated techniques: tree sparsity and graph sparsity. Section 5 explains sparse recovery problem, which is the main application of our algorithms for structured sparsity.

1.5 Notation

Let $[d]$ be the set $\{1, 2, \ldots, d\}$. We say that a vector $x \in \mathbb{R}^d$ is s-sparse if at most s of its coefficients are nonzero. The support of x contains the indices corresponding to nonzero entries in x, i.e., $\text{supp}(x) = \{i \in [d] \,|\, x_i \neq 0\}$. Given a subset $S \subseteq [d]$, we write x_S for the restriction of x to indices in S: we have $(x_S)_i = x_i$ for $i \in S$ and $(x_S)_i = 0$ otherwise. The ℓ_2-norm of x is $\|x\|_2 = \sqrt{\sum_{i \in [d]} x_i^2}$.

2 Structured sparsity models

In this section, we formally define the notion of a structured sparsity model and illustrate the definition with block sparsity as a simple example. We use structured sparsity as a common framework for encoding prior knowledge that goes beyond the assumption that our data of interest is s-sparse. In particular, a structured sparsity model imposes restrictions on the allowed supports, which leads to the following definition:

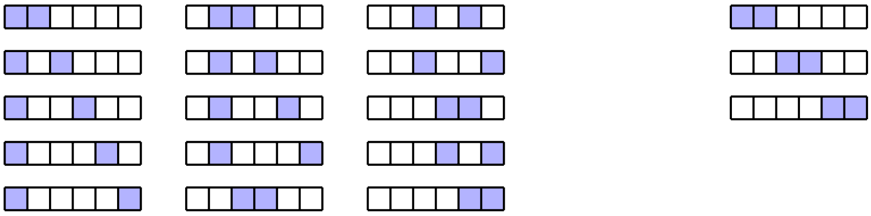
Definition 1 (Structured sparsity model [5]). *Let $\mathbb{M}$ be a family of supports, i.e., $\mathbb{M} = \{S_1, S_2, \ldots, S_L\}$ where each $S_i \subseteq [d]$. Then the corresponding structured sparsity model $\mathcal{M}$ is the set of vectors supported on one of the S_i:*

$$\mathcal{M} = \{x \in \mathbb{R}^d \,|\, \mathrm{supp}(x) \subseteq S \text{ for some } S \in \mathbb{M}\} .$$

Depending on the context, we refer to either the family of supports $\mathbb{M}$ or the corresponding set of vectors $\mathcal{M}$ as a structured sparsity model. It is worth noting that the structure we encode is described entirely by the allowed locations of the nonzero coefficients: a structured sparsity model imposes no restrictions on the actual coefficient values besides whether they are zero.

Usually, the sizes of the individual supports $|S_i|$ are uniformly bounded by a common sparsity parameter s such that the vectors in $\mathcal{M}$ are *both* s-sparse and have a restricted support. For many sparsity models of interest, the "cardinality" of the model, i.e., the number of allowed supports $|\mathbb{M}|$, is significantly smaller than the set of all possible s-sparse supports $\binom{d}{s}$. In sparse recovery, the smaller number of relevant supports $|\mathbb{M}| \ll \binom{d}{s}$ leads to an improved sample complexity and is one of the main reasons for employing a structured sparsity model.

In most cases, the family of allowed supports $\mathbb{M} = \{S_1, S_2, \ldots, S_L\}$ is not given as an explicit family of sets, but $\mathbb{M}$ rather is implicitly defined through a common property shared by all allowed supports. For instance, consider an application where our vectors of interest are split into consecutive blocks of B non-overlapping coefficients. In the *block sparsity* model, we then assume that only s/B blocks contain nonzeros, i.e., the vector is s-sparse and the nonzeros occur only in a small number of contiguous blocks. This model can be seen as a discretization of the cluster sparsity structure present in Figure 1 (a), i.e., the clusters of nonzero coefficients can occur only in a fixed set of locations in the vector (for a more flexible sparsity model, see Section 4). Figure 3 illustrates the block sparsity model with an example.



(a) All 2-sparse vectors.

(b) Block-sparse vectors.

Figure 3: Block sparsity. Subfigure (a) shows all 2-sparse supports for ambient dimension $d = 6$. Subfigure (b) shows all supports that are both 2-sparse and block sparse with block size $B = 2$. The number of structured supports is significantly smaller than the number of all 2-sparse supports. This fact leads to improved sample complexity in sparse recovery.

2.1 Model projection

For a given sparsity model $\mathcal{M}$, one of the main algorithmic problems is to project an arbitrary vector $x \in \mathbb{R}^d$ into the sparsity model, i.e., to return the best approximation of the given vector with a vector $x' \in \mathcal{M}$. Note that once we have fixed a support $\text{supp}(x') = S \in \mathbb{M}$, the approximation error $\|x - x'\|_2^2$ is minimized by setting the coefficients of x' such that they are equal to x for indices in S. Hence the model projection problem can be reduced to finding the best support in $\mathbb{M}$. This leads to the following definition:

Definition 2 (Model projection [5])**.** *Let $\mathbb{M}$ be a structured sparsity model. A model projection oracle for $\mathbb{M}$ is an algorithm $P(x) : \mathbb{R}^d \rightarrow \mathbb{M}$ with the following guarantee: for any $x \in \mathbb{R}^d$, let $S = P(x)$. Then we have*

$$\|x - x_S\|_2^2 \;=\; \min_{S' \in \mathbb{M}} \|x - x_{S'}\|_2^2 \,. \tag{2}$$

Model projection algorithms satisfying the guarantee in Definition 2 are a key ingredient in sparse recovery algorithms. Given such a model projection algorithm, we can instantiate known results from sparse recovery and directly obtain a recovery algorithm for the corresponding sparsity model [5]. Since the model projection algorithm is invoked multiple times in the overall recovery algorithm, achieving a good time complexity for the model projection step is often an important goal. In many cases, the other components of the recovery algorithm run in nearly-linear time, so an ideal model projection algorithm would satisfy Definition 2 and run in nearly-linear time for any value of s.

Model projection algorithms are also used outside of sparse recovery. Directly applying a model projection algorithm to a known vector extracts a representation that captures most of the large coefficients while satisfying the structural constraints encoded in the model. Such an approach can reduce the given data to its essential features or extract interesting structure. For instance, some methods in seismic processing and event detection are essentially model projection algorithms [51, 45].

For the block sparsity model introduced above, the model projection problem can be solved rather easily. First, we observe that we can always only select s/B blocks of the vector due to the structural constraint. Moreover, once we have chosen a set of blocks, we minimize the approximation error $\|x - x_S\|_2$ by including all indices of our selected blocks in the final support S. Since $\|x - x_S\|_2^2 = \|x\|_2^2 - \|x_S\|_2^2$, our task reduces to finding the s/B blocks with the maximum sum of squared coefficients. So for each block $j \in \{0, \ldots, s/B - 1\}$, we compute its weight

$$w(j) = \sum_{i=js/B+1}^{(j+1)s/B} x_i^2$$

and then simply select the s/B blocks with the largest weights $w(j)$. Besides returning the optimal approximation, this algorithm also runs in linear time. While the model projection problem turns out to be simple for block sparsity, other sparsity models lead to significantly more complicated problems.

2.2 Approximate model projections.

In many cases (see e.g., Section 3) the best known algorithms for exact model projections are prohibitively slow to run on large or even medium-scale data sets. In other cases, the exact model projection problem as stated in Definition 2 is even NP-hard. A fruitful way to circumvent these complexity barriers is via *approximation*. The natural relaxation of the exact model projection problem is to allow an approximation factor in Equation (2):

$$\|x - x_S\|_2^2 \leq c \cdot \min_{S' \in \mathbb{M}} \|x - x_{S'}\|_2^2 \qquad (3)$$

where $c > 1$ is a fixed constant. While this guarantee is already useful in some applications such as feature extraction or denoising, it is *not* sufficient for stable sparse recovery, at least using known recovery algorithms. In particular, for any constant $c > 1$, it is possible to construct an adversarial projection oracle that satisfies Equation (3) but prevents sparse recovery algorithms from succeeding for sufficiently large dimension d.

The reason behind this phenomenon is as follows. First, Equation (3) guarantees that the support S is a good approximation to the input vector x as long as *some* support in the model is a good approximation. However, consider the case that the input x is far from all vectors in the model, i.e., $\min_{S'\in\mathbb{M}}\|x - x_{S'}\|_2$ is large. Then even the empty support can satisfy Equation (3) as long as the improvement achieved by the optimal support is less than the slack introduced by allowing $c > 1$. More formally, the approximation error improvement of the optimal support S^* compared to the empty set is

$$\left\|x - x_{\{\}}\right\|_2^2 - \|x - x_{S^*}\|_2^2 \;=\; \|x\|_2^2 - \|x - x_{S^*}\|_2^2 \;=\; \|x_{S^*}\|_2^2 \;.$$

So as long as the approximation slack in Equation (3) is large enough so that it satisfies

$$\|x_{S^*}\|_2^2 \;\leq\; (c-1)\min_{S'\in\mathbb{M}}\|x - x_{S'}\|_2^2 \;=\; (c-1)\|x - x_{S^*}\|_2^2$$

we get

$$\left\|x - x_{\{\}}\right\|_2^2 \;=\; \|x_{S^*}\|_2^2 + \|x - x_{S^*}\|_2^2 \;\leq\; c\cdot\min_{S'\in\mathbb{M}}\|x - x_{S'}\|_2^2 \;.$$

Then an approximate projection oracle can simply return the empty set.

Thus, in order to get provable guarantees for structured sparse recovery, we also need another notion of approximation instead of only relaxing Definition 2. The failure case of Equation (3) suggests a possible fix: to prevent an approximate oracle from simply returning an empty set, we should also require that the support captures at least a constant fraction of the energy compared to the optimal support. This idea leads us to defining a *second* notion of approximate projection oracles, which complements Equation (3). The two resulting definitions are as follows:

Definition 3 (Tail approximation)**.** *Let $\mathbb{M}$ be a structured sparsity model. A tail approximation oracle for $\mathbb{M}$ is an algorithm $T(x) : \mathbb{R}^d \to \mathbb{M}$ with the following guarantee: for any $x \in \mathbb{R}^d$, let $S = T(x)$. Then we have*

$$\|x - x_S\|_2^2 \;\leq\; c_T\cdot\min_{S'\in\mathbb{M}}\|x - x_{S'}\|_2^2 \;,$$

where c_T is an arbitrary fixed constant (independent of d, x, and $\mathbb{M}$).

Definition 4 (Head approximation)**.** *Let $\mathbb{M}$ be a structured sparsity model. A head approximation oracle for $\mathbb{M}$ is an algorithm $H(x) : \mathbb{R}^d \to \mathbb{M}$ with the following guarantee: for any $x \in \mathbb{R}^d$, let $S = H(x)$. Then we have*

$$\|x_S\|_2^2 \;\geq\; c_H\cdot\max_{S'\in\mathbb{M}}\|x_{S'}\|_2^2 \;,$$

where $c_T > 0$ is an arbitrary fixed constant (independent of d, x, and $\mathbb{M}$).

It is worth noting that these two guarantees are complementary and in general, one does not imply the other. The argument above already shows that tail approximation does not imply head approximation. For the opposite direction, consider an input vector x that belongs to the sparsity model, i.e., its tail approximation error is zero. While a tail approximation oracle must return a support with error zero in this case, a head approximation oracle can still return a support that captures only a constant fraction of the optimal solution.

Once we have approximate projection oracles (one satisfying Definition 3 and one satisfying Definition 4), we can instantiate the *approximation-tolerant* sparse recovery framework introduced in [32]. The framework provides general sparse recovery algorithms that invoke approximate projection oracles in a black-box fashion. As long as c_T and c_H are fixed constants, the sample complexity of the sparse recovery algorithm is only affected by constant factors and the final recovery guarantees are essentially the same as in the case of exact projection algorithms. The number of invocations of the head and tail approximation algorithms are the same up to constant factors, so the time complexity of the slower algorithm determines the overall time complexity.

Besides relaxing the approximation error guarantees, the framework in [32] also allows projections into *larger* sparsity models. For instance, consider the case of block sparsity: instead of returning a support formed by s/B blocks with total sparsity s, a projection oracle could also return $2s/B$ blocks with total sparsity $2s$. As long as the size of the relaxed model is comparable to the original model, the sample complexity will only be affected by constant factors. In particular, let $\mathbb{M}'$ be the image of a projection oracle. As long as $\log \mathbb{M}' = O(\log \mathbb{M})$, the sample complexity of the sparse recovery algorithm is unchanged. This second relaxation of the projection guarantees turns out to be crucial for several approximate projection algorithms because it allows us to work with *bicriterion* guarantees that offer trade-offs between the approximation ratio and the support size. Sections 3 and 4 give an overview of several such algorithms.

3 Tree sparsity

The *tree-structured sparsity model* (or *tree-sparsity model*, for short) is perhaps the simplest example of a structured sparsity model where implementing the model projection oracle is non-trivial. In this model, we assume that the d coefficients of the vector x can be arranged as the nodes of a full b-ary tree $\mathcal{T}$ of height h (in what follows, we assume $b = 2$, i.e., we assume that the tree is binary). Then, the tree-sparsity model contains all sets of nodes that form a *rooted subtree* of $\mathcal{T}$. Figure 2 provides an example of one such set.

The tree-sparsity model is motivated by hierarchical decompositions of images, e.g., the wavelet representation depicted in Figure 1 (c). Such representations map each image into several levels of coefficients, where each level corresponds to an appropriate "resolution" of the image. Large coefficients in such representations are typically caused by edges and other image discontinuities. Since each such discontinuity is typically detected in many levels of the hierarchy (all the way to the root), the following property usually holds: if a coefficient in a given node is large, then the coefficient in its parent tends to be large as well. This motivates modeling the supports as subtrees.

Exact model projection. Given a vector x, the model projection oracle for the tree sparsity model needs to compute a rooted subtree T of size s that maximizes the sum of squares of the coefficients x_i (weights) corresponding to the nodes in T. The hierarchical definition of the problem makes it amenable to the dynamic programming approach (first introduced for this problem in [3]).

Specifically, suppose that for a given node i and sparsity s, we want to compute a tree of size t rooted at i that maximizes the total weight of the selected nodes (we denote this maximum weight by $W[i,t]$). This can be accomplished recursively by selecting i and then, for an appropriate parameter r, selecting the maximum weight subtree of size r rooted in the left child of i, as well as the maximum weight subtree of size $t-r-1$ rooted in the left child of i. Since the optimal value of r is not known a priori, the algorithm must enumerate all possible values. This leads to a simple recursive formula

$$W[i,t] = \max_r x_i^2 + W[\text{left}(i), r] + W[\text{right}(i), t-r-1]$$

The values $W[i,t]$ can be computed in a bottom-up fashion after an appropriate initialization. The corresponding trees can be also recovered by “re-tracing” the optimal weight values in a standard manner.

Since i ranges between 1 and d while t ranges between 0 and s, it follows that the array $W[i,t]$ has size $O(ds)$, and therefore the whole algorithm runs in $O(ds^2)$ time. However, it turns out that the above analysis overestimates the real space and time cost of the algorithm. Specifically, observe that for most nodes i, one can obtain an upper bound on feasible values of t that is much lower than s. For example, if i is a leaf, then only possible values of t are 0 and 1, as the left and right subtrees of i are empty. Thus, the total number of feasible entries $W[i,t]$ over all leaves i is $O(d)$. In fact, the same bound holds for any level in the tree $\mathcal{T}$. Since there are $\log d$ levels in $\mathcal{T}$, it follows that the the actual space usage of the algorithm is $O(d \log d)$, not $O(ds)$. This immediately implies a running time bound of $O(ds \log d)$. By using a more careful analysis, one can improve it further to a “rectangular” running time of $O(ds)$ [15]. This is the best known running time for this problem.

Open problem 1: *Is there an algorithm that computes the tree-sparse projection in time faster than $O(ds)$? Alternatively, is there any evidence that such an algorithm does not exist?*

Approximate algorithms. The running time of $O(ds)$ is relatively practical when the value of s is small. However, in many applications the sparsity s is a constant fraction (say, 5%) of d. In such scenarios, the running time of the algorithm becomes quadratic in d, which makes the algorithm impractical for large inputs (e.g., for images with megapixel resolution). This necessitated the development of faster heuristics [2] and approximation algorithms [18, 9, 30, 31, 47] for this problem. In what follows, we present some illustrative examples of such algorithms.

Perhaps the first approximation algorithm of this type is due to Donoho [18]. His algorithm, called *complexity-penalized residual sum-of-squares* (CPRSS), runs in *linear* time. However, instead of solving the above problem, it solves its "Langrangian relaxation". Specifically, for a parameter $\lambda > 0$, the algorithm finds a tree T (not necessarily of size s) that maximizes the sum of weights in T minus $\lambda|T|$, or more formally, $\|x_T\|_2^2 - \lambda|T|$. That is, instead of specifying a fixed sparsity budget, the modified problem instead introduces a penalty for each node used in the tree. The tree that optimizes the new objective can be computed using a dynamic programming approach similar to the one seen earlier. However, since the new problem does not involve a fixed sparsity budget constraint, one only needs a one-dimensional array $W[i]$, as opposed to the two-dimensional $W[i,t]$ described earlier. Furthermore, the computation of $W[i]$ can be accomplished in constant time, as there is no need to enumerate all allocations of sparsity budgets to the children of i. This makes it possible to implement the overall algorithm so that it runs in linear time.

By appropriately increasing (or decreasing) the value of the parameter λ, one can "force" the aforementioned algorithm to output a tree that is "small" (or "large"). In general, it might be impossible to select λ so that the output tree T has exactly the desired size s. However, [30, 31] have shown that one can always select λ so that $|T| \leq 2s$ and the total weight of the nodes *outside of* T is within a factor of 2 from the optimum. In the language of Section 2, the algorithm offers the *tail* guarantee, while increasing the sparsity budget by a factor of at most 2. The algorithm proceeds by performing a binary search on λ in order to bring $|T|$ as close to s as possible. By optimizing the process one can achieve a running time of $O(d \log d)$. Note that this method constitutes a *bicriterion* approximation algorithm, i.e., it outputs a tree of size $2s$.

Open problem 2: *Is there a nearly-linear time single-criterion tail-approximation algorithm for tree sparsity, i.e., an algorithm that returns a tree of size at most s, satisfies the guarantee of Definition 3, and runs in time $O(d \log^{O(1)} d)$?*

The *head* approximation algorithm builds on the exact tree projection algorithm described earlier. However, instead of running the algorithm with the sparsity parameter s, we instead it run it with $s' = O(\log d)$, with reduces the total time to $O(d \log d)$. This enables us to compute a small "chunk" of the optimal tree very quickly; we also keep the dynamic programming table of the algorithm for further use. Our algorithm then uses the greedy approach. Specifically, it maintains a (growing) solution tree, and in each iteration step it finds the small tree chunk that improves the weight of the solution the most. The algorithm then updates the cost of each small tree chunk, which, thanks to the initial preprocessing, can now be performed in only $O(s'^2 \log d)$ time. The process continues until the algorithm constructs a tree of size $O(s)$. It can be seen that the whole algorithm runs in $O(d \log d + s \log^2 d)$ time.

Experiments. In order to demonstrate the computational efficiency of the approximate model-projection algorithms for tree sparsity, we show the results of a running time experiment that originally appeared in [30]. Table 1 compares four algorithms: (i) the exact tree projection algorithm of [15], (ii) the tail-approximation algorithm of [30] (both as described above), (iii) a greedy heuristic (without head- or tail-approximation guarantees), and (iv) an FFT (as a baseline). The input to the algorithms is the wavelet representation of an 512×512 pixel image, i.e., $d \approx 260,000$. The sparsity level is set to $s \approx 35,000$. The experiments were conducted on a laptop computer from 2010 equipped with an Intel Core i7 processor (2.66 GHz) and 8GB of RAM.

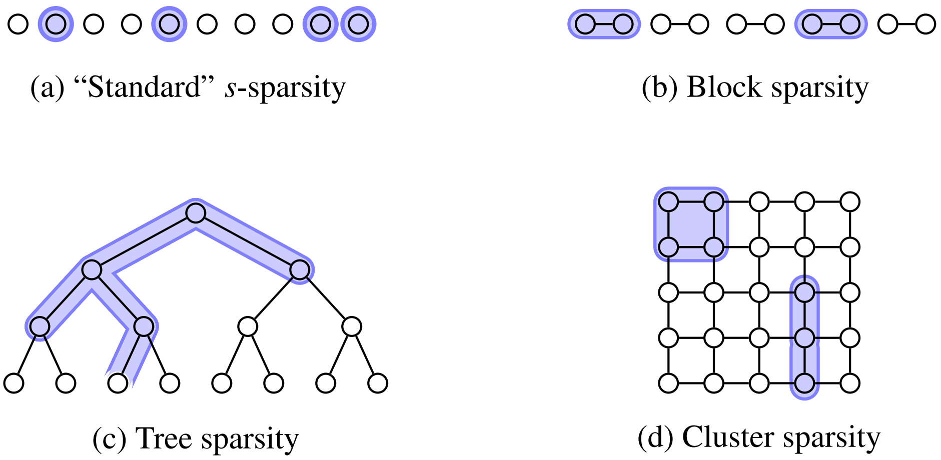
Algorithm	Exact tree	Approx. tree	Greedy tree	FFT
Runtime (sec)	4.4175	0.0109	0.0092	0.0075

Table 1: Running times of various tree projection algorithms on input data with parameters $d \approx$ 260,000, $s \approx$ 35,000. The times are averaged over 10 trials.

The results show that the approximate tree-projection algorithms is more than two orders of magnitude faster than the exact tree-projection algorithm (400× speed-up). Moreover, the approximation algorithm is almost as fast as a heuristic without provable guarantees. The comparison with the FFT running time also shows that two FFTs are more expensive than a single invocation of the tail-approximation algorithm. This is an important comparison because two FFTs are often the other part of the iterations repeated by sparse recovery algorithms. Hence the approximate tree-projection step is not a bottleneck in the overall recovery algorithm.

4 Graph sparsity

The *graph sparsity* model is a general sparsity model that allows us to encode several other sparsity models and give efficient approximation algorithms for all


(a) "Standard" s-sparsity

(b) Block sparsity

(c) Tree sparsity

(d) Cluster sparsity

Figure 4: The graph sparsity model can encode several other structured sparsity models. (a) By using a graph without edges and setting the number of connected components to s, we recover "standard" s-sparsity. (b) We can encode the block sparsity model (see Section 2) by using a line graph and removing the edges between blocks. (c) A single connected component in a tree graph gives the tree sparsity model from Section 3. (d) In order to model clusters in an image such as Figure 1 (a), we can impose a grid graph on the image (each pixel is a node) and require that the support forms a small number of connected components in the grid graph.

these models. Formally, the graph sparsity model is defined as follows [35]:[2] we associate each coefficient index $i \in [d]$ with a node in a given, fixed graph $G = ([d], E)$. For each support $S \subseteq [d]$, we can then consider properties of the induced subgraph $G_S = (S, \{(u, v) \in E \,|\, u \in S \text{ and } v \in S\})$. In particular, we restrict the set of allowed supports by requiring that G_S has at most a given size s (i.e., S has at most cardinality s) and that G_S contains at most g connected components. By imposing these connectivity constraints, the graph sparsity model enforces that the supports form a small number of clusters, where the notion of locality is encoded in the graph G. In order to get an improved sample complexity in sparse recovery, we are usually interested in the regime where the connected components have non-trivial size, i.e., $g \ll s$.

One attractive feature of the graph sparsity model is its generality: by a proper

[2] Recently, a *weighted* version of the graph sparsity model has also been proposed [33]. The weighted graph model generalizes a larger number of structured sparsity models than the unweighted graph sparsity model. In order to simplify our discussion, we focus on the unweighted case and refer the interested reader to [33] for further details. It is worth noting that the algorithmic techniques described here for the unweighted case can also be adapted to the weighted graph model.

choice of the underlying graph G, we can encode several other structured sparsity models. As depicted in Figure 4 (a) – (c), the graph sparsity model generalizes the "standard" s-sparsity model, block sparsity, and tree sparsity. Moreover, we can encode a natural notion of cluster sparsity in images such as Figure 1 (a) by interpreting the image as a grid graph and requiring that the support forms a small number of connected components in this grid graph (see Figure 4 (d)). Hence by finding a *single* efficient (ideally, nearly-linear time) model projection algorithm for the graph sparsity model, we would directly give efficient algorithms for several other structured sparsity models. Such an algorithm would therefore address a weakness in the model-based sparse recovery framework of [5]. While this framework gives an elegant way of combining model projection oracles with known sparse recovery algorithms, it does not prescribe a general "recipe" for designing such model projection oracles. Instead, the projection algorithms have to be invented from scratch for every new model.

Unfortunately, finding *exact* projections for the graph sparsity model turns out to be an NP-hard problem. This can be seen via a reduction from the classical Steiner Tree problem [26]. To be precise, we can encode the decision version of the unweighted Steiner Tree problem in the graph sparsity projection problem: given a graph $G = ([d], E)$ with a set of terminal nodes $T \subseteq [d]$, let $x \in \mathbb{R}^d$ be such that $x_i = 1$ for $i \in T$ and $x_i = 0$ otherwise. Moreover, let s be the threshold for the Steiner Tree size ("is there a Steiner Tree of size at most s?") and set the number of connected components in the graph sparsity model to 1. Now consider the model projection problem

$$OPT \;=\; \min_{S' \in \mathbb{M}} \|x - x_{S'}\|_2^2 \;.$$

If OPT is zero, we know that $\text{supp}(x) \subseteq S$ for some $S \in \mathbb{M}$ and hence there is a Steiner tree of size at most s that connects all the nonzero entries in x. Similarly, if there is a Steiner Tree $S \subseteq [d]$ of size at most s connecting all the terminal nodes in T, then there is also a corresponding support $S \in \mathbb{M}$ such that $x - x_S = 0$ and hence $OPT = 0$. Since solving the model projection problem allows us to test whether $OPT = 0$, model projection for the graph sparsity model is NP-hard.

4.1 Approximation algorithms

Local enumeration. To the best of our knowledge, the first sparse recovery algorithm for the graph sparsity model was given in [35]. Reformulated in our framework, the authors provide a head approximation algorithm that is based on the following idea: in the regime where the graph sparsity model improves the sample complexity in sparse recovery, any support in the graph sparsity model can be decomposed into $O(d/\log d)$ connected components of size $\Omega(\log d)$. The head approximation algorithm then enumerates all such connected components of size $\Theta(\log d)$ and incrementally adds the best remaining component to the support (it is worth noting that this "covering approach" is similar to the head approximation algorithm for the tree sparsity model described in Section 3). Due to the local enumeration, the algorithm has a time complexity of $O(d^c)$, where c quantifies a trade-off between time and sample complexity (c is always greater than 1). Since the paper [35] does not give a tail approximation algorithm, the resulting guarantees for sparse recovery are weaker than those achieved by sparse recovery algorithms with both head- and tail-approximations.

Prize Collecting Steiner Tree. While the Steiner Tree problem is a source of computational hardness for *exact* graph sparsity projections, we can also utilize the rich body of work on approximation algorithms for various Steiner Tree problems in order to design *approximate* model projection algorithms as introduced in Section 2.2. In particular, algorithms for the prize collecting Steiner tree problem (PCST) turn out to be very useful.

The PCST problem is a generalization of the classical Steiner tree problem [8]. Instead of finding the cheapest tree to connect *all* terminal nodes, we can omit some terminals from the solution and pay a specific price for each omitted node. The goal is to find a subtree with the optimal trade-off between the cost paid for edges connecting a subset of the nodes and the price of the remaining, unconnected nodes. More formally, let $G = (V, E)$ be an undirected, weighted graph with edge costs $c : E \rightarrow \mathbb{R}_0^+$ and node prizes $\pi : V \rightarrow \mathbb{R}_0^+$. For a subset of edges $E' \subseteq E$, we write $c(E') = \sum_{e \in E'} c(e)$ and adopt the same convention for node subsets. Moreover, for a node subset $V' \subseteq V$, let $\overline{V'}$ be the complement $\overline{V'} = V \setminus V'$. Then the goal of the PCST problem is to find a subtree $T = (V', E')$ such that $c(E') + \pi(\overline{V'})$ is minimized (we write $c(T)$ and $\pi(T)$ if the node and edge sets are clear from context).

Building on [1], the seminal work of Goemans and Williamson [27] introduced an efficient approximation algorithm for PCST with the following guarantee:

$$c(T) + 2\pi(\overline{T}) \;\leq\; 2 \min_{T' \text{ is a tree}} c(T') + \pi(\overline{T'}) \,. \tag{4}$$

Their overall approach is usually called the Goemans-Williamson (GW) approximation scheme and originally had a time complexity of $O(|V|^2 \log |V|)$ (using the refinement for the unrooted case in [36]). Several papers improved this running time (e.g., [38, 25]), leading to the nearly-linear time variant in [13] (albeit with a slightly worse approximation guarantee).

The PCST problem already captures three important aspects of the graph sparsity model: (i) there is an underlying graph G, (ii) larger trees are penalized through the edge costs, and (iii) nodes have prizes that we wish to pick up. If we set the prizes to correspond to coefficients of our vector $x \in R^d$, i.e., $\pi(i) = x_i^2$, the term $\pi(\overline{T})$ in the PCST objective function becomes $\pi(\overline{T}) = \|x - x_T\|_2^2$, which matches the objective in the tail-approximation problem (see Definition 3).

This guarantee is already close to our desired tail approximation guarantee, but there are two important differences. First, we do not have direct control over the sparsity in our support because the PCST objective only controls the trade-off between node prizes and edge costs. Second, the objective in the PCST problem is to find a *single* tree T, while the graph sparsity model can contain supports defined by *multiple* connected components.

We can address the first issue similar to the case of tree sparsity. By setting all edges to have cost λ, we essentially transform the PCST objective into a Lagrangian relaxation of the cardinality-constrained tail-approximation problem. This gives the following approximation guarantee (now in terms of the support $S \subseteq [d]$):

$$\lambda(|S| - 1) + 2\|x - x_S\|_2^2 \;\leq\; 2 \min_{S' \subseteq [d]} \lambda(|S'| - 1) + \|x - x_{S'}\|_2^2 \;.$$

It is worth noting that the GW scheme does not enable us to solve the Lagrangian relaxation directly, but a binary search similar to the approach for the tree sparsity model still gives a constant-factor bicriterion guarantee [33] (both tail approximation error and the cardinality of the support are within constant factors of the optimum).

Furthermore, it is possible to generalize the GW scheme to the prize-collecting Steiner *forest* problem in which we optimize over a forest with a fixed number of trees as opposed to a single tree. Building on [23] and [13], the authors of [33] show that this modification of the GW scheme achieves the same approximation guarantee as Equation (4) (now minimizing over forests on the right hand side), and that this variant can still be implemented in nearly-linear time. This yields a nearly-linear time bicriterion algorithm with the tail approximation guarantee.

Open problem 3: *Is there a nearly-linear time, single-criterion tail-approximation algorithm for cluster sparsity, i.e., an algorithm that returns g connected components of total size at most s, satisfies the guarantee of Definition 3 for some constant c_T, and runs in time $O(d \log^{O(1)} d)$?*

Interestingly, the forest variant of the GW scheme can also be utilized for a head approximation algorithm. By varying the node prizes in the binary search (instead of the edge costs) and post-processing the final result to extract a "high prize-density" subgraph, the paper [33] also gives a nearly-linear time head approximation algorithm for the graph sparsity model. Combining these results yields an efficient algorithm for sparse recovery with the graph sparsity model.

The aforementioned approach yields a constant factor approximation algorithm for general graphs. However, many applications involve graphs with more specific structure, such as two-dimensional grids or, more generally, planar graphs. It is known that PCST for planar graphs can be solved in $O(d \log d)$ time with an approximation factor arbitrarily close to 1 (i.e., there is a nearly-linear time *approximation scheme*) [21, 6]. Unfortunately, the binary search approach described above yields a tail approximation algorithm with an approximation factor c_T that is strictly greater than 1.

Open problem 4: *Is there a nearly-linear time single-criterion tail-approximation algorithm for cluster sparsity for planar graphs, i.e., an algorithm that returns g connected components of total size at most s, satisfies the guarantee of Definition 3 for some constant c_T arbitrarily close to 1, and runs in time $O(d \log^{O(1)} d)$?*

5 Sparse recovery

In this section, we describe how the aforementioned algorithms for structured sparsity can be fruitfully applied in the context of sparse recovery. Formally, the *robust sparse recovery* problem is defined as follows: Given an s-sparse vector $x \in \mathbb{R}^d$, suppose we obtain $n \ll d$ linear measurements (or *sketches*) of the form $y = Ax + e$, where $A \in \mathbb{R}^{n \times d}$ denotes the measurement matrix and $e \in \mathbb{R}^n$ is the “noise" vector. Then, the goal is to efficiently obtain an estimate $\widehat{x}$ such that:

$$\left\| x - \widehat{x} \right\|_p \leq C \cdot \|e\|_p \tag{5}$$

for some approximation factor C and norm parameter p. Observe that if the noise e is assumed to be zero, then the recovered estimate $\widehat{x}$ must exactly correspond to the vector x. This general formulation addresses a large number of problems arising in compressive sensing [14, 19, 24], statistical regression [39, 53], and data stream algorithms [41]. Depending on the application, the matrix A could be given, or we might be able to design it in order to optimize the recovery process.

At first glance, recovering the unknown vector x from the measurements y appears challenging. Indeed, with no further assumptions on the measurement matrix A, the sparse recovery problem is intractable or impossible [42]. However, seminal results [19, 14] have shown that there exist classes of “good” measurement matrices A that support efficient sparse recovery in conjunction with associated recovery algorithms. In particular, these methods produce an estimate $\widehat{x}$ satisfying (5) with constant approximation factor C and number of measurements $n = O(s \log(d/s))$. Moreover, this bound on the number of measurements is asymptotically tight for constant approximation factor C [16].

The necessity of the “oversampling factor” $\log(d/s)$ is unfortunate since the logarithmic factor increases the number of measurements significantly. This increase is observed both in theory and practice. Therefore, it is a compelling question whether we can reduce the number of measurements even further, ideally down to the information-theoretic limit $n = O(s)$.

We now describe a general algorithmic framework that simultaneously addresses both of the above challenges. In particular, the running time of our recovery algorithms is *nearly-linear* in the ambient dimension d for sparsity s small enough. These constitute the first known nearly-linear time recovery schemes for multiple structured sparsity models.

The first ingredient of our approach are "good" matrices A that support recovery of structured sparse vectors. We achieve this using a popular notion known as the *restricted isometry property* (RIP). A matrix $A \in \mathbb{R}^{n\times d}$ satisfies the (s, δ)-RIP if, for all s-sparse vectors x, the following holds:

$$(1-\delta)\|x\|_2^2 \;\leq\; \|Ax\|_2^2 \;\leq\; (1+\delta)\|x\|_2^2 \;. \tag{6}$$

In most cases, δ is assumed to be a constant. An analogous condition can be defined for specific structured sparsity models. The matrix A satisfies the *model-RIP* if (6) holds for all vectors x in the sparsity model $\mathcal{M}$. It is known that there exist matrices that satisfy the model-RIP with merely $n = O(s + \log|\mathbb{M}|)$ rows [5]. (Recall that $\mathbb{M}$ is the family of allowed supports for the sparsity model $\mathcal{M}$.) So for structured sparsity models that are sufficiently concise (e.g., $|\mathbb{M}| = 2^{O(s)}$), there are indeed measurement matrices with the desired bound $n = O(s)$.

The second, more challenging aspect of our approach is to develop an *algorithmic* framework that utilizes the structure present in the unknown vector for robust sparse recovery. Here, we summarize an approach that specifically makes use of the head- and tail-approximation oracles developed in Sections 3 and 4. For full explanations, see [32] and references therein.

Suppose we wish to recover a vector $x \in \mathcal{M}$ from noisy measurements $y = Ax + e$. Moreover, assume that the recovery algorithm has access to approximate model-projection oracles $T(\cdot)$ and $H(\cdot)$ with approximation factors c_T and c_H, respectively. Then one can recover an estimate of the unknown vector x by applying the following iterative update rule:

$$x_{i+1} = T\left(x_i + H\left(A^T(y - Ax_i)\right)\right).$$

This iterative scheme is based on the Iterative Hard Thresholding (IHT) algorithm of [7]. Therefore, we call this algorithm *approximation-tolerant model-based IHT*, or AM-IHT. Extensions of other iterative algorithms for sparse recovery, such as Compressive Sensing Matching Pursuit (CoSaMP) [44] and expander iterative hard thresholding [4, 24], are also possible. See [32] for details.

Assuming that the measurement matrix satisfies the model-RIP with suitable parameters, the iterates obtained by AM-IHT exhibit *geometric convergence*:

$$\|x - x_{i+1}\|_2 \;\leq\; \alpha\|x - x_i\|_2 + \beta\|e\|_2 \;,$$

where $\alpha < 1$ and β are constants that are controlled by the matrix RIP constant δ and the approximation factors c_T, c_H of the approximate-projection oracles T and H. In other words, the estimation error decreases by a constant factor in each iteration. One can identify tradeoffs between the number of measurements, the approximation factors c_T and c_H, and the convergence ratio α. For example, keeping c_T and c_H fixed, a lower value of δ would lead to a lower value of α and this would mean that the iterates converge faster.

Due to the geometric convergence property, one can show that $O(\log \frac{\|x\|_2}{\|e\|_2})$ iterations suffice to recover a robust estimate $\widehat{x}$. The running time of each iteration of AM-IHT is determined by adding the running times of the head approximation algorithm $H(\cdot)$, the tail approximation algorithm $T(\cdot)$, and the cost of a matrix-vector multiplication with A and A^T.

Putting all the above ingredients together, we can state the following theorem, which holds for both tree sparsity and graph sparsity:

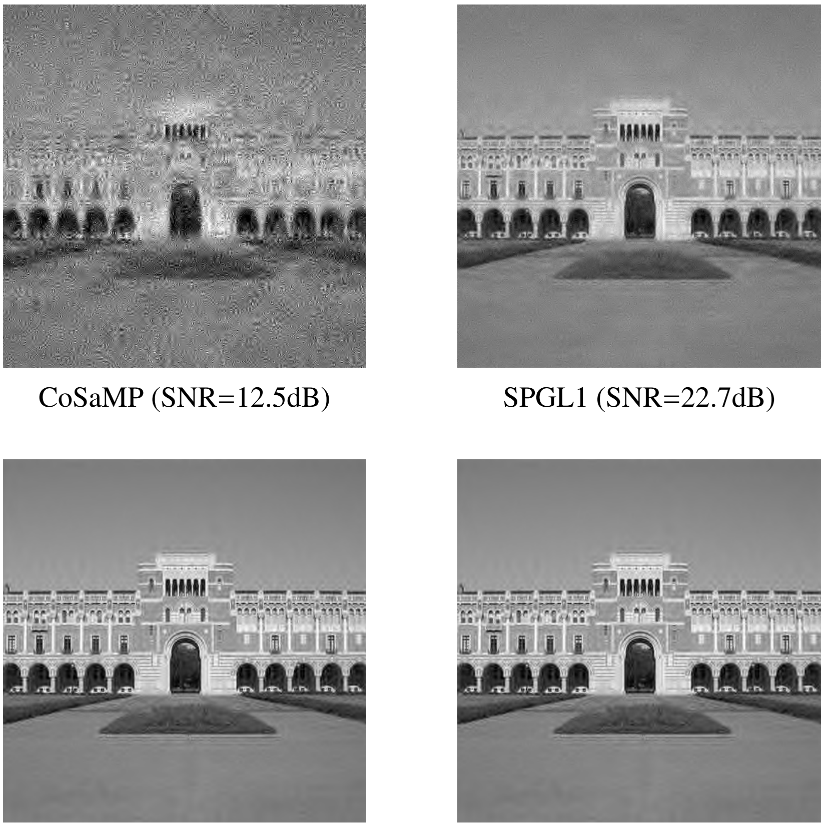
Theorem 5. *Let $A \in \mathbb{R}^{n \times d}$ be a model-RIP matrix as discussed above. Let x be an unknown s-sparse vector having a support belonging to the structured sparsity model $\mathbb{M}$. Let $y = Ax + e$ be noisy measurements of x. Then, there exists an algorithm to recover an estimate $\widehat{x}$ such that $\left\|x - \widehat{x}\right\|_2 \le C \cdot \|e\|_2$ for some fixed constant $C > 0$. The algorithm runs in time $O(d \log^{O(1)} d)$ in the range $s \le d^{1/2-\varepsilon}$ for $\varepsilon > 0$.*

We note that the theoretical guarantee in Theorem 5 is only applicable for $s \le d^{1/2-\varepsilon}$, $\varepsilon > 0$. This is because all known matrices satisfying the model-RIP with only $n = O(s)$ rows utilize fully random matrices, and therefore the matrix-vector multiplication takes at least $\Omega(s^2)$ time (see [31] for details). However, numerical simulations indicate that this limitation does not hold in practice: subsampled Fourier matrices allow matrix-vector multiplication in time $O(d \log d)$ and enable recovery of structured sparse vectors, at least in several experiments (see Subsection 5.1).

Open problem 5: *Are there model-RIP matrices for tree sparsity that have $n = O(s)$ rows and support matrix-vector multiplication in $O(d \log^{O(1)} d)$ time, for all values of s? Alternatively, does a subsampled Fourier matrix with $n = O(d \log^{O(1)} s)$ rows satisfy the model-RIP for tree sparsity?*

5.1 Experiments

We conclude this section with numerical experiments which demonstrate that utilizing structure in sparse recovery leads to an improved sample complexity. The experiments pertain to the tree-sparsity model and originally appeared in the paper



CoSaMP (SNR=12.5dB)

SPGL1 (SNR=22.7dB)

Exact tree (SNR=101.4dB)

Approximate tree (SNR=99.2dB)

Figure 5: Results of recovering an image from linear observations using various algorithms. The parameters are $d = 512 \times 512$, $s = 0.04d \approx 10{,}000$, $n = 3.3s \approx 35{,}000$). Both tree-based algorithms accurately recover the ground truth image.

[30]. We refer the reader to the original paper for a full description of the quantitative results. Analogous numerical experiments for the graph sparsity model are provided in [33].

First, we compare four algorithms on a single test case to illustrate the differences in recovery quality. All algorithms received the same number of linear observations as input; see Figure 5 for the experiment parameters (the linear observations are subsampled Fourier measurements). The four algorithms are:

- A modification of CoSaMP [44] equipped with the exact tree projection approach in [15]. This algorithm exploits structured sparsity but has a slow running time due to the exact projections (see Section 3).

- A modification of CoSaMP with the approximate tail-projection oracle for

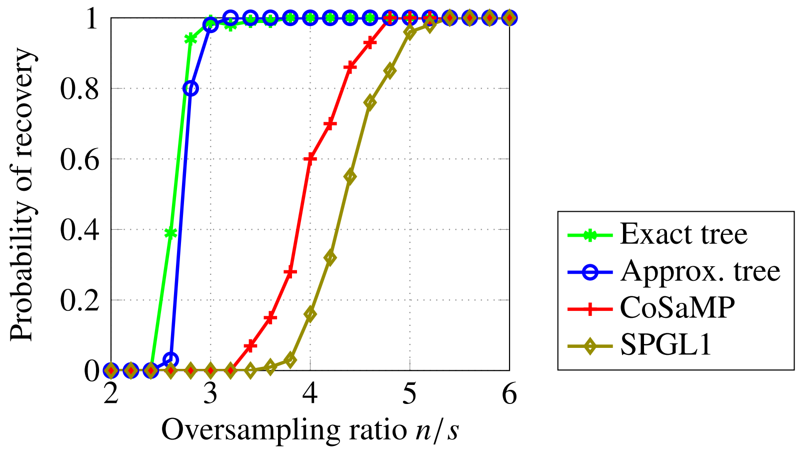


Figure 6: Comparison of sparse recovery algorithms for tree sparsity. The probability of recovery is with respect to the random measurement matrix (i.i.d. Gaussian entries) and generated using 100 trial runs.

tree-sparsity discussed in Section 3 (empirically, it suffices to use the tail algorithm in place of both the head- and tail-approximation oracle). This algorithm exploits structured sparsity and runs in nearly-linear time.

- The sparse recovery algorithm CoSaMP with “standard” s-sparse projections. This algorithm is computationally very efficient but does not utilize the tree structure present in the unknown vector.

- The popular ℓ_1-minimization approach (also called Basis Pursuit) as implemented in the software package SPGL1. This algorithm also does not utilize tree sparsity.

As predicted by the theoretical results, Figure 5 demonstrates that the methods utilizing structured sparsity achieve accurate recovery, while methods utilizing only “standard” sparsity offer worse recovery quality. For all practical purposes, the accuracy achieved by CoSaMP with approximate model projections is equivalent to that of CoSaMP with exact projections. The main benefit of the approximate projections are their better computational complexity: recall from Section 3 that a single approximate projection is 400× faster than an exact tree-projection. The gains in running time for the entire sparse recovery method (not only the projection step) are comparable.

Figure 6 explores the sample complexity of the four algorithms in more detail. For a fixed vector $x \in \mathbb{R}^d$ (unknown to the recovery algorithms), we increase the number of linear observations n available to the recovery algorithms and record the probability of success.[3] The resulting phase transitions indicate at which point (i.e., for how many linear observations) we can expect the algorithms to recover the unknown vector x reliably. In this experiment, the vector x is a discretized piecewise polynomial of dimension $d = 1024$ with sparsity $s = 41$ in a suitably chosen wavelet basis (piecewise polynomial signals are known to be approximately tree-sparse in the wavelet domain). Each data point in Figure 6 was generated by averaging over 100 sample trials using different measurement matrices from the same i.i.d. Gaussian distribution. We observe that the success probability of the approximate method almost matches the performance of CoSaMP with exact tree-projections.

6 Conclusions

In this survey, we presented an overview of structured sparsity models, associated algorithms, and their applications. In particular, we showed how approximation algorithms for combinatorial problems can be used to obtain efficient sparse recovery methods. We have also listed several open problems concerning more efficient algorithms for structured sparsity.

We note that the survey is not meant to be exhaustive. In particular, there are several other important structured sparsity models such as the Constrained EMD model [50] or the Δ-separated model [29] where combinatorial algorithms play an important role. In contrast to the tree and graph sparsity model presented here, there are currently no known nearly-linear time algorithms for these models.

Open problem 6: *Are there nearly-linear time (approximate) model projection algorithms for the EMD and Δ-separated sparsity models?*

Acknowledgments: We thank Tal Wagner for many helpful comments. This work was supported by grants from the MITEI-Shell program and the Simons Investigator Award.

References

[1] Ajit Agrawal, Philip Klein, and R Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing (STOC)*, 1991.

[3] Here, “successful recovery” is defined as the event when the ℓ_2-error of the estimate $\widehat{x}$ is within 5% of the ℓ_2-norm of the original vector x.

[2] Richard G. Baraniuk. Optimal tree approximation with wavelets. In *SPIE Wavelet Applications in Signal and Image Processing*, 1999.

[3] Marko Bohanec and Ivan Bratko. Trading accuracy for simplicity in decision trees. *Machine Learning*, 15(3):223–250, 1994.

[4] Bubacarr Bah, Luca Baldassarre, and Volkan Cevher. Model-based sketching and recovery with expanders. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1529–1543. SIAM, 2014.

[5] Richard G. Baraniuk, Volkan Cevher, Marco F. Duarte, and Chinmay Hegde. Model-based compressive sensing. *IEEE Transactions on Information Theory*, 56(4):1982–2001, 2010.

[6] MohammadHossein Bateni, Chandra Chekuri, Alina Ene, Mohammad T. Hajiaghayi, Nitish Korula, and Daniel Marx. Prize-collecting steiner problems on planar graphs. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1028–1049, 2011.

[7] Thomas Blumensath and Mike E. Davies. Iterative hard thresholding for compressive sensing. *Applied and Computational Harmonic Analysis*, 27(3):265–274, 2009.

[8] Daniel Bienstock, Michel X. Goemans, David Simchi-Levi, and David P. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59(1-3):413–420, 1993.

[9] Richard G. Baraniuk and Douglas L. Jones. A signal-dependent time-frequency representation: optimal kernel design. *IEEE Transactions on Signal Processing*, 41(4):1589–1602, 1993.

[10] Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 4(1):1–106, 2012.

[11] Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. Structured sparsity through convex optimization. *Statistical Science*, 27(4):450–468, 11 2012.

[12] `https://commons.wikimedia.org/wiki/File:Lichtenstein_img_processing_test.png`, 2007.

[13] Richard Cole, Ramesh Hariharan, Moshe Lewenstein, and Ely Porat. A faster implementation of the Goemans-Williamson clustering algorithm. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 17–25, 2001.

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

[15] Coralia Cartis and Andrew Thompson. An exact tree projection algorithm for wavelets. *IEEE Signal Processing Letters*, 20(11):1026–1029, 2013.

[16] Khanh Do Ba, Piotr Indyk, Eric Price, and David P. Woodruff. Lower bounds for sparse recovery. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1190–1197, 2010.

[17] Marco F. Duarte and Yonina C. Eldar. Structured compressed sensing: From theory to applications. *IEEE Transactions on Signal Processing*, 59(9):4053–4085, 2011.

[18] David L. Donoho. Cart and best-ortho-basis: a connection. *Annals of Statistics*, 25(5):1870–1911, 1997.

[19] David L. Donoho. Compressed sensing. *, IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[20] Marwa El Halabi and Volkan Cevher. A totally unimodular view of structured sparsity. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.

[21] David Eisenstat, Philip Klein, and Claire Mathieu. An efficient polynomial-time approximation scheme for steiner forest in planar graphs. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 626–638. SIAM, 2012.

[22] Yonina Eldar and Moshe Mishali. Robust recovery of signals from a structured union of subspaces. *IEEE Transactions on Information Theory*, 55(11):5302–5316, 2009.

[23] Paulo Feofiloff, Cristina G. Fernandes, Carlos E. Ferreira, and José Coelho de Pina. A note on Johnson, Minkoff and Phillips' algorithm for the prize-collecting Steiner tree problem. *Computing Research Repository (CoRR)*, abs/1004.1437, 2010.

[24] Simon Foucart and Holger Rauhut. *A Mathematical Introduction to Compressive Sensing*. Springer, 2013.

[25] Harold N. Gabow, Michel X. Goemans, and David P. Williamson. An efficient approximation algorithm for the survivable network design problem. *Mathematical Programming*, 82(1-2):13–40, 1998.

[26] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[27] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[28] Lihan He and Lawrence Carin. Exploiting structure in wavelet-based bayesian compressive sensing. *IEEE Transactions on Signal Processing*, 57(9):3488–3497, 2009.

[29] Chinmay Hegde, Marco F Duarte, and Volkan Cevher. Compressive sensing recovery of spike trains using a structured sparsity model. In *SPARS'09-Signal Processing with Adaptive Sparse Structured Representations*, 2009.

[30] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. A fast approximation algorithm for tree-sparse recovery. In *International Symposium on Information Theory (ISIT)*, 2014.

[31] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. Nearly linear-time model-based compressive sensing. In *Automata, Languages, and Programming (ICALP)*, volume 8572 of *Lecture Notes in Computer Science*, pages 588–599. 2014.

[32] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. Approximation algorithms for model-based compressive sensing. *IEEE Transactions on Information Theory*, 61(9):5129–5147, 2015. Conference version appeard in the *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2014).

[33] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. A nearly-linear time framework for graph-structured sparsity. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 928–937. JMLR Workshop and Conference Proceedings, 2015.

[34] `http://www.nasa.gov/mission_pages/hubble/science/xdf.html`, 2012.

[35] Junzhou Huang, Tong Zhang, and Dimitris Metaxas. Learning with structured sparsity. *The Journal of Machine Learning Research*, 12:3371–3412, 2011.

[36] David S. Johnson, Maria Minkoff, and Steven Phillips. The prize collecting Steiner tree problem: Theory and practice. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 760–769, 2000.

[37] Laurent Jacob, Guillaume Obozinski, and Jean-Philippe Vert. Group Lasso with overlap and graph Lasso. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pages 433–440, 2009.

[38] Philip Klein. A data structure for bicategories, with application to speeding up an approximation algorithm. *Information Processing Letters*, 52(6):303–307, 1994.

[39] Seyoung Kim and Eric P. Xing. Tree-guided group Lasso for multi-task regression with structured sparsity. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 543–550, 2010.

[40] Julien Mairal, Rodolphe Jenatton, Guillaume Obozinski, and Francis Bach. Convex and network flow optimization for structured sparsity. *The Journal of Machine Learning Research*, 12:2681–2720, 2011.

[41] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 2005.

[42] Balas Kausik Natarajan. Sparse approximate solutions to linear systems. *SIAM journal on computing*, 24(2):227–234, 1995.

[43] Sahand N. Negahban, Pradeep Ravikumar, Martin J. Wainwright, and Bin Yu. A unified framework for high-dimensional analysis of m-estimators with decomposable regularizers. *Statistical Science*, 27(4):538–557, 11 2012.

[44] Deanna Needell and Joel A. Tropp. CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3):301–321, 2009.

[45] Polina Rozenshtein, Aris Anagnostopoulos, Aristides Gionis, and Nikolaj Tatti. Event detection in activity networks. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1176–1185, 2014.

[46] Nikhil S. Rao, Ben Recht, and Robert D. Nowak. Universal measurement bounds for structured sparse signal recovery. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 22 of *JMLR Proceedings*, pages 942–950, 2012.

[47] Siddhartha Satpathi, Luca Baldassarre, and Volkan Cevher. Sparse group covers and greedy tree approximations. In *IEEE Internation Symposium on Information Theory (ISIT)*, 2015.

[48] Noah Simon, Jerome Friedman, Trevor Hastie, and Robert Tibshirani. A sparse-group Lasso. *Journal of Computational and Graphical Statistics*, 22(2):231–245, 2013.

[49] Jerome M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, 1993.

[50] Ludwig Schmidt, Chinmay Hegde, and Piotr Indyk. The constrained Earth Mover Distance model, with applications to compressive sensing. In *International Conference on Sampling Theory and Applications (SAMPTA)*, 2013.

[51] Ludwig Schmidt, Chinmay Hegde, Piotr Indyk, Jonathan Kane, Ligang Lu, and Detlef Hohl. Automatic fault localization using the generalized Earth Mover's Distance. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014.

[52] Martin J. Wainwright. Structured regularizers for high-dimensional problems: Statistical and computational issues. *Annual Review of Statistics and Its Application*, 1(1):233–253, 2014.

[53] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.