Paper ID : 2482

# Node-screening tests for the $\ell_0$-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) solving the $\ell_0$-penalized least-squares problem by detecting nodes of the search tree that cannot yield a global optimizer.

## Introduction

- Sparse decomposition aims at finding some approximation of a vector $\mathbf{y}$ as the linear combination of a few columns of a dictionary $\mathbf{A}$. The $\ell_0$-penalized least-squares problem is one way to achieve this :

$\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2$ Ensures data fidelity

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

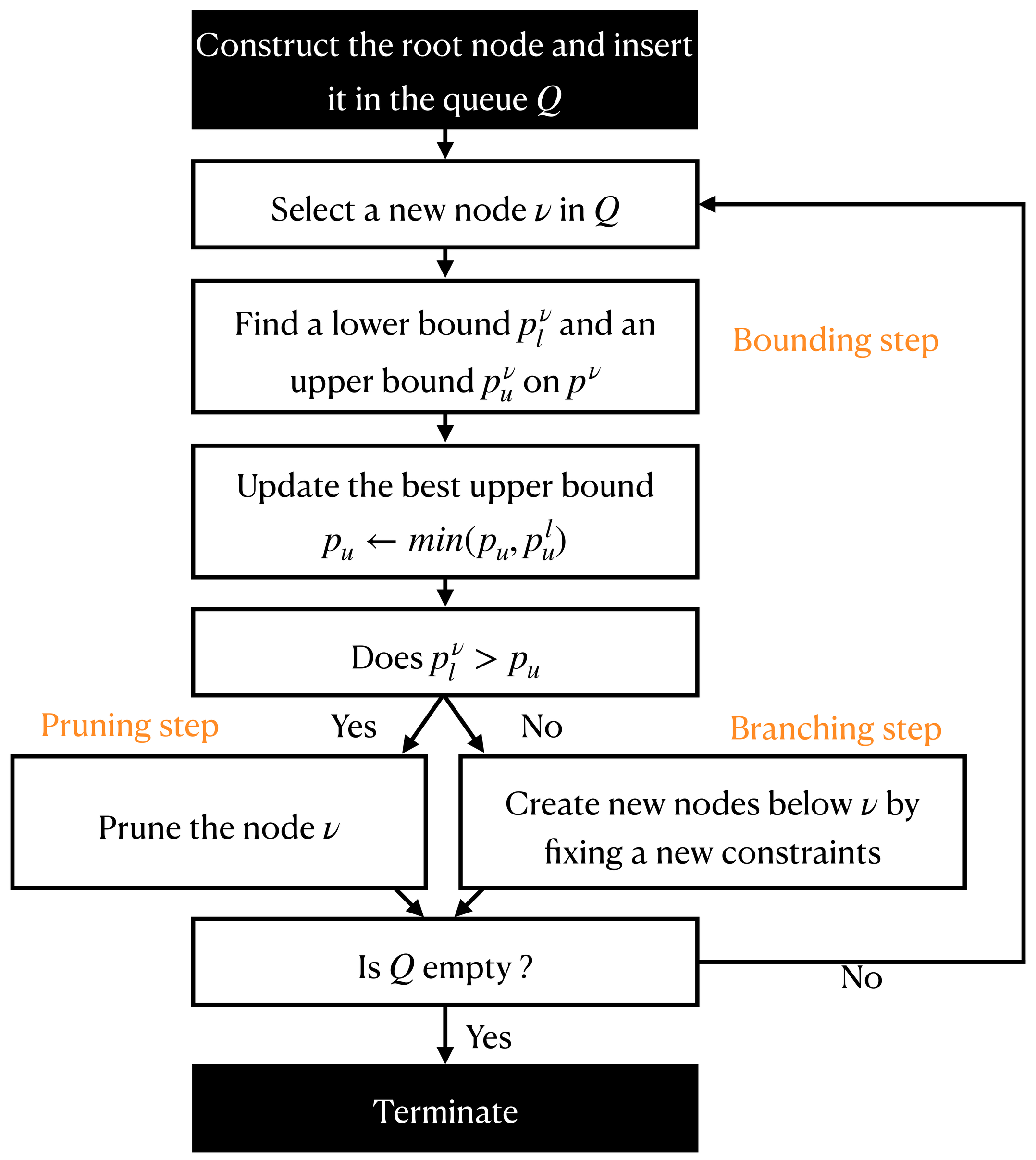
$$p^\star = \begin{cases} \min \frac{1}{2}\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_0 \\ \text{st } \|\mathbf{x}\|_\infty \leq M \end{cases}$$

$\ell_0$-penalized least-squares problem

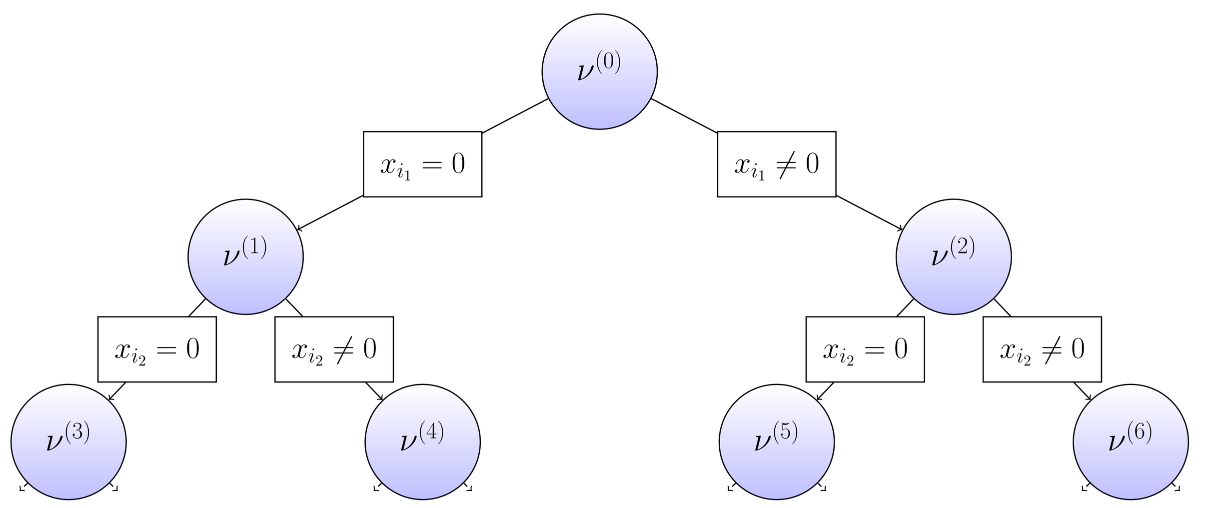
- Highlights :
  - NP-hard problem
  - Mixed-Integer Program (MIP) reformulation thanks to the Big-M constraint
  - Addressable with BnB algorithms
- Recent advances by Atamturk *et. al.* :
  - *Screening tests* to detect zero and non-zero element of the solution
  - This allows a dimensionality reduction in *pre-processing*
- Our contributions :
  - *Node-screening tests* to detect *combinations* of zero and non-zero elements that cannot yield an optimal solution.
  - This allows a dimensionality reduction at *any step of the optimization process*.

## BnB procedures

- Generic procedure :




- Particularized to our problem :
  - Binary decision tree where new decision concerning the *nullity of a coefficient* is taken at each node
  - Each node is defined by $(\mathcal{S}_0, \mathcal{S}_1, \bar{\mathcal{S}})$ where $\mathcal{S}_0$ and $\mathcal{S}_1$ contain the indices which are forced to be *zero* and *non-zero* and where $\bar{\mathcal{S}}$ gathers all the *unfixed* indices.
- What impacts the efficiency of the algorithm :
  - Number of nodes processed
  - Ability to process nodes quickly




## Node-screening tests

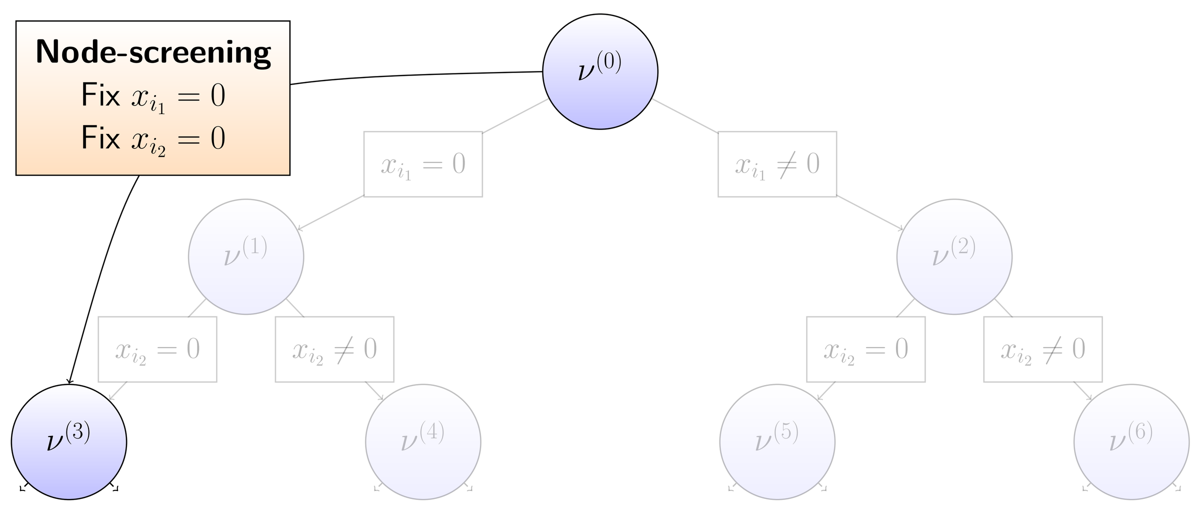
- Underlying idea :
  i) We are at a given node $\nu$
  ii) We select an unfixed index $i$
  iii) We compute a lower bound $d_l^{\nu\cup\{i\}}$ on $p_l^{\nu\cup\{i\}}$ at a *very low computational cost*
  iv) If $d_l^{\nu\cup\{i\}} > p_u$, then the node $\nu\cup\{i\}$ needs not be explored

But ... It is like the pruning step isn't it ?

- Differences with the pruning step :
  - We can test *any* unfixed index
  - Node-screening tests *do not bring any computational overhead*
  - They allow to fix *multiple variables simultaneously*

### Main ingredients

- In the BnB, $p_l^\nu$ is obtained by solving the *convex relaxation* of the initial problem with the current node constraints
- We construct the *dual* of this relaxation
- Duals have a very similar expression between consecutive nodes
- This allows compute the lower bound $d_l^{\nu\cup\{i\}}$ with no additional cost for several $i$
- And therefore to prune *multiple nodes* simultaneously !




## Numerical results

- Data generation :
  1) Generate a random matrix $\mathbf{A}\in\mathbf{R}^{500\times 1000}$ with a correlation $\rho$ between the columns
  2) Generate a $k$-sparse vector $\mathbf{x}^\dagger\in\mathbf{R}^n$
  3) Set $\mathbf{y}=\mathbf{A}\mathbf{x}^\dagger$ + noise with 10dB SNR
  4) Calibrate $\lambda$ 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+scr` | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\rho$ | $k$ | N | T | F | N | T | F | N | T | F |
| Low | 5 | 96 | 25.9 | 0 | 70 | 1.5 | 0 | **56** | **0.7** | **0** |
| | 7 | 292 | 60.8 | 0 | 180 | 5.1 | 0 | **152** | **3.0** | **0** |
| | 9 | 781 | 102.6 | 10 | 483 | 15.6 | 0 | **412** | **9.8** | **0** |
| High | 5 | 1,424 | 10.2 | 0 | 965 | 6.4 | 0 | **725** | **4.2** | **0** |
| | 7 | 17,647 | 106.5 | 0 | 10,461 | 79.3 | 0 | **7,881** | **52.2** | **0** |
| | 9 | 80,694 | 353.4 | 50 | 47,828 | 346.4 | 48 | **41,166** | **267.0** | **40** |

Table: Number of nodes explored (N), solving time in sec (T) and number of instances not solved within $10^3$ sec (F).

- Observations :
  - `BnB+scr` outperforms the two other methods
  - The reduction in the solution time is more important than the reduction in the number of nodes explored
  - Double kiss-cool effect : the bounding step is performed all the faster as *many variables are fixed* by the node-screening tests

### Take home message

In a BnB applied to a sparse problem, there is not always need to spend too much computation in the bounding process. Many nodes that can be *easily pruned* by performing *simpler and cheaper tests* like the node-screening one.