Techniques for accelerating Branch-and-Bound algorithms dedicated to sparse optimization

Gwenaël SAMAIN[a], Sébastien BOURGUIGNON[a] and Jordan NININ[b]

[a]Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000;
[b]Lab-STICC, CNRS UMR 6285, ENSTA Bretagne, Brest Cedex, France

ARTICLE HISTORY
Compiled April 14, 2022

KEYWORDS
branch-and-bound, cardinality, convex relaxation, duality, exploration strategy.

Abstract
Sparse optimization—fitting data with a low-cardinality linear model—is addressed through the minimization of a cardinality-penalized least-squares function, for which dedicated branch-and-bound algorithms clearly outperform generic mixed-integer-programming solvers. Three acceleration techniques are proposed for such algorithms. Convex relaxation problems at each node are addressed with dual approaches, which can early prune suboptimal nodes. Screening methods are implemented, which fix variables to their optimal value during the node evaluation, reducing the subproblem size. Numerical experiments show that the efficiency of such techniques depends on the node cardinality and on the structure of the problem matrix. Last, different exploration strategies are proposed to schedule the nodes. Best-first search is shown to outperform the standard depth-first search used in the related literature. A new strategy is proposed which first explores the nodes with the lowest least-squares value, which is shown to be the best at finding the optimal solution—without proving its optimality. A C++ solver with compiling and usage instructions is made available.

Introduction

Adjusting a linear model with low cardinality to a data set has found many applications, for example in statistics [6], finance [13] and signal processing [10]. The corresponding sparse optimization problem is often formulated as the minimization of the least-squares misfit function between the data vector $y \in \mathbb{R}^N$ and the model Ax, with known matrix $A \in \mathcal{M}_{N\times Q}(\mathbb{R})$ and unknown vector $x \in \mathbb{R}^Q$. Sparsity is then enforced on x, that is, the number of nonzero values, the so-called ℓ_0 "norm" $\|x\|_0 := \mathrm{Card}(\{i|x_i \neq 0\})$, is limited. In this paper, we focus on the penalized formulation [4, 5, 10, 23, 24, 38, 39, 41]:

$$\min_{x\in\mathbb{R}^Q} \tfrac{1}{2}\|y - Ax\|_2^2 + \mu\|x\|_0, \qquad (\mathcal{P})$$

which balances between the two contradictory objectives through the value of parameter $\mu > 0$.

Due to the discrete ℓ_0-norm term, this problem is essentially combinatorial and NP-hard [29]. Many heuristic local search methods have been proposed [2, 42, 43],

CONTACT G. SAMAIN. Author. Email: gwenael.samain@ls2n.fr

as well as algorithms that solve continuous relaxations of $(\mathcal{P})$. In particular, many algorithms were designed for solving the convex, non-smooth, ℓ_1-norm-penalized problem (see [2] for a review). Locally solving continuous, non-convex, relaxations of the ℓ_0-norm problem is also a recent trend (see the discussion in [38] about different such relaxations and their properties).

All such methods are fast, they can scale to high-dimensional problems, and they do find sparse solutions. However, conditions that guarantee their optimality according to $(\mathcal{P})$, mostly requiring that the matrix A is nearly orthogonal and that the solution is highly sparse, are very restrictive [43]. In practice, their performance decreases as the problem complexity increases (highly correlated columns in A, high level of noise), even in small dimension[10].

On the other hand, exact optimization techniques of $(\mathcal{P})$ were considered, which guarantee the global optimality of the solution. In [6, 10, 41], $(\mathcal{P})$ was reformulated as a mixed integer program (MIP) and solved by a generic solver. In [5, 7, 23], dedicated optimization algorithms based on the branch-and-bound framework were proposed, which outperform generic MIP resolution, by exploiting the specific structure of the problem (note that earlier ideas in this direction appeared in [8]).

This papers studies different techniques aiming to accelerate exact sparse optimization by such dedicated branch-and-bound algorithms. The two first contributions focus on reducing the computational burden associated to the relaxation problems evaluated at each explored node, based on exploiting convex duality. First, we leverage weak duality in order to prune suboptimal nodes without resorting to the exact computation of the relaxed subproblems. Then, convex screening methods [30, 34] are studied in order to discard optimization variables prior to (or during) such computations, which reduce the size of the relaxation problems. Last, we consider different possible strategies for node exploration (depth-first of best-first searches based on the different terms of the cost function, and mixed strategies) and we study their impact on the number of explored nodes.

Section 1 introduces the notations and definitions, and defines the global branch-and-bound architecture dedicated to sparse optimization. Exploiting weak duality is addressed in Section 2. Section 3 is devoted to the implementation of screening tests in our context, and exploration strategies are studied in Section 4. The performance of all such techniques is evaluated through numerical experiments exposed in Section 5. Finally, Section 6 provides installation and usage instructions for the corresponding solver. Concluding remarks and a discussion for future research close the paper.

1. Dedicated Branch-and-Bound for sparse optimization

1.1. Problem formulation

The sparse approximation problem $(\mathcal{P})$ is essentially combinatorial, due to the discrete-valued ℓ_0-norm counting function. Dedicated resolution approaches in the literature [5, 7, 8] rely on branch-and-bound algorithms, which require the computation of relaxations of $(\mathcal{P})$. In particular, in order to compute lower bounds of $(\mathcal{P})$ (or of subproblems of $(\mathcal{P})$), a standard approach computes convex relaxations of such problems. However, due to the non-coercivity of the ℓ_0-norm term (since $\forall \lambda \neq 0$, $\|\lambda x\|_0 = \|x\|_0$), no convex relaxation of the ℓ_0 norm can be obtained. One solution then consists in incorporating additional constraints to $(\mathcal{P})$ that will bound the values of x, see Figure 1 for such convex relaxations in a bounded domain.

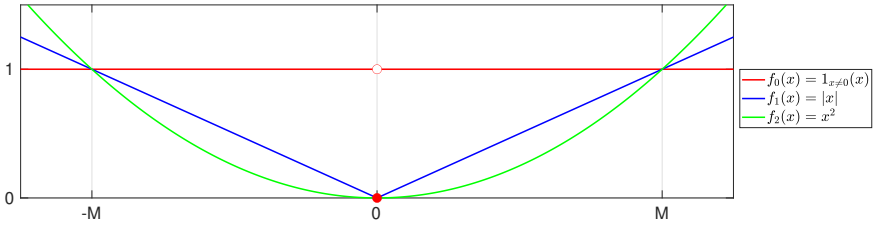

Figure 1.: The ℓ_0 counting function in 1D ($f_0(x) = 1$ if $x \neq 0$ and 0 otherwise), and some of its continuous convex relaxations, only possible in a bounded domain—here, $-M \leq x \leq M$.

For example, in [23, 36], the ℓ_2-norm penalty term $\|x\|_2^2$ is added to the cost function, meaning that amplitudes are enforced to lie in an ℓ_2 ball—also see much earlier works that motivate such a ridge regression term based on a statistical Bernoulli-Gaussian models on vector x [24]. In this paper, we follow another standard approach, constraining the amplitudes of x to lie in a symmetric box: $\|x\|_\infty := \max_{i\in\{1,\dots,Q\}} |x_i| \leq M$. In the rest of this paper, we therefore consider the following optimization problem:

$$\hat{x} := \underset{x\in\mathbb{R}^Q}{\arg\min}\ \tfrac{1}{2}\|y - Ax\|_2^2 + \mu\|x\|_0, \quad \text{subject to (s.t.)} \quad \|x\|_\infty \leq M. \tag{$\mathcal{P}_{2+0}$}$$

Note that such a choice was also used *e.g.* in [5, 6, 8], where problem ($\mathcal{P}_{2+0}$) was reformulated as a Mixed Integer Program (MIP) with binary variables b_i, $i \in \{1, \dots, Q\}$, encoding the nullity/non-nullity of x_i by the linear constraints: $x_i \leq M b_i$, such that $\|x\|_0 = \sum_i b_i$.

Let S denote the *support* of a given vector x, that is, the index set of non-zero components: $S := \{i | x_i \neq 0\}$. Finding the corresponding non-zero values amounts to solving:

$$\begin{aligned} &\min_x \tfrac{1}{2}\|y - Ax\|_2^2 + \mu|S| \quad \text{s.t.} \quad \begin{cases} \|x\|_\infty \leq M, \\ x_j = 0,\ \forall j \notin S \end{cases} \\ \Leftrightarrow\ &\min_{x_S} \tfrac{1}{2}\|y - A_S x_S\|_2^2 \quad \text{s.t.} \quad \|x_S\|_\infty \leq M, \end{aligned}$$

where A_S denotes the submatrix formed by the columns of A with indices in S, and x_S denotes the corresponding subvector of non-zero variables which contribute to the solution. For a given support S, this is a box-constrained least-squares problem. Consequently, the main challenge to solve ($\mathcal{P}_{2+0}$) is the search for the optimal support. Therefore, we build a branch-and-bound algorithm to this end, whose search space is the space of supports.

1.2. Support space structuring and branch-and-bound architecture

Within the branch-and-bound procedure, a given node is divided by choosing a component and make it belong to, or reject from, the support of the solution. At each node, the support space is partitioned into three subsets:

- S_1: the set of components which contribute to the solution,
- S_0: the set of components which are forced to 0,
- $\bar{S}$: the set of free/undetermined components.

We denote a node by $\mathbf{N}(S_1, S_0, \bar{S})$, or simply $\mathbf{N}$ when there is no ambiguity. Dividing a node then means taking a component $j \in \bar{S}$ and putting it into S_1 for the left child and into S_0 for the right child. An example is shown in Figure 2.

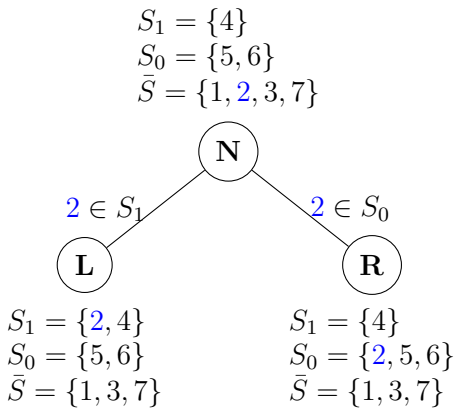

Figure 2.: A branching example with $x \in \mathbb{R}^7$, which operates on the variable x_2. The sets S_1, S_0, $\bar{S}$ are correspondingly updated on the children nodes **L** and **R**.

Algorithm 1 Branch & Bound algorithm for optimization of $(\mathcal{P}_{2+0})$.

1: **procedure** BRANCHANDBOUND($\mathcal{L}$)
2: $\overline{\text{lb}} \leftarrow -\infty$, $\overline{\text{ub}} \leftarrow +\infty$ and $\hat{x} \leftarrow Null$
3: **while** $\mathcal{L}$ is not empty or *another stopping condition is not met* **do**
4: Pop a node $\mathbf{N}$ from $\mathcal{L}$. ▷ Exploration strategy
5: Divide $\mathbf{N}$ into two sub-nodes $\mathbf{L}$ and $\mathbf{R}$. ▷ Branching strategy
6: **for all** sub-node $\mathbf{N}_i$ in $\{\mathbf{L}, \mathbf{R}\}$ **do**
7: Compute a lower bound $\text{lb}_{\mathbf{N}_i}$ of $\mathbf{N}_i$ with solution $x_{\text{lb}}^{\mathbf{N}_i}$. ▷ Bounding
8: **if** $\text{lb}_{\mathbf{N}_i} < \overline{\text{ub}}$ **then**
9: Compute an upper bound $\text{ub}_{\mathbf{N}_i}$ of $\mathbf{N}_i$ with solution $x_{\text{ub}}^{\mathbf{N}_i}$. ▷ Bounding
10: **if** $\text{ub}_{\mathbf{N}_i} < \overline{\text{ub}}$ **then**
11: Update the best solution found: $\hat{x} \leftarrow x_{\text{ub}}^{\mathbf{N}_i}$ and $\overline{\text{ub}} \leftarrow \text{ub}_{\mathbf{N}_i}$.
12: **end if**
13: Push $\mathbf{N}_i$ in $\mathcal{L}$. ▷ Exploration strategy
14: **end if**
15: **end for**
16: **end while**
17: Compute the lowest lower bound: $\overline{\text{lb}} = \min_{\mathbf{N} \in \mathcal{L}} \text{lb}_{\mathbf{N}}$
18: **return** $(\overline{\text{lb}}, \overline{\text{ub}}, \hat{x})$
19: **end procedure**

Algorithm 1 describes the main steps of the branch-and-bound procedure. It returns the global minimizer $\hat{x}$ of $(\mathcal{P}_{2+0})$ if the algorithm is run until the node list $\mathcal{L}$ is empty. If another stopping condition is used (step 3), such as ϵ precision, a maximum number of iteration or a time limit, then $\hat{x}$ is the best solution found, and a certified enclosure of the global minimum of $(\mathcal{P}_{2+0})$, denoted $[\overline{\text{lb}}, \overline{\text{ub}}]$, is obtained. The basic components of such branch-and-bound algorithm are the following.

- The *exploration strategy* selects which node $\mathbf{N}$ should be divided first among the remaining search domain $\mathcal{L}$: it defines the scheduling of nodes (step 4 in Algorithm 1).
- The *bounding procedures* consist in computing, at each node $\mathbf{N}$:
  i) a lower bound $\text{lb}_{\mathbf{N}}$ of the corresponding subproblem (step 7);
  ii) a feasible solution $x_{\text{ub}}^{\mathbf{N}}$ and an upper bound $\text{ub}_{\mathbf{N}}$ of the global minimum (step 9).
- The *branching strategy* defines how the node is divided into children, that is, it selects which variable $j \in \bar{S}$ is moved into S_1 and S_0 (step 5).

Basic bounding procedures and the branching strategy used in this paper are discussed in Sections 1.3 and 1.4, respectively.

1.3. Bounding the nodes

At a given node $\mathbf{N}(S_1, S_0, \bar{S})$, that is, for a given configuration of the support space, the ℓ_0-norm term reads $\|x\|_0 = \|x_{S_0}\|_0 + \|x_{S_1}\|_0 + \|x_{\bar{S}}\|_0$ where, by definition, $x_{S_0} = 0$ and $\|x_{S_1}\|_0 = |S_1|$ is fixed, so that the corresponding subproblem reads:

$$\min_x \tfrac{1}{2}\|y - Ax\|_2^2 + \mu|S_1| + \mu\|x_{\bar{S}}\|_0 \quad \text{s.t.} \quad \left\{ \begin{array}{c} \|x\|_\infty \leq M, \\ x_{S_0} = 0. \end{array} \right. \tag{$\mathcal{P}^{\mathbf{N}}_{2+0}$}$$

This problem is still NP-hard due to the term $\|x_{\bar{S}}\|_0$. As usual in a branch-and-bound setting, the node $\mathbf{N}$ will be evaluated by providing an upper and a lower bound on $(\mathcal{P}^{\mathbf{N}}_{2+0})$.

An upper bound and a feasible solution can be computed as in [5] by considering $x_{\bar{S}} = 0$. This gives the following problem:

$$\mathrm{ub}_{\mathbf{N}} := \min_x \tfrac{1}{2}\|y - Ax\|_2^2 + \mu|S_1|, \quad \text{s.t.} \quad \left\{ \begin{array}{c} \|x\|_\infty \leq M, \\ x_{S_0} = 0, x_{\bar{S}} = 0. \end{array} \right. \tag{1}$$

Problem (1) gives an upper bound $\mathrm{ub}_{\mathbf{N}}$ and a feasible solution $x^{\mathbf{N}}$ for $(\mathcal{P}^{\mathbf{N}}_{2+0})$. It is a box-bounded least-squares problem depending only on variables x_{S_1}. We keep the lowest upper bound currently known in variable $\overline{\mathrm{ub}}$, as shown in Algorithm 1 (step 11). At any given iteration in the branch-and-bound procedure, $\overline{\mathrm{ub}}$ is our best candidate for the global minimum of $(\mathcal{P}_{2+0})$—it will be the global optimum if the support S_1 in (1) is proved to be the optimal one.

For the lower bound, we use convex relaxation, substituting the ℓ_0-norm term by an ℓ_1-norm one. Indeed, due to the ℓ_∞ box constraints, at any feasible point for $(\mathcal{P}^{\mathbf{N}}_{2+0})$, we have:

$$\|x_{\bar{S}}\|_0 = \sum_{\substack{i \in \bar{S} \\ x_i \neq 0}} 1 \geq \sum_{\substack{i \in \bar{S} \\ x_i \neq 0}} \frac{|x_i|}{M} = \frac{1}{M} \sum_{i \in \bar{S}} |x_i| = \frac{1}{M}\|x_{\bar{S}}\|_1.$$

Figure 1 illustrates this property. The lower bound at a given node $\mathbf{N}$ is therefore computed by solving:

$$\mathrm{lb}_{\mathbf{N}} := \min_x \tfrac{1}{2}\|y - Ax\|_2^2 + \mu|S_1| + \tfrac{\mu}{M}\|x_{\bar{S}}\|_1 \quad \text{s.t.} \quad \left\{ \begin{array}{c} \|x\|_\infty \leq M, \\ x_{S_0} = 0, \end{array} \right. \tag{$\mathcal{P}^{\mathbf{N}}_{2+1}$}$$

and we denote by $x^{\mathbf{N}}_{\mathrm{lb}}$ the corresponding minimizer. Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$ is a box-constrained ℓ_1-norm-penalized least-squares problem, sharing similarities with the LASSO in statistics [22] and basis pursuit denoising in signal processing [12], for which many dedicated efficient algorithms have been proposed over the two past decades [2].

1.4. Branching strategy

The branching strategy used in this paper is based on the maximum of amplitudes described in [4]. We select $j \in \bar{S}$ with the highest absolute value in the solution of the relaxed problem:

$$j \in \arg\max_{i \in \bar{S}} |x^{\mathbf{N}}_{\mathrm{lb}\ i}|$$

Contrary to variable selection rules proposed by generic MIP solvers, *e.g.* based on maximal or minimal infeasibility, this branching strategy is particularly suited to sparse approximation: it relies on the assumption that a component with high amplitude in the relaxed problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$ is likely to belong to the support of the original problem $(\mathcal{P}^{\mathbf{N}}_{2+0})$, therefore it tries to fill S_1 with the correct support in priority.

1.5. Contributions

Based on this framework, we now present our contributions for accelerating such branch-and-bound procedure. In Section 2, some state-of-the-art ℓ_1-norm optimization strategies are adapted to $(\mathcal{P}^{\mathbf{N}}_{2+1})$ (here, the ℓ_1 norm only operates on some variables and box constraints are added). Then, we leverage convex duality in order to speed up the pruning of suboptimal nodes by iteratively refining a lower approximation of $\mathrm{lb}_{\mathbf{N}}$. In Section 3, we exploit the non-differentiability of $(\mathcal{P}^{\mathbf{N}}_{2+1})$ in order to reduce the problem dimension by fixing some variables to 0 or $\pm M$, aiming to accelerate the resolution of $(\mathcal{P}^{\mathbf{N}}_{2+1})$. Finally, non-standard exploration strategies are designed in Section 4 and compared against state-of-the-art ones on instances of $(\mathcal{P}_{2+0})$.

2. Convex duality for early node pruning

2.1. Principle

At a given node $\mathbf{N}$, the relaxation problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$ is considered, whose optimal value gives a lower bound $\mathrm{lb}_{\mathbf{N}}$ of the corresponding subproblem $(\mathcal{P}^{\mathbf{N}}_{2+0})$. If this lower bound is greater than the lowest upper bound known, $\overline{\mathrm{ub}}$, then this node can be safely discarded since it cannot contain the optimal solution. We call such a node a suboptimal node.

The standard approach for pruning a node requires to exactly solve $(\mathcal{P}^{\mathbf{N}}_{2+1})$ in order to guarantee the value of the bound $\mathrm{lb}_{\mathbf{N}}$. We leverage convex duality to reduce the computational complexity required for proving such suboptimality. Indeed, $(\mathcal{P}^{\mathbf{N}}_{2+1})$ is convex and can be written as:

$$\begin{aligned} \min_{x\in\mathbb{R}^Q} \quad & P(x) := f(Ax) + g(x), \\ \text{with } & f(Ax) := \tfrac{1}{2}\|y - Ax\|_2^2 \\ \text{and } & g(x) := \mu|S_1| + \tfrac{\mu}{M}\|x_{\bar{S}}\|_1 + I_{[-M,M]^Q}(x) + I_{\{0\}^{|S_0|}}(x_{S_0}), \end{aligned} \tag{2}$$

where $I_{\mathcal{C}}(x)$ is the indicator function equal to 0 if $x \in \mathcal{C}$ and $+\infty$ otherwise. Let $\phi^*(w) := \sup_x w^T x - \phi(x)$ denote the Fenchel conjugate of any convex function ϕ. We use the Fenchel-Rockafellar theorem ([35], theorem 31.2) to get the dual problem associated with Problem (2), which reads:

$$\max_{w\in\mathbb{R}^N} \quad D(w) := -f^*(w) - g^*(-A^T w). \tag{3}$$

It comes that:

$$f^*(w) = \tfrac{1}{2}(\|w + y\|_2^2 - \|y\|_2^2)$$

and the Fenchel conjugate of g can be obtained by:

$$\begin{aligned}
g^*(u) &= \sup_x \underbrace{\left(u^T x - \mu|S_1| - \tfrac{\mu}{M}\|x_{\bar{S}}\|_1 - I_{[-M,M]^Q}(x) - I_{\{0\}^{|S_0|}}(x_{S_0})\right)}_{\text{separable}}, \\
&= \sum_{i\in\bar{S}} \sup_{-M\le x_i\le M} (u_i x_i - \tfrac{\mu}{M}|x_i|) + \sum_{i\in S_1} \sup_{-M\le x_i\le M} u_i x_i \\
&\quad + \sum_{i\in S_0} \sup_{-M\le x_i\le M} \underbrace{(u_i x_i - I_{\{0\}}(x_i))}_{=0} - \mu|S_1|, \\
&= \sum_{i\in\bar{S}} \sup_{0\le|x_i|\le M} (|u_i||x_i| - \tfrac{\mu}{M}|x_i|) + \sum_{i\in S_1} \sup_{0\le|x_i|\le M} |u_i||x_i| - \mu|S_1|, \\
&= \sum_{i\in\bar{S}} \sup_{0\le|x_i|\le M} \left(|x_i|(|u_i| - \tfrac{\mu}{M})\right) + \sum_{i\in S_1} |u_i|M - \mu|S_1|, \\
&= M\left(\sum_{i\in\bar{S}} \max(0, |u_i| - \tfrac{\mu}{M}) + \sum_{i\in S_1} |u_i|\right) - \mu|S_1|.
\end{aligned}$$

The duality gap is defined as:

$$G(x,w) = P(x) - D(w) = f(Ax) + g(x) + f^*(w) + g^*(-A^T w) \tag{4}$$

Let $x^\star = \arg\min_{x\in\mathbb{R}^Q} P(x)$ and $w^\star = \arg\max_{w\in\mathbb{R}^N} D(w)$. We have the following KKT necessary and sufficient optimality conditions:

$$w^\star \in \partial f(Ax^\star) \tag{5a}$$
$$-A^T w^\star \in \partial g(x^\star) \tag{5b}$$
$$Ax^\star \in \partial f^*(w^\star) \tag{5c}$$
$$x^\star \in \partial g^*(-A^T w^\star) \tag{5d}$$

where ∂f denotes the subdifferential of function f, defined as [35]:

$$\partial f(x) := \{u \in \mathbb{R}^N \,|\, \forall y \in \mathbb{R}^N,\, f(y) \ge f(x) + u^T(y-x)\}$$

Weak duality states that $P(x) \ge D(w),\ \forall x, w$. In particular, this is true at the optimal value of the primal problem: $P(x^\star) \ge D(w),\ \forall w$. Therefore we have the following property:

Proposition 2.1. *If $\exists\, w \in \mathbb{R}^N$ such that $D(w) \ge \overline{ub}$, then $P(x^\star) \ge \overline{ub}$, therefore the addressed node can be pruned.*

We use Proposition 2.1 to early stop a given algorithm implemented for solving Problem ($\mathcal{P}^{\mathbf{N}}_{2+1}$). Let $\{x^k\}_k$ be a sequence of iterates produced by such algorithm. Equation (5a) means that, at optimality, we have $w^\star = Ax^\star - y$. Therefore, we compute the dual objective function $D(w^k)$ at points of the form $w^k = Ax^k - y$. Then, if at an iteration k, we have $D(w^k) \ge \overline{\text{ub}}$, then the algorithm solving ($\mathcal{P}^{\mathbf{N}}_{2+1}$) can be stopped and the node can be safely pruned, as illustrated in Figure 3 (left).

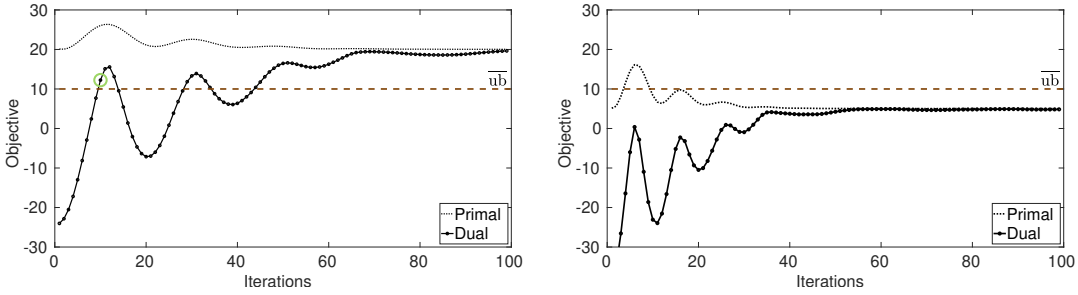

Figure 3.: Early pruning illustration: primal and dual iterates for the optimization of $(\mathcal{P}^{\mathbf{N}}_{2+1})$ (dashed and full lines, respectively), and the best known upper bound (dotted horizontal line). Left: pruning of the node is achieved after convergence of the primal descent algorithm minimizing $(\mathcal{P}^{\mathbf{N}}_{2+1})$ (here in 100 iterations), but the dual value after 10 iterations informs that the node can be pruned (green circle). Right: in this case, the lower bound $\text{lb}_{\mathbf{N}}$ is too low and the node cannot be pruned.

2.2. Dual bound quality of ℓ_1-norm algorithms

Many algorithms were developed to solve ℓ_1-norm-penalized least-squares problems, that is, instances of Problem (2) with $g(x) = \|x\|_1$. Indeed, such convex, non-smooth, optimization problems, benefit from analytical properties that make dedicated approaches much more efficient than generic quadratic programming solvers. In this section, several such methods are extended to the optimization of $(\mathcal{P}^{\mathbf{N}}_{2+1})$, recast as Problem (2). Their efficiency to obtain satisfactory dual bounds will be compared in the experimental Section 5.3.

2.2.1. Proximal algorithms

Proximal algorithms rely on the monotone operator theory [3] to optimize problems of the form (2), with g non-differentiable, by iteratively performing a descent step on f first, and then on g. If f is differentiable, which is the case for $(\mathcal{P}^{\mathbf{N}}_{2+1})$, we can perform a gradient step on f. The descent step on g is performed thanks to the proximal operator [27]:

$$\underset{\lambda g}{\text{Prox}}(x^k) := \arg\min_x \tfrac{1}{2}\|x - x^k\|_2^2 + \lambda g(x). \tag{6}$$

Forward-backward splitting. In the case where $g(x) = \|x\|_1$, the resulting *forward-backward splitting* (FBS) algorithm[16] is called *Iterative Soft-Thresholding Algorithm* (ISTA) [14]. Such a procedure can be extended to our optimization problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$. Let

$$v := \underset{\lambda g}{\text{Prox}}(x^k) = \arg\min_x \tfrac{1}{2}\|x - x^k\|_2^2 + \lambda \left(\tfrac{\mu}{M}\|x_{\bar{S}}\|_1 + I_{[-M,M]^Q}(x) \right).$$

We can show that this operator reads componentwise:

$$\forall i \in S_1, v_i = \arg\min_{x_i} \tfrac{1}{2}(x_i - x_i^k)^2 + \lambda I_{[-M,M]}(x_i) = C_{[-M,M]}(x_i^k), \tag{7}$$

with $C_{[-M,M]}$ the *capping* operator : $C_{[-M,M]}(u) := \begin{cases} u & \text{if } u \in [-M, M] \\ M & \text{if } u \geq M \\ -M & \text{if } u \leq -M \end{cases}$,

and:

$$\forall i \in \bar{S}, v_i = \arg\min_{x_i} \tfrac{1}{2}(x_i - x_i^k)^2 + \lambda \tfrac{\mu}{M}|x_i| + \lambda I_{[-M,M]}(x_i) = CST^{\mu/M}_{[-M,M]}(x_i^k), \tag{8}$$

with $CST^{\alpha}_{[-M,M]}$ the *capped thresholding operator:*

$$CST^{\alpha}_{[-M,M]}(u) := \begin{cases} 0 & \text{if } |u - \alpha \text{sign}(u)| \leq \alpha \\ C_{[-M,M]}(u - \alpha \text{sign}(u)) & \text{otherwise} \end{cases}.$$

The resulting FBS procedure is given in Algorithm 2, where x^0 is any intial point and parameter $L > 0$ must be greater than the spectral radius of matrix $A^T A$ in order to ensure convergence [14]. This convergence is defined as an ϵ precision on the duality gap: $G(x^k, Ax^k - y) \leq \epsilon$, with G defined in Equation (4). The computation cost of one iteration is mainly that of the matrix-vector products involved at step 4.

Algorithm 2 Forward-Backward Splitting algorithm for evaluation of node **N** (lower bound)

1: **procedure** FBS(y, A, μ, x^0, L, $\mathbf{N}(S_1, S_0, \bar{S})$)
2: $k \leftarrow 0$
3: **while** not convergence **do**
4: $x^{k+1/2} = x^k - \frac{1}{L}A^T(Ax^k - y)$ ▷ Gradient step on f
5: $x^{k+1} \leftarrow \text{Prox}_{g/L}(x^{k+1/2})$ ▷ Proximal step on g with (7)–(8)
6: $k \leftarrow k + 1$
7: **end while**
8: **return** x^k
9: **end procedure**

Chambolle-Pock (primal-dual) algorithm. Primal-dual algorithms use the primal and the dual problems to perform the optimization. The Chambolle-Pock algorithm is acknowledged as an efficient algorithm in this family [11], which aims to find the saddle point of:

$$\min_x \max_w w^T Ax + g(x) - f^*(w). \tag{9}$$

It does so by taking a proximal step on f^* and then a proximal step on g. Such primal-dual algorithm generates valid dual points during the optimization, which can be directly used for our early pruning strategy.

In our case, the proximal step on f^* reads:

$$\begin{aligned} u := \underset{\lambda f^*}{\text{Prox}}(w^k) &= \arg\min_w \tfrac{1}{2}\|w - w^k\|_2^2 + \lambda(\tfrac{1}{2}\|w + y\|_2^2 - \underbrace{\tfrac{1}{2}\|y\|_2^2}_{\text{constant}}), \\ &= \frac{w^k - \lambda y}{1 + \lambda}. \end{aligned} \tag{10}$$

Algorithm 3 gives the pseudo-code of the Chambolle-Pock algorithm, with notations similar to Algorithm 2, and $w^0 \in \mathbb{R}^N$ any feasible initial dual point. In our case where f^* is 1-strongly convex, acceleration steps (6–8) are included (see [11]), with additional parameter $\gamma = 1$ (the strong convexity modulus of f^*), $\tau^0 > 0$ and $\sigma^0 > 0$ initial proximal parameters such that $\tau^0\sigma^0 L^2 \leq 1$. The computation cost

per iteration is similar to that of FBS, mainly corresponding to the matrix-vector products involved at steps 4 and 5 of Algorithm 3.

Algorithm 3 Chambolle-Pock algorithm for evaluation of node **N** (lower bound)

1: **procedure** CHAMBOLLE-POCK(y, A, μ, x^0, L, γ, w^0, τ^0, σ^0, $\mathbf{N}(S_1, S_0, \bar{S})$)
2: $k \leftarrow 0$, $\overline{x} \leftarrow x^0$
3: **while** not convergence **do**
4: $w^{k+1} \leftarrow \text{Prox}_{\sigma^k f^*}(w^k + \sigma^k A\overline{x}^k)$ ▷ Proximal step on f^* with (10)
5: $x^{k+1} \leftarrow \text{Prox}_{\tau^k g}(x^k - \tau^k A^T w^{k+1})$ ▷ Proximal step on g with (7)–(8)
6: $\phi^k \leftarrow \frac{1}{\sqrt{1+2\gamma\tau^k}}$
7: $\tau^{k+1} \leftarrow \phi^k \tau^k$
8: $\sigma^{k+1} \leftarrow \sigma^k / \phi^k$
9: $\overline{x}^{k+1} = x^{k+1} + \phi^k(x^{k+1} - x^k)$
10: $k \leftarrow k + 1$
11: **end while**
12: **return** (x^k, w^k)
13: **end procedure**

2.2.2. Coordinate descent algorithms

Coordinate descent (CD) algorithms were also proposed for ℓ_1-norm-penalized least squares [1]—see [44] for a convergence proof. In general, CD is a rather inefficient scheme compared to first-order methods. However, in this setting, scalar subproblems have an analytical solution, which can be computed very efficiently. Indeed, let $x_i^* = \arg\min_{x_i} P(x)$ for $i \in \bar{S} \cup S_1$ (for $i \in S_0$, $x_i^* = 0$). Let A_i denote the i-th column of matrix A and let $e_i := y - \sum_{j \neq i} x_j A_j$. One can show that each scalar update reads:

$$\forall i \in S_1, x_i^* = \arg\min_{x_i} \tfrac{1}{2}\|e_i - x_i A_i\|_2^2 + I_{[-M,M]}(x_i) = C_{[-M,M]}\left(\frac{A_i^T e_i}{A_i^T A_i}\right), \tag{11}$$

$$\text{and } \forall i \in \bar{S}, x_i^* = \arg\min_{x_i} \tfrac{1}{2}\|e_i - x_i A_i\|_2^2 + \tfrac{\mu}{M}|x_i| + I_{[-M,M]}(x_i) = CST_{[-M,M]}^{\mu/M}\left(\frac{A_i^T e_i}{A_i^T A_i}\right), \tag{12}$$

where C and CST respectively denote the capping and the capped thresholding operators introduced in Section 2.2.

Moreover, smart cycling rules enable strong accelerations (see [20] for standard ℓ_1-norm problems). In particular, we can rely on the sparse nature of the solutions by introducing coordinate sweeping rules that most frequently update nonzero values, and only periodically update all variables. The resulting algorithm is given in Algorithm 4, where parameter J controls the period at which a full update is performed.

The complexity for updating all variables (steps 11–13) is essentially that of two matrix-vector products with matrix A and one with matrix A^T. When cycling is performed only on current nonzero variables, computations are restricted to the corresponding variables and are therefore strongly reduced. However, each CD iteration is usually less efficient than a gradient step to decrease the objective function.

The three methods proposed in Algorithms 2 to 4 are *asymptotic* ones, that is, they converge as the number of iterates $k \to +\infty$. We now consider two *exact* methods, meaning that the minimizer is found in a finite (although potentially large) number of iterations.

Algorithm 4 Coordinate descent algorithm for evaluation of node **N** (lower bound)

procedure ICD(y, A, μ, J, x^0, $\mathbf{N}(S_1, S_0, \bar{S})$)
  $k \leftarrow 0$
  $e \leftarrow y - Ax^0$
  while not convergence **do**
    if $k \mod J = 0$ **then**
      CyclingIndices $\leftarrow \{1, ..., Q\}$ ▷ Full cycling
    else
      CyclingIndices $\leftarrow \{i | x_i^k \neq 0\}$ ▷ Partial cycling
    end if
    for i in CyclingIndices **do**
      $e \leftarrow e + A_i x_i^k$ ▷ Cancel the contribution of x_i^k
      Compute $x_i^{k+1} = x_i^*$ using (11)–(12)
      $e \leftarrow e - A_i x_i^{k+1}$ ▷ Add the contribution of x_i^{k+1}
    end for
    $k \leftarrow k + 1$
  end while
  return x^k
end procedure

2.2.3. Active sets

Active set methods [32] were initially proposed for linearly-constrained quadratic programs. They handle the different inequality constraints of a given (quadratic) problem by choosing a number of constraints to be active (saturated), the rest being inactive. Given the set of constraints believed to be active, the problem of interest is reformulated by equating the constraints in the active set and dropping the remaining ones. After a descent step, the active set of constraints is updated to take into account violated constraints, and the procedure is iterated until all optimality conditions are satisfied.

Active set methods have been be proposed for problems involving the ℓ_1 norm [25, 37], and an adaptation to $(\mathcal{P}_{2+1}^{\mathbf{N}})$ was proposed in [4], that will be used in this paper.

2.2.4. Homotopy continuation

The homotopy continuation method was specifically designed for ℓ_1-norm-penalized least squares [17, 33]. Indeed, the minimizer can be shown to be a piecewise linear function of the penalty parameter, say μ. Starting from sufficiently high μ^0 for which the solution is identically zero, a decreasing sequence of critical values for μ is built (*breakpoints* μ^k), at which the configuration of the support is modified. The support is then updated at each breakpoint, and the procedure runs until the target penalty parameter is reached. In [5], an homotopy continuation algorithm was built for solving problem $(\mathcal{P}_{2+1}^{\mathbf{N}})$. The reader is referred to this paper for implementation details.

3. Leveraging convex screening

Screening methods aim to assign optimal values to some variables in a given optimization problem, prior to (or during) its numerical resolution, thanks to some insight provided by the problem structure. Therefore, it reduces the dimension of the problem to be solved, hoping to reduce the computational load. In this section, we discuss the use of screening methods for the convex relaxation problem $(\mathcal{P}_{2+1}^{\mathbf{N}})$, involved at each node of the branch-and-bound Algorithm 1.

3.1. Principle

Due to the non-differentiability of $|x_i|$ at 0, the solution to the standard ℓ_1-norm-penalized least-squares problem contains zero values [28]. Related screening methods therefore aim to fix some variables to zero. We can date back the first works in this field to [18]. In our case, with $(\mathcal{P}^{\mathbf{N}}_{2+1})$, particular points include zero values for x_i, $i \in \bar{S}$, but also values at the bounds $\pm M$ for x_i, $i \in \bar{S} \cup S_1$.

The literature distinguishes between *safe* screening, where the values of fixed variables are proved to be optimal, and *aggressive* or *strong* screening, which only benefits from a probabilistic guarantee (that is, optimality of the resulting solution may be lost) but, in return, it may discard more variables. In our case, we need the node evaluation to provide a guaranteed lower bound on problem $(\mathcal{P}^{\mathbf{N}}_{2+0})$ to ensure the validity of the branch-and-bound procedure, therefore we only consider safe screening.

We consider *dynamic* screening, in which screening tests are performed throughout the iterations of the optimization procedure [9]. Such tests rely on the knowledge of a feasible dual point (w with notations of Section 2.1), around which a subspace containing the optimal dual point $w^\star$ is created, called the *safe region*. To fix a variable, the screening test must hold for any dual point in the safe region, whose size depends on the distance between w and $w^\star$. As w gets closer to $w^\star$, the safe region gets smaller and screening becomes more powerful (that is, more variables can be fixed). In this paper, we resort to gap-safe screening [19, 30], which builds safe regions based on the duality gap (4). Such a procedure is based on two steps:

- building *screening rules*, based on the optimality conditions of the problem;
- defining a *safe region*, which contains the optimal dual point.

These two steps are now addressed in Sections 3.2 and 3.3, respectively.

3.2. Screening rules for relaxation problems

The screening rules exploit the KKT optimality conditions (5), and in particular Equation (5b), which states that $-A^T w^\star \in \partial g(x^\star)$, with $(x^\star, w^\star)$ the optimal primal-dual pair of Problems (2) and (3), respectively.

Proposition 3.1. *We have the following screening rules for Problem* $(\mathcal{P}^{\mathbf{N}}_{2+1})$*:*

$$\forall i \in \bar{S}, \text{ if } |A_i^T w^\star| > \tfrac{\mu}{M}, \text{ then } x_i^\star = -M \operatorname{sign}(A_i^T w^\star); \tag{13a}$$

$$\forall i \in \bar{S}, \text{ if } |A_i^T w^\star| < \tfrac{\mu}{M}, \text{ then } x_i^\star = 0; \tag{13b}$$

$$\forall i \in S_1, \text{ if } A_i^T w^\star \neq 0, \text{ then } x_i^\star = -M \operatorname{sign}(A_i^T w^\star). \tag{13c}$$

Proof. From the expression of g in Problem (2), its sub-differential reads:

$$\nu := \partial g(x) = \partial \left(\tfrac{\mu}{M} \|x_{\bar{S}}\|_1 \right) + \partial \left(I_{[-M,M]^Q}(x) \right).$$

As it is separable, its i-th component reads:

$$\begin{aligned} \forall i \in \bar{S},\ \nu_i &= \partial \left(\tfrac{\mu}{M} |x_i| \right) + \partial \left(I_{[-M,M]}(x_i) \right) \\ \text{and } \forall i \in S_1,\ \nu_i &= \partial \left(I_{[-M,M]}(x_i) \right), \end{aligned}$$

with $\partial |x_i| = \begin{cases} \operatorname{sign}(x_i) & \text{if} \quad x_i \neq 0 \\ [-1,1] & \text{if} \quad x_i = 0 \end{cases}$ and $\partial I_{[-M,M]}(x_i) = \begin{cases} [0, +\infty[& \text{if} \quad x_i = M \\ 0 & \text{if} \quad |x_i| < M \\ ]-\infty, 0] & \text{if} \quad x_i = -M \end{cases}$.

Therefore, the optimality condition $-A^T w^\star \in \partial g(x^\star)$ can be separated into the following cases:

$$\begin{cases} \forall i \in \bar{S} & \text{with} & x_i^\star = M, & -A_i^T w^\star \in [\frac{\mu}{M}, +\infty[\\ \forall i \in \bar{S} & \text{with} & x_i^\star \in]0, M[, & -A_i^T w^\star = \frac{\mu}{M} \\ \forall i \in \bar{S} & \text{with} & x_i^\star = 0, & -A_i^T w^\star \in [-\frac{\mu}{M}, \frac{\mu}{M}] \\ \forall i \in \bar{S} & \text{with} & x_i^\star \in]-M, 0[, & -A_i^T w^\star = -\frac{\mu}{M} \\ \forall i \in \bar{S} & \text{with} & x_i^\star = -M, & -A_i^T w^\star \in]-\infty, -\frac{\mu}{M}] \\ \forall i \in S_1 & \text{with} & x_i^\star = M, & -A_i^T w^\star \in [0, +\infty[\\ \forall i \in S_1 & \text{with} & x_i^\star \in]-M, M[, & -A_i^T w^\star = 0 \\ \forall i \in S_1 & \text{with} & x_i^\star = -M, & -A_i^T w^\star \in]-\infty, -0] \end{cases} . \tag{14}$$

From Equation (14), the rules of Proposition 3.1 follow. □

These rules are illustrated in Figure 4.

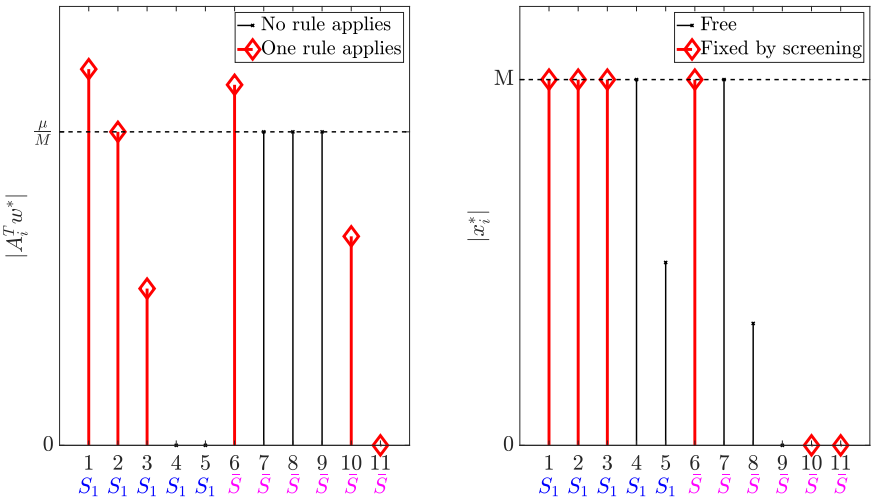

Figure 4.: An example of screening possibilities (◇), for components with indices $\{1, \ldots, 5\} \in S_1$ and $\{6, \ldots, 11\} \in \bar{S}$. For $i \in S_1$, if $A_i^T w^\star \neq 0$, then $x_i^\star$ is fixed to $\pm M$. If $A_i^T w^\star = 0$, then $x_i^\star$ can have any value, including $\pm M$. For $i \in \bar{S}$, if $|A_i^T w^\star| > \frac{\mu}{M}$, then $x_i^\star$ is fixed to $\pm M$. If $|A_i^T w^\star| = \frac{\mu}{M}$, then $x_i^\star$ can have any value, including 0 and $\pm M$. If $|A_i^T w^\star| < \frac{\mu}{M}$, then $x_i^\star = 0$.

3.3. From screening rules to screening tests

The screening rules of Proposition 3.1 require the knowledge of $w^\star$ the optimal dual point, which is unknown during the optimization. Given a dual point w, safe screening methods create a region $\mathcal{R}$ from w such that $w^\star \in \mathcal{R}$, by exploiting the structure of the problem. If one of the rules in Proposition 3.1 is valid for every point in $\mathcal{R}$, then it is valid for $w^\star$, which means that the corresponding variable x_i can be fixed according to the rule. Testing every point of a region $\mathcal{R}$ is inefficient, therefore sphere regions $\mathcal{B}(w, r)$, centered at w and with some radius r are usually considered [18, 30], based on a worst-case analysis: one just has to test the center of the sphere c, and modify the thresholds of rules in Proposition 3.1 by the radius r of the sphere. Indeed, for every dual point w, we have:

$$|A_i^T w^\star| \leq |A_i^T w^\star - A_i^T w| + |A_i^T w|$$

Without loss of generality, we can suppose that the columns of matrix A have unit ℓ_2 norm. Then, using Cauchy-Schwarz inequality, we have:

$$|A_i^T w^\star - A_i^T w| \leq \|A_i\|_2 \|w^\star - w\|_2 = \|w^\star - w\|_2,$$

so that $|A_i^T w^\star| \leq |A_i^T w| + \|w^\star - w\|_2$. Consequently, if $|A_i^T w| < \frac{\mu}{M} - r$, with $\|w^\star - w\|_2 \leq r$, then $|A_i^T w^\star| < \frac{\mu}{M}$, and according to Proposition 3.1, $x_i^\star = 0$. The same reasoning applies for the rules (13a) and (13c).

We will use gap-safe screening, which uses the duality gap $G(x, w) = P(x) - D(w)$ defined in Equation (4) to build such safe spheres. We have the following proposition, adapted from [30, 34] to Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$:

Proposition 3.2. *For Problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$, using any feasible primal point x and any dual point w, we have the following gap-safe tests:*

$$\forall i \in \bar{S}, \text{if } |A_i^T w| > \tfrac{\mu}{M} + \sqrt{2G(x,w)}, \text{ then } x_i^\star = -M \operatorname{sign}(A_i^T w^\star) \tag{15}$$

$$\forall i \in \bar{S}, \text{if } |A_i^T w| < \tfrac{\mu}{M} - \sqrt{2G(x,w)}, \text{ then } x_i^\star = 0 \tag{16}$$

$$\forall i \in S_1, \text{if } |A_i^T w| > \sqrt{2G(x,w)}, \text{ then } x_i^\star = -M \operatorname{sign}(A_i^T w^\star) \tag{17}$$

Proof. Using an arbitrary dual point w, we have [34]:

$$G(x,w) = P(x) - D(w) \geq P(x^\star) - D(w) = D(w^\star) - D(w) = -f^*(w^\star) - g^*(-A^T w^\star) + f^*(w) + g^*(-A^T w).$$

Since f^* is 1-strongly convex, we have:

$$f^*(w) \geq f^*(w^\star) + \nabla f^*(w^\star)^T (w - w^\star) + \tfrac{1}{2}\|w - w^\star\|_2^2.$$

Since g^* is convex, we have:

$$g^*(-A^T w) \geq g^*(-A^T w^\star) + \left(\partial g^*(-A^T w^\star)\right)^T (-A^T w + A^T w^\star).$$

Therefore we have:

$$\begin{aligned} G(x,w) \geq & -f^*(w^\star) - g^*(-A^T w^\star) \\ & + f^*(w^\star) + \nabla f^*(w^\star)^T (w - w^\star) + \tfrac{1}{2}\|w - w^\star\|_2^2 \\ & + g^*(-A^T w^\star) + \left(\partial g^*(-A^T w^\star)\right)^T (-A^T w + A^T w^\star), \\ = & \nabla f^*(w^\star)^T (w - w^\star) + \left(\partial g^*(-A^T w^\star)\right)^T (-A^T w + A^T w^\star) + \tfrac{1}{2}\|w - w^\star\|_2^2. \end{aligned}$$

Due to the first optimality condition, we have:

$$\nabla f^*(w^\star)^T (w - w^\star) + \left(\partial g^*(-A^T w^\star)\right)^T (-A^T w + A^T w^\star) \geq 0.$$

Thus $G(x,w) \geq \frac{1}{2}\|w - w^\star\|_2^2$, so $\|w - w^\star\|_2 \leq \sqrt{2G(x,w)}$ which implies $w^\star \in \mathcal{B}(w, \sqrt{2G(x,w)})$. Then, using the rules of Proposition 3.1, the tests of Proposition 3.2 follow. □

As the duality gap decreases, the safe region shrinks, and the screening tests get better at fixing variables. Therefore, screening tests are expected to be more efficient when used with algorithms that fastly increase the dual objective, as for the early pruning rules described in Section 2.1. Their practical performance will be studied in the numerical experiments of Section 5.4.

Both the early pruning described in Section 2 and the screening method discussed in this section focused on the computation of the lower bound at each node: these

are local enhancements. On the contrary, exploration rules discussed in Section 4 affect the global behavior of the branch-and-bound procedure.

4. Exploration strategies

4.1. Motivation

The *exploration* strategy defines which node should be divided first, that is, it generates a scheduling of the nodes in the list (see step 4 in Algorithm 1). This node scheduling impacts the global performance of the branch-and-bound algorithm [31]. Figure 5 gives an illustrative example, where a binary tree with four depth levels (that is, 16 leaves for a total of 31 nodes) is explored either by a depth-first search (top) or by a best-first search (bottom) strategy. This tree follows the convention of Figure 2 regarding the left and right childs (one variable is put in S_1 and in S_0, respectively). Depth-first search seeks to fully develop a branch of the search tree before jumping to another branch. On the contrary, best-first search picks the node with the highest potential of improvement over the current objective value, that is, it selects the node with the lowest (best) lower bound. Depth-first search tends to fast provide good feasible solutions by favoring the nodes that are considered as "good" by the branching strategy. In this example (where the optimal objective value is 16), a solution with objective value equal to 18 is found in only 4 iterations, but it requires 26 nodes to prove optimality. Indeed, 3 nodes were pruned, avoiding the creation of 6 nodes. On the other hand, best-first search proves optimality more quickly, with the optimal solution proved in 22 nodes: 8 nodes were pruned and the creation of 10 nodes was avoided, by first increasing the lowest lower bound in the scheduling. However, it is not very efficient to quickly find good solutions: in 4 iterations, the best solution found has an objective value of 32.

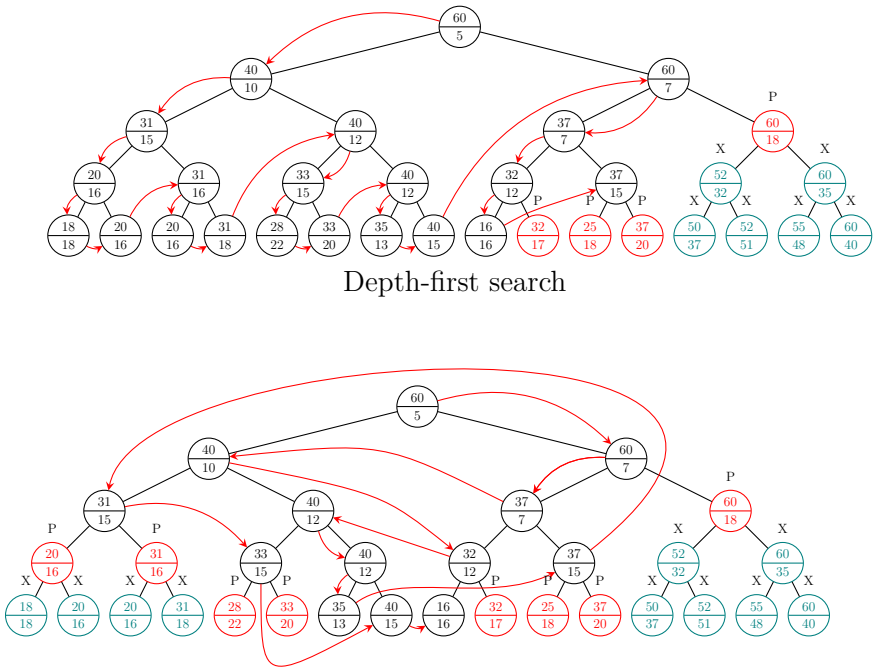

Figure 5.: Two different exploration strategies operating on a branch-and-bound tree. Depth-first search (top) and best-first search (bottom). Numerical values in the upper (respectively lower) part of each node $\mathbf{N}$ indicates its corresponding upper bound $\text{ub}_{\mathbf{N}}$ (respectively its lower bound $\text{lb}_{\mathbf{N}}$) . The red arrows illustrate the path followed by the exploration strategy. A node denoted by P is a pruned node: its children are not created, saving all the nodes marked by an X.

4.2. Analogy with the data structure

Each exploration strategy defines a scheduling of the nodes that are stored in the variable $\mathcal{L}$ in Algorithm 1. Instead of considering $\mathcal{L}$ as a list and the exploration strategy as a sorting function on $\mathcal{L}$, we merge both concepts by defining exploration strategies through their corresponding data structure. Indeed, the scheduling of nodes is implemented by the Push and Pop functions in Algorithm 1 (steps 4 and 13), defining a specific data structure representing a specific exploration strategy.

Standard exploration strategies [31] include *breadth-first search*, *depth-first search* and *best-first search*. Breadth-first search gives priority to the closest node to the root node. In our search tree, it would symmetrically explore the branches where a variable is included in the support (left child) and branches where such variable is removed from the support (right child, see Section 1.2). This rather seems unsuitable to our problem, whose structure is not symmetrical since we seek for sparse solutions, that is, the size of the support is small. On the contrary, as introduced in Section 4.1, depth-first search first explores the deepest node in the tree, which was shown to be an efficient strategy in the context of sparse optimization [5, 23]. It amounts to first choosing one of the children of the node that was just divided. From a data structure viewpoint, this corresponds to $\mathcal{L}$ being a *stack*, also known as *LIFO* (Last In First Out). Best-first search (also introduced in Section 4.1) first selects the node with the lowest lower bound of the objective function. It can be implemented with a *heap*, which is a partially ordered data structure such that the first element of the heap (the one extracted by the Pop function) always has the lowest score among all elements. In the best-first search case, the score used in the heap is the lower bound, and we say this is a heap *sorted* on the lower bound (note that a heap is always only

partially sorted, not fully sorted).

4.3. Dedicated exploration strategies

Building on this data structure view, we propose new exploration strategies specific to our problem. The first two ones are variations of best-first search. Indeed, best-first search schedules nodes according to their lower bound, obtained by solving problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$, which is the sum of a least-squares term and an ℓ_1-norm term. In order to analyze if one of the two terms dominates in the exploration efficiency, we consider the two following strategies:

- *least-squares first*, which is a heap sorted on the value of the least-squares term at the solution of problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$, that is, $\frac{1}{2}\|y - Ax^{\mathbf{N}}_{\text{lb}}\|^2_2$,
- ℓ_1 *first*, which is a heap sorted on the ℓ_1-norm term at the solution of problem $(\mathcal{P}^{\mathbf{N}}_{2+1})$, that is, $\frac{\mu}{M}\|x^{\mathbf{N}}_{\text{lb}}\|_1$.

Additionally, in order to investigate the benefits brought by both depth-first search (that quickly finds good feasible solutions) and best-first search (that is better for proving optimality), we design *mixed strategies*, which behave like a stack (depth-first) at the beginning of the branch-and-bound algorithm, and then switch to a heap sorted on either the lower bound, its least-squares term, or its ℓ_1-norm term.

5. Experimental results

This section is devoted to numerical experiments evaluating the different contributions proposed in Sections 2 to 4. We first describe the numerical setup. Then, Section 5.2 evaluates the performance of the different algorithms described in Section 2.2 in their ability to increase the dual objective function. The homotopy method is shown to outperform its competitors, and is therefore chosen as the best algorithm for the next Sections 5.3 and 5.4 exploiting duality, where the efficiency of node pruning and variable screening are respectively studied. Finally, Section 5.5 compares the different exploration strategies.

5.1. Data-set description

We consider synthetic problems with random matrices as in [5, 6]: the rows of the problem matrix A are drawn according to a multivariate centered normal distribution $\mathcal{N}(0, \Sigma)$, with covariance matrix Σ defined by $\Sigma_{ij} = \rho^{|i-j|}$. As ρ increases, the correlation between the columns of A increases and the sparse recovery problem becomes more difficult, both from informational and computational points of view [5]. We consider two correlation levels $\rho \in \{0.8, 0.92\}$. Columns are scaled so that $\|A_i\|_2 = 1\ \forall i \in \{1, ..., Q\}$. The dimension of y is $N = 500$, the dimension of x is $Q = 100$. We build problem instances by generating a sparse vector x_{truth} with a random support of K components equal to 1, and K varying from 3 to 9[1]. Unless otherwise stated, results are averaged over 10 instances for each value of K. Synthetic data are generated as $y = Ax_{\text{truth}} + \xi$, with noise vectors $\xi \sim \mathcal{N}(0, \sigma^2 I)$, σ controlling the amount of noise in the data such that $\text{SNR} := \|Ax_{\text{truth}}\|^2_2/(N\sigma^2) = 6$. The penalization parameter μ in Problem $(\mathcal{P}_{2+0})$ is tuned empirically so that all solutions have the expected ℓ_0 pseudo-norm, K. We set M to $M := 1.1\|A^T y\|_\infty$ for each instance.

All experiments were performed with our own branch-and-bound code implemented in C++ named `Mimosa`, running on a single thread, using a laptop computer on Ubuntu 20.04.1 with 32 Go RAM and an Intel Core i7-10610U processor clocked at 1.8 GHz. The installation procedure of `Mimosa` is given in Section 6.

[1]Dataset available at `https://gitlab.univ-nantes.fr/samain-g/mimosa-oms-dataset`

5.2. *Optimization for primal and dual objective functions*

In this section, we empirically evaluate the convergence speed, for both the primal and the dual objective functions (Problems (2) and (3) respectively), for each algorithm described in Section 2.2: foward-backward splitting (FBS), Chambolle-Pock primal-dual iterations (ChaPo), coordinate descent (CoordDesc), active set (ActSet) and homotopy continuation (HomCont).

By construction, ChaPo generates primal and dual iterates. FBS, CoordDesc and ActSet generate sequences of primal iterates x^k, for which we consider the corresponding sequence of dual points using the KKT condition (5a), that is, $w^k = Ax^k - y$. HomCont requires an additional rescaling step that is described hereafter. Recall that HomCont generates iterates x^k which are the solutions to problem (2) for a decreasing sequence of values of penalization parameter $\mu^k > \mu$. Optimality conditions (14) show that the corresponding dual solution scales as μ^k. Therefore, we build a sequence of dual candidates for Problem (3) as $w^k = \frac{\mu}{\mu^k}(Ax^k - y)$. The reader is referred to [46] for similar arguments for implementing screening tests.

The left part of Figure 6 shows the typical evolution of the iterates of primal and dual objective values for the implemented algorithms, run at the root node (that is, $S_0 = S_1 = \varnothing$), for one instance extracted from the dataset generated in Section 5.1 with $\rho = 0.8, K = 9$. At first iterations, the primal objective function decreases faster with asymptotic methods (FBS, ChaPo, CoordDesc) than with exact ones (ActSet, HomCont), with FBS being the best and HomCont the worst. However, the exact methods converge in fewer iterations than asymptotic ones, with both ActSet and HomCont terminating in 31 iterations, while CoordDesc converges after 40 iterations and proximal algorithms (FBS and ChaPo) did not terminate after 100 iterations.

Looking at the dual iterates, HomCont is by far the best at first iterations and also converges first. CoordDesc first dual iterates are quite poor, then they quickly become close to the optimal value, although a quite large number of iterations is required for convergence. The first ActSet iterates are also very far from the optimal value, but then they behave similarly to HomCont. Finally, proximal algorithms suffer from slow dual convergence.

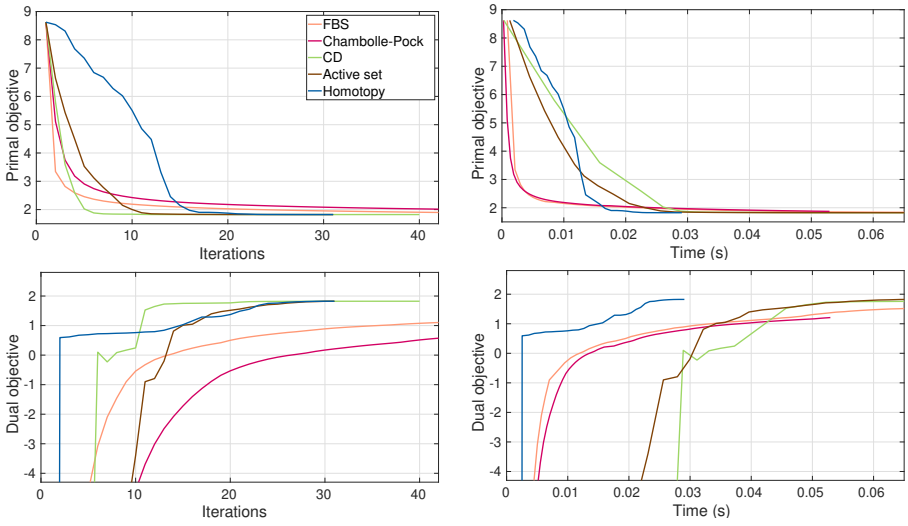

Figure 6.: Evolution of the primal (top) and dual (bottom) objective functions of a relaxation problem at the root node of the branch-and-bound algorithm, for different optimization strategies, as a function of the iteration number (left) and of the computating time (right).

Let us now compare the algorithms as a function of their computation time (right part of Figure 6). Indeed, the cost of one iteration of HomCont or ActSet is numerically more complex than one iteration of FBS, ChaPo or CoordDesc. CoordDesc is clearly slower than the proximal algorithms at first, then it becomes more

competitive at the very end, both for primal and dual iterates. ActSet primal iterates decrease finally slower than those of HomCont: if both strategies require the same number of iterations to converge, ActSet iterations are computationally more heavy. The behavior of dual iterates definitely disqualifies CoordDesc and ActSet methods, and HomCont is by far the fastest method that increases the dual objective.

In conclusion, due to its fast convergence to optimality as well as its best performance for improving the dual objective objective, we will select the homotopy continuation algorithm for the experiments in the next sections.

5.3. Duality-based early pruning

We now evaluate the performance of the duality-based early pruning strategy described in Section 2. For each problem instance, the branch-and-bound procedure given in Algorithm 1 is run, and at each node $\mathbf{N}$:

- we consider the number of iterations required by the homotopy method to compute the corresponding lower bound, say, $I^p_{\mathbf{N}}$;
- we consider the number of iterations after which the dual objective function exceeds the best known upper bound, say, $I^d_{\mathbf{N}}$;
- we define the relative saving in iterations, expressed as $\dfrac{I^p_{\mathbf{N}} - I^d_{\mathbf{N}}}{I^p_{\mathbf{N}}}$.

We only focus on the most difficult instances with $K = 9$ non-zero values (that is, we discard instances with lower cardinality). Figure 7 (top) reports the corresponding savings, drawn as a function of the cardinality of subset S_1 (the number of non-zero components included in the solution at a given node), obtained over 10 problem instances, with $\rho = 0.8$. Although very few iterations can be saved for small values of $|S_1|$, early pruning becomes more efficient as $|S_1|$ increases. In particular, when $|S_1| = K$, more than 50% iterations are saved for 50% of the instances. Some nodes are even pruned from the very first iteration of HomCont (100% saved iterations, see the black dash on the figure). A possible explanation for this behavior comes from our branching strategy, the maximum of amplitudes given in Equation (1.4): as we branch on variables which are likely to be non-zero at the optimal solution, the variables in S_1 are likely to belong to the optimal support. Therefore, lower bounds are expected to be more accurate as S_1 grows. Conversely, when S_1 is small, we are either at the top of the branch-and-bound tree, where few decisions have been made, or we are in the right side of the tree, filling S_0, which means that we exclude variables with high amplitudes. In both cases, the lower bounds are likely to be of poor quality (see the example on the right in Figure 3), and the dual iterates are therefore of no interest.

Figure 7 (bottom) similarly shows the results obtained with $\rho = 0.92$. Although the same trend can be observed (more iterations are saved for higher cardinality of S_1), the pruning performance is much weaker in this case, where only a few percent of iterations can be saved. Such a behavior can be explained by noting that the quality of the lower bounds obtained by the ℓ_1-norm relaxation of the ℓ_0 pseudo-norm worsens as the columns of matrix A become more correlated (see [45] and references therein). The higher ρ, the worse the relaxation. Here again, most explored nodes are likely to correspond to the example on Figure 3 right, where computing dual values cannot bring any improvement. Finally, we note that even in this difficult setting, pruning could still save all iterations at some nodes with support size close to the true cardinality.

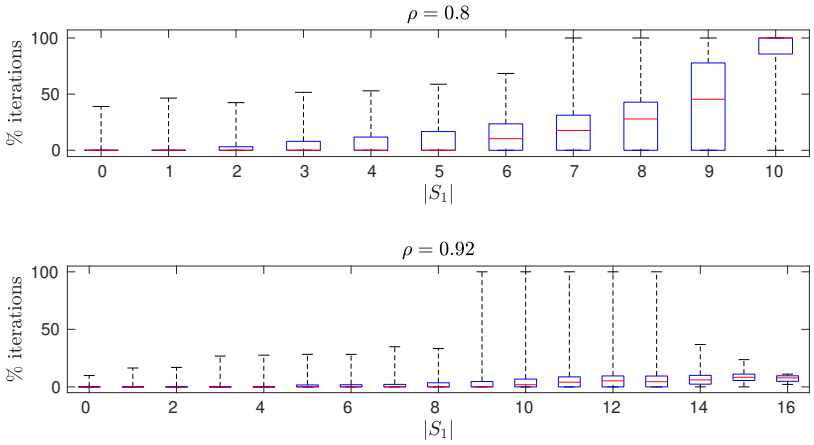

Figure 7.: Results for duality-based early pruning as a function of the number of non-zero variables at the corresponding node, for two correlations levels ρ in matrix A. The cardinality of the solution is $K = 9$. Each box shows the first and third quartiles on the fraction of saved iterations, the red dash indicates the median value, and black dashes indicate the extreme values.

5.4. Variable screening

The efficiency of the screening method described in Section 3 is now studied, using a similar methodology to that in former Section 5.3. At each node **N** of a given branch-and-bound execution, we consider the fraction of variables that are removed along the relaxation strategy. To this end:

- for a problem instance **I**, let $N_{\mathbf{I}}^{|S_1|}$ denote the number of nodes with a given cardinality of S_1 (the number of non-zero variables fixed at the node);
- let $V_{\mathbf{I}}^{|S_1|}$ denote the total number of variables that were screened in such nodes;
- we define the instance-wise average ratio of variables that could be screened depending on the cardinality of S_1: $\mathcal{F}_{\mathbf{I}|S_1|} := \frac{V_{\mathbf{I}}^{|S_1|}}{N_{\mathbf{I}}^{|S_1|}}$;
- we define the global average ratio of variables that could be screened depending on the cardinality of S_1 as $\mathcal{F}_{G|S_1|} := \frac{\sum_{\mathbf{I}} V_{\mathbf{I}}^{|S_1|}}{\sum_{\mathbf{I}} N_{\mathbf{I}}^{|S_1|}}$.

In our experiments, the screening method only set variables $x_i, i \in \bar{S}$, to 0.

Figure 8 (top) shows the screening performance on problem instances with $\rho = 0.8$ and $K = 9$. Contrary to the early pruning efficiency, variable screening appears to be globally more efficient for nodes with few variables in S_1 (that is, upper nodes in the decision tree and nodes on the right part of the tree). It generally achieves elimination of 10 to 15 % of variables, with a relatively high dispersion among the different problem instances. Figure 8 (bottom) shows that the screening performance severely drops for $\rho = 0.92$, where less than 2 % of the variables are removed. A similar interpretation to that in Section 5.3 can be given: when the correlation between the columns of A increases, it becomes harder to discriminate between the variables, and in particular to decide that some variables are irrelevant (that is, screened to 0).

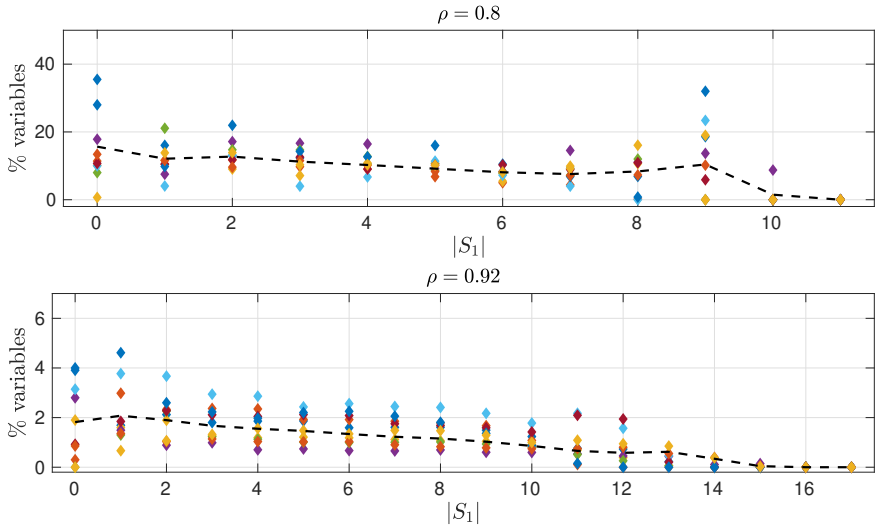

Figure 8.: Screening performance as a function of the number of non-zero variables at the corresponding nodes for two correlations levels in matrix A. The cardinality of the solution is $K = 9$. For each instance, one point represents the average ratio of screened variables $\mathcal{F}_{\mathbf{I}^{|S_1|}}$ over the nodes with cardinality of $|S_1|$. The dotted lined shows the average ratio of screened variables over all instances, $\mathcal{F}_G$.

5.5. Exploration strategies

In this section, we finally compare the performance of the different exploration strategies proposed in Section 4, which are implemented as data structures. The standard depth-first search and best-first search strategies are respectively implemented as a stack and a heap, which is sorted on the lower bound $\text{lb}_{\mathbf{N}}$ (denoted $\texttt{heap}^{\texttt{lb}}$). Problem-specific strategies are also implemented, namely $\texttt{heap}^{\texttt{LS}}$ and $\texttt{heap}^{\ell_1}$, corresponding to heaps sorted on the values of the lest-squares term and of the ℓ_1-norm term in $\text{lb}_{\mathbf{N}}$, respectively. Finally, mixed strategies are implemented, that switch from stack data structure (depth-first search) to heaps, with the three previously described sorting options. The switch is performed after a given number of nodes, say $\texttt{N}_{\texttt{switch}}$, have been explored, for which three values are considered: 20, 200 and 2000. In the following, such strategies are correspondingly named $\texttt{stack}^{\texttt{N}_{\texttt{switch}}}$ `+` $\texttt{heap}^{\texttt{lb}}$, $\texttt{stack}^{\texttt{N}_{\texttt{switch}}}$ `+` $\texttt{heap}^{\texttt{LS}}$, and $\texttt{stack}^{\texttt{N}_{\texttt{switch}}}$ `+` $\texttt{heap}^{\ell_1}$.

We first measure the total number of nodes that were created by each exploration strategy to solve the instances to optimality (in the limit of one hour), aggregating all instances with $K \in \{3, 5, 7, 9\}$, which amounts to 40 instances for each value of ρ.

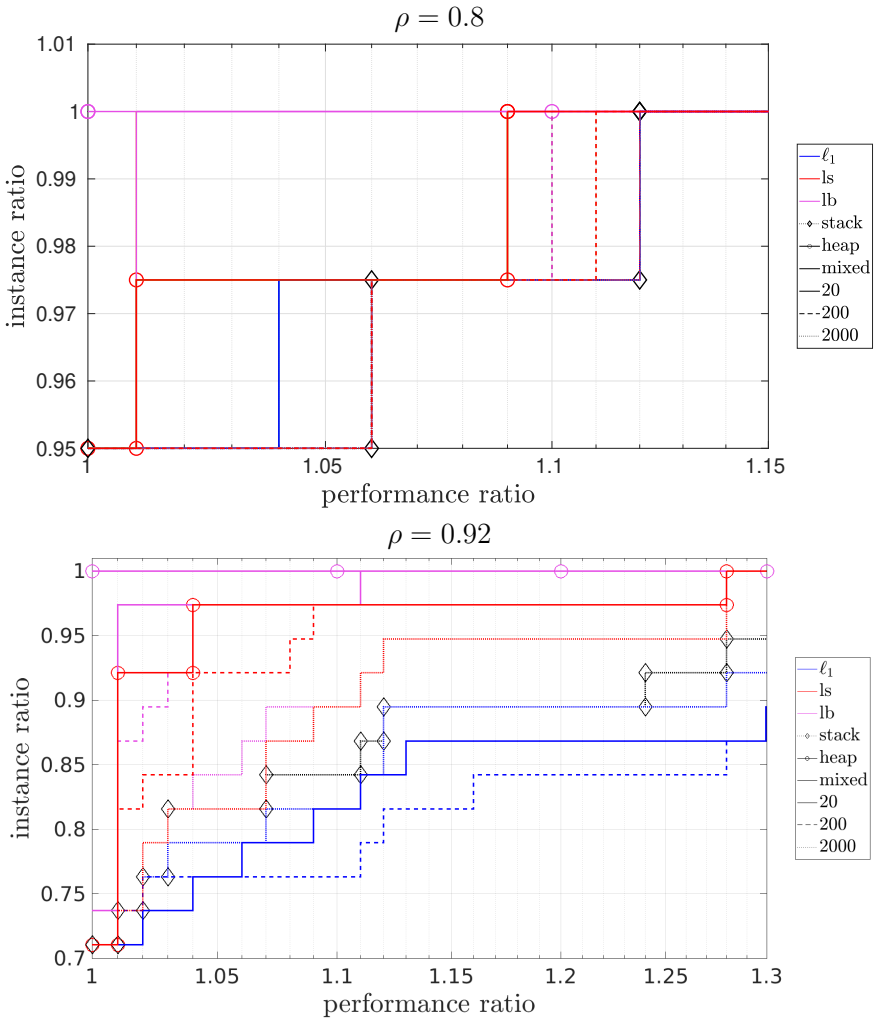

Figure 9.: Performance profiles comparing the number of created nodes for different exploration strategies, for moderately difficult problems ($\rho = 0.8$, top) and highly difficult problems ($\rho = 0.92$, bottom). Exploration is run until the entire search tree has been explored (computation time limited to one hour). The black dotted line with diamonds represents the `stack` implementation, and the full line with circles represents the `heap` implementation, sorted on the lower bound (magenta), its least-squares term (red), or its ℓ_1-norm term (blue). Mixed strategies $\texttt{stack}^{\texttt{N}_{\texttt{switch}}}$ `+ heap` use the same color code for the heap sorting, and are represented with full lines for $\texttt{N}_{\texttt{switch}} = 20$, dashed lines for $\texttt{N}_{\texttt{switch}} = 200$, and dotted lines for $\texttt{N}_{\texttt{switch}} = 2000$.

Figure 9 shows the corresponding performance profiles [15], for $\rho = 0.8$ (top) and $\rho = 0.92$ (bottom). In both cases, best-first search ($\texttt{heap}^{\texttt{lb}}$) achieves the best results, that is, it always explores less nodes than its competitors. Sorting the heap based on the least-squares part of the lower bound ($\texttt{heap}^{\texttt{LS}}$) is slightly less efficient, although still better than the `stack` implementation (which is yet the strategy implemented in the related works [5, 21, 23]). As could be expected, mixed strategies $\texttt{stack}^{\texttt{N}_{\texttt{switch}}}$ `+` $\texttt{heap}^{\texttt{lb}}$ achieve intermediate performance between $\texttt{heap}^{\texttt{lb}}$ and `stack`, with parameter $\texttt{N}_{\texttt{switch}}$ tuning the behavior closer to $\texttt{heap}^{\texttt{lb}}$ (which corresponds to $\texttt{N}_{\texttt{switch}} = 0$) or to `stack` ($\texttt{N}_{\texttt{switch}} = +\infty$). Similarly, $\texttt{stack}^{\texttt{N}_{\texttt{switch}}}$ `+` $\texttt{heap}^{\texttt{LS}}$ behaves between $\texttt{heap}^{\texttt{LS}}$ and `stack`. $\texttt{heap}^{\ell_1}$ is by far the worst strategy. Starting with some depth-first search iterations substantially improves its performance, but still achieves the worst results among all tested strategies. A possible explanation is that explor-

ing first the nodes with the lowest ℓ_1-norm value favors too much the sparsity of the solution, therefore exploring in priority the right part of the branch-and-bound tree where S_0 grows. That is, nodes are explored where high values in $x_{\text{lb}}^{\mathbf{N}}$ are put to zero, which is likely to generate poor solutions.

We also note that the impact of the exploration strategy highly depends on the problem difficulty: for $\rho = 0.8$, all strategies tend to align and give similar results, with the best strategy outperforming the second one for only 5% of the instances, while for $\rho = 0.92$ this is the case for 26% of the instances. Here again, this may be due to the quality of the ℓ_1 norm relaxation, which improves when ρ decreases: in that case, the lower bounds are better, and it becomes easier, for all strategies, to identify the nodes which should be pruned.

Finally, we study the performance of the different exploration strategies in their ability to *find* the optimal solution, without proving its optimality. This may be of interest in order to use the branch-and-bound procedure as a heuristic algorithm with limited computation time. Figure 10 shows similar behaviors both for $\rho = 0.8$ (top) and $\rho = 0.92$ (bottom), although results are more contrasted in the second case: the best-first search strategy $\texttt{heap}^{\texttt{lb}}$ is now the most inappropriate one for finding the optimum. Interestingly, depth-first search, yet acknowledged for its ability to quickly discover good solutions, is not the best option either, as it is outperformed by $\texttt{heap}^{\texttt{LS}}$.

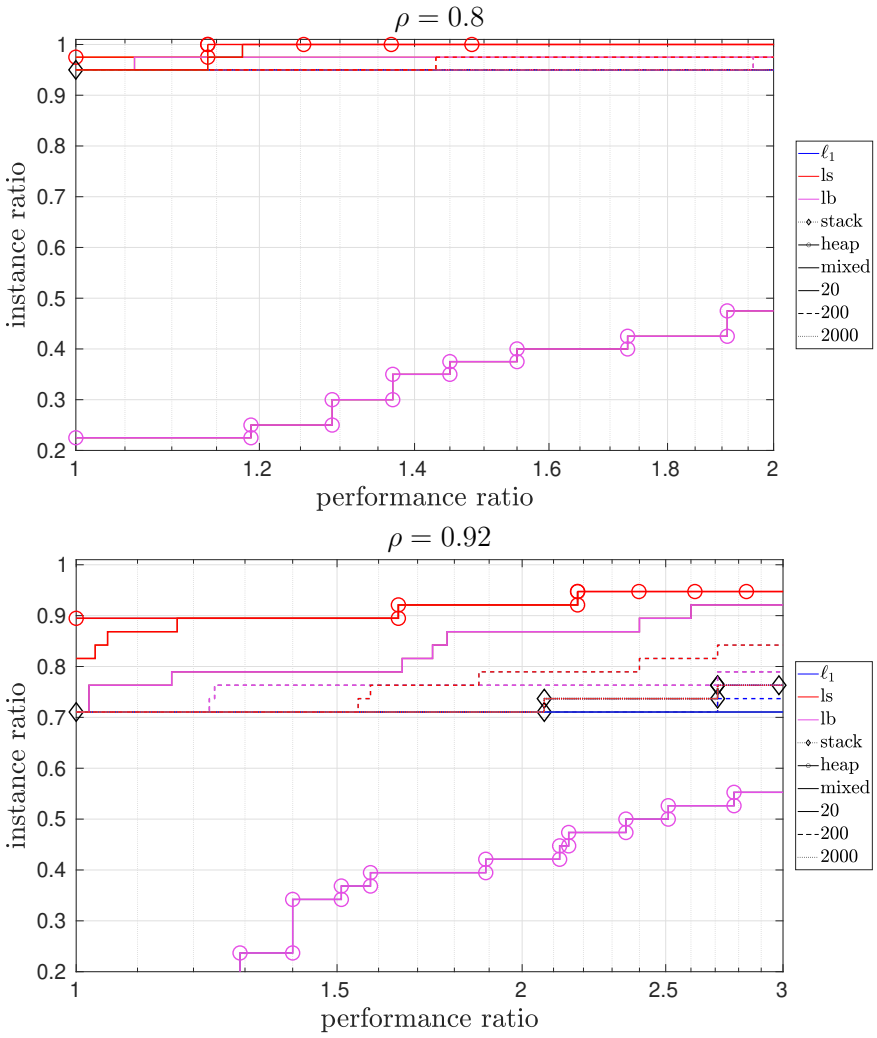

Figure 10.: Performance profiles comparing the number of created nodes required **to find the optimal solution without proving its optimality**, for different exploration strategies, for moderately difficult problems ($\rho = 0.8$, top) and highly difficult problems ($\rho = 0.92$, bottom). The black dotted line with diamonds represents the `stack` implementation, and the full line with circles represents the `heap` implementation, sorted on the lower bound (magenta), its least-squares term (red), or its ℓ_1-norm term (blue). Mixed strategies $\texttt{stack}^{\texttt{N}_{\texttt{switch}}}$ `+ heap` use the same color code for the heap sorting, and are represented with full lines for $\texttt{N}_{\texttt{switch}} = 20$, dashed lines for $\texttt{N}_{\texttt{switch}} = 200$, and dotted lines for $\texttt{N}_{\texttt{switch}} = 2000$.

6. Software installation and usage

We distribute our source code for reproducibility and further usage. Section 6.1 first describes the installation of a pre-packaged version for Ubuntu users, which handles some prescribed dependencies and solvers. Section 6.2 provides more advanced compiling options. Then, the overall object architecture is described in Section 6.3. Finally, Section 6.4 gives usage instructions with both executable and shell scripts.

The code, called `Mimosa` for *Mixed Integer programming Methods for Optimization of Sparse Approximation criteria*, is a `C++` software licensed under the GNU Lesser

General Public License v3 terms[2]. Instructions are given for UNIX systems.

6.1. Ubuntu package installation

For Ubuntu users, a package named `Mimosa_2.0.0_amd64.deb` is already available at `https://box.ec-nantes.fr/index.php/s/Ki7xb7TxKHk4zW5`.

Installing the code along with its dependencies is as simple as:

```
$ sudo apt install <path/to/Mimosa_2.0.0_amd64.deb>
```

6.2. Compiling

To compile the code, you will need the Dlib (`http://dlib.net/`) and Armadillo (`http://arma.sourceforge.net/`) libraries installed.

First retrieve the source:

```
$ git clone -b oms_europt21
    https://gitlab.univ-nantes.fr/samain-g/mimosa-solver
```

Then, in the project folder, create a build directory:

```
$ mkdir cmake_build && cd cmake_build
```

After this comes the `cmake` invocation:

```
$ cmake -DUSE_CPLEX=<0|1> -DCPLEX_ROOT_DIR=<path>
    -DDLIB_ROOT_DIR=<path> -DArmadillo_ROOT_DIR=<path>
    -DLOG_PER_NODE_STAT=<0|1>  -DCMAKE_INSTALL_PREFIX=<path>  ..
```

The last `..` is mandatory. These options set some CMake variables `VAR` with the syntax `-DVAR=value`.

Set `USE_CPLEX` to 1 in order to enable the CPLEX solver support for comparison purposes (see Section 6.4). In that case, `CPLEX_ROOT_DIR` specifies the path to CPLEX. If `USE_CPLEX = 0`, CPLEX support is disabled, which makes `CPLEX_ROOT_DIR` unused. Leaving `USE_CPLEX` unspecified triggers a "best-effort" build. CMake will try to find CPLEX libraries, and otherwise it will fall back to a no-CPLEX build.

Set `DLIB_ROOT_DIR` (respectively `Armadillo_ROOT_DIR`) to the correspond paths for Dlib and Armadillo. If they are installed to a standard location in your system, specifying such values is not required.

The `LOG_PER_NODE_STAT` variable controls whether statistics on individual node should be printed or not. This is switched off by default because of the huge amount of data generated. If you wish to reproduce the results in Section 5.3, you need to set `LOG_PER_NODE_STAT` to 1.

Finally, the option `CMAKE_INSTALL_PREFIX` is used to specify the installation path of `Mimosa`, otherwise `Mimosa` is installed in the default path of your system (`/usr/local/bin`).

After this step, you should have a `Makefile` in the current folder. It can now be compiled with:

```
$ make
```

This will create the `Mimosa` binary. Then:

```
$ sudo make install
```

will put the `Mimosa` binary and its companion shell scripts in `$CMAKE_INSTALL_PREFIX/bin`, by default in `/usr/local/bin`.

6.3. Architecture

`Mimosa` uses an object-oriented paradigm favoring abstract interfaces, and was designed to ease the integration of new variants for classic components of a branch-and-bound algorithm.

[2]Terms available at `https://opensource.org/licenses/LGPL-3.0`

The file `interfaces.hpp` gives the global overview of these interfaces. The branch-and-bound main loop is defined in `Optimizer.cpp`, see Algorithm 1.

This loop manipulates nodes. The nodes interface is declared in `interfaces.hpp`, the implementations are declared in `NodeImplementation.hpp` and defined in `NodeImplementation.cpp`. The node definition defines which quantities are stored in a node, and which ones are computed on-the-fly.

The implementations for dividing a node are declared in `Split.hpp` and defined in `Split.cpp`. These two files contain the branching strategies available in `Mimosa`, see Section 1.4.

For the implementations computing the upper bounds (see Section 1.3), `UB.hpp` and `UB.cpp` gives the necessary declarations and definitions.

For the computations of lower bounds (see Section 2), `LB.hpp` and `LB.cpp` are the main entry points. The organization of the source files is described in `HACKING.md`.

The data structures coding the different exploration strategies (see Section 4.2) are declared in `node_container.hpp` and defined in `node_container.cpp`.

6.4. Usage

Upon compilation, one program `Mimosa` is generated, along with some shell scripts to provide handy shortcuts. The `Mimosa` executable comes with all the options being mandatory, while the shell scripts variants use sensible predefined settings.

Executable code. `Mimosa` takes instances from standard input. One instance is a folder containing the following files:

- `y.dat`: the data measurements y
- `A.dat`: the problem matrix A
- `mu.dat`: the ℓ_0-norm penalty parameter μ defined in $(\mathcal{P}_{2+0})$.

It provides some kind of activity log to standard output, including the solution $\hat{x}$. A CSV file is also generated containing per-instance metrics used for the experiments in Section 5.4 and 5.5.

`Mimosa` also takes some command line arguments, in this order:

```
$ Mimosa l2pl0 <solver> <dualmod> <scrmod> <explo> <exploarg>
```

<solver> The solver to use, which is one element of:
- `full_cplex`,
- `bb_cplex`,
- `bb_homotopy_warm`,
- `bb_homotopy_cold`,
- `bb_activeset_warm`,
- `bb_activeset_cold`,
- `bb_icd_warm`,
- `bb_icd_cold`

among which:
- `full_cplex` stands for a direct call to CPLEX solving a MIP formulation of the problem (see [5] for more details). This is provided as a wrapper around CPLEX if you wish to benchmark our code versus CPLEX.
- `bb_cplex` stands for the CPLEX quadratic programming solver used to compute the lower bounds, and the lower bounds only. As for all solvers with names `bb_$$` , the `Mimosa` Branch-and-Bound framework described in Section 1 is used, the algorithm to compute the lower bounds being specified in the `$$` part.
- `bb_homotopy_$$`, `bb_activeset_$$`, `bb_icd_$$` stand for the `Mimosa` Branch-and-Bound framework, with one of the three algorithms described in Section 2.2 used to compute the lower bounds. The `_warm` suffix stands for warm start, where initial guesses are taken from the output of the parent node evaluation. The `_cold` suffix stands for no warm start. Experiments in this paper were done with `bb_homotopy_cold`.

<dualmod> The `dualmod` parameter is the period at which the dual objective function is evaluated for early pruning within the iterations computing the lower bounds (by the previously described solvers). Setting it to 0 disables dual computations. Experiments in Sections 5.3 and 5.4 were performed with `dualmod` = 1 (that is, at each iteration), while experiments in Section 5.5 were performed with `dualmod` = 0.

<scrmod> The `scrmod` parameter is the period at which screening tests are implemented. Setting `scrmod` = 1 computes the screening tests every time the dual objective function is evaluated, `scrmod` = 2 computes them every two such evaluations, *etc.* `scrmod` = 0 disables screening. Experiments in Section 5.4 were performed with `scrmod` = 1.

<explo> The exploration strategy to use, among:

- `stack`: depth first search,
- `heap_on_lb`: best first search,
- `heap_on_l1`: ℓ_1 first search,
- `heap_on_ls`: least-square first search,
- `stack_then_heap_on_lb_iteration_threshold`: mixed strategy, depth first search then best first search after $\texttt{N}_{\texttt{switch}}$ iterations (see below),
- `stack_then_heap_on_l1_iteration_threshold`: mixed strategy, depth first search then ℓ_1 first search after $\texttt{N}_{\texttt{switch}}$ iterations,
- `stack_then_heap_on_ls_iteration_threshold`: mixed strategy, depth first search then least-square first search after $\texttt{N}_{\texttt{switch}}$ iterations,
- `stack_then_lds_iteration_threshold`: mixed strategy depth first search then limited discrepancy search after $\texttt{N}_{\texttt{switch}}$ iterations.

<exploarg> The iteration threshold $\texttt{N}_{\texttt{switch}}$ for mixed strategies (set to 0 for non-mixed strategies), see Section 4.2.

An invocation example is then:

```
$ echo one_instance_here | Mimosa l2pl0 bb_homotopy_cold 1 2
    stack_then_heap_on_lb 20 my_results.csv
```

Shell scripts. The shell scripts give shorter ways to call `Mimosa`. All scripts use these predefined values:

- `bb_homotopy_cold` solver,
- `dualmod` = 0,
- `scrmod` = 0,
- `heap_on_lb` exploration strategy,
- $\texttt{N}_{\texttt{switch}} = 0$,

that can be changed by editing the script.

Three scripts are provided, with different ways to take instances.

- `Mimosa_l2pl0_1inst` takes one instance as an argument (and pipes it to `Mimosa`), as in the following example:

  ```
  $ Mimosa_l2pl0_1inst <path/to/one/instance/folder>
  ```

- `Mimosa_l2pl0_listinst` takes a file containing a list of instances as an argument. With a file `inst_list.txt` listing instances like this:

  ```
  path/to/one/instance/folder
  path/to/another/instance/folder
  ```

  an invocation example is:

  ```
  $ Mimosa_l2pl0_listinst inst_list.txt
  ```

- Finally, `Mimosa_l2pl0_pipeinst` takes a list of instances on standard input, as `Mimosa` does. An invocation using the previous list would be:

  ```
  $ cat inst_list.txt | Mimosa_l2pl0_pipeinst
  ```

  A dynamically generated list can also be used with this third script, for example:

  ```
  $ find /path/to/directory/containing/instances | grep K9 | sort |
      head -n 20 | Mimosa_l2pl0_pipeinst
  ```

Discussion

In this paper, three acceleration techniques for branch-and-bound algorithms dedicated to linear sparse regression have been explored.

The first two ones relate to the computation of lower bounds involved at each node of the branch-and-bound tree, which are obtained by solving convex, non-smooth, box-constrained, optimization problems mixing least-squares and ℓ_1-norm terms. Early pruning exploits convex duality in order to reject suboptimal nodes thanks to the computation of dual bounds. Screening methods use dual iterates to prove the nullity of some variables at the optimal point, therefore discarding them from the remaining computation of the lower bound. Several dedicated algorithms were studied in such a context, among which the homotopy continuation method appeared to be the most efficient one for increasing the dual objective function. A detailed performance analysis revealed that early pruning and screening are more efficient for settings where the correlation between the columns of the problem matrix is lower, and that their acceleration power depends on the considered node inside the tree: while screening performs better for nodes with the lowest number of variables included in the support, early pruning has more impact for nodes including a higher number of active variables, close to the true sparsity level of the solution.

The third acceleration technique concerns the exploration strategy, where alternatives to standard methods were investigated, and implemented as specific data structures for node scheduling. Contrary to early pruning and screening, the choice of the exploration strategy was shown to have more impact on the more difficult problems. In such cases, best-first search offers the lowest number of explored nodes if the branch-and-bound procedure is run up to optimality. However, exploring in priority the nodes with the lowest least-squares term was found to be more efficient in order to quickly find good solutions. It is therefore a choice of interest if one accepts to trade the loss of the optimality proof against a subsequent reduction of the computation time. To the best of our knowledge, such choices were never discussed in the related literature, where depth-first search always seems to be chosen.

We provide a dedicated `C++` implementation called `Mimosa`, where the different proposals discussed in the paper can be applied or not. Outside of `Mimosa`, other branch-and-bound solvers for sparse optimization [21, 23] could also benefit from these accelerations.

These works pave the way for future ones. First, the efficiency of the proposed techniques was shown to strongly vary with the problem structure, the type of node in the search tree, the depth of the node, etc. In order to optimize the resulting computation time, such methods should only be applied when they are potentially useful, that is, when the potential acceleration brought compensates for the extra computation cost. The resulting choices may highly depend on the problem, and strategies that would learn such decisions for a given problem class are of first interest.

More aggressive screening types for relaxed problems are also worth being studied. Strong screening strategies [26, 40] increase the number of variables that are fixed, and hence reduce the computation time, but with the risk of making the solution found sub-optimal. In that case, the optimal value of the reduced relaxation problem may not be a lower bound of the considered original ℓ_0 subproblem. Fortunately, such cases could be detected by a non-zero duality gap. A valid strategy would then consist in first applying strong screening and find a good, potentially sub-optimal solution and, if necessary, include again the previously fixed variables for the last iterations of the relaxation algorithm.

Finally, the discrepancy between the performance of best-first, least-squares first and most notably ℓ_1-norm first exploration strategies may be a hint that using the ℓ_1 norm to relax the ℓ_0 cardinality term is too loose. This motivates the use of non convex relaxation to improve the quality of lower bounds, *e.g.,* based on the many related works in the sparse approximation literature [38]. This would certainly raise new methodological issues in order to guarantee the validity of the obtained bounds and, subsequently, of the resulting branch-and-band procedure.

References

[1] S. Alliney and S. A. Ruzinsky. An algorithm for the minimization of mixed ℓ_1 and ℓ_2 norms with application to bayesian estimation. *IEEE Transactions on Signal Processing*, 1994.

[2] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. *Optimization with Sparsity-Inducing Penalties*. 2011.

[3] H. Bauschke and P. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Space*. 2011.

[4] R. Ben Mhenni, S. Bourguignon, M. Mongeau, J. Ninin, and H. Carfantan. Sparse Branch and Bound for Exact Optimization of ℓ_0-Norm Penalized Least Squares. In *ICASSP 2020, IEEE International Conference on Acoustics, Speech and Signal Processing*, Barcelona, Spain, 2020.

[5] R. Ben Mhenni, S. Bourguignon, and J. Ninin. Global optimization for sparse solution of least squares problems. *Optimization Methods and Software*, 2021.

[6] D. Bertsimas, A. King, and R. Mazumder. Best subset selection via a modern optimization lens. *The Annals of Statistics*, 2016.

[7] D. Bertsimas and R. Shioda. Algorithm for cardinality-constrained quadratic optimization. *Computational Optimization and Applications*, 2009.

[8] D. Bienstock. Computational study of a family of mixed-integer quadratic programming problems. In *5th International IPCO Conference*, 1995.

[9] A. Bonnefoy, V. Emiya, L. Ralaivola, and R. Gribonval. A dynamic screening principle for the lasso. In *European Signal Processing Conference EUSIPCO*, 2014.

[10] S. Bourguignon, J. Ninin, H. Carfantan, and M. Mongeau. Exact sparse approximation problems via mixed-integer programming: formulations and computational performance. *IEEE Transactions on Signal Processing*, 2016.

[11] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 2011.

[12] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Review*, 2001.

[13] X. Cui, X. Zheng, S. S. Zhu, and X. Sun. Convex relaxations and MIQCQP reformulations for a class of cardinality-constrained portfolio selection problems. *Journal of Global Optimization*, 2013.

[14] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 2004.

[15] E. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 2002.

[16] J. Eckstein. *Splitting methods for monotone operators with applications to parallel optimization*. PhD thesis, Massachusetts Institute of Technology, 1989.

[17] B. Efron, T. J. Hastie, I. M. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 2004.

[18] L. El Ghaoui, V. Viallon, and T. Rabbani. Safe Feature Elimination for the LASSO and Sparse Supervised Learning Problems. Technical report, EECS Department, University of California, Berkeley, 2010.

[19] O. Fercoq, A. Gramfort, and J. Salmon. Mind the duality gap: safer rules for the LASSO. In *Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, 2015.

[20] J. H. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 2010.

[21] T. Guyard, C. Herzet, and C. Elvira. Node-screening tests for ℓ_0-penalized least-squares problem with supplementary material, 2021. URL: `https://arxiv.org/abs/2110.07308`.

[22] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. New York, NY, USA, 2001.

[23] H. Hazimeh, R. Mazumder, and A. Saab. Sparse regression at scale: branch-and-bound rooted in first-order optimization. *Mathematical Programming*, 2021.

[24] J. J. Kormylo and J. M. Mendel. Maximum likelihood detection and estimation of bernoulli - gaussian processes. *IEEE Transactions on Information Theory*, 1982.

[25] H. Lee, A. Battle, R. Raina, and A. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems*, 2006.

[26] M. Massias. From safe screening rules to working sets for faster lasso-type solvers. In *10th NIPS Workshop on Optimization for Machine Learning*, 2017.

[27] J.-J. Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences, Paris*, 1962.

[28] P. Moulin and J. Liu. Analysis of multiresolution image denoising schemes using generalized gaussian and complexity priors. *IEEE Transactions on Information Theory*, 1999.

[29] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 1995.

[30] E. Ndiaye, O. Fercoq, A. Gramfort, and J. Salmon. Gap safe screening rules for sparsity enforcing penalties. *Journal of Machine Learning Research*, 2017.

[31] G. Nemhauser and L. Wolsey. *General algorithms*. In *Integer and Combinatorial Optimization*. 1988. Chapter II.4.

[32] J. Nocedal and S. J. Wright. *Numerical Optimization*. New York, 2nd edition, 2006.

[33] M. Osborne, B. Presnell, and B. Turlach. A new approach to variable selection in least squares problems. *IMA Journal of Numerical Analysis*, 2000.

[34] A. Raj, J. Olbrich, B. Gärtner, B. Schölkopf, and M. Jaggi. Screening rules for convex problems. In *9th NIPS Workshop on Optimization for Machine Learning*, 2016.

[35] R. T. Rockafellar. *Convex Analysis*. Princeton, 1970.

[36] D. X. Shaw, S. Liu, and L. Kopman. Lagrangian relaxation procedure for cardinality-constrained portfolio optimization. *Optimization Methods Software*, 2008.

[37] S. Solntsev, J. Nocedal, and R. Byrd. An algorithm for quadratic ℓ_1-regularized optimization with a flexible active-set strategy. *Optimization Methods and Software*, 2015.

[38] E. Soubies, L. Blanc-Féraud, and G. Aubert. A unified view of exact continuous penalties for ℓ_2-ℓ_0 minimization. *SIAM Journal on Optimization*, 2017.

[39] C. Soussen, J. Idier, D. Brie, and J. Duan. From Bernoulli-Gaussian deconvolution to sparse signal restoration. *IEEE Transactions on Signal Processing*, 2011.

[40] R. Tibshirani, J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. Tibshirani. Strong rules for discarding predictors in lasso-type problems. *Journal of the Royal Statistical Society Series B (Statistical Methodology)*, 2010.

[41] A. M. Tillmann, D. Bienstock, A. Lodi, and A. Schwartz. Cardinality Minimization, Constraints, and Regularization: A Survey, 2021. URL: `https://arxiv.org/abs/2106.09606`.

[42] J. A. Tropp. Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 2004.

[43] J. A. Tropp and S. J. Wright. Computational methods for sparse solution of linear inverse problems. *Proceedings of the IEEE*, 2010.

[44] P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 2001.

[45] M. J. Wainwright. Sharp thresholds for high-dimensional and noisy sparsity recovery using ℓ_1-constrained quadratic programming (lasso). *IEEE Transactions on Information Theory*, 2009.

[46] Z. Xiang, H. Xu, and P. J. Ramadge. Learning sparse representations of high dimensional data on large scale dictionaries. In *Advances in Neural Information Processing Systems*, 2011.