Group-Sparse Model Selection: Hardness and Relaxations

Luca Baldassarre, Nirav Bhan, Volkan Cevher, *Senior Member, IEEE*, Anastasios Kyrillidis, and Siddhartha Satpathi

***Abstract*— Group-based sparsity models are instrumental in linear and non-linear regression problems. The main premise of these models is the recovery of "interpretable" signals through the identification of their constituent groups, which can also provably translate in substantial savings in the number of measurements for linear models in compressive sensing. In this paper, we establish a combinatorial framework for group-model selection problems and highlight the underlying tractability issues. In particular, we show that the group-model selection problem is equivalent to the well-known NP-hard weighted maximum coverage problem. Leveraging a graph-based understanding of group models, we describe group structures that enable correct model selection in polynomial time via dynamic programming. Furthermore, we show that popular group structures can be explained by linear inequalities involving totally unimodular matrices, which afford other polynomial time algorithms based on relaxations. We also present a generalization of the group model that allows for within group sparsity, which can be used to model hierarchical sparsity. Finally, we study the Pareto frontier between approximation error and sparsity budget of group-sparse approximations for two tractable models, among which the tree sparsity model, and illustrate selection and computation tradeoffs between our framework and the existing convex relaxations.**

***Index Terms*— Signal approximation, structured sparsity, interpretability, tractability, dynamic programming, compressive sensing.**

I. Introduction

INFORMATION in many natural and man-made signals can be exactly represented or well approximated by a sparse set of nonzero coefficients in an appropriate basis [1]. Compressive sensing (CS) exploits this fact to recover signals from their compressive samples, which are dimensionality reducing, non-adaptive random measurements. According to the CS theory, the number of measurements for stable recovery is proportional to the signal sparsity, rather than to its Fourier bandwidth as dictated by the Shannon/Nyquist theorem [2]–[4]. Unsurprisingly, the utility of sparse representations also goes well-beyond CS and permeates a lot of fundamental problems in signal processing, machine learning, and theoretical computer science.

Recent results in CS extend the simple sparsity idea to consider more sophisticated *structured* sparsity models, which describe the interdependency between the nonzero coefficients [5]–[8]. There are several compelling reasons for such extensions: The structured sparsity models allow to significantly reduce the number of required measurements for perfect recovery in the noiseless case and be more stable in the presence of noise. Furthermore, they also facilitate the interpretation of the signals in terms of the chosen structures.

An important class of structured sparsity models is based on groups of variables that should either be selected or discarded together [8]–[12]. These structures naturally arise in applications such as neuroimaging [13], [14], gene expression data [11], [15], bioinformatics [16], [17] and computer vision [7], [18]. For example, in cancer research, the groups might represent genetic pathways that constitute cellular processes. Identifying which processes lead to the development of a tumor can allow biologists to directly target certain groups of genes instead of others [15]. Incorrect identification of the active/inactive groups can thus have a rather dramatic effect on the speed at which cancer therapies are developed.

In this paper, we consider *group-based* sparsity models, denoted $\mathfrak{G}$. These structured sparsity models feature collections of groups of variables that could overlap arbitrarily, that is $\mathfrak{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_M\}$ where each $\mathcal{G}_j$ is a subset of the index set $\{1, \ldots, N\}$, with N being the dimensionality of the signal that we model. Arbitrary overlaps mean that we do not restrict the intersection between any two sets from $\mathfrak{G}$.

We address the *signal approximation*, or projection, problem based on a known group structure $\mathfrak{G}$. That is, given a signal $\mathbf{x} \in \mathbb{R}^N$, we seek an $\hat{\mathbf{x}}$ closest to it in the Euclidean sense, whose *support* (*i.e.*, the index set of its non-zero coefficients) consists of the union of at most

Manuscript received March 28, 2013; revised March 28, 2016; accepted July 30, 2016. Date of publication August 24, 2016; date of current version October 18, 2016. This work was supported in part by the European Commission under Grant MIRG-268398, in part by ERC Future Proof, and in part by SNF under Grant 200021-132548.

L. Baldassarre and V. Cevher are with LIONS Laboratory, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland (e-mail: luca.baldassarre@epfl.ch; volkan.cevher@epfl.ch).

N. Bhan was with LIONS Laboratory, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland. He is now with the Laboratory of Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: niravb@mit.edu).

A. Kyrillidis was with LIONS Laboratory, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland. He is now with the Wireless Networking and Communications Group, The University of Texas at Austin, Austin, TX 78712 USA (e-mail: anastasios@utexas.edu).

S. Satpathi was with LIONS Laboratory, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland. He is now with the Coordinated Science Laboratory, University of Illinois at Urbana–Champaign, Champaign, IL 61801 USA (e-mail: ssatpth2@illinois.edu).

Communicated by G. Matz, Associate Editor for Detection and Estimation.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2016.2602222

G groups from $\mathfrak{G}$, where $G > 0$ is a user-defined group budget:

$$\hat{\mathbf{x}} \in \underset{\mathbf{z}\in\mathbb{R}^N}{\operatorname{argmin}} \ \|\mathbf{x}-\mathbf{z}\|_2^2 \quad \text{subject to } \operatorname{supp}(\mathbf{z}) \subseteq \bigcup_{\mathcal{G}\in\mathcal{S}} \mathcal{G} \quad \mathcal{S} \subseteq \mathfrak{G}, \quad |\mathcal{S}| \leq G,$$

where $\operatorname{supp}(\mathbf{z})$ is the support of the vector $\mathbf{z}$. We call such an approximation *G-group-sparse* or in short *group-sparse*. The projection problem is a fundamental step in Model-based Iterative Hard-Thresholding algorithms for solving inverse problems by imposing group structures [7], [19].

More importantly, we seek to also identify the *G-group-support* of the approximation $\hat{\mathbf{x}}$, that is the G groups that constitute its support. We call this the group-sparse *model selection* problem. The G-group-support of $\hat{\mathbf{x}}$ allows us to interpret the original signal and discover its properties so that we can, for example, target specific groups of genes instead of others [15] or focus more precise imaging techniques on certain brain regions only [20]. In this work, we study under which circumstances we can correctly and tractably identify the G-group-support of the approximation of a given signal. In particular, we show that this problem is equivalent to an NP-hard combinatorial problem known as the weighted maximum coverage problem and we propose a novel polynomial time algorithm for finding its solutions for a certain class of group structures.

If the original signal is affected by noise, *i.e.*, if instead of $\mathbf{x}$, we measure $\mathbf{z} := \mathbf{x} + \boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon}$ is some random noise, the G-group support of $\hat{\mathbf{z}}$ may not exactly correspond to the one of $\hat{\mathbf{x}}$. Although this is a paramount statistical issue, here we are solely concerned with the computational problem of finding the G-group support of a given signal, irrespective of whether it is affected by noise or not, because any group-based interpretation would necessarily require such computation.

A. Previous Work

Recent works in compressive sensing and machine learning with group sparsity have mainly focused on leveraging group structures for lowering the number of samples required for recovering signals [5]–[8], [11], [21]–[23]. While these results have established the importance of group structures, many of these works have not fully addressed model selection.

For the special case of non-overlapping groups, dubbed the block-sparsity model, the problem of model selection does not present computational difficulties and features a well-understood theory [21]. The first convex relaxation for group-sparse approximation [24] considered only non-overlapping groups. Its extension to overlapping groups [25], however, selects supports defined as the complement of a union of groups (see also [10]), which is the opposite of what applications usually require, where groups of variables need to be selected together, instead of discarded.

For overlapping groups, Eldar and Mishali [5] consider the union of subspaces framework and cast the model selection problem as a block-sparse model selection one by duplicating the variables that belong to overlaps between the groups. Their uniqueness condition [5, Proposition 1], however, is infeasible for any group structure with overlaps, because it requires that the subspaces intersect only at the origin, while two subspaces defined by two overlapping groups of variables intersect on a subspace of dimension equal to the number of elements in the overlap.

The recently proposed convex relaxations [11], [23] for group-sparse approximations select group-supports that consist of union of groups. However, the group-support recovery conditions in [11] and [23] should be taken with care, because they are defined with respect to a particular subset of group-supports and are not general. As we numerically demonstrate in this paper, the group-supports recovered with these methods might be incorrect. Furthermore, the required consistency conditions in [11] and [23] are unverifiable *a priori*. For instance, they require case-specific tuning parameters to obtain the correct group-support, which cannot be known a-priori.

Huang et al. [22] use coding complexity schemes over sets to encode sparsity structures. They consider linear regression problems where the coding complexity of the support of the solution is constrained to be below a certain value. Inspired by Orthogonal Matching Pursuit, they then propose a greedy algorithm, named StructOMP, that leverages a block-based approximation to the coding complexity. A particular instance of coding schemes, namely graph sparsity, can be used to encode both group and hierarchical sparsity. Their method only returns an approximation to the original discrete problem, as we illustrate via some numerical experiments.

Obozinski and Bach [26] consider a penalty involving the sum of a combinatorial set function F and the ℓ_p norm. In order to derive a convex relaxation of the penalty, they first find its tightest positive homogeneous and convex lower bound, which is $F(\operatorname{supp}(\mathbf{x}))^{\frac{1}{q}}\|\mathbf{x}\|_p$, with $\frac{1}{p}+\frac{1}{q}=1$. They also consider set-cover penalties, based on the weighted set cover of a set. Given a set function F, the weighted set cover of a set $\mathcal{A}$ is the minimum sum of weights of sets that are required to cover $\mathcal{A}$. With a proper choice of the set function F, the weighted set cover can be shown to correspond to the group ℓ_0-"norm" that we define in the following. They establish that the latent group lasso norm as defined in [23] is the tightest convex relaxation of the function $\mathbf{x} \mapsto \|\mathbf{x}\|_p \tilde{F}(\operatorname{supp}(\mathbf{x}))^{\frac{1}{q}}$, where $\tilde{F}(\operatorname{supp}(\mathbf{x}))$ is a properly designed weighted set cover of the support of $\mathbf{x}$.

B. Contributions

This paper is an extended version of a prior submission to the IEEE International Symposium on Information Theory (ISIT), 2013. This version contains all the proofs that were previously omitted due to lack of space, refined explanations of the concepts, and provides the full description of the proposed dynamic programming algorithms.

In stark contrast to the existing literature, we take an explicitly discrete approach to identifying group-supports of signals given a budget constraint on the number of groups. This fresh perspective enables us to show that the group-sparse model selection problem is NP-hard: if we can solve

the group model selection problem in general, then we can solve any weighted maximum coverage (WMC) problem instance in polynomial time. However, WMC is known to be NP-Hard [27]. Given this, we can only hope to characterize a subset of instances which are tractable or find guaranteed and tractable approximations.

We present group structures that lead to computationally tractable problems via dynamic programming. We do so by exploiting the properties of a graph-based representation of the groups. In particular, we present and describe a novel polynomial-time dynamic program that solves the WMC problem for group structures whose graph representation is a tree or a forest. This result is of interest by itself.

We identify tractable discrete relaxations of the group-sparse model selection problem that lead to efficient algorithms. Specifically, we relax the constraint on the number of groups into a penalty term and show that, if the remaining group constraints can be described by linear inequalities involving totally unimodular matrices [28]–[30], then the relaxed problem can be efficiently solved using linear program solvers. Furthermore, if the graph induced by the group structure is a tree or a forest, we can solve the relaxed problem in linear time by the sum-product algorithm [31].

We extend the discrete model to incorporate an overall sparsity constraint and allowing to select individual elements from each group, leading to within-group sparsity. Furthermore, we discuss how this extension can be used to model hierarchical relationships between variables. We present a novel polynomial-time dynamic program that solves the hierarchical model selection problem exactly and discuss a tractable discrete relaxation.

We also interpret the implications of our results in the context of other group-based recovery frameworks. For instance, the convex approaches proposed in [5], [11], and [23] also relax the discrete constraint on the cardinality of the group support. However, they first need to decompose the approximation into vector atoms whose support consists only of one group and then penalize the norms of these atoms. It has been observed [11] that these relaxations produce approximations that are group-sparse, but their group-support might include irrelevant groups. We concretely illustrate these cases via Pareto frontier examples on two different group structures.

C. Paper Structure

The paper is organized as follows. In Section 2, we present definitions of group-sparsity and related concepts, while in Section III, we formally define the approximation and model-selection problems and connect them to the WMC problem. We present and analyze discrete relaxations of the WMC in Section IV and consider convex relaxations in Section V. In Section VI, we illustrate via a simple example the differences between the original problem and the relaxations. The generalized model is introduced and analyzed in Section VII, while in Section VIII, we provide an outline of our two dynamic programming algorithms. Numerical simulations are presented in Section IX and we conclude the paper with some remarks in Section X. The appendices contain the detailed descriptions of the dynamic programs.

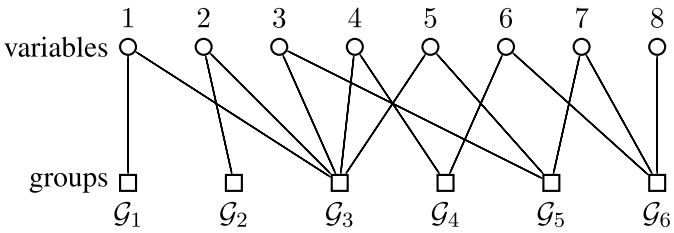

Fig. 1. Example of bipartite graph between variables and groups induced by the group structure $\mathfrak{G}^1$, see Example 1 for details.

II. Basic Definitions

Let $\mathbf{x} \in \mathbb{R}^N$ be a vector, with $\dim(\mathbf{x}) = N$, and $\mathcal{N} = \{1, \ldots, N\}$ be the set of its indices. We use $|\mathcal{S}|$ to denote the cardinality of an index set $\mathcal{S}$. Given a vector $\mathbf{x} \in \mathbb{R}^N$ and a set $\mathcal{S}$, we define $\mathbf{x}_\mathcal{S} \in \mathbb{R}^{|\mathcal{S}|}$, such that the components of $\mathbf{x}_\mathcal{S}$ are the components of $\mathbf{x}$ indexed by $\mathcal{S}$. We use $\mathbb{B}^N$ to represent the space of N-dimensional binary vectors and define $\iota : \mathbb{R}^N \to \mathbb{B}^N$ to be the indicator function of the nonzero components of a vector in $\mathbb{R}^N$, *i.e.*, $\iota(\mathbf{x})_i = 1$ if $x_i \neq 0$ and $\iota(\mathbf{x})_i = 0$, otherwise. We let $\mathbf{1}_N$ to be the N-dimensional vector of all ones and $\mathbf{I}_N$ the $N \times N$ identity matrix. The support of $\mathbf{x}$ is defined by the set-valued function $\text{supp}(\mathbf{x}) = \{i \in \mathcal{N} : x_i \neq 0\}$. Note that we normally use bold lowercase letters to indicate vectors and bold uppercase letters to indicate matrices.

We start with the definition of total unimodularity, a property of matrices that will turn out to be key for obtaining efficient relaxations of integer linear programs.

Definition 1: *A* **totally unimodular matrix** *(TU matrix) is a matrix for which every square non-singular submatrix has determinant equal to* -1 *or* 1.

We now define the main building block of group sparse model selection, the group structure.

Definition 2: *A* **group structure** $\mathfrak{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_M\}$ *is a collection of index sets, named* groups*, with* $\mathcal{G}_j \subseteq \mathcal{N}$ *and* $|\mathcal{G}_j| = g_j$ *for* $1 \leq j \leq M$ *and* $\bigcup_{\mathcal{G} \in \mathfrak{G}} \mathcal{G} = \mathcal{N}$.

We can represent a group structure $\mathfrak{G}$ as a bipartite graph, where on one side we have the N variables nodes and on the other the M group nodes. An edge connects a variable node i to a group node j if $i \in \mathcal{G}_j$. Fig. 1 shows an example. The bi-adjacency matrix $\mathbf{A}^\mathfrak{G} \in \mathbb{B}^{N \times M}$ of the bipartite graph encodes the group structure,

$$A^\mathfrak{G}_{ij} = \begin{cases} 1, & \text{if } i \in \mathcal{G}_j; \\ 0, & \text{otherwise.} \end{cases}$$

Another useful representation of a group structure is via an *intersection graph* $(\mathcal{V}, \mathcal{E})$ where the nodes $\mathcal{V}$ are the groups $\mathcal{G} \in \mathfrak{G}$ and the edge set $\mathcal{E}$ contains e_{ij} if $\mathcal{G}_i \cap \mathcal{G}_j \neq \emptyset$, that is an edge connects two groups that *overlap*. A sequence of connected nodes $v_1, v_2, \ldots, v_n$, is a *cycle* if $v_1 = v_n$.

Example 1: *In order to illustrate these concepts, consider the group structure* $\mathfrak{G}^1$ *defined by the following groups,* $\mathcal{G}_1 = \{1\}$, $\mathcal{G}_2 = \{2\}$, $\mathcal{G}_3 = \{1, 2, 3, 4, 5\}$, $\mathcal{G}_4 = \{4, 6\}$, $\mathcal{G}_5 = \{3, 5, 7\}$ *and* $\mathcal{G}_6 = \{6, 7, 8\}$. $\mathfrak{G}^1$ *can be represented by the variables-groups bipartite graph of Fig. 1 or by the*

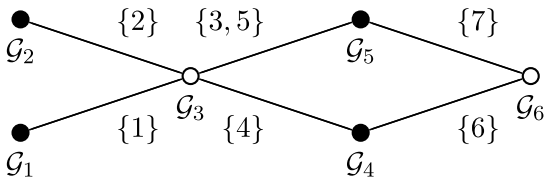

Fig. 2. Bipartite intersection graph with cycles induced by the group structure $\mathfrak{G}^1$, where on each edge we report the elements of the intersection.

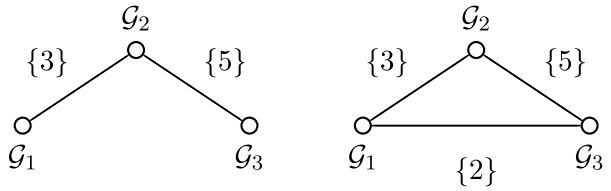

Fig. 3. (Left) Acyclic group structure. (Right) By adding one element from $\mathcal{G}_1$ into $\mathcal{G}_3$, we introduce a cycle in the graph.

intersection graph of Fig. 2, which is bipartite and contains cycles.

An important class of group structures is given by groups whose intersection graph is acyclic (*i.e.*, a tree or a forest) and we call them *acyclic group structures*. A necessary, but not sufficient, condition for a group structure to have an acyclic intersection graph is that each element of $\mathcal{N}$ occurs in at most two groups, *i.e.*, the groups are at most *pairwise overlapping*. Note that a tree or a forest is a bipartite graph, where the two partitions contain the nodes that belong to alternate levels of the tree/forest. For example, consider $\mathcal{G}_1 = \{1, 2, 3\}$, $\mathcal{G}_2 = \{3, 4, 5\}$, $\mathcal{G}_3 = \{5, 6, 7\}$, which can be represented by the intersection graph in Fig. 3(Left). If $\mathcal{G}_3$ were to include an element from $\mathcal{G}_1$, for example $\{2\}$, we would have the cyclic graph of Fig. 3(Right). Note that $\mathfrak{G}^1$ is pairwise overlapping, but not acyclic, since $\mathcal{G}_3, \mathcal{G}_4, \mathcal{G}_5$ and $\mathcal{G}_6$ form a cycle.

We anchor our analysis of the tractability of interpretability via selection of groups on covering arguments. Most of the definitions we introduce here can be reformulated as variants of set covers on the support of a signal $\mathbf{x}$, however we believe it is more natural in this context to talk about group covers of a signal $\mathbf{x}$ directly.

Definition 3: *A* **group cover** $\mathcal{S}(\mathbf{x})$ *for a signal* $\mathbf{x} \in \mathbb{R}^N$ *is a collection of groups such that* $\text{supp}(\mathbf{x}) \subseteq \bigcup_{\mathcal{G}\in\mathcal{S}(\mathbf{x})} \mathcal{G}$. *An alternative equivalent definition is given by*

$$\mathcal{S}(\mathbf{x}) = \{\mathcal{G}_j \in \mathfrak{G} : \boldsymbol{\omega} \in \mathbb{B}^M,\ \omega_j = 1,\ \mathbf{A}^{\mathfrak{G}}\boldsymbol{\omega} \geq \imath(\mathbf{x})\}\ .$$

The binary vector $\boldsymbol{\omega}$ indicates which groups are active and the constraint $\mathbf{A}^{\mathfrak{G}}\boldsymbol{\omega} \geq \imath(\mathbf{x})$ makes sure that, for every non-zero component of $\mathbf{x}$, there is at least one active group that covers it. We also say that $\mathcal{S}(\mathbf{x})$ *covers* $\mathbf{x}$. Note that the group cover is often not unique and $\mathcal{S}(\mathbf{x}) = \mathfrak{G}$ is a group cover for any signal $\mathbf{x}$. This observation leads us to consider more restrictive definitions of group covers.

Definition 4: *A* G**-group cover** $\mathcal{S}^G(\mathbf{x}) \subseteq \mathfrak{G}$ *is a group cover for* $\mathbf{x}$ *with at most* G *elements,*

$$\mathcal{S}^G(\mathbf{x}) = \{\mathcal{G}_j \in \mathfrak{G} : \boldsymbol{\omega} \in \mathbb{B}^M,\ \omega_j = 1,\ \mathbf{A}^{\mathfrak{G}}\boldsymbol{\omega} \geq \imath(\mathbf{x}),\ \sum_{j=1}^{M} \omega_j \leq G\}.$$

It is not guaranteed that a G-group cover always exists for any value of G. Finding the smallest G-group cover lead to the following definitions.

Definition 5: *The* **group** ℓ_0**-"norm"** *is defined as*

$$\|\mathbf{x}\|_{\mathfrak{G},0} := \min_{\boldsymbol{\omega}\in\mathbb{B}^M} \left\{ \sum_{j=1}^{M} \omega_j : \mathbf{A}^{\mathfrak{G}}\boldsymbol{\omega} \geq \imath(\mathbf{x}) \right\}. \tag{1}$$

A similar definition of group sparsity is also considered in [22]; however, there are also key differences in the concepts used for such definition. The authors use *coding complexity* as a lower bound on the "cost" required to cover a given subset of $\mathcal{N}$. Particularly, in the group sparsity case as defined in [22], each predefined group is assigned the coding complexity $\log_2(2m)$, where m is the total number of groups in the model. Then, based on [22, Definition 2], the ℓ_0-"norm" for group sparsity is defined as the minimum coding length of the selected subset of groups: *i.e.*, the summation of coding lengths for each group, such that the union of groups encoded "covers" the given support set. Thus, the resulting coding length in this case is $g \log_2(2m)$, where g is the number of groups used in the covering. In our definition of group ℓ_0-"norm", we assign a unitary cost to each selected group, such that each non-zero element in $\mathbf{x}$ is covered by at least one active group.

Definition 6: *A* **minimal group cover** *for a signal* $\mathbf{x} \in \mathbb{R}^N$ *is defined as* $\mathcal{M}(\mathbf{x}) \equiv \{\mathcal{G}_j \in \mathfrak{G} : \hat{\omega}(\mathbf{x})_j = 1\}$, *where* $\hat{\boldsymbol{\omega}}$ *is a minimizer for* (1)*,*

$$\hat{\boldsymbol{\omega}}(\mathbf{x}) \in \operatorname*{argmin}_{\boldsymbol{\omega}\in\mathbb{B}^M} \left\{ \sum_{j=1}^{M} \omega_j : \mathbf{A}^{\mathfrak{G}}\boldsymbol{\omega} \geq \imath(\mathbf{x}) \right\}.$$

A *minimal group cover* $\mathcal{M}(\mathbf{x})$ is a group cover for the support of $\mathbf{x}$ with minimal cardinality. Note that there exist pathological cases where for the same group ℓ_0-"norm", we have different minimal group cover models. The minimal group cover can also be seen as the minimum set cover of the support of $\mathbf{x}$.

Definition 7: *A signal* $\mathbf{x}$ *is* G**-group sparse** *with respect to a group structure* $\mathfrak{G}$ *if* $\|\mathbf{x}\|_{\mathfrak{G},0} \leq G$.

In other words, a signal is *G-group sparse* if its support is contained in the union of at most G groups from $\mathfrak{G}$.

III. Tractability of Interpretations

Although real signals may not be exactly group-sparse, it is possible to give a group-based interpretation by finding a group-sparse approximation and identifying the groups that constitute its support. In this section, we establish the hardness of group-constrained approximations of signals in general and characterize a class of group structures that lead to tractable approximations. In particular, we present a polynomial time algorithm that finds the correct G-group-support of the G-group-sparse approximation of $\mathbf{x}$, given a positive integer G and the group structure $\mathfrak{G}$.

We first define the G-group sparse approximation $\hat{\mathbf{x}}$ and then show that it can be easily obtained from its G-group cover $\mathcal{S}^G(\hat{\mathbf{x}})$, which is the solution of the model selection problem. We then reformulate the model selection problem as the weighted maximum coverage problem. Finally, we present

our main result, the polynomial time dynamic program for acyclic group structures.

Signal Approximation Problem: Given a signal $\mathbf{x} \in \mathbb{R}^N$, *a best G-group sparse approximation* $\hat{\mathbf{x}}$ *is given by*

$$\hat{\mathbf{x}} \in \operatorname*{argmin}_{\mathbf{z}\in\mathbb{R}^N} \left\{ \|\mathbf{x}-\mathbf{z}\|_2^2 : \|\mathbf{z}\|_{\mathfrak{G},0} \leq G \right\}. \tag{2}$$

If we already know the G-group cover of the approximation $\mathcal{S}^G(\hat{\mathbf{x}})$, we can obtain $\hat{\mathbf{x}}$ as $\hat{\mathbf{x}}_{\mathcal{I}} = \mathbf{x}_{\mathcal{I}}$ and $\hat{\mathbf{x}}_{\mathcal{I}^c} = 0$, where $\mathcal{I} = \bigcup_{\mathcal{G}\in\mathcal{S}^G(\hat{\mathbf{x}})} \mathcal{G}$ and $\mathcal{I}^c = \mathcal{N}\setminus\mathcal{I}$. Therefore, we can solve (2) by solving the following discrete problem.

Model Selection Problem: Given a signal $\mathbf{x} \in \mathbb{R}^N$, *a G-group cover model for its G-group sparse approximation is expressed as follows*

$$\mathcal{S}^G(\hat{\mathbf{x}}) \in \operatorname*{argmax}_{\mathcal{S}\subseteq\mathfrak{G}} \left\{ \sum_{i\in\mathcal{I}} x_i^2 : \mathcal{I} = \bigcup_{\mathcal{G}\in\mathcal{S}} \mathcal{G},\ |\mathcal{S}| \leq G \right\}. \tag{3}$$

To show the connection between the two problems, we first reformulate (2) as

$$\begin{aligned} &\min_{\mathbf{z}\in\mathbb{R}^N} \ \|\mathbf{x}-\mathbf{z}\|_2^2 \\ &\text{subject to } \ \operatorname{supp}(\mathbf{z}) = \mathcal{I}, \quad \mathcal{I} = \bigcup_{\mathcal{G}\in\mathcal{S}} \mathcal{G} \\ &\qquad\qquad \mathcal{S} \subseteq \mathfrak{G}, \quad |\mathcal{S}| \leq G, \end{aligned} \tag{4}$$

which can be rewritten as

$$\min_{\substack{\mathcal{S}\subseteq\mathfrak{G} \\ |\mathcal{S}|\leq G \\ \mathcal{I}=\bigcup_{\mathcal{G}\in\mathcal{S}}\mathcal{G}}} \ \min_{\substack{\mathbf{z}\in\mathbb{R}^N \\ \operatorname{supp}(\mathbf{z})=\mathcal{I}}} \|\mathbf{x}-\mathbf{z}\|_2^2.$$

The optimal solution is not changed if we introduce a constant, change sign of the objective and consider maximization instead of minimization

$$\max_{\substack{\mathcal{S}\subseteq\mathfrak{G} \\ |\mathcal{S}|\leq G \\ \mathcal{I}=\bigcup_{\mathcal{G}\in\mathcal{S}}\mathcal{G}}} \ \max_{\substack{\mathbf{z}\in\mathbb{R}^N \\ \operatorname{supp}(\mathbf{z})=\mathcal{I}}} \left\{ \|\mathbf{x}\|_2^2 - \|\mathbf{x}-\mathbf{z}\|_2^2 \right\}.$$

The internal maximization is achieved for $\hat{\mathbf{x}}$ as $\hat{\mathbf{x}}_{\mathcal{I}} = \mathbf{x}_{\mathcal{I}}$ and $\hat{\mathbf{x}}_{\mathcal{I}^c} = 0$, so that we have, as desired,

$$\mathcal{S}^G(\hat{\mathbf{x}}) \in \operatorname*{argmax}_{\substack{\mathcal{S}\subseteq\mathfrak{G} \\ |\mathcal{S}|\leq G \\ \mathcal{I}=\bigcup_{\mathcal{G}\in\mathcal{S}}\mathcal{G}}} \|\mathbf{x}_{\mathcal{I}}\|_2^2.$$

The following reformulation of (3) as a binary problem allows us to characterize its tractability.

Lemma 1: *Given* $\mathbf{x} \in \mathbb{R}^N$ *and a group structure* $\mathfrak{G}$, *we have that* $\mathcal{S}^G(\hat{\mathbf{x}}) = \{\mathcal{G}_j \in \mathfrak{G} : \omega_j^G = 1\}$, *where* $(\boldsymbol{\omega}^G, \mathbf{y}^G)$ *is an optimal solution of*

$$\max_{\boldsymbol{\omega}\in\mathbb{B}^M,\ \mathbf{y}\in\mathbb{B}^N} \left\{ \sum_{i=1}^N y_i x_i^2 : \mathbf{A}^{\mathfrak{G}}\boldsymbol{\omega} \geq \mathbf{y}, \sum_{j=1}^M \omega_j \leq G \right\}. \tag{5}$$

Proof: The proof follows along the same lines as the proof in [29]. Note that in (5), $\boldsymbol{\omega}$ and $\mathbf{y}$ are binary variables that specify which groups and which variables are selected, respectively. The constraint $\mathbf{A}^{\mathfrak{G}}\boldsymbol{\omega} \geq \mathbf{y}$ makes sure that for every selected variable at least one group is selected to cover it, while the constraint $\sum_{j=1}^M \omega_j \leq G$ restricts choosing at most G groups. □

Problem (5) can produce all the instances of the weighted maximum coverage problem (WMC), where the weights for each element are given by x_i^2 $(1 \leq i \leq N)$ and the index sets are given by the groups $\mathcal{G}_j \in \mathfrak{G}$ $(1 \leq j \leq M)$. Since WMC is NP-hard [27] and given Lemma 1, the tractability of (3) directly depends on the hardness of (5), which leads to the following result.

Proposition 1: *The model selection problem* (3) *is NP-hard.*

It is possible to approximate the solution of (5) using the greedy WMC algorithm [32]. At each iteration, the algorithm selects the group that covers new variables with maximum combined weight until G groups have been selected. However, we show next that for certain group structures we can find an exact solution.

Our main result is an algorithm for solving (5) for acyclic group structures.

Theorem 1: *Given an acyclic group structure* $\mathfrak{G}$ *with M groups and a group budget G, the dynamic programming algorithm described in Section VIII-B solves* (5) *in* $\mathcal{O}(GM^2)$ *time.*

The proof of a more general algorithm, which also includes a sparsity budget, is given in Appendix A, while an intuitive description of the algorithm is given in Section VIII-B.

Remark 1: *It is also possible to consider the case where each group* $\mathcal{G}_i$ *has a cost* C_i *and we are given a maximum group cost budget C. The problem then becomes the Budgeted Maximum Coverage [33]. However, this problem is NP-hard, even in the non-overlapping case, because it generalizes the knapsack problem. However, similarly to the pseudo-polynomial time algorithm for knapsack [34], we can easily devise a pseudo-polynomial time algorithm for the weighted group sparse problem, even for acyclic overlaps. The only condition is that the costs must be integers. The time complexity of the resulting algorithm is then polynomial in C, the maximum group cost budget. The algorithm is almost the same as the one given in Appendix A: instead of keeping track of selecting g groups, where g varies from* 1 *to G; we keep track of selecting groups with total weight equal to c, where c varies from* 1 *to C.*

IV. Discrete Relaxations

Relaxations are useful techniques that allow to obtain approximate, or sometimes even exact solutions. In the specific cases below, relaxations provide less computationally demanding solutions. In our case, we relax the constraint on the number of groups in (5) into a regularization term with parameter $\lambda > 0$, which amounts to paying a penalty of λ for each selected group. We then obtain the following binary linear program

$$(\boldsymbol{\omega}^\lambda, \mathbf{y}^\lambda) \in \operatorname*{argmax}_{\substack{\boldsymbol{\omega}\in\mathbb{B}^M,\ \mathbf{y}\in\mathbb{B}^N \\ \mathbf{A}^{\mathfrak{G}}\boldsymbol{\omega}\geq\mathbf{y}}} \left\{ \sum_{i=1}^N y_i x_i^2 - \lambda \sum_{j=1}^M \omega_j \right\} \tag{6}$$

We can rewrite the previous program in standard form. Let $\mathbf{u}^\top = [\mathbf{y}^\top \ \boldsymbol{\omega}^\top] \in \mathbb{B}^{N+M}$, $\mathbf{w}^\top = [x_1^2, \ldots, x_N^2, -\lambda\mathbf{1}_M^\top] \in \mathbb{R}^{N+M}$ and $\mathbf{C} = [\mathbf{I}_N, \ -\mathbf{A}^{\mathfrak{G}}] \in \mathbb{B}^{N\times(N+M)}$. We then have

that (6) is equivalent to

$$\mathbf{u}^\lambda \in \operatorname*{argmax}_{\mathbf{u}\in\mathbb{B}^{N+M}} \left\{ \mathbf{w}^\top \mathbf{u} : \mathbf{C}\mathbf{u} \leq 0 \right\} \tag{7}$$

In general, (7) is NP-hard, however, it is well known [28] that if the constraint matrix $\mathbf{C}$ is Totally Unimodular (TU), then it can be solved in polynomial-time. While the concatenation of two arbitrary TU matrices is not TU, the concatenation of the identity matrix with a TU matrix results in a TU matrix. Thus, due to its structure, $\mathbf{C}$ is TU if and only if $\mathbf{A}^{\mathfrak{G}}$ is TU [28, Proposition 2.1].

The next lemma characterizes which group structures lead to totally unimodular constraints.

Proposition 2: *Group structures whose intersection graph is bipartite lead to constraint matrices* $\mathbf{A}^{\mathfrak{G}}$ *that are TU.*

Proof: We first use a result that establishes that if a matrix is TU, then its transpose is also TU [28, Proposition 2.1]. We then apply [28, Corollary 2.8] to $\mathbf{A}^{\mathfrak{G}}$, swapping the roles of rows and columns. Given a $\{0, 1, -1\}$ matrix whose columns can be partitioned into two sets, $\mathcal{S}_1$ and $\mathcal{S}_2$, and with no more than two nonzero elements in each row, this corollary provides two sufficient conditions for it being totally unimodular:

1) If two nonzero entries in a row have the same sign, then the column of one is in $\mathcal{S}_1$ and the other is in $\mathcal{S}_2$.
2) If two nonzero entries in a row have opposite signs, then their columns are both in $\mathcal{S}_1$ or both in $\mathcal{S}_2$.

In our case, the columns of $\mathbf{A}^{\mathfrak{G}}$, which represent groups, can be partitioned in two sets, $\mathcal{S}_1$ and $\mathcal{S}_2$ because the intersection graph is bipartite. The two sets represents groups which have no common overlap so that each row of $\mathbf{A}^{\mathfrak{G}}$ contains at most two nonzero entries, one in each set. Furthermore, the entries in $\mathbf{A}^{\mathfrak{G}}$ are only 0 or 1, so that condition 1) is satisfied and condition 2) does not apply. □

Corollary 1: *Acyclic group structures lead to totally unimodular constraints.*

Proof: Acyclic group structures have an intersection graph which is a tree or a forest, which is bipartite. □

The worst case complexity for solving the linear program (7), via a primal-dual method [35], is $\mathcal{O}(N^2(N+M)^{1.5})$, which is greater than the complexity of the dynamic program of Theorem 1. However, in practice, using an off-the-shelf LP solver may still be faster, because the empirical performance is usually much better than the worst case complexity.

Another way of solving the linear program for acyclic group structures is to reformulate it as an energy maximization problem over a tree, or forest. In particular, let $\psi_i = \|\mathbf{x}_{\mathcal{G}_i}\|_2^2$ be the energy captured by group $\mathcal{G}_i$ and $\psi_{ij} = \|\mathbf{x}_{\mathcal{G}_i\cap\mathcal{G}_j}\|_2^2$ the energy that is double counted if both $\mathcal{G}_i$ and $\mathcal{G}_j$ are selected, which then needs to be subtracted from the total energy. Consider first problem (3). Given the energy functions defined above, the objective in the maximization can be rewritten as:

$$\begin{aligned}\sum_{i\in\mathcal{I}} x_i^2 = \|\mathbf{x}_{\mathcal{I}}\|_2^2 &= \sum_{i=1}^{M} \omega_i \|\mathbf{x}_{\mathcal{G}_i}\|_2^2 - \sum_{(i,j)\in\mathcal{E}} \omega_i\omega_j \|\mathbf{x}_{\mathcal{G}_i\cap\mathcal{G}_j}\|_2^2 \\ &= \sum_{i=1}^{M} \omega_i \psi_i - \sum_{(i,j)\in\mathcal{E}} \omega_i\omega_j \psi_{ij}\,. \end{aligned} \tag{8}$$

Here, the first term corresponds to the energy contributed by the active groups and the second term corresponds to the excessive energy contributed by the *overlapping* active groups, and thus needs to be removed. The regularized version, problem (6), can then be formulated as

$$\max_{\boldsymbol{\omega}\in\mathbb{B}^M} \sum_{i=1}^{M} \omega_i(\psi_i - \lambda) - \sum_{(i,j)\in\mathcal{E}} \omega_i\omega_j\psi_{ij}\,.$$

This problem is equivalent to finding the most probable state of the binary variables ω_i, where their probabilities can be factored into node and edge potentials. These potentials can be computed in $\mathcal{O}(N)$ time via a single sweep over the elements, then the most probable state can be exactly estimated by the max-sum algorithm in only $\mathcal{O}(M)$ operations, by sending messages from the leaves to the root and then propagating other message from the root back to the leaves [31].

The next lemma establishes when the regularized solution coincides with the solution of (5).

Lemma 2: *If the value of the regularization parameter* λ *is such that the solution* $(\boldsymbol{\omega}^\lambda, \mathbf{y}^\lambda)$ *of* (6) *satisfies* $\sum_j \omega_j^\lambda = G$, *then* $(\boldsymbol{\omega}^\lambda, \mathbf{y}^\lambda)$ *is also a solution for* (5).

Proof: This lemma is a direct consequence of Proposition 3 below. □

However, as we numerically show in Section IX, given a value of G it is not always possible to find a value of λ such that the solution of (6) is also a solution for (5). Let the set of points $\mathcal{P} = \{G, (f(G))\}_{G=1}^{M}$, where $f(G) = \sum_{i=1}^{N} y_i^G x_i^2$, be the Pareto frontier of (5) between approximation quality and group budget. We then have the following characterization of the solutions of the discrete relaxation.

Proposition 3: *The discrete relaxation* (6) *yields only the solutions that lie on the intersection between the Pareto frontier* $\mathcal{P}$ *of* (5)*, as defined above, and the boundary of the convex hull of* $\mathcal{P}$.

Proof: The solutions of (5) for all possible values of G are the Pareto optimal solutions [36, Sec. 4.7] of the following vector-valued minimization problem with respect to the positive orthant of $\mathbb{R}^2$, which we denote by $\mathbb{R}_+^2$,

$$\begin{aligned} &\min_{\boldsymbol{\omega}\in\mathbb{B}^M,\ \mathbf{y}\in\mathbb{B}^N} \ \mathbf{f}(\boldsymbol{\omega}, \mathbf{y}) \\ &\text{subject to } \mathbf{A}^{\mathfrak{G}}\boldsymbol{\omega} \geq \mathbf{y} \end{aligned} \tag{9}$$

where $\mathbf{f}(\boldsymbol{\omega}, \mathbf{y}) = \left(\|\mathbf{x}\|^2 - \sum_{i=1}^{N} y_i x_i^2, \sum_{j=1}^{M} \omega_j\right) \in \mathbb{R}_+^2$. Specifically, the two components of the vector-valued function $\mathbf{f}$ are the approximation error E, and the number of groups G that cover the approximation. It is not possible to simultaneously minimize both components, because they are somehow adversarial: unless there is a group in the group structure that covers the entire support of $\mathbf{x}$, lowering the approximation error requires selecting more groups. Therefore, there exists the so called Pareto frontier of the vector-valued optimization problem defined by the points (E_G, G) for each choice of G, *i.e.*, the second component of $\mathbf{f}$, where E_G is the minimum approximation error achievable with a support covered by at most G groups.

The scalarization of (9) yields the following discrete problem, with $\lambda > 0$

$$\min_{\boldsymbol{\omega}\in\mathbb{B}^M,\ \mathbf{y}\in\mathbb{B}^N} \ \|\mathbf{x}\|^2 - \sum_{i=1}^{N} y_i x_i^2 + \lambda \sum_{j=1}^{M} \omega_j$$
$$\text{subject to } \mathbf{A}^{\mathfrak{G}}\boldsymbol{\omega} \geq \mathbf{y} \tag{10}$$

whose solutions are the same as for (6). Therefore, the relationship between the solutions of (5) and (6) can be inferred by the relationship between the solutions of (9) and (10). It is known that the solutions of (10) are also Pareto optimal solutions of (9), but only the Pareto optimal solutions of (9) that admit a supporting hyperplane for the feasible objective values of (9) are also solutions of (10) [36, Sec. 4.7]. In other words, the solutions obtainable via scalarization belong to the intersection of the Pareto optimal solution set and the boundary of its convex hull. □

V. Convex Relaxations

For tractability and analysis, convex proxies to the group ℓ_0-norm have been proposed (e.g., [23]) for finding group-sparse approximations of signals. Given a group structure $\mathfrak{G}$, an example generalization is defined as

$$\|\mathbf{x}\|_{\mathfrak{G},\{1,p\}} := \inf_{\substack{\mathbf{v}^1,\ldots,\mathbf{v}^M \in \mathbb{R}^N \\ \forall j,\ \mathrm{supp}(\mathbf{v}^j)=\mathcal{G}_j \\ \sum_{j=1}^M \mathbf{v}^j = \mathbf{x}}} \sum_{j=1}^{M} d_j \|\mathbf{v}^j\|_p \tag{11}$$

where $\|\mathbf{x}\|_p = \left(\sum_{i=1}^N x_i^p\right)^{1/p}$ is the ℓ_p-norm, and d_j are positive weights that can be designed to favor certain groups over others [11]. This norm, also called Latent Group Lasso norm in the literature, can be seen as a weighted instance of the atomic norm described in [8], where the authors leverage convex optimization for signal recovery, but not for model selection.

One can use (11) to find a group-sparse approximation under the chosen group norm

$$\hat{\mathbf{x}} \in \operatorname*{argmin}_{\mathbf{z}\in\mathbb{R}^N} \left\{ \|\mathbf{x}-\mathbf{z}\|_2^2 : \|\mathbf{z}\|_{\mathfrak{G},\{1,p\}} \leq \lambda \right\}, \tag{12}$$

where $\lambda > 0$ controls the trade-off between approximation accuracy and group-sparsity. However, solving (12) does not yield a group-support for $\hat{\mathbf{x}}$: though we can recover one through the decomposition $\{\mathbf{v}^j\}$ used to compute $\|\hat{\mathbf{x}}\|_{\mathfrak{G},\{1,p\}}$, it may not be unique as observed in [11] for $p = 2$. In order to characterize the group-support for $\mathbf{x}$ induced by (11), in [11] the authors define two group-supports for $p = 2$. The *strong group-support* $\check{\mathcal{S}}(\mathbf{x})$ contains the groups that constitute the supports of each decomposition used for computing (11). The *weak group-support* $\mathcal{S}(\mathbf{x})$ is defined using a dual-characterisation of the group norm (11). If $\check{\mathcal{S}}(\mathbf{x}) = \mathcal{S}(\mathbf{x})$, the group-support is uniquely defined. However, [11] observed that for some group structures and signals, even when $\check{\mathcal{S}}(\mathbf{x}) = \mathcal{S}(\mathbf{x})$, the group-support does not capture the minimal group-cover of $\mathbf{x}$.

Hence, the equivalence of ℓ_0 "norm" and ℓ_1 norm minimization [2], [3] in the standard compressive sensing setting does not hold in the group-based sparsity setting. Therefore, even for acyclic group structures, for which we can obtain exact identification of the group support of the approximations via dynamic programming, the convex relaxations are not guaranteed to find the correct group support. We illustrate this case via a simple example in the next section. It remains an open problem to characterize which classes of group structures and signals admit an exact identification via convex relaxations.

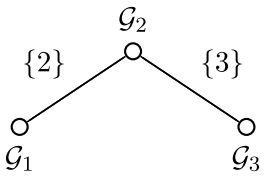

Fig. 4. The intersection graph for the example in Section VI.

VI. Case Study: Discrete vs. Convex Interpretability

The following stylized example illustrates situations that can potentially be encountered in practice. In these cases, the group-support obtained by the convex relaxation will not coincide with the discrete definition of group-cover, while the dynamical programming algorithm of Theorem 1 is able to recover the correct group-cover.

Let $\mathcal{N} = \{1,\ldots,4\}$ and let $\mathfrak{G} = \{\mathcal{G}_1 = \{1,2\},\ \mathcal{G}_2 = \{2,3\},\ \mathcal{G}_3 = \{3,4\}\}$ be an acyclic group structure structure with three groups of equal cardinality. Its intersection graph is represented in Fig. 4. Consider the 2-group sparse signal $\mathbf{x} = [1\ 2\ 2\ 1]^\top$, with minimal group-cover $\mathcal{M}(\mathbf{x}) = \{\mathcal{G}_1, \mathcal{G}_3\}$.

The dynamic program of Theorem 1, with group budget $G = 2$, correctly identifies the groups $\mathcal{G}_1$ and $\mathcal{G}_3$. The TU linear program (6), with $0 < \lambda \leq 2$, also yields the correct group-cover. By contrast, the decomposition obtained via (11) with unitary weights and $p = 2$ is unique, but is not group sparse. In fact, we have $\mathcal{S}(\mathbf{x}) = \check{\mathcal{S}}(\mathbf{x}) = \mathfrak{G}$. We can only obtain the correct group-cover if we use the weights $[1\ d\ 1]$ with $d > \sqrt{\frac{8}{5}}$, that is knowing beforehand that $\mathcal{G}_2$ is irrelevant. These results are obtained directly from (11), exploiting the symmetry in both the group structure and $\mathbf{x}$ to simplify the problem to one variable depending on d.

Remark 2: *This is an example where the correct minimal group-cover exists, but cannot be directly found by the Latent Group Lasso approach. There may also be cases where the minimal group-cover is not unique. We leave to future work, to investigate which of these minimal covers are obtained by the proposed dynamic program and characterize the behavior of relaxations.*

VII. Generalizations

In this section, we first present a generalization of the discrete approximation problem (5) by introducing an additional overall sparsity constraint. Secondly, we show how this generalization encompasses approximation with hierarchical constraints that can be solved exactly via dynamic programming. Finally, we show that the generalized problem can be

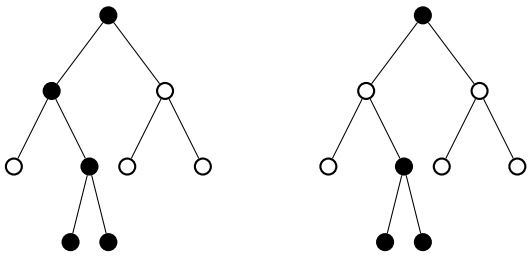

Fig. 5. Hierarchical constraints. Each node represent a variable, in black selected variables. (Left) A valid selection of nodes. (Right) An *invalid* selection of nodes.

relaxed into a linear binary problem and that hierarchical constraints lead to totally unimodular matrices for which there exist efficient polynomial time solvers.

A. Sparsity Within Groups

In many applications, for example genome-wide association studies [17], it is desirable to find approximations that are not only group-sparse, but also sparse in the usual sense (see [37] for an extension of the group lasso). To this end, we generalize our original problem (5) by introducing a sparsity constraint K and allowing to individually select variables within a group; then, one could use such projection step in linear regression frameworks, in order to impose such structure in the final solution. The generalized integer problem then becomes

$$\begin{aligned} \max_{\boldsymbol{\omega}\in\mathbb{B}^M,\ \mathbf{y}\in\mathbb{B}^N} \ & \sum_{i=1}^N y_i x_i^2 \\ \text{subject to } & \mathbf{A}^{\mathfrak{G}}\boldsymbol{\omega} \geq \mathbf{y} \\ & \sum_{i=1}^N y_i \leq K \\ & \sum_{j=1}^M \omega_j \leq G\ . \end{aligned} \tag{13}$$

The problem described above is a generalization of the well-known Weighted Maximum Coverage (WMC) problem. The latter does not have a constraint on the number of indices chosen, so we can simulate it by setting $K = N$. WMC is also well-known to be NP-hard, so that our present problem is also NP-hard, but it turns out that it can be solved in polynomial time for the same group structures that allow to solve (5).

Theorem 2: *Given an acyclic group structure $\mathfrak{G}$ with M groups, a group budget G and a sparsity budget K, Algorithm 1 solves* (13) *in $\mathcal{O}(M^2GK^2)$ time.*

An intuitive description of the algorithm is given in Section VIII-A, together with its pseudocode. The dynamic program is thoroughly described in Appendix A, which also contains the proof of its time complexity.

B. Hierarchical Constraints

The generalized model allows to deal with hierarchical structures, such as regular trees, frequently encountered in image processing (*e.g.*, denoising using wavelet trees). In such cases, we often require to find K-sparse approximations such that the selected variables form a rooted connected subtree of the original tree, see Fig. 5. Given a tree $\mathcal{T}$, the rooted-connected approximation can be cast as the solution of the following discrete problem

$$\max_{\mathbf{y}\in\mathbb{B}^N} \left\{ \sum_{i=1}^N y_i x_i^2 : \operatorname{supp}(\mathbf{y}) \in \mathcal{T}_K \right\}, \tag{14}$$

where $\mathcal{T}_K$ denotes all rooted and connected subtrees of the given tree $\mathcal{T}$ with at most K nodes.

This type of constraints can be represented by a group structure, where for each node in the tree we define a group consisting of that node and all its ancestors. When a group is selected, we require that all its elements are selected as well. We impose an overall sparsity constraint K, while discarding the group constraint G.

Relaxed and greedy approximations have been proposed [38]–[40] for this particular problem. In Section VIII-C, we describe a dynamic program, Algorithm 2, that runs in polynomial time and yields an exact solution, while Appendix B contains a formal description and proofs of its correctness and time-space complexity.

Theorem 3: *Given a tree with N nodes and maximum degree D, Algorithm 2 solves* (14) *in $\mathcal{O}(NK^2D)$ time.*

While preparing the final version of this manuscript, [41] independently proposed a similar dynamic program for tree projections on D-regular trees with time complexity $\mathcal{O}(NKD)$. Following their approach, we improved the time complexity of our algorithm to $\mathcal{O}(NKD)$ for D-regular trees. We also prove that its memory complexity is $\mathcal{O}(N \log_D K)$. A computational comparison of the two methods, both implemented in Matlab, is provided in Section IX, showing that our

Algorithm 1 *Tree-WMC With Sparsity* (TWMCS)

Inputs: Acyclic group structure $\mathfrak{G}$ consisting of M groups defined over N elements, weights of N elements, group budget G, sparsity budget K.

Output: Set of K elements contained in G groups with maximum combined weight.

1: Initialize rooted tree $\mathcal{T}$ with vertex set $\mathfrak{G}$ corresponding to the M groups, edge set $\mathcal{E}$ corresponding to edges between overlapping groups, and root chosen arbitrarily.
2: Compute the *Graph Exploration Rule* using Algorithm 5 (Appendix A-G).
3: Recursively compute the table of optimal values via the *Value Update Rule* using Algorithm 3 (Appendix A-F).
4: *Backtrack* the optimal selection of G groups and K elements from Algorithm 4 (Appendix A-F).

Algorithm 2 *Rooted Connected K-Sparse Trees*

Input: Rooted tree $\mathcal{T}(\mathcal{V}, \mathcal{E}, \text{root})$ of N nodes, weights of nodes, sparsity budget K.

Output: A subtree of $\mathcal{T}$ with the same root as $\mathcal{T}$, having at most K nodes with maximum combined weight.

1: Recursively compute the table of optimal values with the *Value Update Rule* using Algorithm 6 (Appendix B-C).
2: *Backtrack* the optimal selection of at most K nodes using Algorithm 7 (Appendix B-E).

dynamic program can be up to 60 times faster, despite having similar *worst-case* time complexity.

Proposition 4: *The time complexity of Algorithm 2 on D-regular trees is $\mathcal{O}(NKD)$.*

Proposition 5: *The space complexity of Algorithm 2 on D-regular trees is $\mathcal{O}(N \log_D K)$.*

The proofs of both propositions can be found in Appendix B.

C. Additional Relaxations

By relaxing both the group budget and the sparsity budget in (13) into regularization terms, we obtain the following binary linear program

$$\begin{aligned}(\boldsymbol{\omega}^\lambda, \mathbf{y}^\lambda) \in \underset{\boldsymbol{\omega}\in\mathbb{B}^M, \mathbf{y}\in\mathbb{B}^N}{\operatorname{argmax}} \quad & \mathbf{w}^\top \mathbf{u} \\ \text{subject to } \mathbf{u}^\top = [\mathbf{y}^\top \ \boldsymbol{\omega}^\top \ \mathbf{y}^\top] \\ \mathbf{C}\mathbf{u} \leq 0 \end{aligned} \tag{15}$$

where $\mathbf{w}^\top = [x_1^2, \ldots, x_N^2, -\lambda_G \mathbf{1}_M^\top, -\lambda_K \mathbf{1}_N^\top]$ and $\mathbf{C} = [\mathbf{I}_N, -\mathbf{A}^{\mathfrak{G}}, \ \mathbf{0}_N]$ and $\lambda_G, \lambda_K > 0$ are two regularization parameters that indirectly control the number of active groups and the number of selected elements. (15) is of the same form as (7), therefore can be solved in polynomial time if the constraint matrix $\mathbf{C}$ is totally unimodular. Due to its structure, by [28, Proposition 2.1] and that concatenating a matrix of zeros to a TU matrix preserves total unimodularity, $\mathbf{C}$ is totally unimodular if and only if $\mathbf{A}^{\mathfrak{G}}$ is totally unimodular. As a characteristic example of when such cases appear in practice, we provide the following proposition.

Proposition 6: *Hierarchical group structures lead to totally unimodular constraints.*

Proof: We use the fact that a binary matrix is totally unimodular if there exists a permutation of its columns such that in each row the 1s appear consecutively, which is a combination of [28, Corollary 2.10 and Proposition 2.1]. For hierarchical group structures, such permutation is given by a depth-first ordering of the groups. In fact, a variable is included in the group that has it as the leaf and in all the groups that contain its descendants. Given a depth-first ordering of the groups, the groups that contain the descendants of a given node will be consecutive. □

The regularized hierarchical approximation problem, in particular

$$\max_{\mathbf{y}\in\mathbb{B}^N} \left\{ \sum_{i=1}^N y_i x_i^2 - \lambda \|\mathbf{y}\|_0 : \ \operatorname{supp}(\mathbf{y}) \in \mathcal{T}_N \right\}, \tag{16}$$

for $\lambda \geq 0$, has already been addressed by Donoho *et al.* [42] as the *dyadic CART*, which can find a solution in $\mathcal{O}(N)$ time. However, it is not clear how to set the regularization parameter λ in order to obtain solutions with a given sparsity budget K. The *condensing sort and select algorithm* (CSSA) [38], with complexity $\mathcal{O}(N \log N)$, solves problem (16) where the indicator variable $\mathbf{y}$ is relaxed to be continuous in $[0, 1]$ and the penalty term is substituted with the constraint that $\|\mathbf{y}\|_1$ be smaller than a given threshold γ, yielding rooted connected approximations that might have more than K elements. Moreover, applying the heuristic to threshold the final solution to obtain a K-sparse solution does not guarantee to return the best K-sparse tree projection.

VIII. Algorithms

In this section, we provide an outline of our two dynamic programming algorithms for solving problems (13) and (14), respectively. Additional algorithmic considerations can be found in the appendices.

A. Tree-WMC With Sparsity

One of the key contributions of this work is a new dynamic programming algorithm which solves problem (13). Our algorithm addresses the following question: Given N items contained in M groups, and each item associated with a non-negative weight, how can we choose G groups, and K items within these G groups, in order to maximize the sum of weights of the chosen items? This problem is NP-hard since it generalizes the Weighted Maximum Coverage (WMC) problem. However, if the intersection graph is a tree (or a forest), then the problem can be solved in polynomial time, as we briefly describe next and prove in Appendix A. We note that our dynamic programming algorithm can be easily generalized to the case where groups and items have integral costs instead of unit costs, giving us a pseudo-polynomial time scheme.

Broadly speaking, the proposed framework is a dynamic program, *i.e.*, it builds the solution to the global problem from solutions of sub-problems. In our case, the sub-problem involves a subset of the groups. To introduce the key steps of our approach gradually, let us first consider the following idea: Looking at a subset $\mathcal{S}_n$ of groups, one might compute a table of optimal g-group sparse, k-element sparse solutions for all $1 \leq g \leq G, 1 \leq k \leq K$. Then, we might try to extend this table by adding one new group G_{n+1} to $\mathcal{S}_n$, as in standard dynamic programming approaches. Unfortunately, such an approach fails to find the optimal solution. To see this, observe that in order to perform such an extension, it is essential to know whether the new items appearing in G_{n+1} have already been considered for $\mathcal{S}_n$; otherwise, it could lead to double-counting, which happens when G_{n+1} overlaps with some groups in $\mathcal{S}_n$.[1]

To rectify this, we introduce the notion of a *boundary group*, *i.e.*, a group within the current selected subset of groups, that may overlap with future groups. Thus, the boundary groups of $\mathcal{S}_n$ are all the groups in $\mathcal{S}_n$ which overlap with some group in $\mathfrak{G} \setminus \mathcal{S}_n$; see also Figure 6. Our idea is to store a separate table of solutions for each possible selection of boundary groups, *i.e.*, all 2^b combinations for b boundary groups. The fact that we can avoid double-counting in this manner is non-obvious: It relies on an optimal substructure property that we describe in Appendix A-B. Leveraging this property, we show the existence of a *value update rule* (Appendix A-F), which allows us to extend our solutions by incorporating one new

[1] While such argument provides intuition, it does not rigorously prove that the above method could not succeed; we refer the reader to appendix A for a detailed analysis.

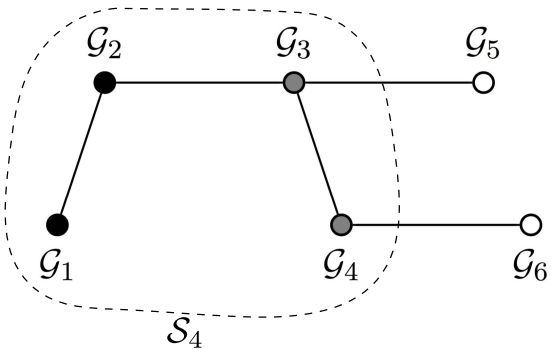

Fig. 6. Boundary groups: In the intersection graph above, the set of explored groups is $\mathcal{S}_4 = \{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \mathcal{G}_4\}$. The boundary groups are $\{\mathcal{G}_3, \mathcal{G}_4\}$.

group at a time. Repeating this rule, we eventually obtain the global optimal solution.

Remark 3: *Sets that are included in one another can be excluded because choosing the larger set would be a strictly dominant strategy, making the smaller set redundant. However, the correctness of the dynamic program is unaffected even if such sets are present, as long as the intersection graph remains acyclic.*

The above procedure adds a factor of 2^b to our running time, which can be exponential.[2] In Appendix A-G, we show that we can design a *graph exploration rule* for tree intersection graphs, which prevents the number of boundary groups from growing too large. In fact, the maximum number of boundary groups, when an acyclic intersection graph is explored using our method, is bounded by $\log_2 M$ for M groups. This further implies that the factor 2^b remains bounded by M throughout our algorithm, which leads to a polynomial running time of $\mathcal{O}(M^2K^2G)$.

The Tree-WMC with Sparsity (TWMCS) pseudocode is provided in Algorithm 1, while Appendices A-G and A-F describe the graph exploration rule and the value update rule in Algorithms 5 and 3, respectively. Once the final table of optimal values has been computed, it is necessary to *backtrack* the algorithm's steps in order to obtain the optimal solution. This backtracking procedure is outlined in Algorithm 4 in Appendix A-F.

B. Tree-WMC Without Sparsity

If there is no sparsity budget K, the dynamic program described in the previous section can be simplified to efficiently solve (5) for acyclic group structures. The first key observation is that it is sufficient to keep a smaller table of optimal values without considering the sparsity variable. Secondly, in the value update step, when the new group is selected, we add *all* its elements after removing any overlap with the boundary groups.

Here, we encounter a potential problem—if there exist groups with a large number of elements, the value update step could add a factor of $\mathcal{O}(N)$ to the complexity. This problem can be easily avoided with some pre-processing, where we combine elements into equivalent groups based on the overlap structure. Since sparsity is no longer a constraint, if two elements are contained in the same set of groups, they can be treated as one element with the combined weight. Due to the tree structure, this will result in at most $\mathcal{O}(M)$ different elements[3] once the pre-processing is complete *i.e.*, same order of elements as number of groups. It can be shown that after this operation, adding element weights will not increase the complexity. The final time complexity of Algorithm 1 in this case is $\mathcal{O}(GM^2)$.

C. Rooted, Connected K-Sparse Trees

We describe here the Dynamic Programming algorithm that solves problem (14). In this case, we are interested in the following question: Given a rooted tree with a non-negative weight associated with each node, how can we pick K nodes that form a rooted, connected subtree in order to maximize the sum of weights of the nodes? By rooted, connected subtree, we mean that the root must be selected, and for any selected node, all of its ancestors up to the root must also be chosen.

An interesting feature of the rooted-connected subtree constraint is the following: Let T_0 be the maximum-weight, K-node, rooted, connected subtree in our graph. For any node X, consider the subset of nodes in T_0 which are descendants of X. Then, these form a rooted, connected subtree at X (call it T_X). Further, if T_X comprises k nodes, then T_X is the maximum-weight, k-node, rooted, connected subtree at X. This suggests a natural choice of sub-problems for dynamic programming.

Briefly, our objective is to compute the optimal weights of all k-node, rooted, connected subtrees at any node X, for each $k \in \{1, \ldots, K\}$. If X is a leaf node, this is trivial. If X is not a leaf node, then we shall inductively assume that these solutions have been computed for all of its subtrees. To combine the subtrees at X, we use the following *Value Update Rule*: The optimal way to choose k nodes from two subtrees equals the optimal way to choose $k - j$ nodes from the first subtree and j nodes from the second subtree, maximized over all $j \in \{0, \ldots, k\}$. A detailed analysis of this algorithm leads to a running time of $\mathcal{O}(NKD)$ for D-regular trees, and $\mathcal{O}(NK^2D)$ for trees which have maximum degree D, but may not be D-regular (see Appendix B-D). Algorithm 2 summarizes the key steps of the dynamic program, whose details and analysis are given in Appendix B.

IX. Pareto Frontier Examples

The purpose of these numerical simulations is to illustrate the limitations of relaxations and of greedy approaches for correctly estimating the G-group cover of an approximation.

A. Acyclic Constraints

We consider the problem of finding a G-group sparse approximation of the wavelet coefficients of a given image,

[2]Indeed, if the factor was always polynomial, we could solve an NP-hard problem.

[3]After pre-processing, there will be at most one element corresponding to each node and each edge of the tree intersection graph.

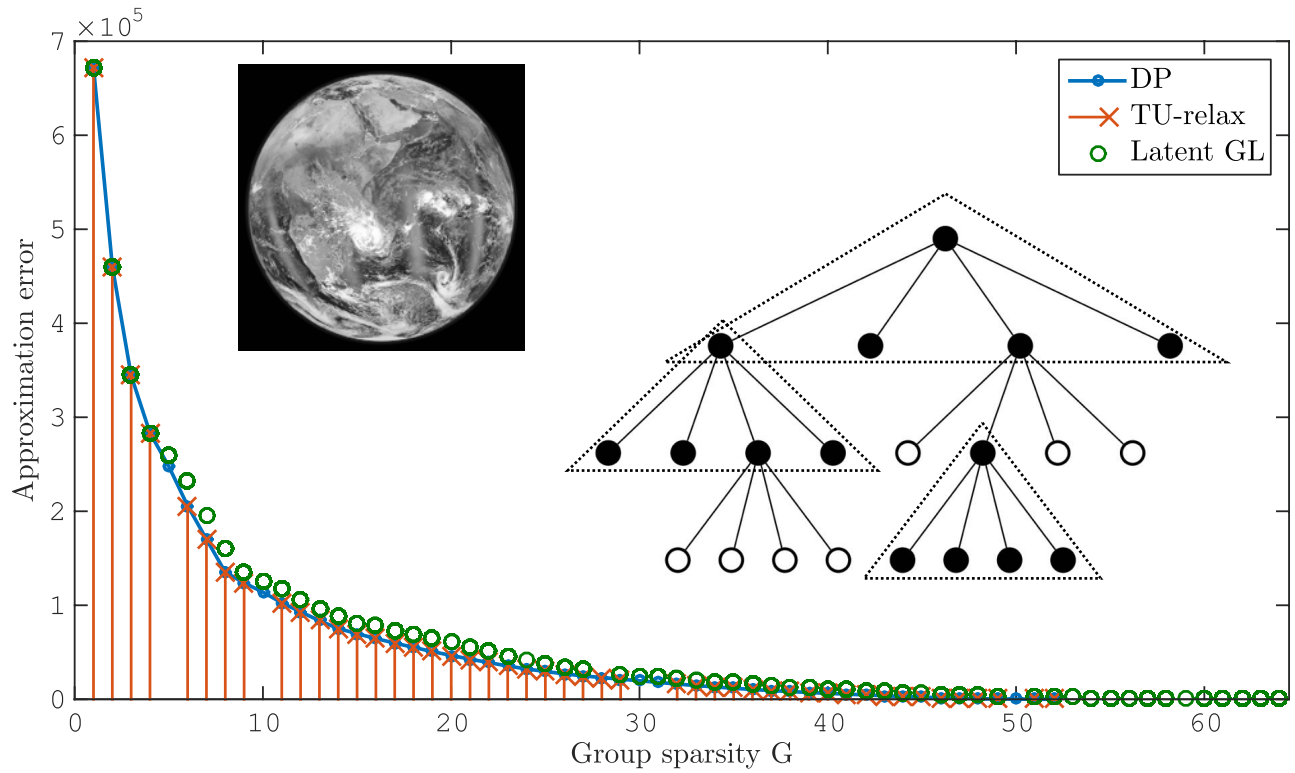

Fig. 7. Insets: (Left) Earth image used for the numerical simulation, before being resized to 16×16 pixels. (Right) Example of allowed support on one of the three wavelet quad-trees: The black circles represent selected variables and the empty circles unselected ones, while the dotted triangles indicate the active groups. Main plot: 2D Wavelet approximation on three quad-trees. The original signal is the wavelet decomposition of the 16×16 pixels Earth image. The blue line is the Pareto frontier of the dynamic program for all group budgets G. Note that, for $G \geq 52$, the approximation error for the dynamic program is zero, while the Latent Group Lasso approach needs to select all 64 groups to yield a zero error approximation. The totally unimodular relaxation only yields the points in the Pareto frontier of the dynamic program that lie on its convex hull.

in our case a view of the Earth from space, see left inset in Fig. 7. We consider a group structure defined over the 2D wavelet tree. The wavelet coefficients of a 2D image can naturally be organized on three regular quad-trees, corresponding to a multi-scale analysis with wavelets oriented vertically, horizontally and diagonally, respectively [1]. We define groups consisting of a node and its four children, therefore each group has 5 elements, apart from the topmost group that contains the scaled average term and the first nodes of each of the three quad-trees. These groups overlap only pairwisely and their intersection graph is a tree itself, therefore leading to a totally unimodular constraint matrix. An example is given in the right inset in Fig. 8. We resize the image to 16×16 pixels and compute its Daubechies-4 wavelet coefficients. At this size, there are 64 groups, but actually 52 are sufficient to cover all the variables, since it is possible to ignore the penultimate layer of groups.

Figures 7 and 8 show the Pareto frontier of the approximation error $\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2$ with respect to the group sparsity G for the proposed dynamic program, Algorithm 1. We also report the approximation error for the solutions obtained via the totally unimodular linear relaxation (TU-relax) (7) and the latent group lasso formulation (Latent GL) (12) with $p = 2$, which we solved with the method proposed in [43]. Fig. 8 shows the performance of StructOMP [22] using the same group structure and of the greedy algorithm for solving the corresponding weighted maximum coverage problem.

We observe that there are points in the Pareto frontier of the dynamic program, for $G = 5, 10, 30, 31, 50$, that are not achievable by the TU relaxation, since they do not belong to its convex hull. Furthermore, the latent group lasso approach often does not yield the optimal selection of groups, leading to a greater approximation error for the same number of active groups and it needs to select all 64 groups in order to achieve zero approximation error. It is interesting to notice that the greedy algorithm outperforms StructOMP (see inset of Fig. 8), but still does not achieve the optimal solutions of the dynamic program. Furthermore, StructOMP needs to select all 64 groups for obtaining zero approximation error, while the greedy algorithm can do with one less, namely 63.

B. Hierarchical Constraints

We now consider the problem of finding a K-sparse approximation of a signal imposing hierarchical constraints. We generate a piecewise constant signal of length $N = 64$, to which we apply the Haar wavelet transformation, yielding a 25-sparse vector of coefficients $\mathbf{x}$ that satisfies hierarchical constraints on a binary tree of depth 5, see Fig. 9(Top).

We compare the proposed dynamic program (DP), Algorithm 2, to the regularized totally unimodular linear

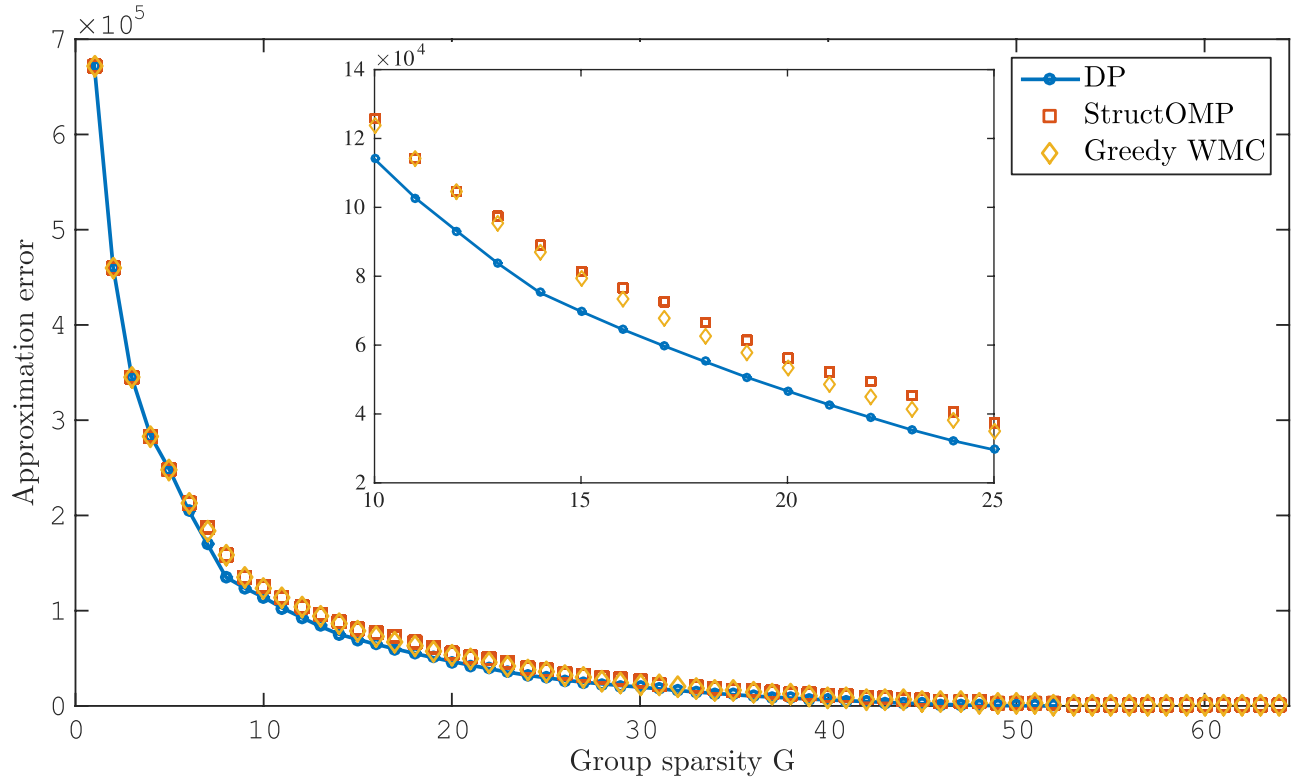

Fig. 8. 2D Wavelet approximation on three quad-trees. The original signal is the wavelet decomposition of the 16×16 pixels Earth image. The blue line is the Pareto frontier of the dynamic program for all group budgets G. Note that for $G \geq 52$ the approximation error for the dynamic program is zero, while StructOmp needs to select all 64 groups to yield a zero error approximation. The greedy algorithm for solving the corresponding weighted maximum coverage problem obtains better solutions than StructOmp, but it still requires 63 groups to yield a zero-error approximation.

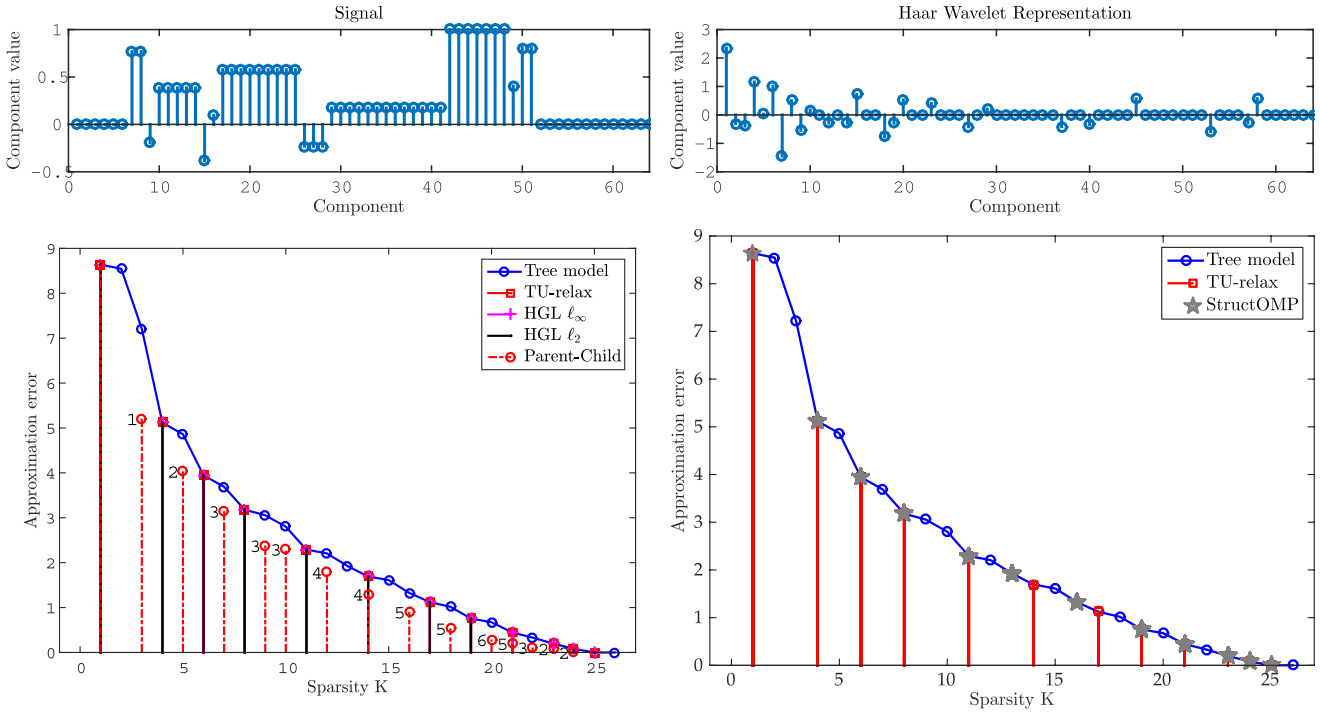

Fig. 9. (Top) Original piecewise constant signal and its Haar wavelet representation. (Bottom) Signal approximation on the binary tree. The original signal is 25-sparse and satisfies hierarchical constraints. The numbers next to the Parent-Child solutions indicate the number of hierarchial constraint violations, *i.e.*, a node is selected but not its parent.

program approach, two convex relaxations that use group-based norms and the StructOMP greedy approach [22]. The first convex relaxation [8] uses the Latent Group Lasso norm (11) with $p = 2$ as a penalty and with groups defined as all parent-child pairs in the tree. We call this approach Parent-Child. This formulation will not enforce all hierarchical

N	Algorithm 2	[41]	Speed-up
2^9	0.007	0.14	20×
2^{10}	0.012	0.29	23×
2^{11}	0.025	0.62	25×
2^{12}	0.048	1.21	25×
2^{13}	0.093	2.55	27×
2^{14}	0.19	5.35	29×
2^{15}	0.37	11.8	32×
2^{16}	0.76	26.4	35×
2^{17}	1.54	66.5	43×
2^{18}	3.14	196	62×

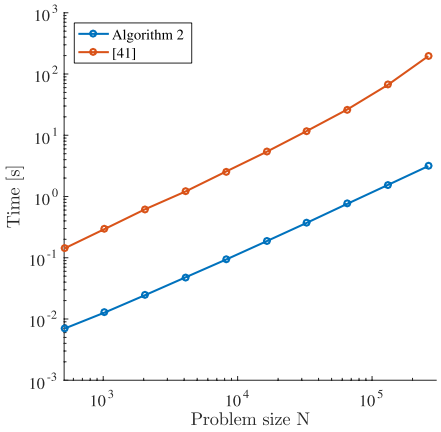

Fig. 10. Running times in seconds of the proposed dynamic program for hierarchical constraints and the one proposed by Cartis and Thompson [41]. The sparsity budget is kept constant to $K = 200$ for all problem sizes.

constraints to be satisfied, but will only favor them. Therefore, we also report the number of hierarchical constraint violations. The second convex relaxation [40] considers a hierarchy of groups where $\mathcal{G}_j$ contains node j and all its descendants. Hierarchical constraints are enforced by the group lasso penalty $\Omega_{GL}(\mathbf{x}) = \sum_{\mathcal{G}\in\mathfrak{G}} \|\mathbf{x}_{\mathcal{G}}\|_p$, where $\mathbf{x}_{\mathcal{G}}$ is the restriction of $\mathbf{x}$ to $\mathcal{G}$, and we assess $p = 2$ and $p = \infty$. We call this method Hierarchical Group Lasso. As shown in [30], solving $\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{x}\|_2^2 + \lambda\Omega_{GL}(\mathbf{x})$, for $p = \infty$, is actually equivalent to solving the totally unimodular relaxation with the same regularization parameter. Once we determine the support of the solution, we assign to the components in the support the values of the corresponding components of the original signal. Finally, for the StructOMP[4] method, we define a block for each node in the tree. The block contains that node and all its ancestors up to the root. By finely varying the regularization parameters for these methods, we obtain solutions with different levels of sparsity.

In Figures 9(Bottom), we show the approximation error $\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2$ as a function of the solution sparsity K for the methods. The values of the DP solutions form the discrete Pareto frontier of the optimization problem controlled by the parameter K, indicated as Tree model in the figure. Note that there are points in the Pareto frontier that do not lie on its convex hull, hence these solutions are not achievable by the TU linear relaxation. As expected, the Hierarchical Group Lasso[5] with $p = \infty$ obtains the same solutions as the TU linear relaxation, while with $p = 2$ it also misses the solutions for $K = 21$ and $K = 23$. The Parent-Child[6] approach achieves more levels of sparsity, but still misses the solutions for $K = 2, 13$ and 15. However, it also violates some of the hierarchical constraints, *i.e.*, we count one violation when one node is selected but not its parent. The StructOMP approach yields only few of the solutions on the Pareto frontier, but without violating any constraints. These observations lead us to conclude that, in some cases, relaxations of the original discrete problem or other greedy approaches might not be able to find the correct group-based interpretation of a signal.

In Fig. 10, we report a computational comparison between our dynamic program, Algorithm 2, and the one independently proposed by Cartis and Thompson [41]. We consider the problem of finding the $K = 200$ sparse rooted connected tree approximation on a binary tree of a signal of length 2^L, with $L = 9, \ldots, 18$, whose components are randomly and uniformly drawn from $[0, 1]$. Despite the two algorithms have the same computational complexity, $\mathcal{O}(NKD)$, and are both implemented in Matlab, our dynamic program is from 20 to 60 times faster.

X. Conclusions

Several applications benefit from group sparse representations. Unfortunately, our main result shows that finding a group-based interpretation of a signal is an NP-hard integer optimization problem. To this end, we characterize group structures for which a dynamical programming algorithm can find a solution in polynomial time and also delineate discrete relaxations for special structures (i.e., totally unimodular constraints) that can obtain correct solutions.

Our examples and numerical simulations show the deficiencies of relaxations, both convex and discrete, and of greedy approaches. We observe that relaxations only recover group-covers that lie in the convex hull of the Pareto frontier determined by the solutions of the original integer problem for different values of the group budget G (and sparsity budget K for the generalized model). This, in turn, implies that convex and non-convex relaxations might miss some important groups or include spurious ones in the group-sparse model selection. We summarize our findings in Fig. 11.

There remain several interesting open questions which deserves further studies. Firstly, an intuitive understanding of under which circumstances the relaxations are able to yield the correct solutions is still missing. Secondly, our analysis implicitly assumes an orthogonal basis for the description of signals. In many machine learning and compressive sensing applications however, the structures in signals emerge only after representing them onto an overcomplete basis,

[4]We used the code provided at http://ranger.uta.edu/~huang/R_StructuredSparsity.htm

[5]We used the code provided at http://spams-devel.gforge.inria.fr/.

[6]We used the algorithm proposed in [43].

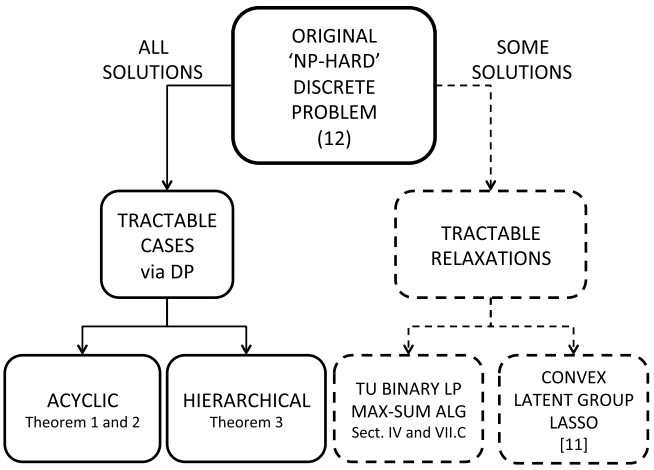

Fig. 11. A summary of tractability of group-based approximations.

e.g., shearlets or sparse coding techniques. Therefore, it is interesting to explore to which extent our results can be generalized to the overcomplete setting.

APPENDIX A
DYNAMICAL PROGRAMMING FOR SOLVING (13) FOR LOOPLESS PAIRWISE OVERLAPPING GROUPS

Here, we give the proof of Theorem 2. The proof of Theorem 1 follows along similar lines. We start by giving an intuitive understanding of the algorithm, followed by a formal description and proofs of correctness and complexity, both in time and space. The pseudocode of the main algorithm is provided in Section VIII.

Problem (13) can be equivalently described by the following problem.

Sparse Group Selection Problem (SGSP): Given a signal $\mathbf{x} \in \mathbb{R}^N$ and a group structure $\mathfrak{G}$ consisting of M groups defined over the index set $\mathcal{N} = \{1, \ldots, N\}$, with each index having an associated (non-negative) weight (*e.g.*, x_i^2, $\forall\ i \in \mathcal{N}$), find the optimal selection of at most K indices, to maximize the sum of their weights, such that the indices are contained in a union of at most G groups. In this paper, we frequently use the term *elements* in place of indices, and use the term *weight of the* i^{th} *element* to refer to the i^{th} entry of the weights vector.[7]

In this form, the problem described above is a generalization of the well-known Weighted Maximum Coverage (WMC) problem, which is NP-hard. In fact WMC is just a special case of SGSP with $K = N$. Although this makes it intractable in general, we show that SGSP has an interesting structure that allows us to build a dynamic program which can obtain the exact solution in polynomial time, for certain special group structures.

A. Our Dynamic Programming Approach: The Intuition

We first present an informal account of the ideas behind our method. The basic idea we use is dynamic programming, *i.e.*, we build the solution to the global optimization problem from solutions to subproblems. For understanding the algorithm for Problem (13), our starting point shall be a simpler dynamic program, that works when the groups are non-overlapping. We give an outline of this algorithm, describe why it fails if applied to the general problem, and provide a way to remedy these issues.

1) Dynamic Program for Non-Overlapping Groups: Let us start with the following problem: Given M *disjoint* groups containing N elements in total, how do we pick G groups and K elements within these G groups, to maximize the total weight of chosen elements? That this problem can be solved efficiently by a dynamic program, is a consequence of the following observation. Suppose that the optimal solution contains g groups and k elements from the first m groups. Then, these groups and elements must represent the optimal selection of g groups and k elements from the first m groups. In other words, if the global optimal solution is projected onto the first m groups, we get a partial optimal solution for the following subproblem: Given the first m groups and the contained elements, how do we pick g groups, and k elements within these g groups, to maximize the total weight?

The above fact suggests that there should be a way to build the global optimal solution from partial solutions. Indeed, there is such an algorithm, and we can define it using induction. Assume that we have computed all partial solutions for the first m groups, *i.e.*, we have computed the optimal way to choose g-groups from among the first m groups, and k-elements within these g groups, for each $g \in \{1, \ldots, G\}$ and each $k \in \{1, \ldots, K\}$. Then we can efficiently extend these partial solutions to $m + 1$ groups, by noting that there are only 2 possibilities for choosing g groups and k elements from the $m + 1$ groups:

1) Do not choose the $m + 1$-th group, *i.e.*, select all the groups (and elements) from the first m groups.
2) Choose the $m + 1$-th group. Select a positive integer $j \in \{0, \ldots, k\}$, and choose j elements from the $m + 1$-th group, and choose $k - j$ elements contained in $g - 1$ groups from the first m groups.

The optimal g-group, k-element selection is then given by maximizing over the $k + 2$ numbers obtained in the above steps. Both these cases are easy to compute, because the values corresponding to m groups are already available from the induction step. Here, we are using the important structural property discussed above, which ensures that while computing an optimal selection for $m+1$ groups, the selection of the first m groups is an optimal solution for the m-group subproblem. Thus, these partial solutions can be used as building blocks to obtain solutions for progressively larger subproblems, until we eventually reach the global optimal solution. In the next section, we look at how this algorithm[8] performs if used naïvely on problem (13).

2) Failure of the Naïve Dynamic Program: Could the dynamic program described above solve the general problem with overlaps? The answer must be negative, since otherwise

[7]Note that, since each element is non-negative, we can assume that the optimal solution will contain the maximum allowed G groups, as well as K elements, except in trivial cases. We will therefore often assume that the optimal solution has exactly G groups and exactly K elements. However, no generality is lost in our theorems by removing this assumption.

[8]To distinguish it from our later algorithm, we call the above naïve dynamic program.

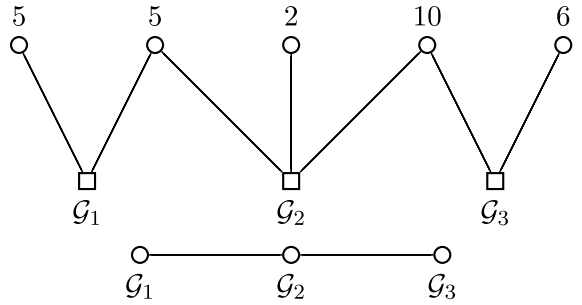

Fig. 12. Failure of naïve DP example: (left) Group structure, where the numbers above the variable nodes indicate their weights. (right) Intersection graph. When we have only seen groups $\mathcal{G}_1$ and $\mathcal{G}_2$, the optimal solution to every subproblem involves choosing $\mathcal{G}_2$. After we explore $\mathcal{G}_3$, the optimal solution for $G = 2, K = 4$ no longer involves selecting $\mathcal{G}_2$.

we could solve an NP-hard problem. Nevertheless, we find it useful to explore the reasons for this failure in some detail, since rectifying these failures will allow us to design efficient algorithms for certain cases. We do this through two examples.

Example 1: *Consider the case of $N = 5$, with the weights being the vector* $[5, 5, 2, 10, 6]$. *For the sake of illustration, let the group structure be $\mathfrak{G} = \{\mathcal{G}_1, \mathcal{G}_2\}$, where $\mathcal{G}_1 = \{1, 2\}$, $\mathcal{G}_2 = \{2, 3, 4\}$. We wish to find the optimal solutions for the cases:*

1) $K = 2, G = 1$*; and*
2) $K = 3, G = 2$.

The optimal solutions can be found simply by observation.

1) *The optimal solution for $K = 2, G = 1$, has weight* 15*, and involves selecting group $\mathcal{G}_2$, and elements* $\{2, 4\}$.
2) *The optimal solution for $K = 3, G = 2$, has weight* 20*, and involves selecting both groups and elements* $\{1, 2, 4\}$.

Example 2: *Consider the case of $N = 5$, with the weights being the vector* $[5, 5, 2, 10, 6]$. *Let the set of groups be $\mathfrak{G} = \{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3\}$, where $\mathcal{G}_1 = \{1, 2\}$, $\mathcal{G}_2 = \{2, 3, 4\}$, $\mathcal{G}_3 = \{4, 5\}$. We wish to find the optimal solution for the case: $K = 4$, $G = 2$. Once again, we can see that the optimal solution involves selecting groups $\mathcal{G}_1$ and $\mathcal{G}_3$, and elements* $\{1, 2, 4, 5\}$*, for a total value of* 26.

In the examples described above, the set of elements and their weights are the same. However, in the first example, any optimal solution for any meaningful values of the parameters G and K, involves $\mathcal{G}_2$. Yet, in the second example, we have a situation where the optimal selection does not involve $\mathcal{G}_2$, see Figure 12. This implies that there is no way to extend partial solutions from the 2-group subproblem to the 3-group subproblem. Hence, the naïve dynamic program cannot solve problem (13).

3) Boundary-Aware DP: As we illustrated above, the naïve DP approach does not work. This is not surprising because of the following reason: When we look at a subset of groups, some of which overlap with as yet unexplored groups, decisions regarding the overlapping groups are difficult to make. This is because the quality of a group in the view of the algorithm may decrease, if the high-weight elements in the group also happen to be contained in another overlapping group, which is seen in the future. While building partial solutions, we then need to consider both possibilities: An overlapping group is either included or excluded from the putative solution. We now introduce some notation which allows us to describe these ideas more concretely.

Our algorithm is heavily based on the intersection graph of the group structure. Thus, we refer to the groups as 'nodes' in our algorithm and in the sequel. Our approach involves exploring the nodes of the intersection graph one at a time and storing a list of optimal values from the explored nodes. These optimal values constitute the optimal weight of a g-group, k-element selection from the explored groups, for all $1 \leq g \leq G$ and for all $1 \leq k \leq K$. Further, we need to store these optimal values for each possible selection of the overlapping groups, so that we do not make decisions concerning such groups at the current step. In terms of the intersection graph, these overlapping groups are simply those nodes which belong to our currently explored set, but are also adjacent to some node which is not in the explored set. We call such nodes *boundary nodes*. Since our algorithm explores the intersection graph, keeping track of all possibilities at the boundary nodes, it may fittingly be called a *boundary-aware Dynamic Program.*

Although this trick of being boundary-aware helps us get the correct solution, it can be expensive. Suppose we have b boundary nodes at a certain step of the algorithm. Then, the table of optimal values we seek to store has size $GK2^b$, which is exponential in b. For an arbitrary intersection graph, this factor can indeed be exponential; for example, a complete graph with M nodes will always have $M - 1$ boundary nodes at the penultimate step. However, if we restrict the intersection graph to be a tree, it turns out there is a way to explore the graph such that the number of boundary nodes in a graph with M nodes is only $O(\log M)$. This property allows our algorithm to run in polynomial time on such graphs.

B. Optimal Substructure

We expose the optimal substructure of this problem below by highlighting two key properties: *Groups-elements dichotomy* property and *independence given the boundary* property. These provide sufficient evidence that an optimal solution to our problem can be efficiently constructed from optimal solutions to subproblems, indicating the correctness of the dynamic programming approach. Further, we will use a slight generalization of the second property in the proof of correctness of our algorithm.

1) *Groups-Elements Dichotomy:* Suppose we had access to an oracle which provides us the set of G groups that comprise the optimal solution to SGSP. Then we can easily recover the full solution using this information, by picking the K largest-weight elements contained in the union of these G groups.

   Interestingly, the above does not hold in the other direction. If the oracle returns the list of K elements contained in the optimal selection, but not the groups, the problem remains hard. Finding the G groups that comprise the optimal solution is equivalent to finding a G-group cover for these K elements, given that such a cover exists.

If we could solve this task in polynomial time, the same algorithm would also solve the NP-Hard *Set Cover* problem in polynomial time.[9] In a certain sense, the above shows that the difficult part of finding the optimal solution is selecting the groups. However, this does not imply that the element sparsity constraint is insignificant. It is easy to create problem instances where even a small change in K significantly changes the optimal selection.

2) *Independence Given the Boundary:* Let $\mathfrak{G}$ be the complete set of groups, and let $\mathcal{S} \subset \mathfrak{G}$ be a subset of these groups. Let $\mathcal{B}(\mathcal{S})$ be the boundary nodes of $\mathcal{S}$, that is the nodes in $\mathcal{S}$ that are connected to nodes in its complement, $\mathcal{S}^c$. Once again, we assume the existence of an oracle who knows the true solution. Suppose this oracle tells us the following information:
   a) The number of groups in $\mathcal{S}$ which are included in the optimal solution. Call this quantity G_1.
   b) The number of elements in the optimal solution, which occur in any of the groups in $\mathcal{S}$. Call this quantity K_1.
   c) The boundary nodes included in the optimal solution.

   Then this information allows us to recover the optimal solution, by solving two independent optimization problems on the sets $\mathcal{S}$ and $\mathcal{S}^c$ respectively.

   For ease of explanation, we refer to the set of boundary nodes included in the optimal selection as the set of 'active boundary nodes', $\mathcal{B}_A(\mathcal{S})$. Note that $\mathcal{B}_A(\mathcal{S})$ is known as it is given to us by the oracle. Further, we call the set of elements included in $\mathcal{B}_A(\mathcal{S})$ the set of active boundary elements, or $\mathcal{E}_A$.

   Recovery Method: In order to recover the global optimal solution, we need to recover the selection of groups and elements in $\mathcal{S}$ and $\mathcal{S}^c$ respectively. We first describe the procedure for $\mathcal{S}$. Consider all possible ways of choosing K_1 elements contained in G_1 groups from $\mathcal{S}$, such that the set of chosen groups in $\mathcal{B}(\mathcal{S})$ exactly matches $\mathcal{B}_A(S)$. Among these choices, the choice which has the maximum total weight of chosen elements gives us the selections of groups and elements in $\mathcal{S}$.

   Now we describe the procedure for $\mathcal{S}^c$. We know that the total number of selected groups in the set $\mathcal{S}^c$ equals $G_2 \triangleq G - G_1$. Similarly, we know that the total number of selected elements, from elements contained *only* in $\mathcal{S}^c$ equals $K_2 \triangleq K - K_1$. We perform a 'cleaning' operation on groups in $\mathcal{S}^c$, where we remove elements in $\mathcal{E}_A$ from these groups. Let the new set of groups thus obtained be called $\tilde{\mathcal{S}}^c$ (note that $\tilde{\mathcal{S}}^c$ is in general not a subset of $\mathfrak{G}$). Then, we can recover the optimal selection of groups and elements in $\mathcal{S}^c$, by finding the maximum-weight G_2-group, K_2-element selection in $\tilde{\mathcal{S}}^c$.

   Proof: The proof of these two statements is straightforward. First, we formally show how to break the true optimal solution into two disjoint components. After this, we argue that the two components constitute optimal solutions to smaller optimization problems. Let us denote the set of groups and elements in the global optimal solution by $\mathfrak{G}^*$ and $\mathcal{E}^*$, respectively. We create two new group-element selections, roughly corresponding to $\mathcal{S}$ and $\mathcal{S}^c$, which we shall denote by $(\mathfrak{G}_1, \mathcal{E}_1)$ and $(\mathfrak{G}_2, \mathcal{E}_2)$ respectively. These two components are constructed as follows:
   a) The set of selected groups in $\mathcal{S}$ and $\mathcal{S}^c$ are already disjoint, so these are directly assigned to $\mathfrak{G}_1$ and $\mathfrak{G}_2$ respectively.
   b) For any element in $\mathcal{E}^*$ which occurs only in (groups in) $\mathcal{S}$, assign it to $\mathcal{E}_1$.
   c) For any element in $\mathcal{E}^*$ which occurs only in $\mathcal{S}^c$, assign it to $\mathcal{E}_2$.
   d) For any element in $\mathcal{E}^*$ which occurs in $\mathcal{S}$ as well as $\mathcal{S}^c$ (and hence in $\mathcal{B}(\mathcal{S})$), first try to assign it to $\mathcal{E}_1$. That is, check if this element is contained in $\mathfrak{G}_1$, and if so assign the element to $\mathcal{E}_1$. If not, we assign it to $\mathcal{E}_2$.

   We can verify the following properties:
   i. $\mathfrak{G}_1$ and $\mathfrak{G}_2$ form a partition of $\mathfrak{G}^*$, and similarly $\mathcal{E}_1$ and $\mathcal{E}_2$ form a partition of $\mathcal{E}^*$.
   ii. $(\mathfrak{G}_1, \mathcal{E}_1)$, $(\mathfrak{G}_2, \mathcal{E}_2)$ represent valid group-element selections over the sets of groups $\mathcal{S}$ and $\mathcal{S}^c$ respectively (i.e. $\mathcal{E}_1$ is contained in the union of groups in $\mathfrak{G}_1$, and similarly $\mathcal{E}_2$ is contained in $\mathfrak{G}_2$.)
   iii. $(\mathfrak{G}_2, \mathcal{E}_2)$ can also be thought of as a valid selection over the set $\tilde{\mathcal{S}}^c$. This is because our definition of the components assigns any element in the active boundary groups to $\mathcal{E}_1$.
   iv. $|\mathfrak{G}_1| = G_1$, $|\mathcal{E}_1| = K_1$, $|\mathfrak{G}_2| = G_2$, $|\mathcal{E}_2| = K_2$, where G_1, K_1, G_2, K_2 are defined as above.

   We are now ready to prove the correctness of the recovery method. Let us first consider $\mathcal{S}^c$. Suppose that contrary to our claim above, $(\mathfrak{G}_2, \mathcal{E}_2)$ does not constitute an optimal solution for $\tilde{\mathcal{S}}^c$, i.e., there exists another G_2-group, K_2-element selection on $\tilde{\mathcal{S}}^c$, namely $(\mathfrak{G}'_2, \mathcal{E}'_2)$, such that the total weight of elements in $\mathcal{E}'_2$, i.e., $weight(\mathcal{E}'_2)$, is larger than that in $\mathcal{E}_2$. Then we could improve the optimal solution by considering the group-element selection $(\mathfrak{G}_1 \cup \mathfrak{G}'_2, \mathcal{E}_1 \cup \mathcal{E}'_2)$. Note that it is impossible for $\mathcal{E}_1$ and $\mathcal{E}'_2$ to select the same element twice, and hence the above represents a valid G-group, K-element selection over $\mathfrak{G}$. Also, since $weight(\mathcal{E}'_2) > weight(\mathcal{E}_2)$, we have $weight(\mathcal{E}_1 \cup \mathcal{E}'_2) > weight(\mathcal{E}_1 \cup \mathcal{E}_2) = weight(\mathcal{E}^*)$, so this is an improvement over the selection $(\mathfrak{G}^*, \mathcal{E}^*)$. But this contradicts the optimality of the latter solution. Hence, our assumption must be false, *i.e.*, $(\mathfrak{G}_2, \mathcal{E}_2)$ comprises an optimal solution to the G_2-group, K_2-element selection problem for $\tilde{\mathcal{S}}^c$. An identical argument shows that $(\mathfrak{G}_1, \mathcal{E}_1)$ represents an optimal G_1-group, K_1-element selection over $\mathcal{S}$, among all group-element selections for which the set of chosen nodes from $\mathcal{B}(\mathcal{S})$ equals exactly $\mathcal{B}_A(\mathcal{S})$. This proves the correctness of our recovery method. □

[9]The reduction described here is not formal. It is possible that the additional structure possessed by the optimal solution would allow us to recover the groups in polynomial time. However, there seems to be no clear way to use this additional structure, so the only obvious way to recover the groups is to solve a set-cover problem, which is NP-hard.

C. Overview of Our Algorithm

Our algorithm explores the acyclic intersection graph one node at a time, storing the optimal solution among the visited

Algorithm 3 *Value Update Rule* for Tree-WMC With Sparsity

Inputs: Rooted tree $\mathcal{T}(\mathfrak{G}, \mathcal{E}, \text{root})$, group budget G, sparsity budget K.
Output: Table of optimal values.
Initialize: $\mathfrak{G} = (\mathcal{G}_1, \ldots, \mathcal{G}_M)$ where $(\mathcal{G}_1, \ldots, \mathcal{G}_M) = Explore(\mathcal{T}(\text{root}))$ (see Algorithm 5).
Initialize: Explored Set $\mathcal{S}_0 = \emptyset$, Boundary Set Vector $b_0 = \emptyset$, Boundary Indicator Vector $\mathbf{I}_{b_0} = \emptyset$, Table of Values $F(\mathcal{S}_0, g, k, b_0, \mathbf{I}_{b_0}) = 0$, Backtracking Table $L(\mathcal{S}_0, g, k, b_0, \mathbf{I}_{b_0}) = 0, \forall\ 1 \le g \le G, 1 \le k \le K$
Initialize: sum of weight of $1 \le l \le |\mathcal{S}|$ largest elements in set $\mathcal{S}$ is $H(\mathcal{S}, l)$.
For $i \in \{1, \ldots, M\}$
  $\mathcal{S}_i = \mathcal{S}_{i-1} \cup \mathcal{G}_i$
  Initialize: b_i as set of nodes $\{\mathcal{G}_j \in \mathcal{S}_i : (\mathcal{G}_j, \mathcal{G}_k) \in \mathcal{E}$ for any $\mathcal{G}_k \in \mathfrak{G} \setminus \mathcal{S}_i\}$.
    Update Table
  For $g \in \{1, \ldots, G\}, k \in \{1, \ldots, K\}, \mathbf{I}_{b_{i-1}} \in \{0, 1\}^{|b_{i-1}|}$
    $\overline{\mathcal{G}}_i = \mathcal{G}_i \setminus \left(\bigcup_{j : \mathbf{I}_{b_{i-1}}(j) = 1} b_{i-1}(j) \right)$
    $F(S_i, g, k, \{b_{i-1}, \mathcal{G}_i\}, \{\mathbf{I}_{b_{i-1}}, 0\}) = F(S_{i-1}, g, k, b_{i-1}, \mathbf{I}_{b_{i-1}})$
    $F(S_i, g, k, \{b_{i-1}, \mathcal{G}_i\}, \{\mathbf{I}_{b_{i-1}}, 1\}) = \max_{1 \le l \le k} F(S_{i-1}, g-1, k-l, b_{i-1}, \mathbf{I}_{b_{i-1}}) + H(\overline{\mathcal{G}}_i, l)$
    $L(S_i, g, k, \{b_{i-1}, \mathcal{G}_i\}, \{\mathbf{I}_{b_{i-1}}, 1\}) = \arg\max_{1 \le l \le k} F(S_{i-1}, g-1, k-l, b_{i-1}, \mathbf{I}_{b_{i-1}}) + H(\overline{\mathcal{G}}_i, l)$
  End For
  Condense Table
  Initialize: $b = b_{i-1}$
  For $\mathcal{G}_j \in b_i \setminus b_{i-1}$
    $b = b \setminus \mathcal{G}_j$
    For $g \in \{1, \ldots, G\}, k \in \{1, \ldots, K\}, \mathbf{I}_b \in \{0, 1\}^{|b|}$
      $F(S_i, g, k, b, \mathbf{I}_b) = \max\left(F(S_{i-1}, g, k, \{b, \mathcal{G}_j\}, \{\mathbf{I}_b, 0\}), F(S_{i-1}, g, k, \{b, \mathcal{G}_j\}, \{\mathbf{I}_b, 1\})\right)$
    End For
  End For
End For
Return: F, L

nodes and eventually leading to the optimal solution for the entire graph. Its pseudocode is provided in Section VIII, Algorithm 1. It is based on two rules: the *Value Update Rule* and the *Graph Exploration Rule*.

1) *Graph Exploration Rule*: This rule takes as input a given rooted tree graph, and outputs a sequence of nodes to explore in order, so as to minimize the number of encountered boundary nodes. See Algorithm 5. The root can be chosen arbitrarily.
2) *Value Update Rule*: The Value Update Rule determines how to update the list of optimal values when we explore a new node. See Algorithm 3.

We will focus most of our attention on the *value update rule*, and deal with the *graph exploration rule* only toward the end. Before we describe these two rules, we define the table of optimal values maintained by our algorithm, as well as some notation to be used in our subsequent discussions.

D. Table of Optimal Values

We describe the set of optimal solutions stored by our Table of optimal values. Abstractly, this table can be thought of as a mathematical function with 5 different parameters. These are described below:

- **Explored Set** : $\mathcal{S} \subseteq \mathfrak{G}$
  This is any subset of nodes of the intersection graph. It represents the set of nodes currently visited by our algorithm.
- **Group Count** : $g \in \{1, 2, \ldots, G\}$.
  This is the maximum number of groups we are allowed to select.
- **Element Count** : $k \in \{1, 2, \ldots, K\}$.
  This is the maximum number of elements we are allowed to select.
- **Boundary Set Vector** : $\mathbf{b} = (b_1, b_2, \ldots, b_B),\ b_i \in \mathcal{S}\ \forall i$, with $B = \dim(\mathbf{b})$.
  This is any subset of the explored set $\mathcal{S}$, represented in vector form. We allow $\mathbf{b}$ to be an *empty vector*, which we denote by $\emptyset$.[10]
- **Boundary Indicator Vector** : $\mathbf{I_b} \in \{0, 1\}^B$.
  This is a binary vector of size B. Given a boundary set vector, $\mathbf{b}$, for each $i \in \{1, \ldots, B\}$, the i-th component of $\mathbf{I_b}$ is either 0 or 1, representing whether the group $\mathbf{b}_i$ is excluded or selected in the optimal selection. We also allow $\mathbf{I_b}$ to be an empty vector.

Then, $\mathbf{F}(\mathcal{S}, \mathbf{g}, \mathbf{k}, \mathbf{b}, \mathbf{I_b})$ represents the maximum weight obtainable by selecting at most k elements contained in a union of at most g groups from the set $\mathcal{S}$, with the choice of selections among the set of boundary nodes $\mathbf{b}$ given by $\mathbf{I_b}$. This function is defined for the entire range of its arguments mentioned above.

Although the function is defined for all $\mathcal{S} \subseteq \mathfrak{G}$, in practice we explore the nodes one at a time, in serial order. Thus, we

[10] We do not give a precise definition of empty vector in this text. Informally, it can be thought of as a vector of 0 elements, very similar to an empty set.

only need to keep track of M different sets of explored nodes, where the i-th set, $\mathcal{S}_i$, consists of groups $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_i$. Furthermore, we only see M different sets of boundary nodes for a given intersection graph viz., $\mathcal{B}(\mathcal{S}_i)$ for all $i \in \{1, \ldots, M\}$. In certain intermediate steps we shall find it convenient to use in place of $\mathcal{B}(\mathcal{S}_i)$, a different set than the actual set of boundary nodes.

If we fix $\mathcal{S} = \mathcal{S}_i$ and $\mathbf{b} = \mathbf{b}_i$[11] and vary other parameters over their respective ranges, we obtain the complete list of values stored by our algorithm at the i-th step. Note that the number of such stored values equals $G \cdot K \cdot 2^B$, with $B = \dim(\mathbf{b})$.

E. Data Format and Notation

Without loss of generality, we can assume that each group has no more than K elements. This is because no element besides the top K will ever be selected in the optimal solution. Further, we will assume that the indices in each group are specified in decreasing order of weights.

In case the above assumptions are not met a-priori, we can do some preprocessing on the given data. Since we know that each group consists of at most N elements, we can pick the largest K elements and then sort them in $O(N + K \log N)$ time.[12] Since we need to do this for each one of the M groups, this leads to a total complexity of $O(MN + MK \log N)$. While describing the complexity of our main algorithm, we will assume that the groups are already represented in the above canonical form. Hence, we will not consider the above term in our expression for time complexity. Another reason for neglecting this term is that it is unlikely to be relevant in practice, since the running time of the algorithm would be the dominant term, unless N is extraordinarily large. Next, we formally define some notation that we use in our description of the value update rule.

1) Concatenation Operator: Given two vectors $\mathbf{x}$ and $\mathbf{y}$ of lengths m and n respectively, we define the vector '$\mathbf{x}$ concatenated with $\mathbf{y}$', written as $\mathbf{x}.\mathbf{y}$, to be an $m + n$-length vector which consists of entries of $\mathbf{x}$ followed by entries of $\mathbf{y}$.

2) Best-k Operator: We define a function $H(\mathcal{S}, k)$ to represent the optimal value for choosing k elements from a set $\mathcal{S}$. The set $\mathcal{S}$ could be a single group, a union of groups, or any well-defined collection of elements. As noted earlier, $H(\mathcal{S}, k)$ simply equals the sum of the k largest weight elements in $\mathcal{S}$.

F. Value Update Rule

We shall formally describe the Value Update Rule in this section, see also the pseudocode of Algorithm 3. This determines how to find the optimal solution to problem (13), which is represented by the value: $F(\mathfrak{G}, G, K, \emptyset, \emptyset)$.

Base Case: We start with $\mathcal{S}_0 = \emptyset$. For this case, all values of F are set to 0: $F(\emptyset, g, k, \emptyset, \emptyset) = 0 \ \forall g, k$.

Update: The update case describes how to recompute the list of optimal values when we explore a new node. We shall apply this rule a total of M times, exploring one new node from the graph each time, and updating our table of values. At the end, we can simply read off the solution from the appropriate entry of the table.

Since we explore the nodes in serial order, at the i-th step, our explored set will consist of nodes $1, 2, \ldots, i$. As mentioned earlier, we denote our explored set after the i-th step as $\mathcal{S}_i$, and the boundary set vector at this time as $\mathbf{b}_i$. We use the notation $\mathcal{G}_j$ to refer to the j-th group, which is also the j-th node of the intersection graph as per our chosen ordering. At the end of the i-th step, we assume that we have stored the values of F for the explored set $\mathcal{S}_i$ and boundary set vector $\mathbf{b}_i$ for each possible value of parameters g, k, and the indicator variable $\mathbf{I}_{\mathbf{b}_i}$, in their respective ranges. Thus, the following values are available to us:

$$F(\mathcal{S}_i, g, k, \mathbf{b}_i, \mathbf{I}_{\mathbf{b}_i}) \quad \text{for all } 1 \le g \le G, 1 \le k \le K \text{ and } \mathbf{I}_{\mathbf{b}_i} \in \{0,1\}^{B_i},$$

where $B_i = \dim(\mathbf{b}_i)$. Our objective is to extend these values by exploring the $i + 1$-th node. In other words, defining $\mathcal{S}_{i+1} \triangleq \mathcal{S}_i \cup \{\mathcal{G}_{i+1}\}$, we wish to obtain the following set of values:

$$F(\mathcal{S}_{i+1}, g, k, \mathbf{b}_{i+1}, \mathbf{I}_{\mathbf{b}_{i+1}}) \quad \text{for all } 1 \le g \le G, 1 \le k \le K \text{ and } \mathbf{I}_{\mathbf{b}_{i+1}} \in \{0,1\}^{B_{i+1}},$$

where $\mathbf{b}_{i+1}$ represents the boundary nodes at time $i + 1$ in vector form.

We obtain these numbers in the following manner. When we first consider node $i+1$, we treat it as a new boundary node and compute the optimal values for it being included or excluded from the putative solution. After this, we test for boundary nodes that have fallen into the interior of the explored set. For these redundant boundary nodes, we no longer need to store two separate numbers for the node being included or excluded, so we condense these into a single value. Our update rule thus consists of 3 steps.

1) *The new node is excluded.*
   In this case, we are computing the optimal value for selecting k elements contained in a union of g groups among the first $(i + 1)$ groups when the $(i + 1)$-th group is not selected, and the groups in $\mathcal{B}_i$ are selected as per the indicator variables. Since the $(i + 1)$-th group is not chosen, all our groups and elements must be chosen from among the first i groups, with the same restrictions on the choice of boundary nodes. Hence, all optimal values for this case are equal to the corresponding values for $\mathcal{S}_i$.
   $$F(\mathcal{S}_{i+1}, g, k, \mathbf{b}_i.(\mathcal{G}_{i+1}), \mathbf{I}_{\mathbf{b}_i}.(0)) = F(\mathcal{S}_i, g, k, \mathbf{b}_i, \mathbf{I}_{\mathbf{b}_i})$$
   for all $1 \le g \le G$ and $1 \le k \le K$ and all $\mathbf{I}_{\mathbf{b}_i} \in \{0,1\}^{B_i}$.
2) Case (a): *The new node is included and does not overlap with any explored node.*
   In this case, we are computing the optimal values for the case when the $(i + 1)$-th node is selected. Hence we can choose at most $g-1$ nodes from the first i nodes. We first compute the sum of the optimal value for choosing the best ℓ elements from the new node and the optimal value

[11] Technically $\mathbf{b}_i$ is a vector, and involves both a set of elements and an ordering over the elements. But this ordering is really a matter of notation; we will care only about the set of boundary nodes, and not the order, in our algorithm.

[12] This can be done by building a max-heap of all N elements and then extracting the topmost element K times.

for choosing $k-\ell$ elements from $g-1$ nodes in S_i, for any ℓ such that $1 \le \ell \le k$. Then, the new optimal value for each g and k is given by taking the maximum of these sums over ℓ. To ensure that our optimal values are computed with selections of nodes in B_i being specified by the indicator variables, we use the same values of indicators when computing the second term in the above sum.

$$F(\mathcal{S}_{i+1}, g, k, \mathbf{b}_i.(\mathcal{G}_{i+1}), \mathbf{I}_{\mathbf{b}_i}.(1)) = \max_{1\le\ell\le k} \left\{F(\mathcal{S}_i, g-1, k-\ell, \mathbf{b}_i, \mathbf{I}_{\mathbf{b}_i}) + H(\mathcal{G}_{i+1}, \ell)\right\}$$

for all $1 \le g \le G$ and $1 \le k \le K$ and all $\mathbf{I}_{\mathbf{b}_i} \in \{0,1\}^{B_i}$.

2) Case (b): *The new node is included but overlaps with some explored nodes*.
The update rule is the same as for case (a), but the elements in the region of overlap between the new node and the *selected* explored nodes must not be considered as being part of the new node. For this step, we need to know exactly which nodes have been chosen while computing an optimal value. This is the reason why we need to store separate values for each boundary node.

$$F(\mathcal{S}_{i+1}, g, k, \mathbf{b}_i.(\mathcal{G}_{i+1}), \mathbf{I}_{\mathbf{b}_i}.(1)) = \max_{1\le\ell\le k} \left\{F(\mathcal{S}_i, g-1, k-\ell, \mathbf{b}_i, \mathbf{I}_{\mathbf{b}_i}) + H(\mathcal{C}, \ell)\right\}$$

for all $1 \le g \le G$ and $1 \le k \le K$ and all $I_{B_i} \in \{0,1\}^{B_i}$, where

$$\mathcal{C} \triangleq \mathcal{G}_{i+1} \setminus \bigcup_{\substack{j\in\{1,\ldots,B_i\}\\ \mathbf{I}_{\mathbf{b}_i}(j)=1}} \mathbf{b}_i(j)$$

That is we "clean" $\mathcal{G}_{i+1}$ of the overlap with the currently selected boundary nodes.

3) *Condensation.*
After performing the above steps, the number of stored values will be doubled. We can reduce them: for each boundary node which has fallen into the interior of the explored nodes, we combine the optimal values for it being selected or excluded, into a single value by taking the larger of the two values. Each such operation reduces the number of stored values by half and we perform it after each value update. Unlike the earlier steps, this step may have to be performed multiple times in a single update.
Suppose $\mathbf{b}'_i$ is the current boundary set vector for which we have maintained optimal values. Suppose $\mathcal{G}_j$ is a node in $\mathbf{b}'_i$ which is not present in $\mathbf{b}_{i+1}$. For notational convenience, we will now assume that the group $\mathcal{G}_j$ has been moved to the end of the $\mathbf{b}'_i$ vector. Define $\mathbf{b}''_i$ to be the vector of length $\dim(\mathbf{b}'_i) - 1$, consisting of all entries of $\mathbf{b}'_i$ except the last. Thus, we can write $\mathbf{b}'_i = \mathbf{b}''_i.\{\mathcal{G}_j\}$. Then we can reduce the boundary set vector from $\mathbf{b}'_i$ to $\mathbf{b}''_i$, as follows:

$$F(\mathcal{S}_{i+1}, g, k, \mathbf{b}''_i, \mathbf{I}_{\mathbf{b}''_i}) = \max\{F(\mathcal{S}_{i+1}, g, k, \mathbf{b}'_i, \mathbf{I}_{\mathbf{b}''_i}.(0)), F(\mathcal{S}_{i+1}, g, k, \mathbf{b}'_i, \mathbf{I}_{\mathbf{b}''_i}.(1))\}$$

for all $1 \le g \le G$ and $1 \le k \le K$ and for all $\mathbf{I}_{\mathbf{b}''_i} \in \{0,1\}^{B''_i}$, where $B''_i = \dim(\mathbf{b}''_i)$.

Proof of Correctness: The correctness of our algorithm relies on the correctness of the value update rule. Below, we argue for the correctness of this rule for each of its 3 steps.

- Step 1: The correctness of this step is self-evident.
- Step 2, case (a): Since this step is a special case of step 2, case (b), it is sufficient to prove correctness of the latter.
- Step 2, case (b): We prove the correctness of this step using the optimal substructure property 2 described in section A-B.
  Our task is to find the optimal selection of g-groups and k-elements from the set $\mathcal{S}_{i+1} \triangleq \mathcal{S}_i \cup \mathcal{G}_{i+1}$, when $\mathcal{G}_{i+1}$ is selected, and nodes in $\mathbf{b}_i$ are selected according to $\mathbf{I}_{\mathbf{b}_i}$. We now consider only the graph consisting of nodes in $\mathcal{S}_{i+1}$. With reference to the substructure property, choose the set $\mathcal{S}$ to be equal to $\mathcal{S}_i$. Critically, note that all groups in $\mathcal{B}(\mathcal{S})$ are contained in $\mathbf{b}_i$, and thus we store optimal values separately for these.
  Although the substructure property 2 was derived on a graph with no additional information, it is equally well-applicable when certain groups (such as $\mathcal{G}_{i+1}$, and groups in $\mathbf{b}_i$) are constrained to be selected or excluded in the optimal solution. This property had three preconditions, one of which was the knowledge of boundary nodes in the optimal solution. This is trivially true, since in this particular optimization problem, the selection of groups in $\mathbf{b}_i$ is already fixed by $\mathbf{I}_{\mathbf{b}_i}$. Then, the property shows us that if we also know the number of groups and elements chosen from the two parts of the graph, we can recover the optimal solution over $\mathcal{S}_{i+1}$ by solving two separate optimization problems over $\mathcal{S}_i$ and $\mathcal{G}_{i+1}$ respectively.
  Here, we know that exactly $g-1$ groups must be selected from $\mathcal{S}_i$, and (obviously) one group chosen from $\{\mathcal{G}_{i+1}\}$. However, we do not know the number of elements chosen from $\mathcal{S}_i$. Hence, we consider all possibilities by varying a parameter ℓ for the number of selected elements contained exclusively in $\mathcal{G}_{i+1}$, from 1 up to k. More precisely, ℓ represents the number of elements chosen from $\mathcal{C}$, where $\mathcal{C}$ is the set of elements obtained by cleaning $\mathcal{G}_{i+1}$ of overlap with active boundary nodes in $\mathbf{b}_i$. This leads us to solve two independent optimization problems - find the best selection of ℓ elements from $\mathcal{C}$, and the best $(g-1)$-group, $(k-\ell)$-element selection from $\mathcal{S}_i$, respecting boundary node constraints.
  Solving the optimization problem over $\mathcal{C}$ is trivial: simply choose the top ℓ elements. Solving the problem over $\mathcal{S}_i$ need not actually be carried out, since we have already stored all the relevant optimal solutions in the previous step. This value is stored in the table of optimal values, in the entry $F(\mathcal{S}_i, g-1, k-\ell, \mathbf{b}_i, \mathbf{I}_{\mathbf{b}_i})$. Thus, by maximizing the sum of these optimal values and the best-ℓ selection in $\mathcal{C}$, over all ℓ from 1 up to k, we obtain the optimal solutions for $\mathcal{S}_{i+1}$.
- Step 3: This is the condensation step. The correctness of this step follows from the interpretation of the objective function—$F(\mathcal{S}, g, k, \mathbf{b}, \mathbf{I}_{\mathbf{b}})$ represents the optimal values

for g-group k-element selections, when the choices of groups in $\mathbf{b}$ are fixed by $\mathbf{I}_{\mathbf{b}}$. Thus, for groups that are not in $\mathbf{b}$, we need to consider both whether the node is included or excluded. Therefore, in order to remove a node from the set $\mathbf{b}$, we simply take the maximum value of the two cases.

Running Time: The running time of our algorithm is determined by two steps - Value Update rule and the Graph Exploration algorithm. As we explain later, the exploration rule can be implemented independently and is computationally much faster, so the time complexity is determined by the value update rule. We analyze the complexity of each step of the update rule below.

Complexity of Step 1: All optimal values for this case are simply the optimal values computed before the node is explored. Thus, the update in this case corresponds simply to a table-copying operation. In fact, this copying can be avoided entirely by some clever bookkeeping; all we need to do is remember where the appropriate values are stored in memory. Thus, this step is very inexpensive from a computational point of view.

Complexity of Step 2, Case (a): Observe that the total number of values to be computed on the LHS of the update rule equals $GK2^{B_i}$. To compute one such value, we need to take a maximum over K different numbers on the RHS. We will show that each of these numbers can effectively be obtained in $O(1)$ time. Computing one of these numbers involves the sum of two terms. The first term is an optimal value that is already stored, so it merely involves a table lookup. The second term, $H(\mathcal{G}_{i+1}, \ell)$ involves taking the sum of ℓ largest numbers in the group $\mathcal{G}_{i+1}$. Since the elements in $\mathcal{G}_i$ are described to us in descending order of weights (by assumption), this is equivalent to finding the sum of the first ℓ elements. Since each successive sum differs from the previous sum in only one element, we can compute each sum by doing just one additional operation. Hence, computing the K different numbers on the RHS takes only $\mathcal{O}(K)$ time. Combining this with the total number of values on the LHS, gives us an expression for complexity as $\mathcal{O}(G\,K^2\,2^{B_i})$.

Complexity of Step 2, Case (b): The new operation that we need to perform here, compared to case (a), is the "cleaning" operation on the $i+1$-th node. This operation is independent of the parameters g, k, ℓ, and depends only on the indicator variables $\mathbf{I}_{\mathbf{b}_i}$. Hence, we can perform our updates by first fixing $\mathbf{I}_{\mathbf{b}_i}$, and then varying g and k. In this way we do the cleaning operation a total of 2^{B_i} times. The time required for the cleaning operation is equal to the time required to go through each of the K elements in $\mathcal{G}_{i+1}$, and checking whether the element is also contained in any of the groups whose indicator variable is set to 1. By doing some simple preprocessing (*e.g.*, sorting indices in some canonical order), checking membership of an element in a group can be done in $\mathcal{O}(\log_2 K)$ time, by binary search. Thus, the time required for one cleaning operation is $\mathcal{O}(B_i K \log K)$. Hence the total time required for all cleaning operations in one step equals $\mathcal{O}(2^{B_i} B_i K \log K)$. Combining this with the expression obtained in step 2, case (a), the time complexity of this update step equals $\mathcal{O}(GK^2 2^{B_i} + K B_i 2^{B_i} \log K)$.

Complexity of Step 3: Since condensation removes an explored node from the boundary set forever, it will have to be performed at most M times in the entire algorithm. Since the set of boundary nodes at each step is fully determined by the intersection graph and the exploration ordering, these can be precomputed without significant time cost. Hence, we assume these are available to us and ignore their complexity. Then the complexity of a single condensation step is determined only by the number of values that need to be condensed, and is given by $\mathcal{O}(GK2^{B_i'})$, which also equals $\mathcal{O}(GK2^{B_i})$.

Overall Time Complexity: Among the above, the most expensive case is step 2, case (b). The complexity of this step as obtained earlier equals $\mathcal{O}(GK^2 2^{B_i} + K B_i 2^{B_i} \log K)$, for the $i+1$-th value update. We need to perform this step M times, with the parameter i varying from 0 to $M-1$ in the above expression.

Let B^* be the maximum number of boundary nodes encountered by the algorithm at any step, *i.e.*, $B^* = \max_i B_i$. Then the running time of our update algorithm is bounded by $\mathcal{O}(M(2^{B^*} K^2 G + 2^{B^*} B^* K \log K))$. Our graph exploration rule allows us to explore the graph so that B^* is logarithmic in M, specifically $B^* \leq (\log_2 M + 1)$. Hence $2^{B^*} = \mathcal{O}(M)$. Using this in our above expression, we see that the complexity becomes $\mathcal{O}(M^2 K^2 G + M^2 K \log M \log K)$ which shows that our algorithm is polynomial time. If we ignore logarithmic terms, we can write the complexity more compactly as $\mathcal{O}(M^2 K^2 G)$.

Space Complexity and Backtracking: We now look at the amount of space (memory) required by our algorithm. To account for this, we also need to describe how we will backtrack, *i.e.*, how we find the optimal selection of groups and elements. Note that the method described above yields the optimal value for selecting K elements from G groups, but does not immediately tell us which groups are selected. We choose a backtracking method which is time-efficient, but involves storing a fair amount of data. Specifically, we store the optimal values obtained at each step of the value update rule prior to condensation, i.e., $F(\mathcal{S}_i, g, k, \mathbf{b}_{i-1}.\{\mathcal{G}_i\}, \mathbf{I}_{\mathbf{b}_{i-1}}.\{0\})$ and $F(\mathcal{S}_i, g, k, \mathbf{b}_{i-1}.\{\mathcal{G}_i\}, \mathbf{I}_{\mathbf{b}_{i-1}}.\{1\})$ for all $1 \leq g \leq G$, $1 \leq k \leq K$, $\mathbf{I}_{\mathbf{b}_{i-1}} \in \{0,1\}^{B_{i-1}}$, $i \in \{1, \ldots, M\}$. Thus the number of values we shall need to store is at most $MGK2^{B^*}$, which can be simplified to $\mathcal{O}(M^2 KG)$ using $2^{B^*} = \mathcal{O}(M)$ (due to our graph exploration algorithm).

Algorithm 4 formally defines our backtracking procedure. We start from the M-th node and work backwards, determining the number of elements selected from each group. For the M-th group, we look at the optimal value for G groups and K elements, for the 2 cases when $\mathcal{G}_M$ is selected or unselected. The value which is the larger of these two forms our optimal solution, and thus tells us whether or not $\mathcal{G}_M$ is chosen in the optimal selection. If the optimal values stored at the $M-1$-th step involve other boundary nodes besides node M, we maximize over all selections of these boundary nodes, since we don't care about any particular nodes being selected in the optimal solution. We also remember the assignment of the indicator variables which allows us to obtain the largest value of F, since it tells us which nodes in $\mathbf{b}_{M-1}$ are included in the optimal solution. If we find that $\mathcal{G}_M$ is not chosen in the

Algorithm 4 *Backtracking* for Tree-WMC With Sparsity

Inputs: Tree $\mathcal{T}(\mathfrak{G}, \mathcal{E})$, group budget G, sparsity budget K, tables of values F, L from *Value Update Rule* in Algorithm 3.

Output: Set of K elements contained in G groups with maximum combined weight.

Initialize: $\mathcal{S} = \mathfrak{G}$, $g = G$, $k = K$, $b = b_{M-1}$, $Opt\mathcal{S} = \emptyset$
For $i \in \{M, M-1, \ldots, 1\}$

$$\{\mathbf{I}_b^*, j^*\} = \underset{\substack{\mathbf{I}_b \in \{0,1\}^{|b|} \\ j \in \{0,1\}}}{\operatorname{argmax}} F(S, g, k, \{b, \mathcal{G}_i\}, \{\mathbf{I}_b, j\})$$

  If $j^* = 1$
    $Opt\mathcal{S} = Opt\mathcal{S} \cup \mathcal{G}_i$
    $l = L(S, g, k, \{b, \mathcal{G}_i\}, \{\mathbf{I}^*_{b_{i-1}}, 1\})$
    $g = g - 1, k = k - l$
  End If
  $\mathcal{S} = \mathcal{S} \setminus \mathcal{G}_i, b = b_{i-1}$
End For
Initialize: $Opt\mathcal{K} \leftarrow K$ largest elements in $\cup_{\mathcal{G}_j \in Opt\mathcal{S}} \mathcal{G}_j$
Return: $Opt\mathcal{S}, Opt\mathcal{K}$

optimal selection, then we can ignore that group and simply find the optimal G-group, K-element selection on $M-1$ groups.

If $\mathcal{G}_M$ is chosen, however, we must determine the number of elements that are selected from $\mathcal{G}_M$, after it is cleaned of elements from other selected boundary nodes. We do this by repeating step 2, case (b), of the value update rule, and noting the optimal value of the parameter ℓ which is used in computing a given value on the LHS. For the M-th group, we are specifically concerned with the optimal value for $g = G$ and $k = K$ on the LHS, and we must choose the indicator variables to maximize the value of F. Noting the value of ℓ which gives the optimum on the RHS tells us the number of elements chosen from $\mathcal{G}_M$ in the optimal selection, after cleaning any overlapping selected boundary nodes. Suppose this value is ℓ_1. Then we now need to solve a smaller problem - find the optimal selection of $G-1$ groups and $K-\ell_1$ elements from groups $1, \ldots, M-1$, with the nodes in $\mathbf{b}_{M-1}$ fixed to the maximizing value of their indicator variables. Clearly, we can repeat the above procedure on this smaller problem, and hence recursively determine the entire optimal selection.

It can be verified that the running time of the above algorithm is somewhat smaller than the update rule. Thus, the overall expression for time complexity is unchanged even when we account for backtracking.

G. Graph Exploration Rule

We determine the order with which the nodes are picked by a value associated to each subtree of the graph, which we call the D-value. In the following, we describe how it is computed, how it depends logarithmically on the number of nodes in the graph and how the number of boundary nodes is bounded by the D-value. Pseudocode is provided in Algorithm 5. We start below with some definitions.

Algorithm 5 *Graph Exploration Rule*

Input: Rooted tree $\mathcal{T}(\mathcal{V}, \mathcal{E}, \text{root})$

Output: Sequence of nodes to visit that minimizes the number of boundary nodes.

Initialize: $C(v) \leftarrow$ set of children of v
Initialize: $\mathcal{T}(v) \leftarrow$ sub-tree rooted at v
Return: $Explore(\mathcal{T}(\text{root}))$:

Compute D-value

Function $D(v)$
  If $C(v) = \emptyset$
    $D(v) = 1$
  Else If $|C(v)| = 1$
    $D(v) = D_{C(v)}$
  Else
    $u_1 = \arg\max_{u \in C(v)} D(u)$
    $u_2 = \arg\max_{u \in C(v) \setminus u_1} D(u)$
    $D(v) = \max(D(u_1), D(u_2) + 1)$
  End If
End Function

Exploration rule

Function $Explore(\mathcal{T}(v))$
  Let $C(v) = (u_1, \ldots, u_R)$ with $D(u_1) \geq \ldots \geq D(u_R)$.
  Return:

$$\big(Explore(\mathcal{T}(u_1)), v, Explore(\mathcal{T}(u_2)), \ldots, Explore(\mathcal{T}(u_R))\big)$$

End Function

Definition 8: *Given a graph $\mathbb{G} = (\mathcal{V}, \mathcal{E})$, and an 'explored set' $\mathcal{S} \subseteq \mathcal{V}$ of its nodes, a node $v \in \mathcal{V}$ is said to be a* boundary node *in $\mathbb{G}$ with respect to $\mathcal{S}$ if $v \in \mathcal{S}$ and $\exists u \in \mathcal{V}$ such that $u \notin \mathcal{S}$ and $(u, v) \in \mathcal{E}$.*

Definition 9: *A* rooted tree graph *$\mathcal{T} = (\mathcal{V}, \mathcal{E}, r)$ is a tree graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$, and a specific node $r \in \mathcal{V}$ designated as the root.*

Definition 10: *The* rooted subtrees *of a rooted tree graph $\mathcal{T} = (\mathcal{V}, \mathcal{E}, r)$ are the d rooted tree graphs obtained as components when the root of $\mathcal{T}$ is deleted. The roots of the subtrees are the unique nodes which were adjacent to r in $\mathcal{T}$. Note that d is the degree of r in $\mathcal{T}$.*

Definition 11: *The* D-value *of a rooted tree graph is a non-negative integer associated with the graph. We will define the D-value algorithmically later.*

Exploration Rule: Given a rooted tree graph $\mathcal{T}$, we first order all rooted subtrees with respect to the the D-value, so that $D_1 \geq \ldots \geq D_R$ for subtrees $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_R$. We then pick the subtrees in the order $\{\mathcal{T}_1, \text{root}, \mathcal{T}_2, \ldots, \mathcal{T}_R\}$ and recurse until the explored subtree has only one node, see Fig. 13.

Computing D-Values: The procedure for computing the D-values is also recursive. If the tree has only one node, $D = 1$. Now, assume the R subtrees at a node Q have values $D_1 \geq \ldots \geq D_R$. Then, $D(Q) = \max(D_1, D_2 + 1)$. In case there is no second subtree, $D(Q) = D_1$. We then have the following bound on the D-values.

Lemma 3: *The D-value of a rooted tree graph is logarithmic in the number of nodes,* i.e.*, $D(\mathcal{T}) \leq \log_2(M) + 1$.*

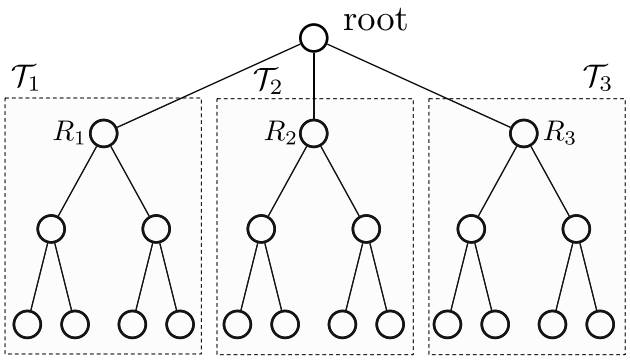

Fig. 13. Graph Exploration Rule: explore nodes in the order $\mathcal{T}_1$, root, $\mathcal{T}_2, \mathcal{T}_3$ where $D_1 \geq D_2 \geq D_3$. For the subtree $\mathcal{T}_1$, the node connected to `root` should be considered the root of $\mathcal{T}_1$, which we denote by R_1; similarly for the other subtrees.

Proof: Let D be a positive integer and $N(D)$ be the minimum number of nodes that a rooted tree must have in order to have D-value of D. We prove by induction that

$$N(D) \geq 2^{D-1}. \tag{17}$$

Base Case: $D = 2$. A tree with only one node will have a D-value of 1. So to have a D-value of 2, we require a graph with at least 2 nodes. Hence (17) is satisfied.

Inductive Case: $D > 2$. Let $\mathcal{T}$ be a smallest (*i.e.*, minimum node) rooted tree graph whose D-value is equal to D. Spread out $\mathcal{T}$ in the form of root and subtrees. Let the subtrees be $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$, with corresponding D-values $D_1, D_2, \ldots, D_k$. Without loss of generality, assume that $D_1 \geq D_2 \geq \ldots \geq D_k$. By definition, $D(\mathcal{T}) = \max(D_1, D_2 + 1)$.

By our assumption, $\mathcal{T}$ is a minimum-node graph with D-value equal to D, hence we cannot have $D_1 = D(\mathcal{T}) = D$, since that would give us a smaller rooted tree graph ($\mathcal{T}_1$) with a D-value of D. This means that $D_1 < D$, but $D = \max(D_1, D_2 + 1)$, hence $D_2 + 1 = D$, *i.e.*, $D_2 = D - 1$. Since $D_1 \geq D_2 = D - 1$ and $D_1 < D$, then $D_1 = D - 1 = D_2$. Thus, the graph $\mathcal{T}$ has 2 subtrees ($\mathcal{T}_1$ and $\mathcal{T}_2$), with D-values of $D-1$ each. By definition, any rooted subtree with a D-value of $D - 1$ must have at least $N(D - 1)$ nodes. By our induction hypothesis, $N(D - 1) \geq 2^{D-2}$. Therefore, $\mathcal{T}$ has at least $2 \times 2^{D-2} = 2^{D-1}$ nodes. But since $\mathcal{T}$ was the smallest rooted tree graph with D-value of D, this means that $N(D) \geq 2^{D-1}$, as required. □

We now link the number of boundary nodes visited by the algorithm to the D-value of the intersection graph.

Lemma 4: *The total number of boundary nodes encountered by the graph exploration algorithm cannot exceed the D-value of the graph.*

Proof: Let $\mathcal{T}$ be the given rooted tree graph, with M nodes. We shall consider the number of boundary nodes when there is a *ghost node* connected to the root node. The ghost node is a hypothetical node which is not really a part of the graph, but still makes adjacent explored nodes count as boundary nodes. The ghost node captures the fact that when we are running the algorithm recursively on a subtree, there will be an additional (potentially unexplored) node connected to the root of the subtree, which may lead to the root being counted as a boundary node. Let $B^*(\mathcal{T})$ denote the maximum number of boundary nodes encountered on $\mathcal{T}$ when we pick nodes according to our algorithm, and let $B^*_G(\mathcal{T})$ represent the same when we also have the ghost node. Clearly, $B^*_G(\mathcal{T}) \geq B^*(\mathcal{T})$, hence it is enough to prove the following:

$$B^*_G(\mathcal{T}) \leq D(\mathcal{T}). \tag{18}$$

We prove this by strong induction on M.

Base Case. Suppose the rooted tree graph $\mathcal{T}$ has only one node. Then the maximum number of boundary nodes encountered is obviously one, which is equal to the D-value of the graph (by definition). Hence $B^*_G(\mathcal{T}) \leq D(\mathcal{T})$.

Inductive Case. When the graph $\mathcal{T}$ consists of M nodes, $M > 1$, consider the graph to be spread out in the form of root and subtrees. Compute the D-values for each rooted subtree, where w.l.o.g., $D_1 \geq D_2 \geq \ldots \geq D_k$. Let $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ be the corresponding subtrees. By definition, our algorithm explores nodes in the sequence: $\mathcal{T}_1$, root, $\mathcal{T}_2, \mathcal{T}_3, \ldots \mathcal{T}_k$.

Since each subtree has strictly fewer than M nodes, each subtree satisfies (18) by the induction hypothesis. Also, notice that when exploring the subtree $\mathcal{T}_1$ of $\mathcal{T}$, the number of boundary nodes encountered is less than or equal to the number of boundary nodes encountered when exploring $\mathcal{T}_1$ as a standalone rooted-tree-graph, with a ghost node connected to its root. By definition, this is exactly equal to $B^*_G(\mathcal{T}_1)$, which by our induction hypothesis is bounded by D_1. Therefore, the number of boundary nodes encountered while exploring $\mathcal{T}_1$ in $\mathcal{T}$ cannot exceed D_1. Once we are finished with $\mathcal{T}_1$, we pick the root, so the total number of boundary nodes is 1. We now proceed to pick $\mathcal{T}_2$. By a similar argument, the maximum number of boundary nodes in $\mathcal{T}_2$ at any point cannot exceed the number of boundary nodes encountered while exploring $\mathcal{T}_2$ as a standalone graph with attached ghost node. In addition, the root of $\mathcal{T}$ can contribute at most one additional boundary node (in fact, the ghost node for $\mathcal{T}$ ensures that the root, once picked, will always contribute an additional boundary node). Therefore, the total number of boundary nodes in $\mathcal{T}$ while exploring $\mathcal{T}_2$ is at most $D_2 + 1$. Similar arguments hold for all other subtrees — the maximum number of boundary nodes while exploring the k-th subtree will be at most $D_k + 1$, which is upper bounded by $D_2 + 1$.

Therefore, the maximum number of boundary nodes encountered at any step while exploring $\mathcal{T}$ is $B^*_G(\mathcal{T}) \leq \max(D_1, D_2 + 1)$. By definition, $D(\mathcal{T}) = \max(D_1, D_2 + 1)$. Therefore $B^*_G(\mathcal{T}) \leq D(\mathcal{T})$. □

Combining Lemmas 3 and 4, we have the following result.

Lemma 5: *The maximum number of boundary nodes at any step of the algorithm is logarithmic in the number of nodes,* i.e.*, $B \leq \log_2(M) + 1$.*

The previous lemma establishes the polynomial time complexity of the dynamic program for solving the generalized integer problem (13). We shall now prove that the exploration rule itself requires minimal computation. This will justify our earlier claim that the running time is determined solely by the value update rule.

Lemma 6: *The running time of the graph exploration rule is $\mathcal{O}(M)$ for an M-node graph.*

Proof: The exploration rule can be algorithmically run in two loops. In the first, we compute all D-values of all the required subtrees in the graph. In the second loop, we find the exploration ordering using these D-values. Note that the subtrees encountered by our recursive D-value computing algorithm are exactly the same set of subtrees encountered by our exploration rule, which makes it possible to compute all the required D-values in a single loop.

For computing D-values at a particular node, we use the formula $D = \max(D_1, D_2 + 1)$, where $D_1 \geq D_2 \geq D_3, D_4, \ldots, D_k$. Thus, we need to find the largest and second largest D-values among the subtrees. For a node with d children, this takes $\mathcal{O}(d)$ time. Since the values $D_1, D_2, \ldots, D_k$ are obtained recursively, this is the only computation which needs to be performed at the current node. Hence, the total time required is proportional to $\sum_{v \in \mathcal{V}} \max(d(v), 1) \leq 2M$, where $d(v)$ represents the number of children that node v has. Hence, this loop runs in $\mathcal{O}(M)$ time.

Obtaining the exploration order is similar. We only need to find the subtree with the largest D-value at the current node, so that we can pick the subtrees in the right order. This takes $\mathcal{O}(d)$ time for a node with d children, and hence $\mathcal{O}(M)$ time for all nodes. Since both the above steps are $\mathcal{O}(M)$, the graph exploration rule itself runs in $\mathcal{O}(M)$, *i.e.*, linear time. □

APPENDIX B
DYNAMICAL PROGRAMMING FOR SOLVING THE HIERARCHICAL SIGNAL APPROXIMATION PROBLEM (14)

Here we describe the dynamic program for solving the hierarchical signal approximation problem (14) and show that its time complexity is $\mathcal{O}(NK^2D)$ for general trees with maximum degree D, and $\mathcal{O}(NKD)$ for D-regular trees. Furthermore, its space complexity for D-regular trees is $\mathcal{O}(N \log_D K)$. Pseudocode is provided in Algorithm 2 in Section VIII.

A. Problem Description

Problem (14) can be equivalently rephrased as the following optimization problem.

Rooted-Connected Subtree Problem: Given a rooted tree $\mathcal{T}$ with each node having at most D children, a non-negative real number (weight) assigned to every node and a positive integer K, choose a subset of its nodes forming a rooted-connected subtree that maximizes the sum of weights of the chosen elements, such that the number of selected nodes does not exceed K.

In our case, (14), the weight of a node is the square of the value of the component of the signal associated to that node. The proposed algorithm leverages the optimal substructure of the problem.

B. Optimal Substructure

Suppose that a particular node X belongs to the optimal K-node rooted-connected subtree. Consider the subtree $\mathcal{T}_{X,d}$ obtained by choosing X, d of its children ($1 \leq d \leq D$) and all descendants of these children. Consider the set of nodes $\mathcal{S}$ consisting of all the nodes of $\mathcal{T}_{X,d}$ which are also present in the optimal K-node rooted-connected subtree. Suppose there are L nodes in $\mathcal{S}$. Then the nodes in $\mathcal{S}$ form the optimal L-node rooted-connected subtree at X, for the subgraph $\mathcal{T}_{X,d}$. See Fig. 14 for an example.

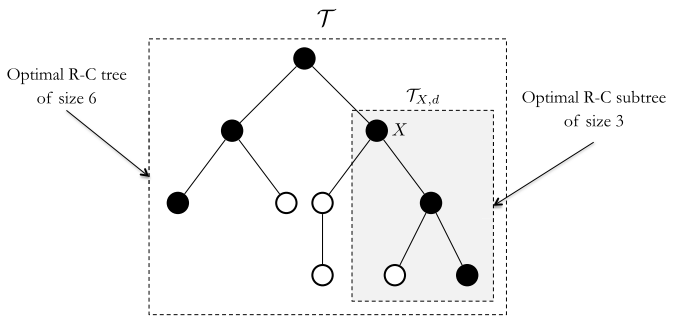

Fig. 14. Example of a nested subproblem in hierarchical groups model.

C. Dynamic Programming Method

For every node X, we store the weight of the optimal k-node rooted-connected subtree at X, using only the nodes in the d rightmost children of X and their descendants, for each k and d such that $1 \leq k \leq K$ and $1 \leq d \leq D$. We define a function $F(X, k, d)$ to store these optimal values. We start from the leaf nodes and move upwards, for each node assessing all its subtrees from right to left, eventually covering the entire tree. At the end, the optimal value will be given by $F(\text{root}, K, D)$, that is the value of the best K-node rooted connected subtree of the root considering all its descendants.

Base Case. For every leaf node X and for all $1 \leq k \leq K$ and $1 \leq d \leq D$, we set $F(X, k, d) = \text{Weight}(X)$.

Inductive Case. By induction, for every non-leaf node X, all the F-values are known for the descendants of X. Let $X_1, X_2, \ldots X_d$ be the d children of X in the right-to-left order, where $1 \leq d \leq D$. Then, we compute the F-values of X using the following update rules.

Value Update Rule:

1) For all $1 \leq k \leq K$

$$F(X, k, 1) = \text{Weight}(X) + F(X_1, k-1, D).$$

The optimal value for choosing a k-node subtree rooted at X, when only the rightmost child X_1 is allowed, equals the weight of X itself (since X must be chosen), plus the optimal value for choosing a rooted connected subtree with $k-1$ nodes from the rightmost child X_1.

2) For all $1 \leq k \leq K$ and $1 < i \leq d$

$$F(X, k, i) = \max_{0 \leq \ell \leq k-1} \{F(X, k-\ell, i-1) + F(X_i, \ell, D)\}.$$

For choosing the best k-node rooted connected subtree from the rightmost i children, choose a positive integer $\ell \leq k$, pick the best $k-\ell$-node subtree at X by including the rightmost $i-1$ children and pick the remaining ℓ nodes from the subtree of the ith child. We then take the maximum over all ℓ, $0 \leq \ell \leq k-1$ (since at least

1 node must be chosen from the rightmost $i-1$ nodes, this node will be the root).

3) For all $1 \le k \le K$ and $d < i \le D$

$$F(X,k,i) = F(X,k,d).$$

For convenience, when a node has only d children, where d is strictly less than D, we set F-values for cases involving more than d children equal to the value for d children.

D. Running Time

Theorem 3: *Given a hierarchical group structure $\mathfrak{G}$, the time complexity of Algorithm 2 is $\mathcal{O}(NK^2D)$, where D is maximum number of children of a node in the tree.*

Proof: The main cost of the dynamic program is evaluating the second value update rule. Let X_i be the i-th node in the tree, d_i the number of its children $X_{i,1}, \ldots, X_{i,d_i}$. Let also K_i be the cardinality of the tree that has X_i as root and $K_{i,j}$ be the cardinality of the tree that has $X_{i,j}$ as root for $1 \le i \le N$ and $1 \le j \le d_i$. Given X_i, evaluating $F(X_i,k,j)$ for $1 \le k \le \min(K, K_i)$ and $1 \le j \le d_i$ requires $\min(k, K_{i,j})$ operations. Therefore, overall we need to compute

$$\begin{aligned}
&\sum_{i=1}^{N} \sum_{k=1}^{\min(K,K_i)} \sum_{j=1}^{d_i} \min(k, K_{i,j}) \\
&\quad \le \sum_{i=1}^{N} \sum_{k=1}^{\min(K,K_i)} \sum_{j=1}^{D} \min(k, K_{i,j}) \\
&\quad \le \sum_{i=1}^{N} \sum_{k=1}^{K} Dk \\
&\quad = \mathcal{O}(NK^2D)
\end{aligned}$$

values, each of which requires a simple operation. □

By leveraging the special structure of D-regular trees, it is possible to prove that the complexity of the dynamic program is linear in K.

Proposition 4: *The time complexity of Algorithm 2 for D-regular trees is $\mathcal{O}(KDN)$.*

Proof: The proof follows the arguments in [41]. Suppose there are J levels in our tree, hence the maximum number of nodes that can be selected for a sub-tree with root in level j is $S_j = 1 + D + D^2 + \cdots + D^{J-j} = \frac{D^{J-j+1}-1}{D-1}$ where $j \in \{1,2,\ldots,J\}$. At each step, the dynamic program considers selecting at most K elements to form a sub-tree. Hence for a sub-tree with root at level j, we can select a maximum number of $\mathcal{O}(l(j)) = \mathcal{O}(\min(K, S_j)) = \mathcal{O}(\min(K, D^{J-j}))$ for $D \ge 3$ and $\mathcal{O}(l(j)) = \mathcal{O}(\min(K, D^{J-j+1}))$ for $D = 2$. Note that we do not require any computation for level J. The update step of $F(X,k,i) = \max_{0 \le \ell \le \min(k, l(j+1))} F(X, k-\ell, i-1) + F(X_i, \ell, D)$ then requires $\mathcal{O}(\min(k, l(j+1)))$ operations and for X in level j this needs to be calculated for all $1 \le k \le l(j)$ and $1 \le i \le D$. This leads to at most $\mathcal{O}\left(D \sum_{k=1}^{l(j)} \min(k, l(j+1))\right)$ operations. By considering that at level j there are at most D^{j-1} nodes, the total number of operations can be written as

$$\mathcal{O}\left(\sum_{j=1}^{J} D^{j-1} D \sum_{k=1}^{l(j)} \min(k, l(j+1))\right) \tag{19}$$

Let j' be such that $K \le D^{J-j}$ for all $j < j'$. We then have $j' = J - \lfloor \log_D K \rfloor$ and $\min(K, D^{J-j}) = K$ for $j < j'$ and $\min(K, D^{J-j}) = D^{J-j}$ for $j \ge j'$. Hence we can break (19) into

$$\begin{aligned}
&\mathcal{O}\left(\sum_{j=1}^{j'-1} D^j \sum_{k=1}^{K} k + \sum_{j=j'}^{J} D^j \sum_{k=1}^{D^{J-j}} \min(k, D^{J-j-1})\right) \\
&= \mathcal{O}\Big(\sum\nolimits_{j=1}^{j'-1} D^j K^2 + \sum\nolimits_{j=j'}^{J} D^j \Big(\sum\nolimits_{k=1}^{D^{J-j-1}} k \\
&\qquad + \sum\nolimits_{k=D^{J-j-1}+1}^{D^{J-j}} D^{J-j-1}\Big)\Big) \\
&= \mathcal{O}\left(\sum_{j=1}^{j'-1} D^j K^2 + \sum_{j=j'}^{J} D^j \left(D^{2J-2j-2} + DD^{J-j-1}\right)\right) \\
&\le \mathcal{O}\left(K^2 \frac{D^{j'}}{D-1} + D^{2J-2} \sum_{j=j'}^{J} D^{-j}\right) \\
&\le \mathcal{O}\left(K^2 \frac{D^{j'}}{D-1} + D^{2J-2} \frac{D^{-j'}}{1-D^{-1}})\right) \\
&= \mathcal{O}\left(K^2 \frac{D^{J-\lfloor \log_D K \rfloor}}{D-1} + K \frac{D^{J-2}}{1-D^{-1}})\right) \\
&\le \mathcal{O}\left(K \frac{D^{J+1}}{D-1} + K \frac{D^{J-2}}{1-D^{-1}})\right) \\
&= \mathcal{O}\left(K \frac{D^{J+1}}{D-1} + K \frac{D^{J-1}}{D-1})\right) \\
&= \mathcal{O}\left(K \frac{D^{J+1}}{D-1}\right).
\end{aligned}$$

For D-regular trees (with $D \ge 3$), we have $N = \frac{D^J - 1}{D-1} \approx \frac{D^J}{D-1}$, so that the time complexity will be $\mathcal{O}(KDN)$. When $D = 2$, we can follow the same steps to show that the complexity is $\mathcal{O}(KD^2N)$. But for small values of D, $\mathcal{O}(ND^2K) = \mathcal{O}(NDK)$. Hence we can say that the overall complexity is $\mathcal{O}(NDK)$. □

E. Space Complexity

Proposition 5: *The memory complexity of Algorithm 2 for D-regular trees is $\mathcal{O}(N \log_D K)$ for our implementation.*

Proof: Suppose there are total of $J \ge 1$ levels in our tree. Hence the maximum number of nodes that can be selected for a sub-tree with root in level j is $\ell(j) = \min(K, 1 + D + D^2 + .. + D^{J-j}) = \min(K, \frac{D^{J+1-j}-1}{D-1})$ or $\mathcal{O}(\ell(j)) = \mathcal{O}(\min(K, D^{J-j}))$, where $j \in \{1,2,\ldots,J\}$. Let $N_j = D^{j-1}$ be the number of nodes at level j.

In order to recover the optimal selection of nodes from the dynamic program, we use a standard backtracking procedure:

Algorithm 6 *Value Update Rule* for Rooted Connected K-Sparse Trees

Input: Rooted tree $\mathcal{T}(\mathcal{V}, \mathcal{E}, \text{root})$, weight of each node $v \in \mathcal{V}$ (Weight(v)), sparsity budget K.
Output: table of values F, L.
Initialize: $C(v) \leftarrow$ set of children of $v \in \mathcal{V}$
Let $\mathcal{V} = (v_1, v_2, \ldots, v_N)$ according to *Breadth First Ordering*
Initialize: table of values $F(v, k, j) = 0$, backtracking table $L(v, k, j) = 0$, $\forall\ v \in \mathcal{V}, 1 \le k \le K, 0 \le j \le |C(v)|$
For $i \in \{N, N-1, \ldots, 1\}$
  For $k \in \{1, 2, \ldots, K\}$
    If $C(v_i) = \emptyset$
      $F(v_i, k, 0) =$Weight(v_i)
    Else
      Let $C(v_i) = (v_{i_1}, v_{i_2}, \ldots, v_{i_R})$ for $i_1 \ge i_2 \ge \ldots \ge i_R$
      For $j \in \{2, \ldots, R\}$
        $F(v_i, k, 1) = \text{Weight}(v_i) + F(v_{i_j}, k-1, |C(v_{i_j})|)$
        $F(v_i, k, j) = \max_{1 \le \ell \le k}\{F(v_i, \ell, j-1) + F(v_{i_j}, k-\ell, |C(v_{i_j})|)\}$
        $L(v_i, k, j) = \arg\max_{1 \le \ell \le k}\{F(v_i, \ell, j-1) + F(v_{i_j}, k-\ell, |C(v_{i_j})|)\}$
      End For
    End If
  End For
End For
return F, L

Algorithm 7 *Backtracking* for Rooted Connected K-Sparse Trees

Input: Rooted tree $\mathcal{T}(\mathcal{V}, \mathcal{E}, \text{root})$, table of values F, L from *Value Update Rule* in Algorithm 6z, sparsity budget K.
Output: A subtree of $\mathcal{T}$ with same root as $\mathcal{T}$ having at most K nodes with maximum combined weight.
Initialize: $C(v) \leftarrow$ set of children for all $v \in \mathcal{V}$
Let $\mathcal{V} = (v_1, v_2, \ldots, v_N)$ according to *Breadth First Ordering*
Initialize: $Opt\mathcal{K}(v) = 0,\ \forall\ v \in \mathcal{V}$
Set $Opt\mathcal{K}(v_1) = K$
For $i \in \{1, \ldots, N\}$ such that $C(v_i) \neq \emptyset$
  Let $C(v_i) = (v_{i_1}, v_{i_2}, \ldots, v_{i_R})$ for $i_1 \ge \ldots \ge i_R$
  For $j \in \{R, R-1, \ldots, 1\}$
    If $j = 1$
      $Opt\mathcal{K}(v_{i_j}) = Opt\mathcal{K}(v_i) - 1$
    Else
      $Opt\mathcal{K}(v_{i_j}) = Opt\mathcal{K}(v_i) - L(v_i, Opt\mathcal{K}(v_i), j)$
    End If
    $Opt\mathcal{K}(v_i) = Opt\mathcal{K}(v_i) - Opt\mathcal{K}(v_{i_j})$
  End For
End For
Initialize: $Opt\mathcal{S} = \{v_i : 1 \le i \le N, Opt\mathcal{K}(v_i) \ge 1\}$
Return: $Opt\mathcal{S}$

for each node, we store the number of selected nodes in each of its subtrees (D numbers) in the optimal selection for $1 \le k \le \ell(j)$.

Hence, the total memory required is

$$\mathcal{O}\left(\sum_{j=1}^{J} D\ell(j)N_j\right) = \mathcal{O}\left(\sum_{j=1}^{J} \min(K, D^{J-j})D^j\right) \quad (20)$$

Let j' be such that $K \le D^{J-j}$ for all $j < j'$. We then have $j' = J - \lfloor \log_D K \rfloor$ and $\min(K, D^{J-j}) = K$ for $j < j'$ and $\min(K, D^{J-j}) = D^{J-j}$ for $j \ge j'$.

(20) now becomes

$$\begin{aligned}
&\mathcal{O}\left(\sum_{j=1}^{j'-1} K D^j + \sum_{j=j'}^{J} D^{J-j} \cdot D^j\right) \\
&= \mathcal{O}\left(K \sum_{j=1}^{j'-1} D^j + \sum_{j=j'}^{J} D^J\right) \\
&= \mathcal{O}\left(K D^{j'}) + D^J (J - j')\right) \\
&\le \mathcal{O}\left(K D^{J+1-\log_D K} + D^J \log_D K\right) \\
&= \mathcal{O}\left(D^{J+1} + D^J \log_D K\right) \\
&= \mathcal{O}\left(N \log_D K\right) \text{ where } N = \mathcal{O}(D^J)
\end{aligned}$$

□

Algorithm 7 describes our backtracking procedure to obtain the optimal set of nodes from the table of optimal values.

ACKNOWLEDGEMENTS

The authors would like to sincerely thank the anonymous reviewers for their detailed and constructive observations and criticisms. They also thank Nikhil Rao for providing the code for block signal recovery with the Latent Group Lasso approach.

REFERENCES

[1] S. Mallat, *A Wavelet Tour of Signal Processing*. New York, NY, USA: Academic, 1999.
[2] D. L. Donoho, "Compressed sensing," *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.

[3] E. J. Candès, "Compressive sampling," in *Proc. Int. Congr. Math.*, Madrid, Spain, Aug. 2006, pp. 1433–1452.
[4] R. G. Baraniuk, "Compressive sensing," *IEEE Signal Process. Mag.*, vol. 24, no. 4, pp. 118–121, Jul. 2007.
[5] Y. C. Eldar and M. Mishali, "Robust recovery of signals from a structured union of subspaces," *IEEE Trans. Inf. Theory*, vol. 55, no. 11, pp. 5302–5316, Nov. 2009.
[6] T. Blumensath and M. E. Davies, "Sampling theorems for signals from the union of finite-dimensional linear subspaces," *IEEE Trans. Inf. Theory*, vol. 55, no. 4, pp. 1872–1882, Apr. 2009.
[7] R. G. Baraniuk, V. Cevher, M. F. Duarte, and C. Hegde, "Model-based compressive sensing," *IEEE Trans. Inf. Theory*, vol. 56, no. 4, pp. 1982–2001, Apr. 2010.
[8] N. Rao, B. Recht, and R. Nowak. (Sep. 2012). "Signal recovery in unions of subspaces with applications to compressive imaging." [Online]. Available: https://arxiv.org/abs/1209.3079
[9] R. Baraniuk, V. Cevher, and M. Wakin, "Low-dimensional models for dimensionality reduction and signal recovery: A geometric perspective," *Proc. IEEE*, vol. 98, no. 6, pp. 959–971, Jun. 2010.
[10] R. Jenatton, J.-Y. Audibert, and F. Bach, "Structured variable selection with sparsity-inducing norms," *J. Mach. Learn. Res.*, vol. 12, pp. 2777–2824, Feb. 2011.
[11] G. Obozinski, L. Jacob, and J. Vert. (Oct. 2011). "Group lasso with overlaps: The latent group lasso approach." [Online]. Available: http://arxiv.org/abs/1110.0413
[12] N. S. Rao, R. D. Nowak, S. J. Wright, and N. G. Kingsbury, "Convex approaches to model wavelet sparsity patterns," in *Proc. 18th. IEEE Int. Conf. Image Process.*, Sep. 2011, pp. 1917–1920.
[13] A. Gramfort and M. Kowalski, "Improving M/EEG source localization with an inter-condition sparse prior," in *Proc. IEEE Int. Symp. Biomed. Imag.*, Jun./Jul. 2009, pp. 141–144.
[14] R. Jenatton, A. Gramfort, V. Michel, G. Obozinski, F. Bach, and B. Thirion, "Multi-scale mining of fmri data with hierarchical structured sparsity," in *Proc. Pattern Recognit. NeuroImag. (PRNI)*, May 2011, pp. 69–72.
[15] A. Subramanian *et al.*, "Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles," *Proc. Nat. Acad. Sci. USA*, vol. 102, no. 43, pp. 15545–15550, Aug. 2005.
[16] F. Rapaport, E. Barillot, and J.-P. Vert, "Classification of arraycgh data using fused svm," *Bioinformatics*, vol. 24, no. 13, pp. i375–i382, Jul. 2008.
[17] H. Zhou, M. E. Sehl, J. S. Sinsheimer, and K. Lange, "Association screening of common and rare genetic variants by penalized regression," *Bioinformatics*, vol. 26, no. 19, p. 2375, Aug. 2010.
[18] V. Cevher, C. Hegde, M. Duarte, and R. Baraniuk, "Sparse signal recovery using Markov random fields," in *Proc. NIPS*, 2009, pp. 257–264.
[19] B. Bah, L. Baldassarre, and V. Cevher, "Model-based sketching and recovery with expanders,"in *Proc. ACM-SIAM Symp. Discrete Algorithms*, 2014, pp. 1529–1543.
[20] V. Michel, A. Gramfort, G. Varoquaux, E. Eger, and B. Thirion, "Total variation regularization for fmri-based prediction of behavior," *IEEE Trans. Med. Imag.*, vol. 30, no. 7, pp. 1328–1340, Jul. 2011.
[21] M. Stojnic, F. Parvaresh, and B. Hassibi, "On the reconstruction of block-sparse signals with an optimal number of measurements," *IEEE Trans. Signal Process.*, vol. 57, no. 8, pp. 3075–3085, Aug. 2009.
[22] J. Huang, T. Zhang, and D. Metaxas, "Learning with structured sparsity," *J. Mach. Learn. Res.*, vol. 12, pp. 3371–3412, Jan. 2011.
[23] L. Jacob, G. Obozinski, and J. Vert, "Group lasso with overlap and graph lasso,"in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 433–440.
[24] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *J. Roy. Statist. Soc.(Statist. Methodol.)*, vol. 68, no. 1, pp. 49–67, 2006.
[25] P. Zhao, G. Rocha, and B. Yu, "The composite absolute penalties family for grouped and hierarchical variable selection," *Ann. Statist.*, vol. 37, no. 6A, pp. 3468–3497, 2009.
[26] G. Obozinski and F. Bach. (May 2012). "Convex relaxation for combinatorial penalties." [Online]. Available: http://arxiv.org/abs/1205.1240
[27] D. S. Hochbaum, *Approximation Algorithms for NP-Hard Problems*, vol. 20. Boston, MA, USA: PWS publishing company, 1997.
[28] L. Wolsey and G. Nemhauser, *Integer and Combinatorial Optimization*. New York, NY, USA: Wiley, 1999.
[29] A. Kyrillidis and V. Cevher, "Combinatorial selection and least absolute shrinkage via the clash algorithm,"in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2012, pp. 2216–2220.
[30] M. E. Halabi and V. Cevher, "A totally unimodular view of structured sparsity,"in *Proc. 18th Int. Conf. Artif. Intell. Statist.*, 2015, pp. 223–231.
[31] C. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.
[32] G. Nemhauser, L. Wolsey, and M. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Math. Program.*, vol. 14, no. 1, pp. 265–294, Dec. 1978.
[33] S. Khuller, A. Moss, and J. S. Naor, "The budgeted maximum coverage problem," *Inf. Process. Lett.*, vol. 70, no. 1, pp. 39–45, Apr. 1999.
[34] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Berlin, Germany: Springer, 2004.
[35] S. Wright, *Primal-Dual Interior-Point Methods*. Philadelphia, PA, USA: SIAM, 1997.
[36] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[37] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani, "A sparse-group lasso," *J. Comput. Graph. Statist.*, vol. 22, no. 2, pp. 231–245, May 2012.
[38] R. G. Baraniuk and D. L. Jones, "A signal-dependent time-frequency representation: Fast algorithm for optimal kernel design," *IEEE Trans. Signal Process.*, vol. 42, no. 1, pp. 134–146, Jan. 1994.
[39] R. G. Baraniuk, "Optimal tree approximation with wavelets," *Proc. SPIE*, vol. 3813, pp. 196–207, Oct. 1999.
[40] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach, "Proximal methods for hierarchical sparse coding," *J. Mach. Learn. Reasearch*, vol. 12, pp. 2297–2334, Jul. 2011.
[41] C. Cartis and A. Thompson, "An exact tree projection algorithm for wavelets," *IEEE Signal Process. Lett.*, vol. 20, no. 11, pp. 1028–1029, Apr. 2013.
[42] D. L. Donoho, "Cart and best-ortho-basis: A connection," *The Ann. Statist.*, vol. 25, no. 5, pp. 1870–1911, 1997.
[43] S. Mosci, S. Villa, A. Verri, and L. Rosasco, "A primal-dual algorithm for group sparse regularization with overlapping groups," in *Proc. Adv. Neural Inf. Process. Syst.(NIPS)*, 2010, pp. 2604–2612.

Luca Baldassarre received the M.Sc. in Physics in 2006 and the Ph.D. in Machine Learning in 2010 at the University of Genoa, Italy. He then joined the Computer Science Department of University College London, UK, to work with Prof. Massimiliano Pontil on structured sparsity models for machine learning and convex optimization. He joined the LIONS of Prof. Volkan Cevher at the École Polytechnique Fédérale de Lausanne (EPFL), Switzerland in 2012, where he is still a guest researcher. Currently, he works as data scientist at Gamaya, an EPFL start-up in the field of precision agriculture. His research interests include structured sparsity in machine learning and compressive sensing, and optimization.

Nirav Bhan is currently a second year graduate student in EECS at Massachusetts Institute of Technology. He is member of the Laboratory of Information and Decision Systems (LIDS). His interests are in optimization, machine learning, graphical models, and applying mathematics to solve problems. Prior to being a graduate student, Nirav obtained a B.Tech degree in Electrical Engineering along with a Minor in Computer Science, from the Indian Institute of Technology- Bombay. He has worked as a research assistant at LIONS, EPFL, during the period of May to July, 2012.

Volkan Cevher (SM'10) received the B.Sc. (valedictorian) in electrical engineering from Bilkent University in Ankara, Turkey, in 1999 and the Ph.D. in electrical and computer engineering from the Georgia Institute of Technology in Atlanta, GA in 2005. He was a Research Scientist with the University of Maryland, College Park from 2006- 2007 and also with Rice University in Houston, TX, from 2008-2009. Currently, he is an Associate Professor at the Swiss Federal Institute of Technology Lausanne and a Faculty Fellow in the Electrical and Computer Engineering Department at Rice University. His research interests include signal processing theory, machine learning, convex optimization, and information theory. Dr. Cevher was the recipient of a Best Paper Award at SPARS in 2009, a Best Paper Award at CAMSAP in 2015, and an ERC StG in 2011.

Anastasios Kyrillidis is a Simons PostDoc at the Electrical and Computer Engineering department of the University of Texas, Austin. He obtained his Ph.D. from the School of Computer and Communication Sciences at Ecole Polytechnique Federale de Lausanne (EPFL) in 2014. Before that, he completed his M.Sc. and Diploma (5-year) studies at the Technical University of Crete, Greece. He closely worked as an intern with the Cognitive Computing department of IBM Zurich, Ruschlikon. His research interests include scalable and provable machine learning algorithms, and, in a broader sense, convex and non-convex analysis and optimization. In the past he has worked on low-dimensional modeling and compressed sensing, as well as developing practical linear algebra tools.

Siddhartha Satpathi is currently a first year graduate student in ECE at University of Illinois at Urbana Champaign. He is member of the Coordinated Science Lab (CSL). He obtained a B.Tech+M.Tech degree in Electrical Engineering and a minor in Computer Science from Indian Institute of Technology Kharagpur. He has worked as a research intern at EPFL, Switzerland, during the period of May to July, 2013. His interests are in compressive sensing, machine learning and energy harvesting communication systems.