Node-screening tests for the ℓ_0-penalized least-squares problem

Théo Guyard*, Cédric Herzet†, Clément Elvira‡

*Applied Mathematics Department, INSA Rennes, France | †SIMSMART team, INRIA Rennes-Bretagne Atlantique, France | ‡SCEE team, CentraleSupelec Rennes, France

Objectives

Reduce the optimization time of a Branch-and-Bound (BnB) algorithm tailored to the ℓ_0-penalized least-squares problem by detecting nodes of the decision tree that cannot yield a global optimizer.

Introduction

- **Sparse decomposition:** Find an approximation of a vector $\mathbf{y}$ as the *linear combination* of a *few* columns of a matrix $\mathbf{A}$.

ℓ_0-penalized least-squares

$\|\mathbf{y} - \mathbf{Ax}\|_2$ — Ensures data-fidelity

$\|\mathbf{x}\|_0$ — Promotes sparsity

$$(\mathcal{P}) \quad \begin{cases} \min & \frac{1}{2}\|\mathbf{y} - \mathbf{Ax}\|_2^2 + \lambda\|\mathbf{x}\|_0 \\ \text{st.} & \|\mathbf{x}\|_\infty \leq M \end{cases}$$

- **Characteristics:**
  - NP-hard problem due to the ℓ_0-norm
  - Mixed-Integer Program reformulation
  - Addressable with BnB algorithms
  - "Big-M" to construct bounded relaxations
- **Recent advances by Atamturk *et. al.*:**
  - Screening tests to detect zeros and non-zeros in the optimizers of the problem
  - Dimensionality reduction in pre-processing
- **Our contributions:**
  - Node-screening tests to detect zeros and non-zeros in the optimizers of any node problem
  - Dimensionality reduction at any step of the optimization process
  - Larger improvement in the solving time

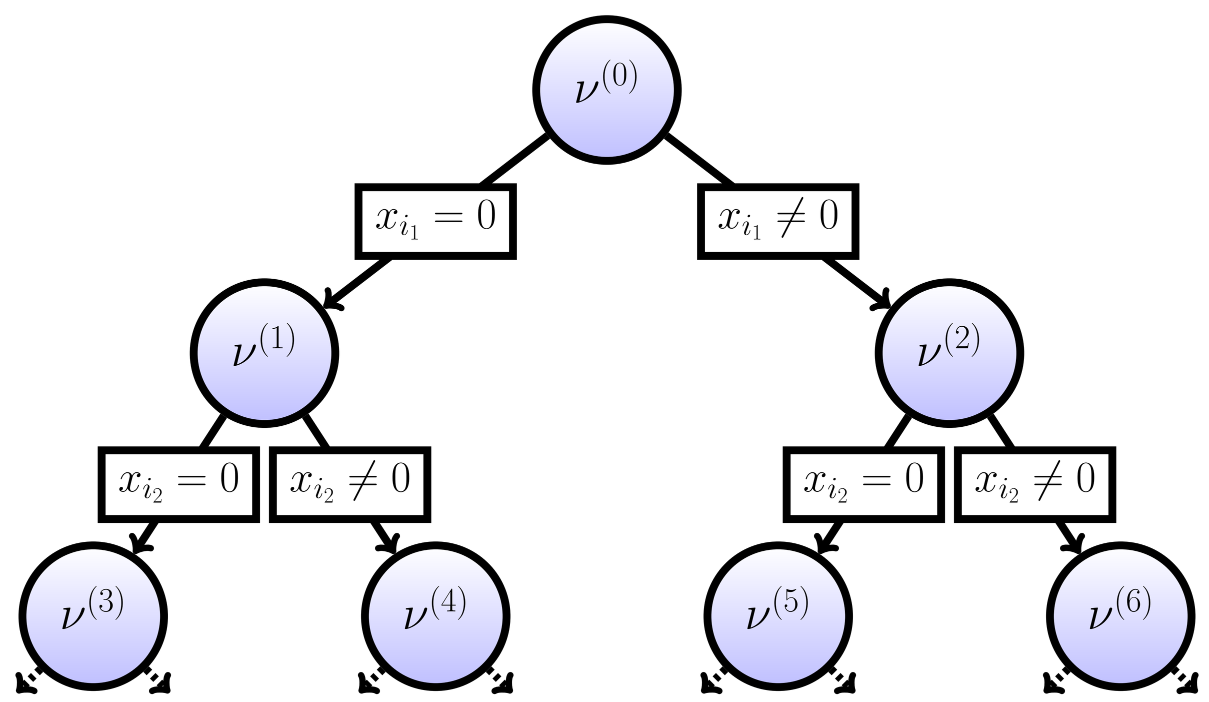
BnB algorithms

- **Generic procedure:**

Algorithm 1: BnB algorithm

Create a root node ν^0 and initialize $\mathcal{Q} = \{\nu^0\}$
while $\mathcal{Q} \neq \emptyset$ **do**
  1) Select a new node ν in $\mathcal{Q}$
  2) Find a lower and an upper bound on the node problem objective value
  3) Update the best upper bound known
  if *node LB* $\leq$ *best UB* **then**
    4) Create child nodes of ν by fixing new constraints and push them in $\mathcal{Q}$
  end
  5) Remove ν from $\mathcal{Q}$
end
return *any node yielding the best UB*

- **Efficiency of the algorithm:**
  - Tightness of the bounds computed
  - Ability to process nodes quickly
  - Number of nodes processed
- **Particularized to our problem:**
  - Binary tree with decision about the nullity of an entry at each node
  - Each node is uniquely defined by the set of entries forced to zero and to non-zero



Node-screening tests

- **Underlying idea:**
  - i) For many nodes, it is obvious that they cannot yield a global solution (too large penalty, *etc.*..)
  - ii) BnB is an exact method so we have to prove that they can be pruned safely
  - iii) Standard pruning methodology : solve a convex relaxation. This is expensive !
  - iv) Why not trying weaker but cheaper methods ?

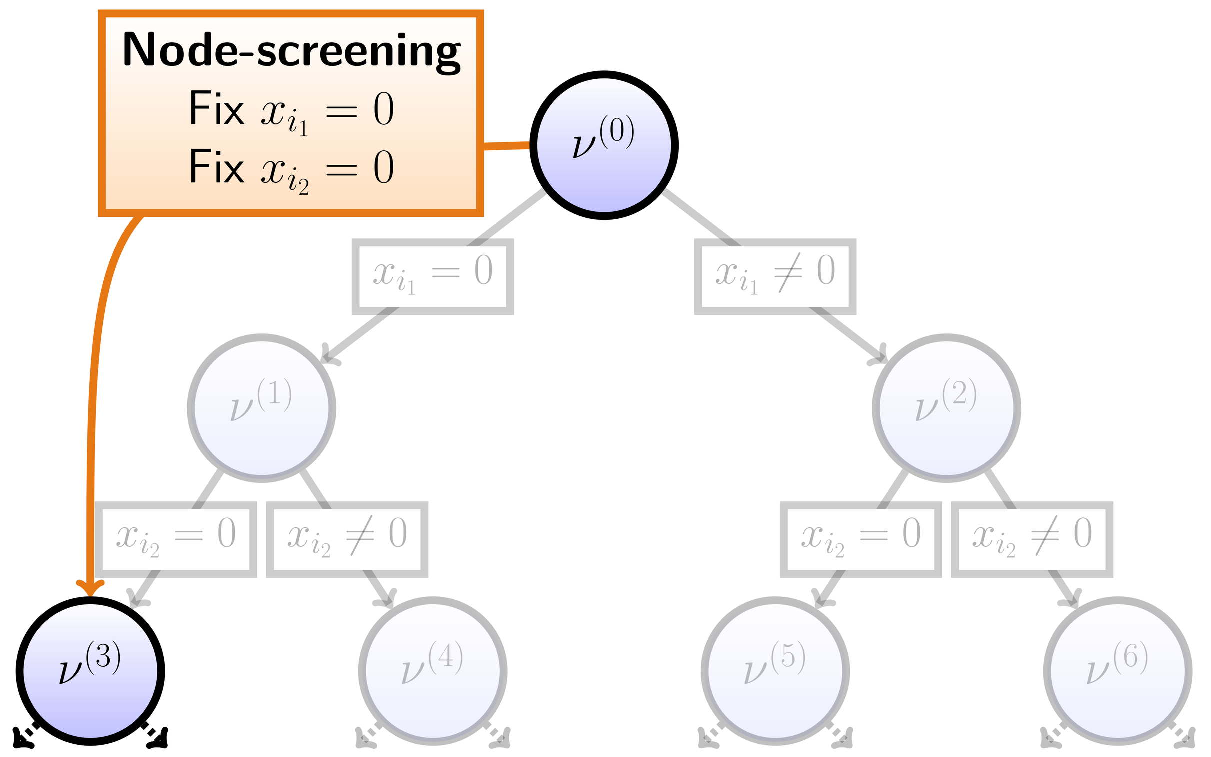
"Non-obvious" nodes → Standard pruning

"Obvious" nodes → Node-screening

- **Differences with the classical method:**
  - Test any unfixed index, no branching rule
  - No computational overhead to prune nodes
  - Can fix multiple variables simultaneously

Main ingredients

- At each node, a convex relaxation is solved
- We construct the dual of this relaxation
- Duals are very similar between two nodes
- At node ν, we use this relation to hopefully prune the child node of ν constructed by branching on a given index



Numerical results

- **Data generation:**
  1) Set $(m, n) = (500, 1000)$
  2) Generate a random matrix $\mathbf{A} \in \mathbf{R}^{m \times n}$ with a correlation ρ between the columns
  3) Generate a k-sparse vector $\mathbf{x}^\dagger \in \mathbf{R}^n$
  4) Set $\mathbf{y} = \mathbf{Ax}^\dagger$ + noise with 10dB SNR
  5) Calibrate λ and M statistically to recover $\mathbf{x}^\dagger$
- **Concurrent methods:**
  - Direct method using CPLEX
  - Tailored BnB algorithm from Mhenni *et. al.*
  - Tailored BnB algorithm with node-screening

		Direct			BnB			BnB + node-scr.		
ρ	k	N	T	F	N	T	F	N	T	F
Low	5	0.1	25.9	0	0.1	1.5	0	**0.05**	**0.7**	**0**
	7	0.3	60.8	0	0.2	5.1	0	**0.1**	**3.0**	**0**
	9	0.8	102.6	10	0.5	15.6	0	**0.4**	**9.8**	**0**
High	5	1.4	10.2	0	1.0	6.4	0	**0.7**	**4.2**	**0**
	7	17.6	106.5	0	10.5	79.3	0	**7.9**	**52.2**	**0**
	9	80.7	353.4	50	47.8	346.4	48	**41.2**	**267.0**	**40**

Table: (N) : Nodes explored $\times 10^{-3}$. (T) : Solving time in sec. (F) : Number of instances not solved within 10^3 sec.

- **Observations:**
  - BnB + node-scr. has the best performances
  - Time gain is larger than node gain
  - Double kiss-cool effect : the bounding step is performed all the faster as many variables are fixed

Take home message

In a BnB tailored to a sparse problem, there is not always need to perform heavy computations to prune nodes. Many nodes can be easily pruned by performing simple and cheap tests like the node-screening one.