Sparse Group Covers and Greedy Tree Approximations

Siddhartha Satpathi
Indian Institute of Technology
Kharagpur - India
sidd.piku@gmail.com

Luca Baldassarre
LIONS - EPFL
Lausanne - Switzerland
luca.baldassarre@epfl.ch

Volkan Cevher
LIONS - EPFL
Lausanne - Switzerland
volkan.cevher@epfl.ch

***Abstract*—We consider the problem of finding a K-sparse approximation of a signal, such that the support of the approximation is the union of sets from a given collection, a.k.a. group structure. This problem subsumes the one of finding K-sparse tree approximations. We discuss the tractability of this problem, present a polynomial-time dynamic program for special group structures and propose two novel greedy algorithms with efficient implementations. The first is based on submodular function maximization with knapsack constraints. For the case of tree sparsity, its approximation ratio of $1-1/e$ is better than current state-of-the-art approximate algorithms. The second algorithm leverages ideas from the greedy algorithm for the Budgeted Maximum Coverage problem and obtains excellent empirical performance, shown by computing the full Pareto frontier of the tree approximations of the wavelet coefficients of an image.**

I. Introduction

Leveraging structure in signals has become paramount in many fields of signal processing, from compressive sensing to machine learning and from graph sketching to denoising. Many structures can be described by a set of index sets, or groups, of variables that should either be selected or discarded together. These group structures encompass models such as the block model, the overlapping group model and the hierarchical model, which have been extensively studied in the recent years, both from theoretical and practical perspectives [1]–[10].

The fundamental discrete problem common to these models is that of finding the best approximation of a signal in a given norm, subject to the support of the approximation be covered by G groups. This problem is in general NP-hard, but there exist special cases, such as the block model [7] and the acyclic overlapping group model [5], which can be solved exactly in polynomial time. Furthermore, convex relaxations such as the group lasso [1], [2] and the latent group lasso [3], [4] provide tractable proxies to the discrete problem, with the disadvantage though of not being able to obtain the entire solution set [5].

An extension to the basic group model is to allow to select only few variables from the active groups, to allow for so called within-group sparsity. This has led to the sparse group lasso norm [11] and also to the general dynamic program for acyclic group structures that finds a K-sparse solution covered by at most G-groups [5].

In some applications, however, the groups might be very heterogenous in size with potentially large overlaps, such as pathways of genes for microarray data analysis [12], rendering the specification of a group budget meaningless. Furthermore, it may be important to select all variables within a group, but define an overall sparsity budget. Interestingly, this model can be used to model hierarchical sparsity, where the components of a signal are arranged on a tree and we aim at finding an approximation whose support is a rooted connected subtree.

Two exact dynamic programs have been concurrently proposed recently for obtaining tree approximations with the same complexity [5], [6]. The hierarchical group lasso [8] norm enforces the same structure via a convex relaxation, with no direct control on the sparsity of the solution. An approximate greedy algorithm was proposed in 1999 by Baraniuk [13] and more recently, approximate algorithms [14] have gained interest in order to achieve nearly linear time reconstruction algorithms for compressed sensing [15].

In this paper, we first formulate the K-sparse group-cover problem as a submodular maximization problem with submodular constraints for which a greedy algorithm with guaranteed approximation factor was recently proposed [16]. We then propose a novel greedy algorithm, which does not yet have approximation guarantees, but whose empirical performance for tree approximation is very competitive. We also present a dynamic program for obtaining exact solutions for special group structures, in a vein similar to [5]. Specializing our results to tree sparsity, we first show that the approximation factor of the greedy algorithm is $1-1/e$, which is much better than the factor of $1/4$ of the greedy algorithm presented in [15]. We then offer an efficient implementation of both greedy algorithms and analyze their time and space complexities. Finally, we illustrate the performance of the novel greedy algorithm by computing the Pareto frontier of the tree approximations of the wavelet coefficient of an image and show that both the relaxation algorithm of [14] and the greedy approach of [15] fail to obtain all approximations.

II. Foundations

A. Notation and basics

Given a universe $\mathcal{U}$, a set function $f : 2^{\mathcal{U}} \to \mathbb{R}$ is submodular if for all subsets $\mathcal{S}, \mathcal{V}$ of $\mathcal{U}$, it holds that $f(\mathcal{S}) + f(\mathcal{V}) \geq f(\mathcal{S} \cup \mathcal{V}) + f(\mathcal{S} \cap \mathcal{V})$. We define $f(j|\mathcal{S}) \triangleq f(\mathcal{S} \cup j) - f(\mathcal{S})$ as the gain of adding the element $j \in \mathcal{U}$ to $\mathcal{S}$. The function is monotone iff $f(j|\mathcal{S}) \geq 0 \ \forall j \notin \mathcal{S}, \mathcal{S} \subseteq \mathcal{U}$. A normalized submodular function is such that $f(\emptyset) = 0$. The total curvature

of f is defined as

$$\kappa_f = 1 - \min_{j \in \mathcal{U}} \frac{f(j|\mathcal{U} \setminus j)}{f(j)} \ . \tag{1}$$

In this paper, we consider signals $\mathbf{x} \in \mathbb{R}^N$, whose components are indexed by the ground set $[N] := \{1, \ldots, N\}$. A group structure $\mathfrak{G} := \{\mathcal{G}_1, \ldots, \mathcal{G}_M\}$ over the ground set is a collection of sets, or groups, $\mathcal{G} \subseteq [N]$ such that $\bigcup_{\mathcal{G} \in \mathfrak{G}} \mathcal{G} = [N]$. The intersection graph of $\mathfrak{G}$ has groups as nodes and edges between nodes whose intersection is non-empty. We call a group structure acyclic if its intersection graph is acyclic.

We define the "neighborhood" of a set of groups $\mathcal{S} \subseteq \mathfrak{G}$, $\mathcal{N}(\mathcal{S})$, as the variables covered by the groups in $\mathcal{S}$, $\mathcal{N}(\mathcal{S}) = \bigcup_{\mathcal{G} \in \mathcal{S}} \mathcal{G}$. Given a vector of non-negative weights $\mathbf{w} \in \mathbb{R}^N$, the "weight" of $\mathcal{S}$ is the sum of the weights of the elements in $\mathcal{N}(\mathcal{S})$, $W(\mathcal{S}) = \sum_{i \in \mathcal{N}(\mathcal{S})} w_i$. Note that both $W(\mathcal{S})$ and $|\mathcal{N}(\mathcal{S})|$ are submodular functions since the groups are allowed to overlap arbitrarily. Without overlaps, both $W(\mathcal{S})$ and $|\mathcal{N}(\mathcal{S})|$ are modular.

B. The K-sparse group-cover problem

Finding the best K-sparse group cover for a signal $\mathbf{x} \in \mathbb{R}^N$ can be formulated as the following problem

$$\max_{\mathcal{S} \in \mathfrak{G}} \{W(\mathcal{S}) : |\mathcal{N}(\mathcal{S})| \leq K\} . \tag{2}$$

where the weights w_i are given by $(x_i)^p$ for $p \geq 0$. This problem can be solved exactly via dynamic programming for acyclic group structures with the algorithm described in Section III, whose complexity is $\mathcal{O}(M^2 K)$.

Next, we establish a connection between the K-sparse group-cover problem with recent results in submodular optimization, in order to adapt a greedy algorithm with approximation guarantees. The problem of maximizing a submodular function subject to a submodular upper bound, or knapsack constraint, has been recently studied in [16] where they name it Submodular Cost Submodular Knapsack (SCSK) problem:

$$\max_{\mathcal{S}} \{g(\mathcal{S}) : f(\mathcal{S}) \leq b\} , \tag{3}$$

where both f and g are monotone non-decreasing and normalized submodular functions and $b > 0$ is the knapsack budget. We say that an algorithm for solving (3) has an approximation factor of $0 \leq \sigma \leq 1$, if it is guaranteed to obtain a set $\widehat{\mathcal{S}}$, such that $g(\widehat{\mathcal{S}}) \geq \sigma g(\mathcal{S}^*)$ and $f(\widehat{\mathcal{S}}) \leq b$, where $\mathcal{S}^*$ is an optimal solution of (3). Specifically, Iyer and Bilmes [16] establish that SCSK (3) is NP-hard and provide a greedy algorithm whose approximation factor depends, among other things, on the total curvature of f and g.

At each step i, the algorithm adds to the current selection $\mathcal{S}_i$ the element that maximizes the gain of g, while still satisfying the knapsack constraint:

$$\operatorname{argmax}\{g(j|\mathcal{S}_{i-1}) | j \notin \mathcal{S}_{i-1}, f(\mathcal{S}_{i-1} \cup j) \leq b\}. \tag{4}$$

In Section V-A, we propose an efficient implementation of the greedy algorithm for hierarchical sparsity on D-regular trees, whose time complexity is $\mathcal{O}(N \min(\log_D N, K))$. The approximation factor of the greedy algorithm is given by the following theorem.

Theorem 1 ([16]). *The greedy algorithm for SCSK obtains an approximation factor of*

$$\frac{1}{\kappa_g}\left(1 - \left(\frac{C_f - \kappa_f}{C_f}\right)^{c_f}\right) \geq \frac{1}{C_f},$$

where

$$C_f = \max\{|\mathcal{S}| : f(\mathcal{S}) \leq b\},$$

$$c_f = \min\{|\mathcal{S}| : f(\mathcal{S}) \leq b, \forall j \in \mathcal{U} \setminus \mathcal{S}, f(\mathcal{S} \cup j) > b\}.$$

C. A new greedy algorithm

We can observe from algorithm (4) that the selection criteria only maximizes the marginal gain $g(j|\mathcal{S}_{i-1}), \forall j \in [N]$, and is independent of the constraint function f. Since our objective is to maximize g while keeping f as low as possible, we propose a greedy algorithm which involves maximizing the ratio of the marginal gains in g and f. At each iteration i, we add an element corresponding to

$$\operatorname*{argmax}_j \left\{ \frac{g(j|\mathcal{S}_{i-1})}{f(j|\mathcal{S}_{i-1})} : j \notin \mathcal{S}_{i-1}, f(\mathcal{S}_{i-1} \cup j) \leq b \right\}. \tag{5}$$

This is a natural extension of the greedy algorithm for the budgeted maximum coverage problem [17], where f is modular. In Section V-B, we provide an efficient implementation and complexity analysis of algorithm (5) on D-regular trees. While we are still unable to provide an approximation factor for this algorithm, its empirical performance is superior to algorithm (4), as shown in the Pareto frontier example of Section VI.

D. Relaxation

Instead of dealing with the sparsity constraint K in (2), it is possible to relax it into a penalty term with parameter λ, to obtain the relaxed problem

$$\max_{\mathcal{S} \in \mathfrak{G}} \{W(\mathcal{S}) - \lambda |\mathcal{N}(\mathcal{S})|\} . \tag{6}$$

Let $\psi_i = W(\mathcal{G}_i)$ and $\psi_{ij} = W(G_i \cap G_j)$. Also define $s_i = |G_i|$ and $s_{ij} = |G_i \cap G_j|$. We then have that the relaxed problem can be rewritten as

$$\max_{\boldsymbol{\omega} \in \mathbb{B}^M} \sum_{i=1}^{M} (\psi_i - \lambda s_i)\omega_i - \sum_{i,j=1}^{M} (\psi_{ij} - \lambda s_{ij})\omega_i \omega_j, \tag{7}$$

where $\boldsymbol{\omega}$ is a M-dimensional binary variable. If the group structure is acyclic, then the above problem can be solved via the sum-product algorithm in $\mathcal{O}(M)$ time [18].

III. Dynamical Programming

The K-sparse group-cover problem can solved exactly via dynamic programming for acyclic group structures with an algorithm which is inspired by the one proposed in [5].

We first describe the optimal substructure of the problem that can be exploited by dynamic programming and which can be easily proved by contradiction. Suppose we have an optimal selection, $\mathcal{S}$, of groups whose neighborhood $\mathcal{N}(\mathcal{S})$

has cardinality less than, or equal to, K. Let now partition $\mathcal{S}$ in two sets, $\mathcal{S}_1$ and $\mathcal{S}_2$, and suppose that $\mathcal{S}_1$ contains the groups in the optimal solution such that the cardinality of their neighborhood is less than, or equal to, K_1. Then, the groups in $\mathcal{S}_2$ are the optimal selection of groups for a budget $K - K_1$ from $\mathfrak{G} \setminus \mathcal{S}_1$ after the elements in $\mathcal{N}(\mathcal{S}_1)$ have been removed.

The algorithm explores every node in the acyclic intersection graph, keeping a table of the best solutions found among the visited nodes. Two rules define its behavior: the **Node Picking Rule** determines the exploration strategy in order to minimize the size of the table, while the **Table Update Rule** describes how the table is updated when a new node is considered. We let an arbitrary node be the root node.

Suppose we have explored m out of the total M nodes. We store the best possible values that can be obtained with a sparsity budget of k from the currently explored set of nodes, for $1 \leq k \leq K$. We define a **boundary node** as an explored node adjacent to an unexplored node. Due to the possible overlaps, we must store all values separately for each possible selection of boundary nodes. We expand the set of explored nodes one new node at a time and duplicate the table of stored values, for the cases that the new node is included in the optimal solution or not. We then update both of these tables according to the following table update rules.

1) *New node is not included.* The table remains unchanged.
2) *New node is included and it does not overlap with any boundary node.* Let the cardinality of the new group be k_g and the cardinality of the union of the selected boundary nodes be k_b. Then for $1 \leq k < k' := k_g + k_b$, the new values are all equal to zero, since the budget does not actually allow to select the new node and the already selected boundary nodes. For $k' \leq k \leq K$, the new value is given by the sum of the weight of the new node and the previous optimal value for budget $k - k'$.
3) *New node is accepted, but it overlaps with some boundary nodes*. The update rule is the same as for case 2, but we first need to "clean" the new node from the elements in the overlap with the selected boundary nodes, so that both its weight and cardinality are reduced.

After these steps, we can condense the new tables: for each boundary node which has fallen into the interior of the explored nodes, we combine the optimal values for it being selected or not, by taking the larger of the 2 values.

Let B be the maximum number of boundary nodes encountered by the algorithm, then the number of steps is bounded by $O(2^B MK)$. We can now use the Node Picking Rule described in [5] in order to bound the number of boundary nodes by $\mathcal{O}(\log M)$, so that the total complexity of the dynamic program is $\mathcal{O}(M^2 K)$.

IV. Hierarchical sparsity

An important structured sparsity model that finds common use in imaging and machine learning, among other fields, is hierarchical sparsity. In this model, the components of a signal are arranged on a tree and one wants to find an approximation whose support is a rooted connected (RC) subtree of the original tree. The wavelet decomposition coefficients of 2D images, for instance, can be naturally organized on three regular trees of degree four, corresponding to multi-scale analysis along the vertical, horizontal and diagonal directions [19]. Furthermore, natural images yield sparse coefficients that form a RC subtree of the wavelet tree and which can be exploited to yield better approximations or reconstruction, both with discrete and convex methods [5], [8], [14], [20], [21].

Hierarchical sparsity can be modeled as a K-sparse group-cover problem, where the groups consist of a node and all its ancestors. This group structure naturally leads to selections of variables that form a RC subtree. Let $W(\mathcal{S}) := g$ and $\mathcal{N}(\mathcal{S}) := f$, then we have the following constants

$$\kappa_g = 1, \quad \kappa_f = 1, \quad C_f = K, \quad c_f = K \ . \tag{8}$$

In [5] and [6], two similar dynamic program were proposed to solve this problem exactly for regular trees, with complexity $\mathcal{O}(NKD)$, where D is the tree degree.

Our implementation of the greedy algorithm (4), described in Section V-A, has complexity $\mathcal{O}(N \min(\log_D N, K))$ and the following approximation guarantee

$$\left(1 - \left(\frac{K-1}{K}\right)^K\right) \geq 1 - 1/e \ .$$

In comparison, the greedy algorithm of [15] has time complexity $\mathcal{O}(N \log_D N + K \log_D^2 N)$ and an approximation guarantee of $\frac{1}{4}$. However, the returned support is guaranteed to be only a subset of a RC subtree with sparsity $K(2D+2)$.

In the next section, we show that the new greedy algorithm (5) has time complexity $\mathcal{O}(N \log_D N)$ for D-regular trees. Although it comes with no approximation guarantees, its empirical performance is very competitive, see Section VI.

V. Efficient implementation and complexity analysis of the greedy algorithms for hierarchical sparsity

In this section, we present the analysis of algorithm (4) and (5) for groups defined under hierarchical sparsity. A naïve implementation of algorithm (4) or (5) would require to calculate the marginal gains of the submodular functions f and g for every element at every iteration, with time complexity $\mathcal{O}(NK)$. In our implementation, we do efficient an pre-processing to calculate all marginal gains of f and g, and then bound the run time complexity to be linear in N or K. We limit our discussion to regular trees where each non-leaf node has D children, however, our algorithm and analysis can be easily extended to non-regular trees with bounded degree.

Let the nodes of the tree be numbered in breadth first ordering form 1 to N. We recall that we define a group $\mathcal{G}_i$ for each node $i \in [N]$ containing the node and all its ancestors. Further, let $\mathcal{D}_i$ denote the set of descendants of node i, p_i denote its parent and $\mathcal{C}_i$ the set of its children. For example, in a 2-regular tree with $N = 7$, $\mathcal{G}_0 = \emptyset, \mathcal{G}_1 = \{1\}, \mathcal{G}_2 = \{1, 2\}$, $G_3 = \{1, 3\}, \mathcal{G}_4 = \{1, 2, 4\}$; $\mathcal{D}_1 = \{1, 2, \ldots, 7\}, \mathcal{D}_2 = \{2, 4, 5\}$, $\mathcal{D}_5 = \emptyset$; $p_1 = 0, p_2 = 1, p_3 = 1$ and $\mathcal{C}_1 = \{2, 3\}$.

A. Implementation and complexity analysis of algorithm (4)

At first, we initialize a variable $\{m_i, n_i\}$ with $n_i = \text{argmax}_{j \in \mathcal{D}_i} W(\mathcal{G}_j \setminus \mathcal{G}_{p_i})$ and $m_i = W(\mathcal{G}_{n_i} \setminus \mathcal{G}_{p_i})$. Note that n_i always corresponds to a leaf node of the subtree that has root i. We can compute m_i recursively as $m_i = \max_{j \in \mathcal{C}_i} (m_j + W(\{i\}))$. Similarly, we can also form a recursive relation to compute n_i. Hence, we can compute $\{m_i, n_i\},\ \forall i \in [N]$ by a single sweep of i from N to 1. Thus, computing all the values for $\{m_i, n_i\}$ requires $\mathcal{O}(N)$ time complexity and $\mathcal{O}(N)$ space complexity.

We then proceed to run algorithm (4). Let us assume that the algorithm runs for ℓ iterations and let us denote the iterations as $i_0, i_1, \ldots, i_\ell$. Let $\mathcal{S}_j$ denote the groups selected until the i_j-th iteration. We break the analysis of the algorithm into two cases. The first involves all iterations until $i_k, k \leq \ell$, such that $k = \text{argmax}_i \{f(\mathcal{S}_i \cup \mathcal{G}_j) \leq K,\ \forall \mathcal{G}_j \notin \mathcal{S}_i\}$ and the second case computes the complexity from iteration i_{k+1} to i_ℓ. Basically, i_k denotes the last iteration for which the addition of any group $\mathcal{G}_j$, which is not already selected, does not violate the budget constraint.

a) Case 1: (Up to iteration i_k) In any iteration $i_j \leq i_k$, let $\mathcal{B}$ denote the set of uncovered nodes which are at the boundary of the selected nodes. For instance, if $\mathcal{S}_1 = \mathcal{G}_1$, then at the end of the first iteration $\mathcal{B} = \mathcal{C}_1$. More formally, at the end of iteration i_j, we have $\mathcal{B} = \left(\cup_{q \in \mathcal{N}(\mathcal{S}_j)} \mathcal{C}_q\right) \setminus \mathcal{N}(\mathcal{S}_j)$. Up to iteration i_k, we maintain a priority queue of $\mathcal{B}$ with priority value m_i of node i. For simpler exposition, we explain the functioning of the algorithm from iteration i_0. The first group to be selected is $\mathcal{G}_{n_1}$. Hence, we push the nodes in $\left(\cup_{j \in \mathcal{N}(\mathcal{G}_{n_1})} \mathcal{C}_j\right) \setminus \mathcal{N}(\mathcal{G}_{n_1})$ into the priority queue $\mathcal{B}$. Then, we extract the highest priority element from $\mathcal{B}$. Let it be node j with priority value m_j. Hence, it is easy to observe that, in iteration i_2, group $\mathcal{G}_{n_j}$ gets selected. We continue the algorithm in this manner until iteration i_k.

For every covered node $q \in \mathcal{N}(\mathcal{S}_k)$, we perform at max $|\mathcal{C}_q| \leq D$ insertions into the priority queue, so $|\mathcal{B}| \leq D|\mathcal{N}(\mathcal{S}_k)|$. Also, since we perform one extract-max operation per iteration, we can upper bound the number of extract-max operations from $\mathcal{B}$ as $i_k \leq |\mathcal{N}(\mathcal{S}_k)|$. Hence, the total complexity is $(D+1)|\mathcal{N}(\mathcal{S}_k)|$ insertions or extract-max operations. Each insertion or extract-max operation from the priority queue $\mathcal{B}$, implemented as binary heap, can take maximum $O(\log_2 |\mathcal{B}|))$ time. With $\mathcal{N}(\mathcal{S}_k) \leq K$, we can upper bound the complexity till iteration i_k as,

$$\mathcal{O}((D+1)|\mathcal{N}(\mathcal{S}_k)| \log_2 |\mathcal{B}|)) \leq \mathcal{O}(DK \log_2(DK)). \quad (9)$$

b) Case 2: (From iteration i_{k+1} to i_ℓ) We perform a naïve implementation of algorithm (4) after the i_k-th iteration without a priority queue. Basically, in every iteration, we compute $\{m_i, n_i\}$ for all boundary nodes in $\mathcal{B}$ and select the group $\mathcal{G}_{n_i}$ corresponding to the maximum value m_i. In any iteration i_q, $\{m_i, n_i\}$ follows the changed definitions, $n_i = \text{argmax}_{j \in \mathcal{D}_i} \{W(\mathcal{G}_j \setminus \mathcal{G}_{p_i}) : |\mathcal{N}(\mathcal{S}_q)| + |\mathcal{G}_j \setminus \mathcal{G}_{p_i}| \leq K\}$ and $m_i = W(\mathcal{G}_{n_i} \setminus \mathcal{G}_{p_i})$. It is easy to observe that in every iteration, computing $\{m_i, n_i\},\ \forall i \in \mathcal{B}$ and selecting the maximum among them, takes at max $\mathcal{O}(N)$ time. Hence the total complexity from iteration i_{k+1} to i_ℓ, can be stated as $\mathcal{O}(N(\ell - k))$. From the definition of i_k, we can say that $|\mathcal{N}(\mathcal{S}_k)| \geq \max(K - h, 0)$, where $h = \mathcal{O}(\log_D N)$ is the height of the tree. Hence $\ell - k \leq K - |\mathcal{N}(\mathcal{S}_k)| \leq \min(h, K)$. Thus, we can upper bound the complexity as $\mathcal{O}(N \min(\log_D N, K))$.

Finally, form *Case 1* and *Case 2*, we can state that algorithm (4) takes total $\mathcal{O}(DK \log_2(DK) + N \min(\log_D N, K))$ time and $\mathcal{O}(N)$ space for a D-regular tree.

B. Implementation and complexity analysis of algorithm (5)

Now, we extend our implementation of algorithm (4) to algorithm (5) for D-regular trees. We initialize $\{m_i, n_i\}$ as follows: $n_i = \text{argmax}_{j \in \mathcal{D}_i} \frac{W(\mathcal{G}_j \setminus \mathcal{G}_{p_i})}{|\mathcal{G}_j \setminus \mathcal{G}_{p_i}|}$ and $m_i = \frac{W(\mathcal{G}_{n_i} \setminus \mathcal{G}_{p_i})}{|\mathcal{G}_{n_i} \setminus \mathcal{G}_{p_i}|}$. This time, computing $\{m_i, n_i\}$ is more involved. We first compute and store the marginal gains $\frac{W(\mathcal{G}_j \setminus \mathcal{G}_i)}{|\mathcal{G}_j \setminus \mathcal{G}_i|}\ \forall i \in \mathcal{G}_j \setminus \{j\}$ for every value of $j \in [N]$.

Computing each of $\{m_i, n_i\}$ takes a maximum of $\mathcal{O}(|\mathcal{D}_i|)$ time. Hence, the total time to find $\{m_i, n_i\}$ for every value of i is $\mathcal{O}(\sum_{i \in [N]} |\mathcal{D}_i|)$. Let the maximum depth of the tree be J (depth of *root* node being 1). Therefore, $|\mathcal{D}_i| = 1 + D^2 + \cdots + D^{J-j} = \frac{D^{J-j+1}-1}{D-1}$ for depth of i being j. Hence, we can write $\mathcal{O}(\sum_{i \in [N]} |\mathcal{D}_i|) \overset{(a)}{=} \mathcal{O}\left(\sum_{j=1}^{J} D^{j-1} \frac{D^{J-j+1}-1}{D-1}\right) \leq \mathcal{O}\left(\sum_{j=1}^{J} \frac{D^J-1}{D-1}\right) = \mathcal{O}\left(|\mathcal{D}_1| J\right) = \mathcal{O}\left(N \log_D N\right)$, where (a) follows since there are D^{j-1} nodes at a depth of j.

Having computed $\{m_i, n_i\}$, we proceed to run algorithm (5) in a similar fashion as algorithm (4). From iteration i_0 to i_k, we can prove that algorithm (5) takes $\mathcal{O}(DK \log_2(DK))$ time. From iteration i_{k+1} to i_ℓ, we recompute $\{m_i, n_i\}$ in every iteration i_q as $n_i = \text{argmax}_{j \in \mathcal{D}_i} \left\{ \frac{W(\mathcal{G}_j \setminus \mathcal{G}_{p_i})}{|\mathcal{G}_j \setminus \mathcal{G}_{p_i}|} : |\mathcal{N}(\mathcal{S}_q)| + |\mathcal{G}_j \setminus \mathcal{G}_{p_i}| \leq K \right\}$ and $m_i = \frac{W(\mathcal{G}_{n_i} \setminus \mathcal{G}_{p_i})}{|\mathcal{G}_{n_i} \setminus \mathcal{G}_{p_i}|}$ for every node $i \in \mathcal{B}$. Note that this takes a maximum of $\mathcal{O}(\sum_{i \in \mathcal{B}} |\mathcal{D}_i|)$ time. Since $\bigcap_{i \in \mathcal{B}} \mathcal{D}_i = \emptyset$, we can write $\mathcal{O}(\sum_{i \in \mathcal{B}} |\mathcal{D}_i|) = \mathcal{O}(N)$. Thus, the time complexity of algorithm (5) from iteration i_{k+1} to i_ℓ is $\mathcal{O}(N(\ell - k))$, which can be reduced to $\mathcal{O}(N \min(\log_D N, K))$ as proved in analysis of algorithm (4).

Finally, algorithm (5) has $\mathcal{O}(DK \log_2(DK) + N \log_D N)$ time and $\mathcal{O}(N \log_D N)$ space complexities.

VI. Numerical results

In order to illustrate the performance of the proposed greedy algorithms, we consider the problem of finding tree approximations of the wavelet coefficients of a 2D image. We first run the exact dynamic program, `ETP`, proposed in [5], to obtain the full Pareto frontier of the problem, that is the maximum achievable weight for every sparsity budget $1 \leq K \leq N$. We then assess the performance of the two greedy algorithms (4) and (5), `Greedy 1` and `Greedy 2`, comparing against the `HeadApprox` greedy algorithm proposed in [15], the bisection algorithm that solves the relaxed problem, `Relax` [14], and Baraniuk's greedy algorithm, `Greedy Bar` [13].

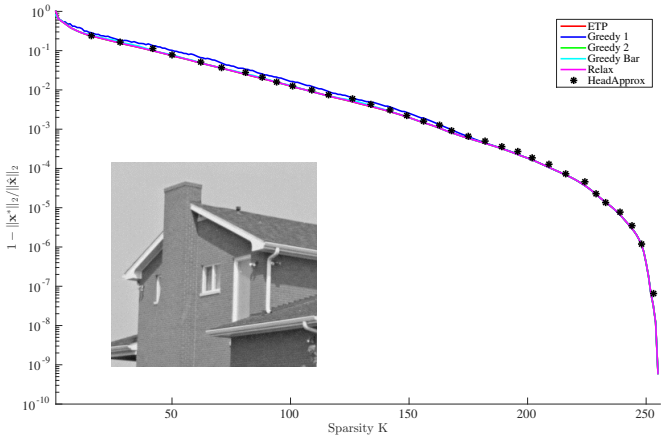

Fig. 1. Pareto frontier of tree approximations of the wavelet coefficients of the house image (inset). The `ETP`, `Greedy 2` and `Relax` solutions are almost identical and overlap in the plot.

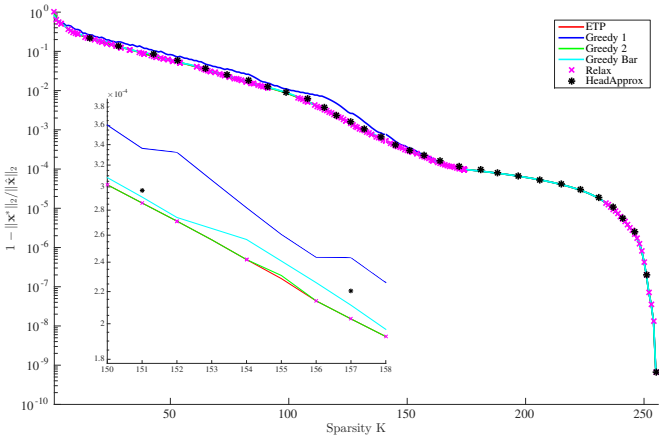

Fig. 2. Pareto frontier of tree approximations of the discretized wavelet coefficients of the house image. The `ETP` and `Greedy 2` solutions are almost identical and overlap in the plot, while the `Relax` solutions do not cover the entire spectrum of sparsity budgets. (inset) Zoom on $K = 151, \ldots, 158$.

We take the `house` image, resized to 16×16 pixels, subtract its mean, and compute its Daubechies-4 wavelet coefficients, which are arranged onto three regular quad-trees. We use this structure to obtain sparse rooted connected approximations.

The results are illustrated in Figure 1, where we plot $1 - \|\mathbf{x}^*\|_2 / \|\hat{\mathbf{x}}\|_2$ with respect to K, with $\mathbf{x}^*$ being the original wavelet coefficients and $\hat{\mathbf{x}}$ the obtained K-sparse approximation. For `HeadApprox`, we set $\alpha = \lceil \log_D N \rceil$ and report the value for the sparsity of the returned solution, since the algorithm is not guaranteed to return an exactly K sparse solution. Furthermore, its solutions do not cover the entire spectrum of desired sparsities. `Greedy 1` is able to achieve each sparsity budget, but yields worse solutions than `HeadApprox`, despite having a better theoretical approximation factor. The new `Greedy 2` and `Relax` algorithms perform equally well and yield solutions that are almost optimal.

While our new greedy algorithm does not yet have any guarantees, we next show that its performance is more robust than other algorithms, especially `Relax`. Indeed, as already noted in [5], the relaxed problem yields only the solutions

TABLE I
RUNNING TIMES [S] FOR VARYING N AND $K = 0.1N$.

Image side	64	128	256	512	1024
Greedy 1	0.0003	0.0008	0.0029	0.012	0.072
Greedy 2	0.0021	0.0074	0.034	0.16	1.06
Relax	0.0003	0.0014	0.0047	0.034	0.096

of the original problem that lie on the convex hull of its Pareto frontier: the more it is non-convex, the fewer solutions are achievable by relaxation. In order to illustrate this phenomenon, we use the same image as in the previous example, but then discretize the wavelet coefficients values into 16 levels, akin to using only 4 bits for storing the value of each coefficient. In this case, see Figure 2, the relaxed problem only yields approximately half of the solutions, 140 out of 256.

REFERENCES

[1] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 68, no. 1, pp. 49–67, 2006.

[2] R. Jenatton, J.-Y. Audibert, and F. Bach, "Structured variable selection with sparsity-inducing norms," *Journal of Machine Learning Research*, vol. 12, pp. 2777–2824, 2011.

[3] L. Jacob, G. Obozinski, and J. Vert, "Group lasso with overlap and graph lasso," in *International Conference on Machine Learning*, 2009.

[4] G. Obozinski, L. Jacob, and J. Vert, "Group lasso with overlaps: The latent group lasso approach," *arXiv preprint arXiv:1110.0413*, 2011.

[5] L. Baldassarre, N. Bhan, V. Cevher, and A. Kyrillidis, "Group-sparse model selection: Hardness and relaxations," *arXiv preprint arXiv:1303.3207*, 2013.

[6] C. Cartis and A. Thompson, "An exact tree projection algorithm for wavelets," *IEEE Signal Processing Letters*, 2013.

[7] R. Baraniuk, V. Cevher, M. Duarte, and C. Hegde, "Model-based compressive sensing," *IEEE Transactions on Information Theory*, 2010.

[8] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach, "Proximal methods for hierarchical sparse coding," *Journal of Machine Learning Reasearch*, vol. 12, pp. 2297–2334, 2011.

[9] M. Duarte and Y. Eldar, "Structured compressed sensing: From theory to applications," *IEEE Transactions on Signal Processing*, 2011.

[10] P. Sprechmann, I. Ramirez, G. Sapiro, and Y. C. Eldar, "C-hilasso: A collaborative hierarchical sparse modeling framework," *Signal Processing, IEEE Transactions on*, vol. 59, no. 9, pp. 4183–4198, 2011.

[11] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani, "A sparse-group lasso," *Journal of Computational and Graphical Statistics*, 2013.

[12] A. Subramanian *et al.*, "Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles," *PNAS*, vol. 102, no. 43, pp. 15 545–15 550, 2005.

[13] R. G. Baraniuk, "Optimal tree approximation with wavelets," in *Internat. Symp. on Optical Science, Engineering, and Instrumentation*, 1999.

[14] C. Hegde, P. Indyk, and L. Schmidt, "A fast approximation algorithm for tree-sparse recovery," in *International Symposium on Information Theory (ISIT)*, 2014.

[15] ——, "Nearly linear-time model-based compressive sensing," *International Colloquium on Automata, Languages, and Programming*, 2014.

[16] R. K. Iyer and J. A. Bilmes, "Submodular optimization with submodular cover and submodular knapsack constraints," in *Advances in Neural Information Processing Systems*, 2013, pp. 2436–2444.

[17] S. Khuller, A. Moss, and J. S. Naor, "The budgeted maximum coverage problem," *Information Processing Letters*, 1999.

[18] C. Bishop, *Pattern Recognition and Machine Learning*. Springer New York, 2006.

[19] S. Mallat, *A wavelet tour of signal processing*. Academic press, 1999.

[20] M. S. Crouse, R. D. Nowak, and R. G. Baraniuk, "Wavelet-based statistical signal processing using hidden markov models," *Signal Processing, IEEE Transactions on*, vol. 46, no. 4, pp. 886–902, 1998.

[21] N. Rao, R. Nowak, S. Wright, and N. Kingsbury, "Convex approaches to model wavelet sparsity patterns," in *IEEE International Conference on Image Processing (ICIP)*, 2011.