Exact Sparse Nonnegative Least Squares

A combinatorial approach

Nicolas Nadisic, Arnaud Vandaele, Nicolas Gillis, Jeremy Cohen

May 4-8th, 2020 — ICASSP 2020 (virtual conference)

University of Mons, Belgium

Paper and slides:
`http://nicolasnadisic.xyz/publication/exact-sparse-nnls/`

Nonnegative Least Squares

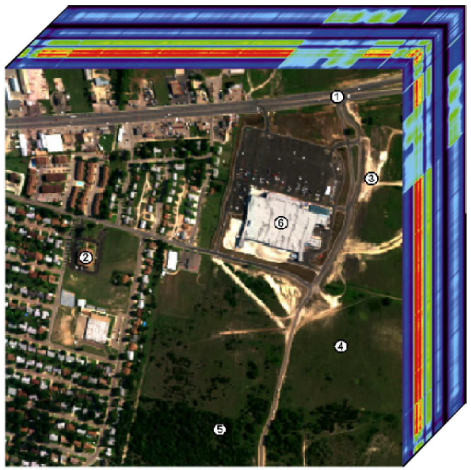
- Nonnegative Least Squares problems of the form

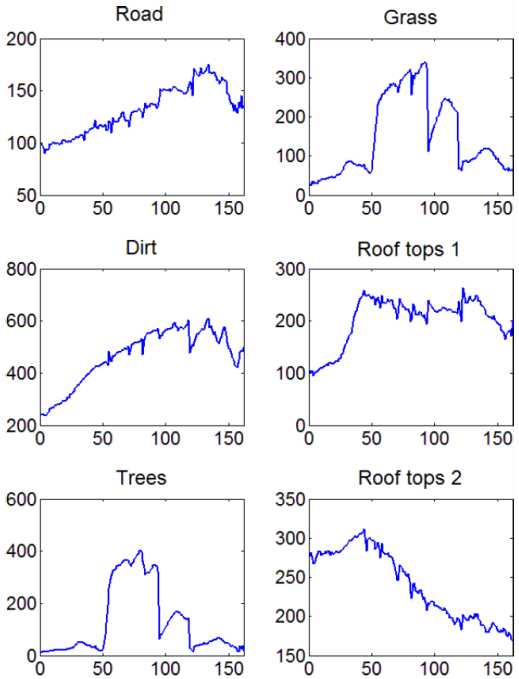
$$\min ||Ax - b||_2^2 \text{ such that } x \geq 0 \tag{1}$$

  are a variant of Least Squares problems where the solution is constrained to be entry-wise nonnegative.
- They are useful when data points are modeled as additive combinations of atoms.

Example of application: Hyperspectral Unmixing

$$\underbrace{\mathbf{M}(:,j)}_{\substack{\text{spectral signature}\\ \text{of } j\text{th pixel}}} \approx \sum_{k=1} \underbrace{\mathbf{U}(:,k)}_{\substack{\text{spectral signature}\\ \text{of } k\text{th endmember}}} \underbrace{\mathbf{V}(k,j)}_{\substack{\text{abundance of } k\text{th endmember}\\ \text{in } j\text{th pixel}}} .$$





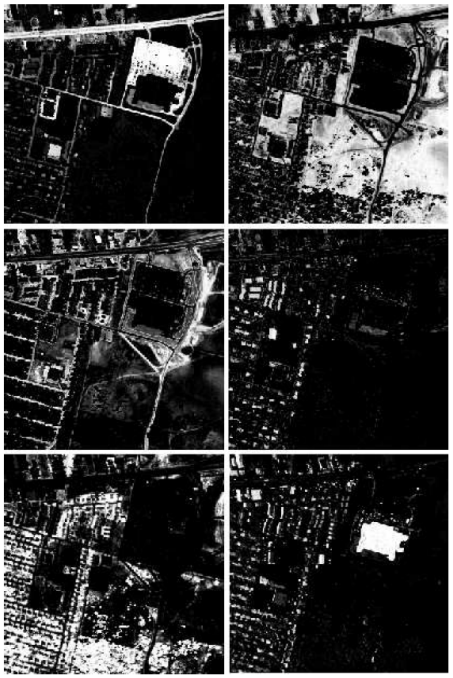


Figure 1: Given M and U, find V.

Sparsity

- Sparsity = Most entries are zero.
- Sparsity improves interpretability.

Sparsity

- Sparsity = Most entries are zero.
- Sparsity improves interpretability.

- Nonnegativity naturally produces sparse solutions...
  ...but without guarantee.
- Controlling the level of sparsity is useful in many applications.
- Ex: we want a pixel to be expressed as a combination of at most 3 materials.

Sparsity

- How to quantify sparsity?
- A natural measure: ℓ_0-"norm"
  $||x||_0 = |\{i : x_i \neq 0\}|$ (number of nonzero entries of x).

Sparsity

- How to quantify sparsity?
- A natural measure: ℓ_0-"norm"
  $||x||_0 = |\{i : x_i \neq 0\}|$ (number of nonzero entries of x).

Problem

$||x||_0$ is non-convex and non-smooth $\Rightarrow$ hard to enforce as a constraint.

Sparsity

- How to quantify sparsity?
- A natural measure: ℓ_0-"norm"
  $||x||_0 = |\{i : x_i \neq 0\}|$ (number of nonzero entries of x).

Problem

$||x||_0$ is non-convex and non-smooth $\Rightarrow$ hard to enforce as a constraint.

Possible relaxation

$||x||_1 = \sum_{i=1}^{n} |x_i|$ is convex $\Rightarrow$ easy to optimize.

It is often used as a sparsity measure.

LASSO

- There are many efficient sparsity-inducing techniques.
- The most famous is LASSO regularization, where the ℓ_1-norm is used as a convex surrogate of ℓ_0 and applied as a penalty to the objective function.

LASSO

- There are many efficient sparsity-inducing techniques.
- The most famous is LASSO regularization, where the ℓ_1-norm is used as a convex surrogate of ℓ_0 and applied as a penalty to the objective function.

The NNLS problem becomes:

$$\min ||Ax - b||_2^2 + \lambda ||x||_1 \text{ such that } x \geq 0, \text{ for a given } \lambda > 0. \quad (2)$$

LASSO

- There are many efficient sparsity-inducing techniques.
- The most famous is LASSO regularization, where the ℓ_1-norm is used as a convex surrogate of ℓ_0 and applied as a penalty to the objective function.

The NNLS problem becomes:

$$\min ||Ax - b||_2^2 + \lambda ||x||_1 \text{ such that } x \geq 0, \text{ for a given } \lambda > 0. \quad (2)$$

Drawbacks

- λ is hard to tune to reach a sparsity target.
- There is no guarantee on the solution sparsity.
- As the optimized problem changes, a bias is introduced.

Hard ℓ_0 constraint

It's better to impose hard ℓ_0 constraints, in order to:

- Be able to define an explicit sparsity target.
- Have guarantees on the solution's sparsity.

Hard ℓ_0 constraint

It's better to impose hard ℓ_0 constraints, in order to:

- Be able to define an explicit sparsity target.
- Have guarantees on the solution's sparsity.

Problem statement

Given $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $k \in \mathbb{N}$, find

$$x^* = \operatorname{argmin} \|Ax - b\|_2^2 \text{ such that } \|x\|_0 \leq k \text{ and } x \geq 0. \tag{3}$$

- This is the Sparse Nonnegative Least Squares problem, also known as Nonnegative Sparse Coding.
- We wish to solve it exactly.

A combinatorial problem

- ℓ_0-"norm" is discrete $\Rightarrow$ the problem is combinatorial in nature.
- It is equivalent to "Find the optimal support of x with cardinality k" (support = set of nonzero entries).
- There are $\binom{n}{k}$ possible supports.

Example

For $n = 4$ and $k = 2$, possible k-sparse solutions are:
$\{00x_3x_4\}$ $\{0x_20x_4\}$ $\{0x_2x_30\}$ $\{x_100x_4\}$ $\{x_10x_30\}$ $\{x_1x_200\}$.

A combinatorial problem

- ℓ_0-"norm" is discrete $\Rightarrow$ the problem is combinatorial in nature.
- It is equivalent to "Find the optimal support of x with cardinality k" (support = set of nonzero entries).
- There are $\binom{n}{k}$ possible supports.

Example

For $n = 4$ and $k = 2$, possible k-sparse solutions are:
$\{00x_3x_4\}$ $\{0x_20x_4\}$ $\{0x_2x_30\}$ $\{x_100x_4\}$ $\{x_10x_30\}$ $\{x_1x_200\}$.

- Exact resolution is possible with a bruteforce approach (solve a NNLS subproblem for every possible support).
- It doesn't scale well: $\binom{n}{k}$ subproblems to solve.

An efficient algorithm for exact resolution

- To avoid exploring all of the possible supports, we can prune the search space using a branch-and-bound strategy.
- Our solution: the `arborescent` algorithm
  `arborescent` Realizes a Branch-and-bound Optimization to Require Explicit Sparsity Constraints to be Enforced in NNLS Tasks

arborescent

root node, unconstrained

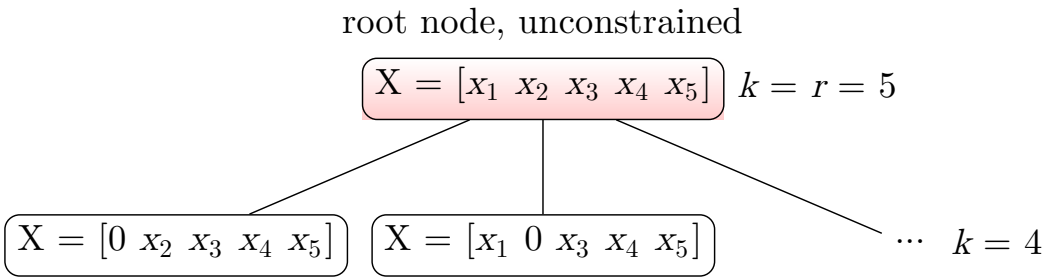
$\boxed{\mathrm{X} = [x_1 \;\; x_2 \;\; x_3 \;\; x_4 \;\; x_5]}$ $k = r = 5$

arborescent

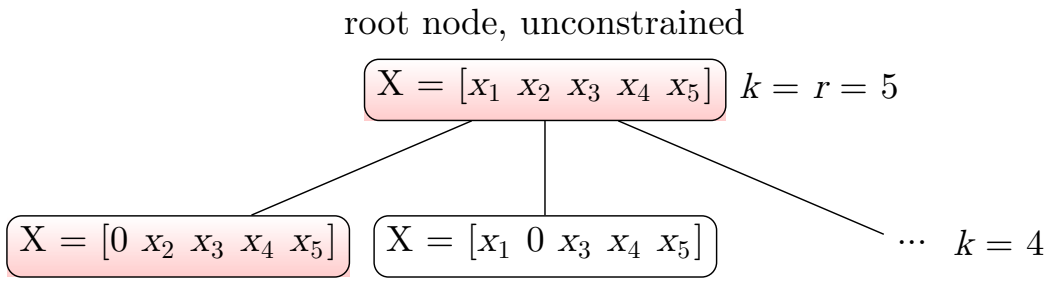
root node, unconstrained

$\boxed{\mathrm{X} = [x_1 \;\; x_2 \;\; x_3 \;\; x_4 \;\; x_5]}$ $k = r = 5$

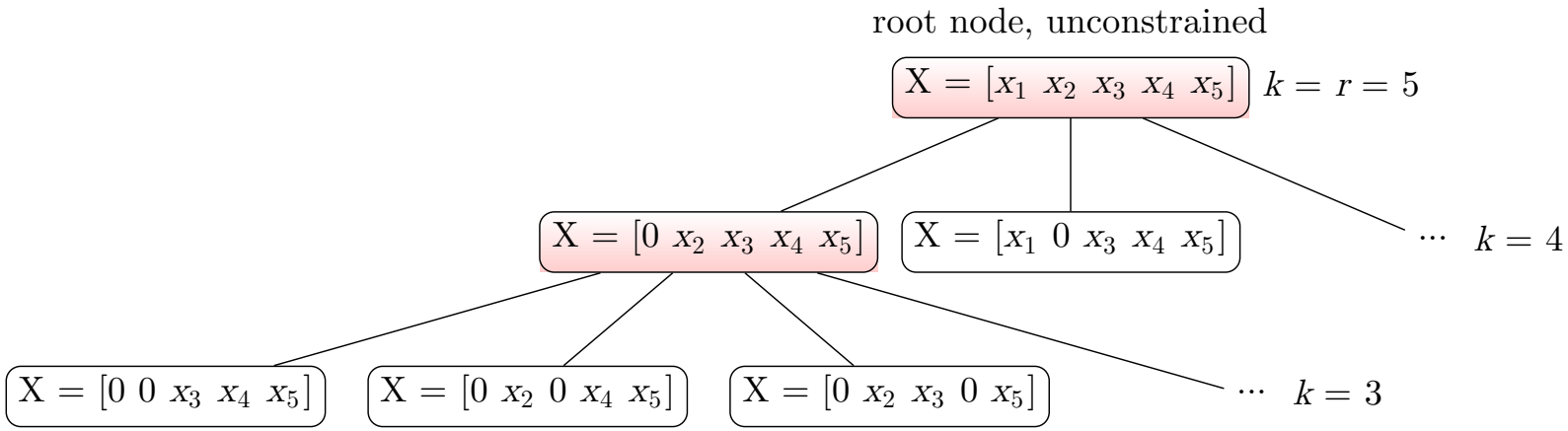
arborescent



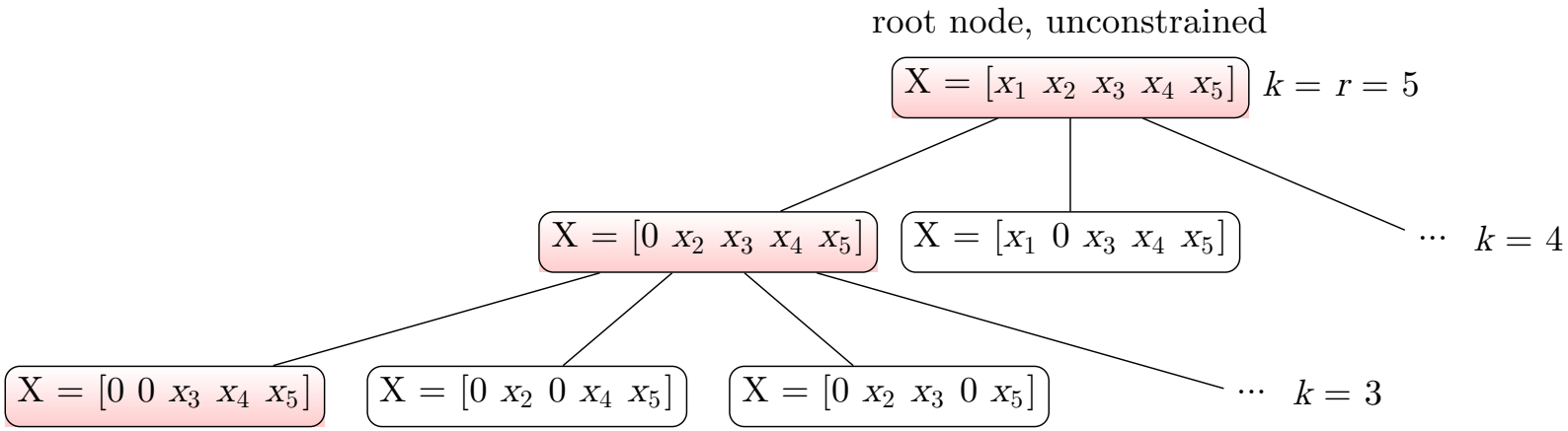
arborescent



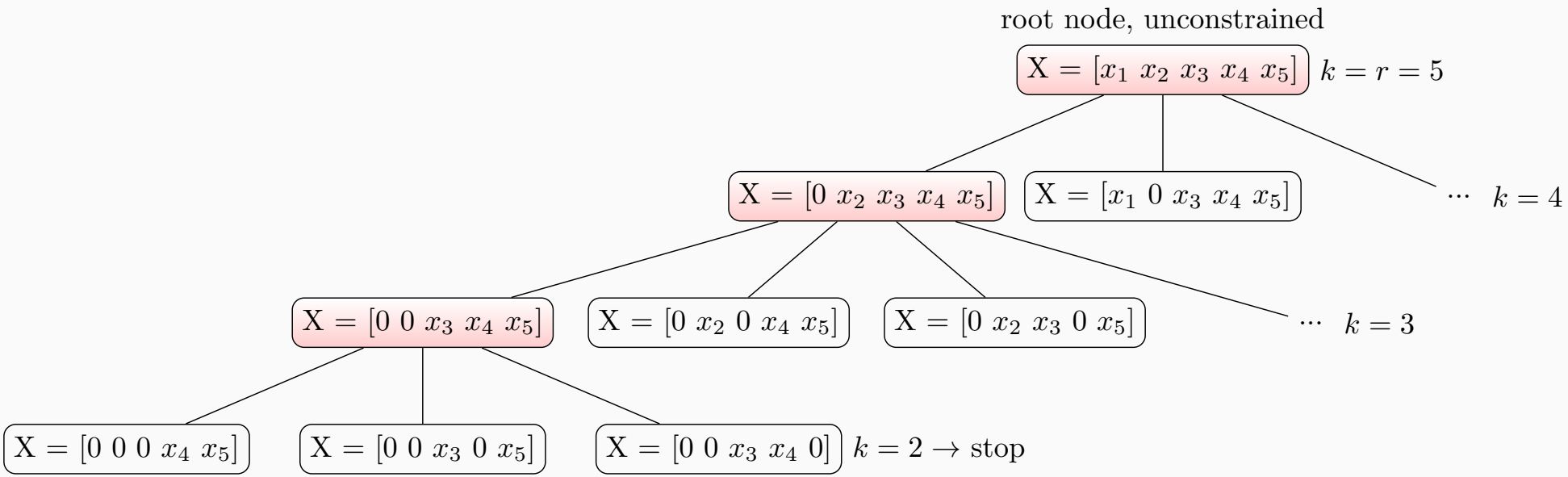
arborescent



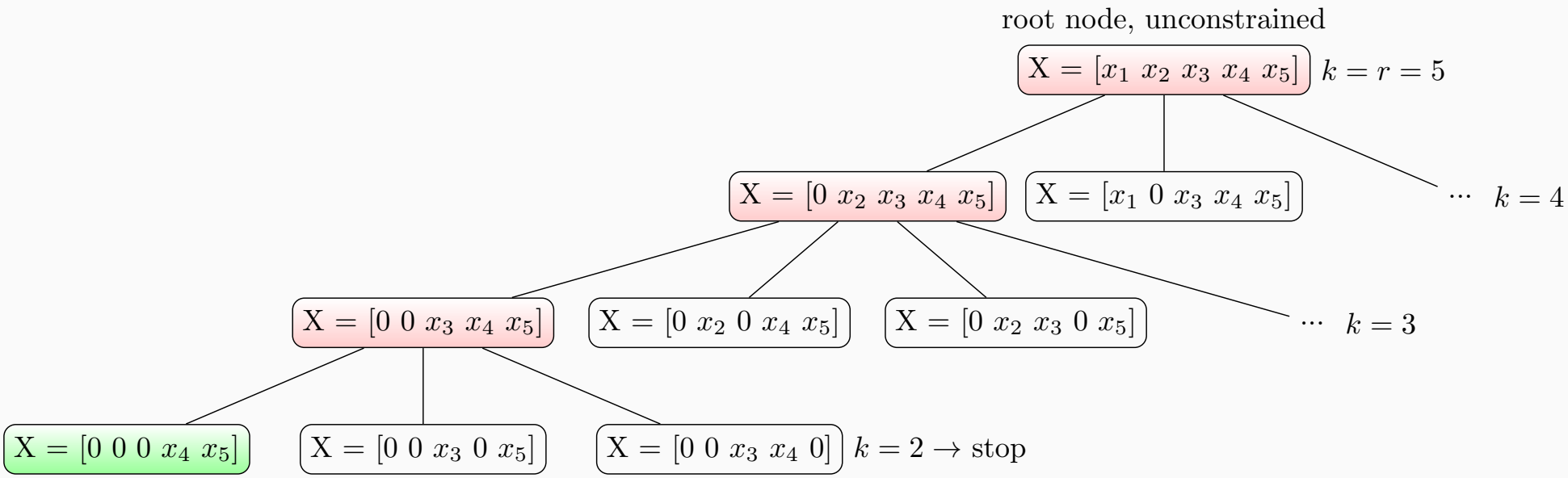
arborescent



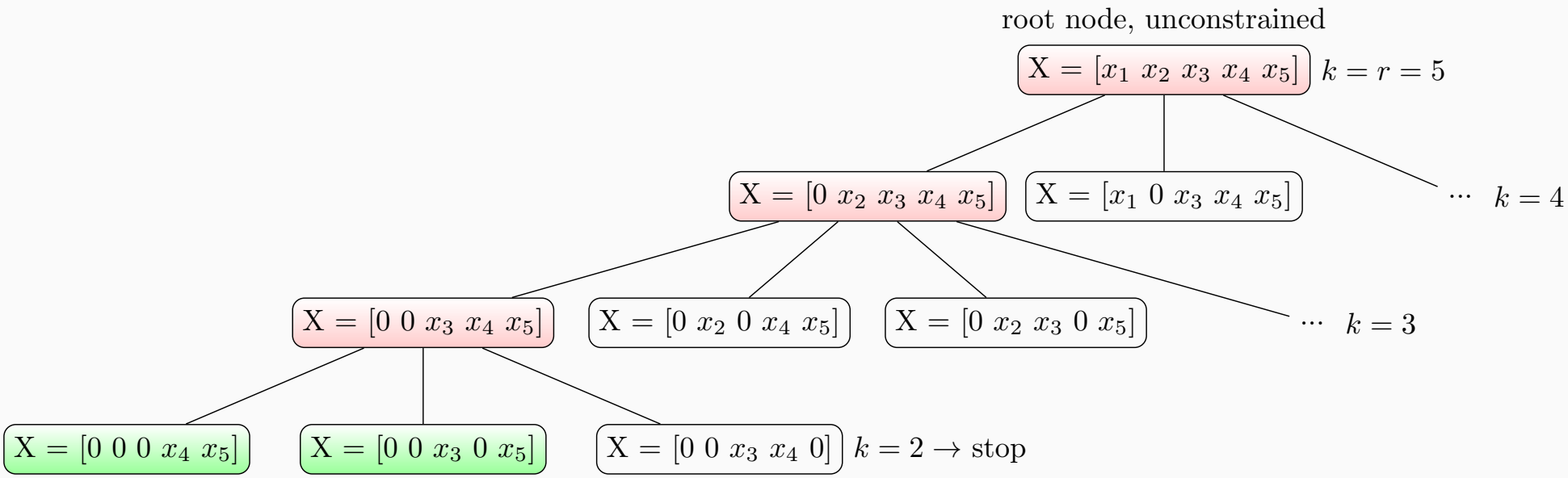
arborescent



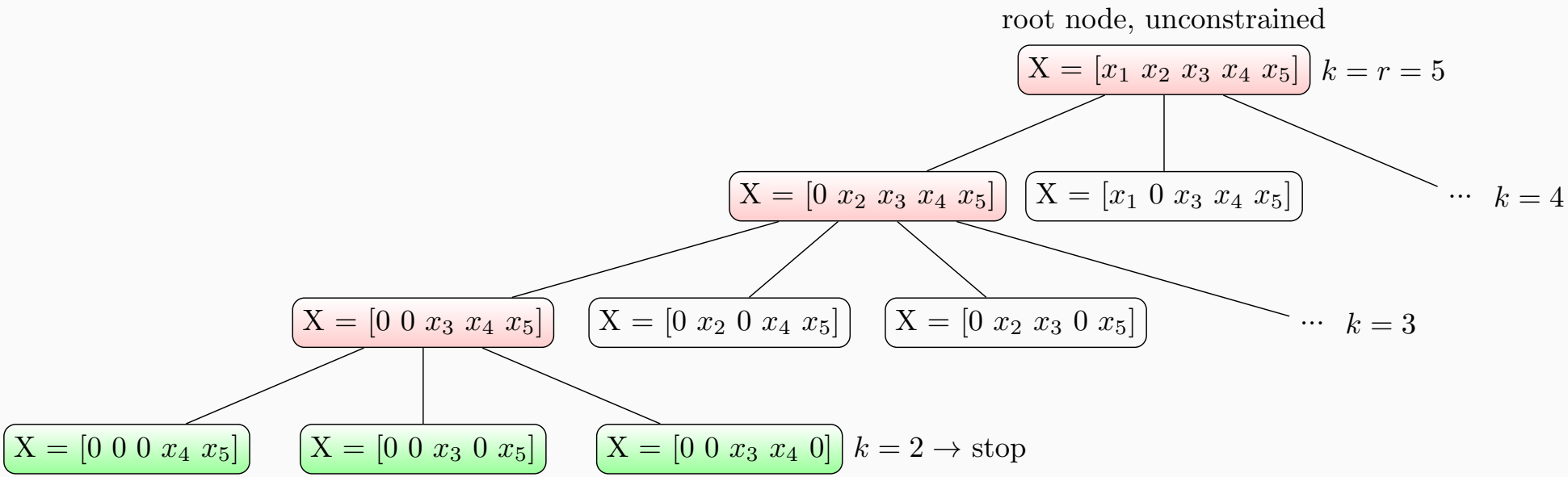
arborescent



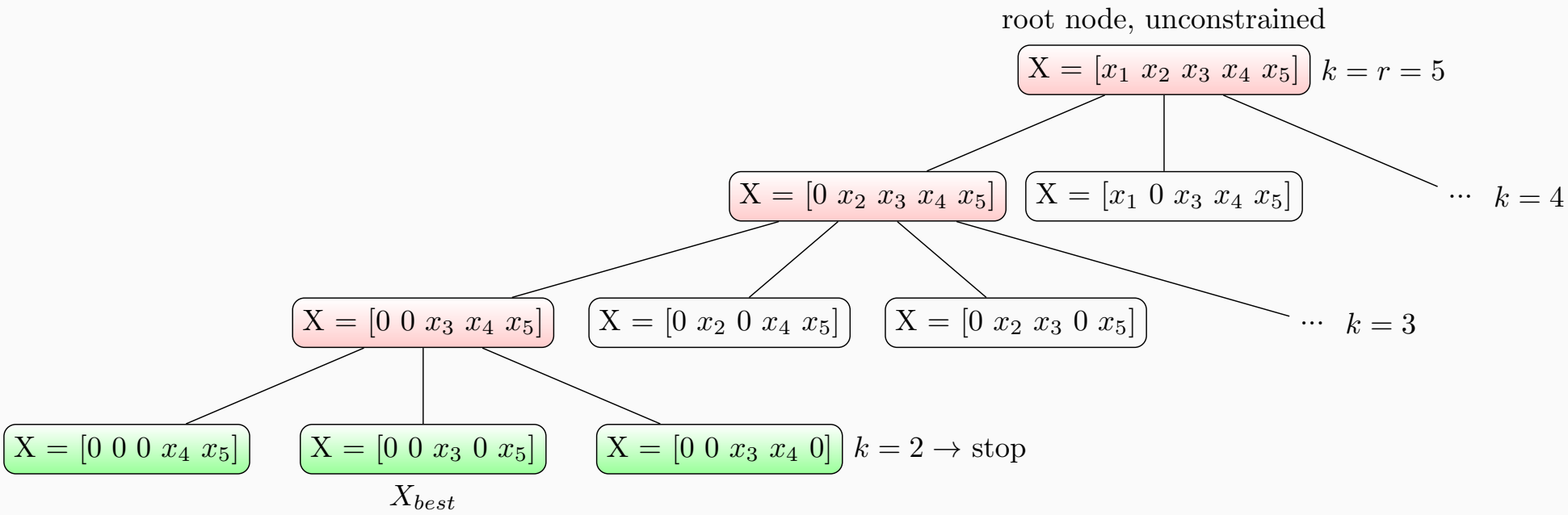
arborescent



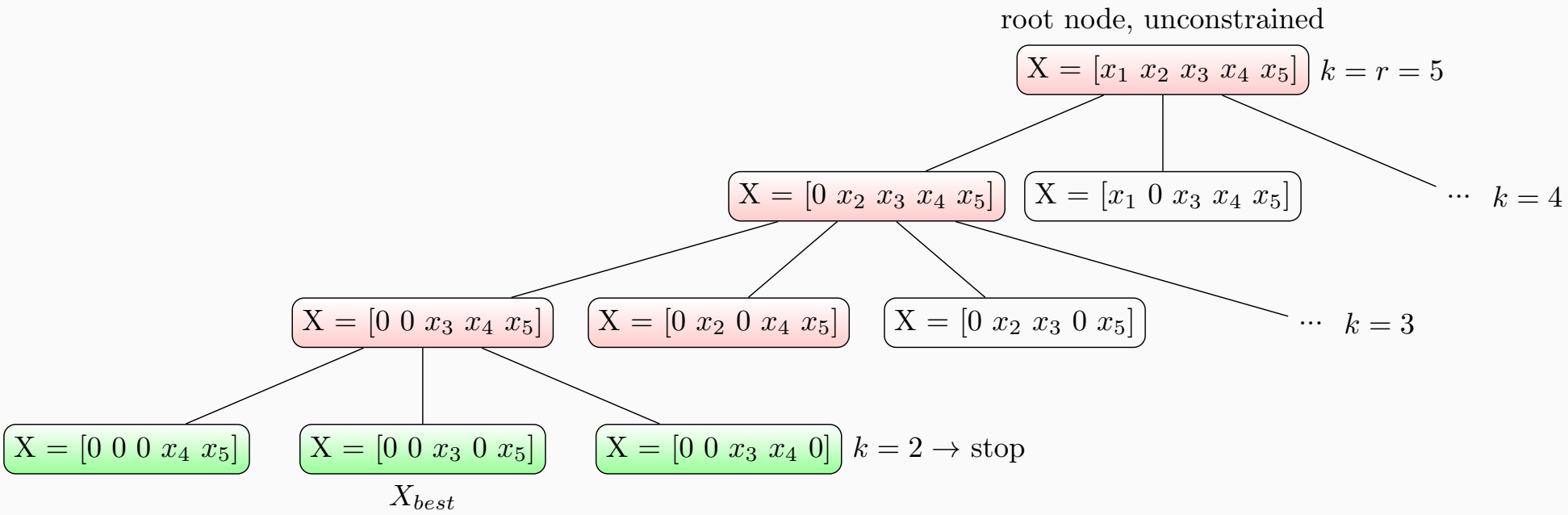
arborescent



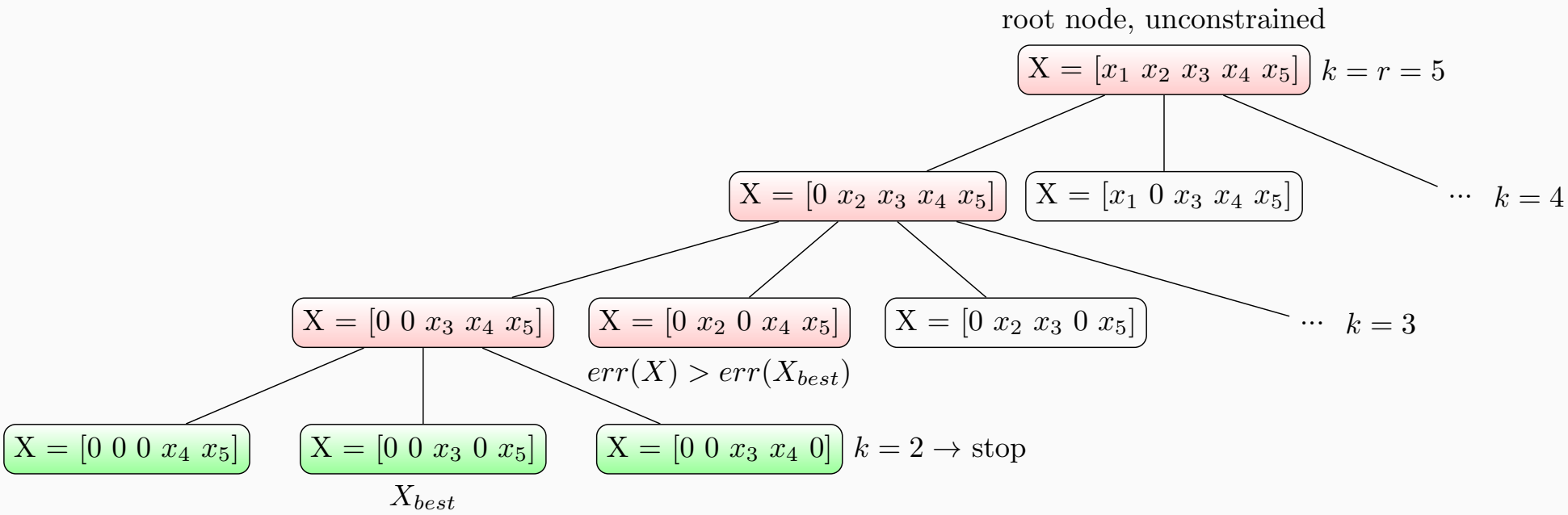
arborescent



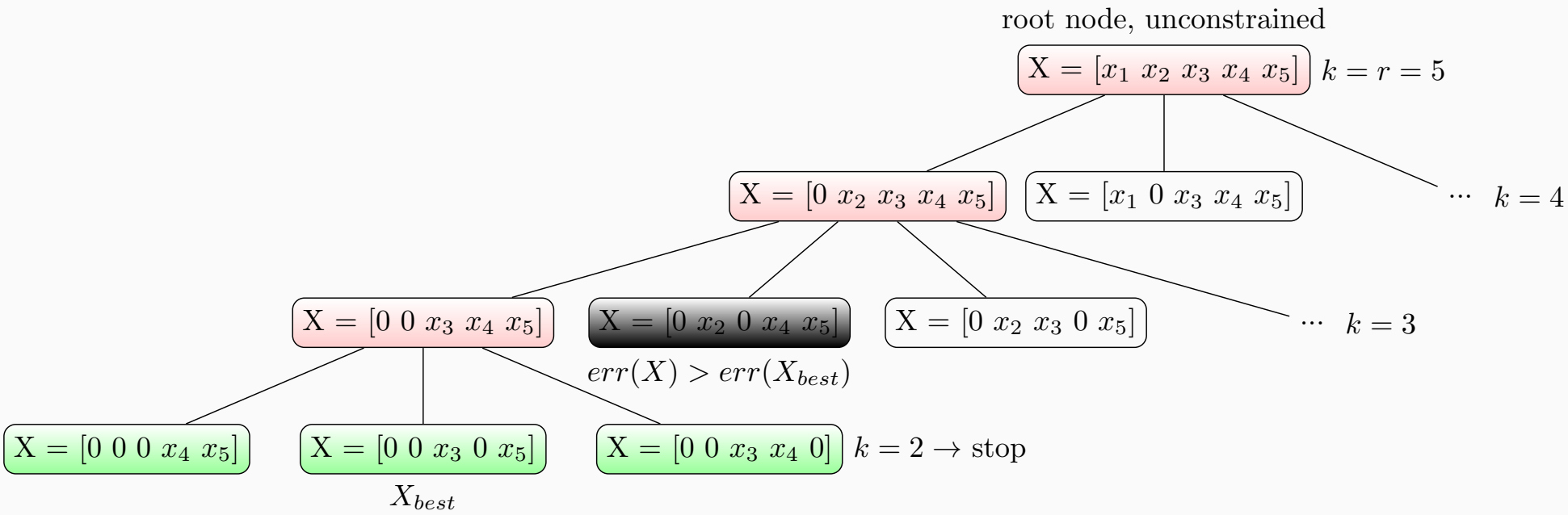
arborescent



arborescent



arborescent



- One node = an over-support + the parent solution.
- NNLS subproblems are solved with an active-set algo, initialized with the parent solution.
- At root node, entries of X are sorted (ascending order of entries of the solution to the unconstrained problem). Then the exploration is depth-first and "left-first".
- Avoid symmetry.

Input: $A \in \mathbb{R}_+^{m \times n}$, $b \in \mathbb{R}_+^m$, $k \in \{1, 2, \ldots, n\}$

Output: $x_{best} = \arg\min_{x \geq 0} ||Ax - b||_2^2$ s.t. $||x||_0 \leq k$

```
1  Init x_0 ← NNLS(A, b); Sort the entries in x_0;
2  Init K_0 ← {1, ..., n}; Init best_error ← +∞; Init P ← {(K_0, x_0)}
3  while P ≠ ∅ do
4      (K, x_parent) = P.select()
5      P ← P \ {(K, x_parent)}
6      (error, x) ← NNLS(A(:, K), b, x_parent(K))
7      if error > best_error then
8          prune (do nothing)
9      else
10         if size(K) > k then
11             foreach i ∈ K do
12                 P ← P ∪ {(K \ {i}, x) }
13         else
14             if error < best_error then
15                 best_error ← error
16                 x_best ← x
```

Competing algorithms

- LASSO: a ℓ_1-penalized coordinate descent.
- Greedy algorithms: Nonnegative versions of OMP and OLS [Nguyen et al., 2019].
- Bruteforce ([Cohen and Gillis, 2019] for Sparse NMF)

Experiments: Comparison on synthetic datasets

Data:

- We generate a matrix $A \in \mathbb{R}_+^{m \times n}$ and a random k-sparse vector $x_{true} \in \mathbb{R}_+^n$, compute $b = Ax_{true}$, and run the algorithms with A, b and k as input.
- Fixed $n = 20$ and $k = 10$.
- Three values of m:1000, 100, 20.
- A is generated well-conditioned or ill-conditioned.
- 5% noise is added to b, or not.
- Total of 12 test settings, we do 100 runs for each one.

Measures:

- Average relative error $\frac{||Ax-b||_2}{||b||_2}$.
- Average computing time (in seconds).
- Success = number of time the support of x_{true} is recovered.

Experiments: Comparison on synthetic datasets i

	Well-cond A, Noiseless b			Well-cond A, Noisy b			Ill-cond A, Noiseless b			Ill-cond A, Noisy b		
	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.
L1-CD	0	6.83	100	5.01	3.13	97	3.50	8.61	16	6.28	4.12	9
CVX	0	796.90	100	4.96	634.62	100	0.08	609.45	96	4.98	571.72	98
NNOMP	0	5.60	100	4.96	3.73	100	2.79	4.14	3	5.83	3.63	3
NNOLS	0	4.67	100	4.96	3.36	100	1.95	4.21	23	5.50	3.13	22
arbo.	0	43.09	100	4.96	1369.30	100	0	29.80	100	4.96	1223.20	100

Results for $m = 1000$, $n = 20$, $k = 10$.

Experiments: Comparison on synthetic datasets ii

	Well-cond A, Noiseless b			Well-cond A, Noisy b			Ill-cond A, Noiseless b			Ill-cond A, Noisy b		
	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.
L1-CD	0.27	1.93	93	4.97	2.48	84	3.65	2.22	11	6.02	1.97	7
CVX	0	536.95	100	4.73	502.00	100	0.02	508.60	98	4.84	489.68	58
NNOMP	0.48	2.83	89	4.98	2.79	83	3.09	2.83	3	5.50	2.80	2
NNOLS	0.20	2.52	95	4.88	2.62	91	2.15	2.55	14	5.11	2.55	18
arbo.	0	46.32	100	4.73	1145.10	100	0	29.39	100	4.71	1304.40	63

Results for $m = 100$, $n = 20$, $k = 10$.

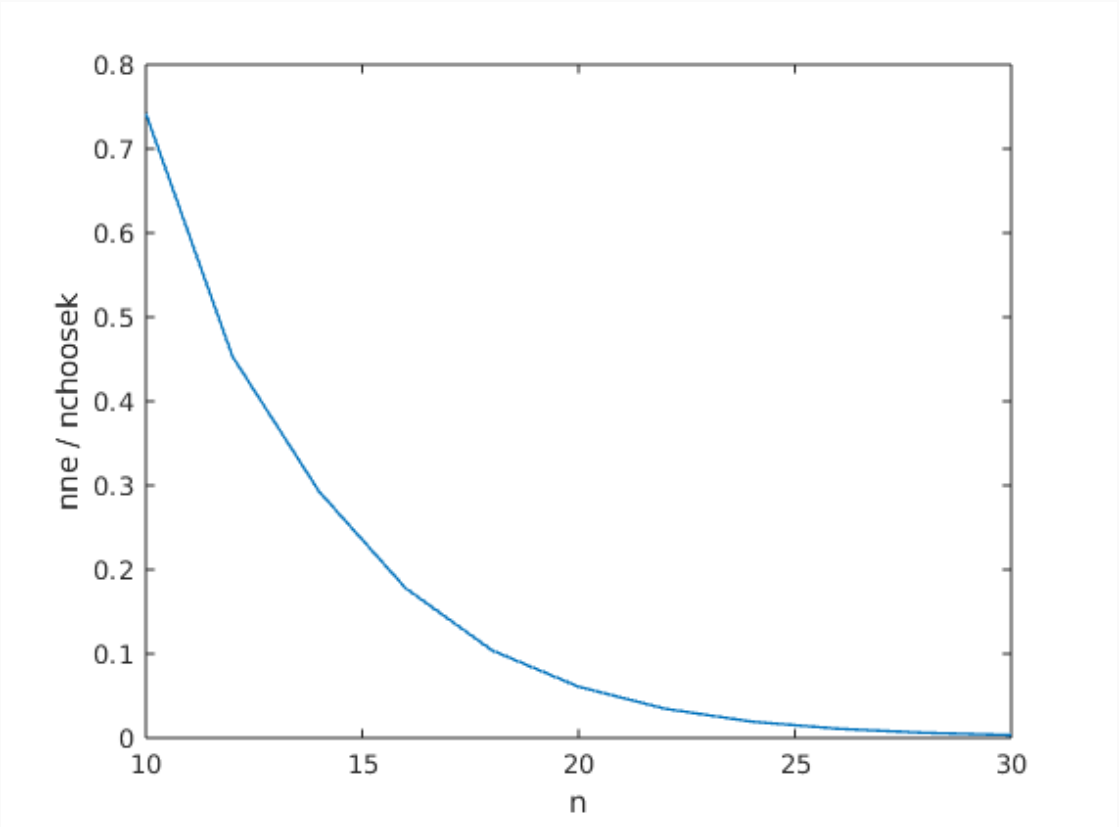
Experiments: Comparison on synthetic datasets iii

	Well-cond A, Noiseless b			Well-cond A, Noisy b			Ill-cond A, Noiseless b			Ill-cond A, Noisy b		
	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.
L1-CD	2.88	1.39	23	4.29	1.31	15	3.41	1.44	5	4.66	1.53	1
CVX	0.00	532.23	100	3.26	495.62	23	0.95	548.41	39	3.07	509.10	7
NNOMP	3.02	2.85	12	3.85	2.74	6	2.07	2.96	2	3.57	2.88	0
NNOLS	2.57	2.59	18	3.73	2.54	10	1.48	3.02	12	3.26	3.08	1
arbo.	0	46.84	100	3.09	1472.20	30	0	34.20	100	2.83	1283.70	11

Results for $m = 20$, $n = 20$, $k = 10$.

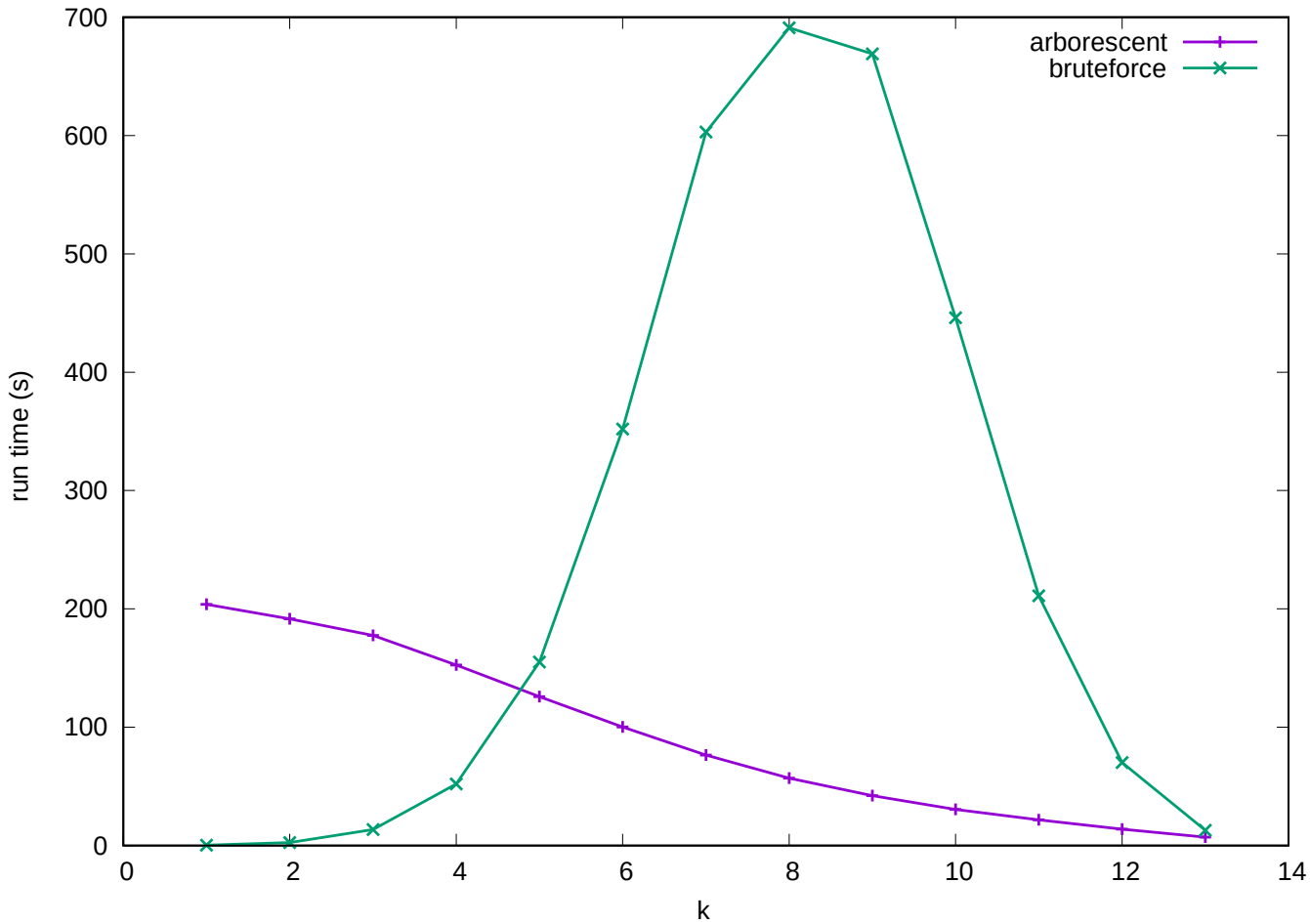
Experiment: scalability of `arborescent`

- For different values of n, and $k = n/2$, we generate 100 random datasets with 5% noise and run `arborescent`.
- We measure the average number of nodes explored by `arborescent`, divided by the number of nodes explored by bruteforce $\binom{n}{k}$.

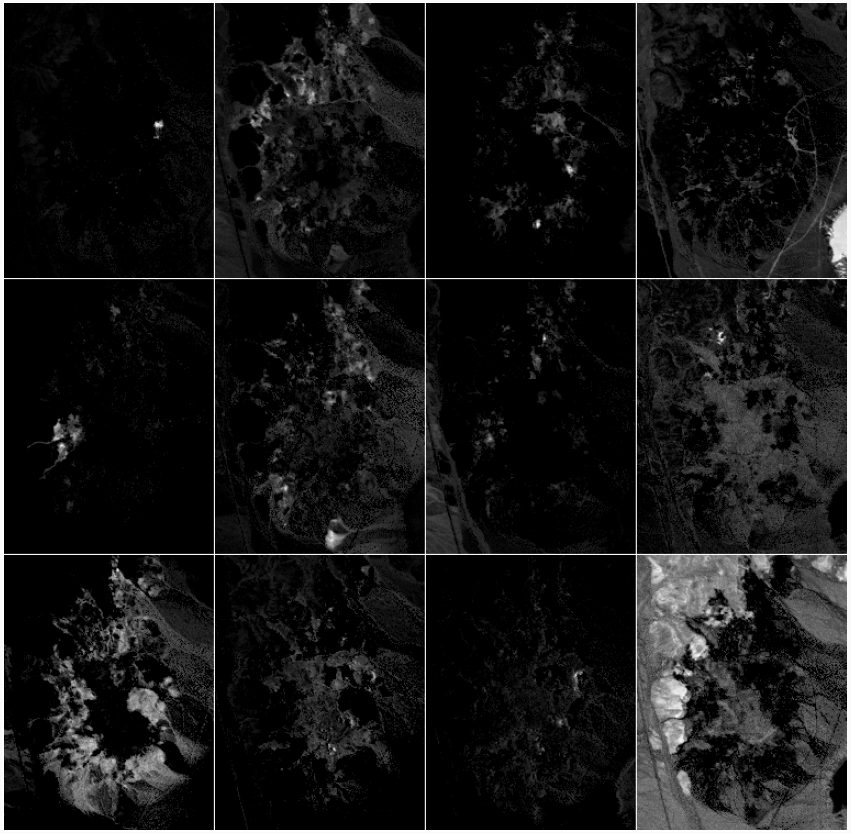


Experiment: `arborescent` vs bruteforce

- For m and n fixed, and k taking all values between 1 and $n-1$, we generate 100 random datasets, run `arborescent` and a bruteforce method, and measure the average running times.



Experiment: Hyperspectral Unmixing on Cuprite Image



	NNLS	LASSO	arbo.
Time (s.)	15.81	22.35	1053
Rel. Error (%)	1.74	4.21	1.78
Sparsity	6.61	4.34	4.77

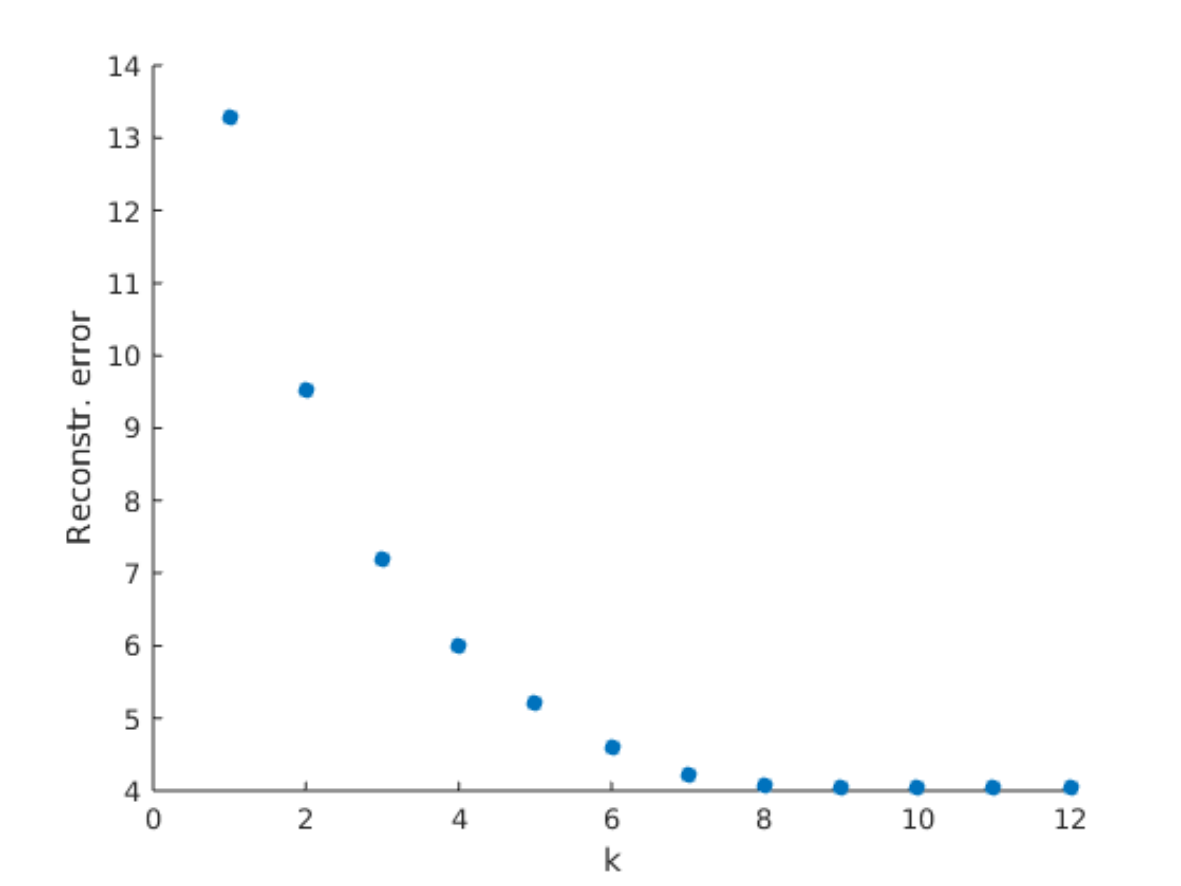
- Given an hyperspectral image of $250 \times 191 = 47750$ pixels in $m = 188$ denoised spectral bands, and a dictionary of $n = 12$ materials, identify the materials in the pixels.
- Algorithms compared: coordinate descent without sparsity constraint (NNLS), with sparsity constraints (LASSO), and `arborescent`.

Conclusion

- We proposed `arborescent`, a branch-and-bound algorithm to solve exactly the k-Sparse Nonnegative Least Squares problem.
- It works in very general settings, when traditional approaches fail.
  - (At the cost of higher computation time)
- Performs well on a real-life hyperspectral unmixing application.
- Generally much faster than bruteforce methods.
  - (Except when k is very small)

WIP: Pareto front

- As a side-effect, when computing the k-sparse solution, `arborescent` also computes all of the k'-sparse solutions, $\forall k' \in \{k, k+1, ...n\}$.
- It gives a kind of Pareto front (minimizing the error and k).
- It allows for "automatic k detection".



References i

Cohen, J. E. and Gillis, N. (2019).
Nonnegative Low-rank Sparse Component Analysis.
In *2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8226–8230.

Nguyen, T. T., Idier, J., Soussen, C., and Djermoune, E.-H. (2019).
Non-Negative Orthogonal Greedy Algorithms.
IEEE Transactions on Signal Processing, pages 1–16.

Contact: nicolas.nadisic@umons.ac.be

Paper and slides:
http://nicolasnadisic.xyz/publication/exact-sparse-nnls/

Code and exp.:
https://gitlab.com/nnadisic/sparse-nmf