Exact Biobjective k-Sparse Nonnegative Least Squares

Nicolas Nadisic, Arnaud Vandaele, Nicolas Gillis
Université de Mons
Mons, Belgium
{firstname.lastname}@umons.ac.be

Jeremy E. Cohen
Université de Rennes, INRIA, CNRS, IRISA
Rennes, France
jeremy.cohen@irisa.fr

***Abstract*—The k-sparse nonnegative least squares (NNLS) problem is a variant of the standard least squares problem, where the solution is constrained to be nonnegative and to have at most k nonzero entries. Several methods exist to tackle this NP-hard problem, including fast but approximate heuristics, and exact methods based on brute-force or branch-and-bound algorithms. Although intuitive, the k-sparse constraint is sometimes limited; the parameter k can be hard to tune, especially in the case of NNLS with multiple right-hand sides (MNNLS) where the relevant k could differ between columns. In this work, we propose a novel biobjective formulation of the k-sparse nonnegative least squares problem. We present an extension of Arborescent, a branch-and-bound algorithm for exact k-sparse NNLS, that computes the whole Pareto front (that is, the set of optimal solutions for all values of k) instead of only the k-sparse solution, for virtually the same computing cost. We also present a method for MNNLS that enforces a matrix-wise sparsity constraint, by first computing the Pareto front for each column and then selecting one solution per column to build a globally optimal solution matrix. We show the advantages of the proposed approach for the unmixing of hyperspectral images.**

***Index Terms*—sparse approximation, ℓ_0 constraint, biobjective optimization, nonnegative least squares**

I. Introduction

Nonnegative least squares (NNLS) problems occur in many signal processing and data mining tasks, when data points can be expressed as additive linear combinations of basis components [1]. For example, in hyperspectral images, the spectral signature of a pixel is the additive linear combination of the spectral signatures of the materials it contains [2]. NNLS problems are also at the heart of many alternating algorithms to compute nonnegative matrix factorization (NMF) [3].

Given $A \in \mathbb{R}^{m \times r}$ and $b \in \mathbb{R}^m$, the standard NNLS problem can be written as follows,

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

where $x \in \mathbb{R}^r$, and $x \geq 0$ means x is entry-wise nonnegative.

Nonnegativity in least squares problems is known to naturally induce sparsity (see Theorem 6.1 in [4]), that is, solutions with few nonzero entries. Sparsity is an appreciated feature, as it often improves the interpretability of the solution even for invertible linear systems. For example, in hyperspectral unmixing, that is, the task of identifying materials in a hyperspectral image, sparsity means a pixel is expressed as a combination of only a few materials. However, there is no guarantee on the sparsity of the solution of a general NNLS problem, while some applications may benefit from explicit sparsity constraints. Leveraging prior knowledge of the sparsity of the solution can help regularize the problem, reduce noise, and improve the results.

The most natural sparsity measure is the ℓ_0-"norm", as it is equal to the number of nonzero entries of a vector, $\|x\|_0 = Card\{i : x_i \neq 0\}$. A vector is said k-sparse if it has at most k nonzero entries. A common way to enforce sparsity is with a *k-sparsity* constraint. Combined with nonnegativity, this leads to the following problem, called k-sparse NNLS,

$$\min_x \|Ax - b\|_2^2 \text{ such that } x \geq 0 \text{ and } \|x\|_0 \leq k. \tag{2}$$

This problem is sometimes called nonnegative sparse coding or cardinality-constrained NNLS.

In hyperspectral unmixing, the k-sparsity constraint means a pixel can be composed of at most k materials. Albeit quite intuitive, this formulation still suffers from the need to choose an appropriate parameter k. Most importantly, in NNLS problems with multiple right-hand sides (MNNLS), the suitable k often varies from one column to another (pixels can contain different numbers of materials), and imposing a single sparsity parameter can produce inadequate results.

To overcome this issue, instead of optimizing the error while constraining the sparsity, one can consider a *biobjective* formulation. Here, the objectives are minimizing the reconstruction error on the one hand, and maximizing the sparsity (that is, minimizing the ℓ_0-"norm") on the other hand,

$$\min_{x \geq 0} \{\|Ax - b\|_2^2, \|x\|_0\}. \tag{3}$$

These objectives are conflicting, hence there is no unique optimal solution to Problem (3), and solutions representing different tradeoffs between error and sparsity are all equally good. Therefore, we consider the notion of *Pareto-optimality*. Given a set of objectives to optimize, a solution x is said Pareto-optimal if and only if it is *not dominated*, that is, there does not exist a solution that is at least as good as x on all objectives and strictly better than x on a least one objective.

NN and NG acknowledge the support by the European Research Council (ERC starting grant No 679515), and by the Fonds de la Recherche Scientifique - FNRS and the Fonds Wetenschappelijk Onderzoek - Vlanderen (FWO) under EOS project O005318F-RG47.

The set of all Pareto-optimal solutions to a problem is called the *Pareto front*, see Figure 1. In Problem (3), the discreteness

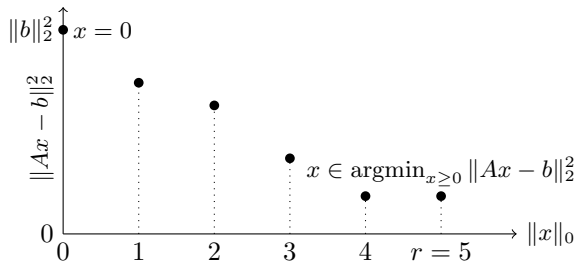


Fig. 1. Example of the Pareto front for a biobjective k-sparse NNLS problem with $r = 5$ variables. The first solution, for $\|x\|_0 = 0$, corresponds to the zero vector. The last solution, for $\|x\|_0 = 5$, corresponds to the NNLS problem with no sparsity constraint. Here the penultimate solution is identical to the last one, meaning that the solution with no sparsity constraint has naturally 1 zero entry.

of the ℓ_0-"norm" actually makes the computation of the Pareto front easier, as it suffices to solve Problem (2) for all possible values of $\|x\|_0$ in $\{1, 2, \ldots, r\}$. By computing the Pareto front instead of just one solution, we provide the user with a set of solutions to choose from, representing different tradeoffs between error and sparsity, and remove the need to define a parameter k a priori. Note that, when $\text{rank}(A) < m$, which includes the underdetermined case $m < r$, the optimal solution of Problem (2) is not necessarily unique. This does not change the principles of the methods presented in this work; they return one optimal solution among the possible ones.

In this paper, we tackle Problem (3) exactly. In section II, we review briefly existing approaches for sparse NNLS. In section III, we describe the existing branch-and-bound algorithm Arborescent (abbreviated Arbo), upon which our approach is built. In section IV, we introduce our novel extension of Arbo to tackle Problem (3) exactly. In section V, we present our approach to leverage this extension in the context of sparse MNNLS. In section VI, we illustrate the proposed method with the unmixing of hyperspectral images.

II. Related work

The discreteness of the ℓ_0-"norm" makes problems like (2) combinatorial, and thus hard to solve. For this reason, many approximate methods have been used.

The most common one is to use the ℓ_1-norm as a convex relaxation of the ℓ_0-"norm" in order to leverage the efficient algorithms and strong theoretical results from convex optimization. The ℓ_1-penalized problem $\min_x \|Ax-b\|_2+\lambda\|x\|_1$ is called the LASSO, and several nonnegative variants have been studied, see for example [5]. However, these methods suffer from several drawbacks. Tuning the parameter λ to reach a target sparsity can be tricky, especially in MNNLS where the adequate λ can vary between columns. Although there exist conditions under which ℓ_1 methods are guaranteed to produce a solution with the same support as the ℓ_0 method, they are quite restrictive in practice [6].

Greedy heuristics are also widely used. These methods start with an empty support, and select entries one by one to add to the support, until the target sparsity k is reached. The selection is done greedily, by choosing at each iteration the entry that maximizes the decrease of the error. Orthogonal variants make sure that entries are not selected more than once; orthogonal matching pursuit (OMP) and orthogonal least squares (OLS) are the most popular algorithms. Recently, nonnegative variants have been studied, see [7] and the references therein. They solve (2) approximately, and recovery guarantees depend on conditions that can be restrictive in practice.

A few methods were proposed to solve exactly ℓ_0-constrained problems, similar to (2) but with different constraints. Reference [8] introduced a branch-and-cut algorithm using continuous relaxations of the ℓ_0-"norm". It was latter extended and improved, see [9] and the references therein. Reference [9] introduced mixed-integer programming (MIP) formulations for several variants involving the ℓ_0-"norm" (to be able to solve them with a generic MIP solver), and [10] proposed dedicated branch-and-bound algorithms to solve them. Finally, [11] introduced a branch-and-bound algorithm specifically designed for k-sparse NNLS; this is the foundation our work is based upon.

III. The arborescent algorithm

In this section, we briefly describe the algorithm Arbo introduced in [11]. This algorithm solves k-sparse NNLS (2) exactly using a branch-and-bound strategy. Instead of enumerating all possible supports, it uses the structure of the problem to prune large parts of the search space. This search space is mapped on a tree, see Figure 2. Every node of the tree represents an over-support $\mathcal{K}$ of x, that is, the set of entries of x not constrained to be zero, with $\mathcal{K} \subseteq \{1, 2, \ldots, r\}$. Exploring a node means solving the NNLS subproblem

$$f^*(\mathcal{K}) = \min ||A(:, \mathcal{K})x(\mathcal{K}) - b||_2^2 \text{ such that } x(\mathcal{K}) \geq 0,$$

where $x(\mathcal{K})$ is the subvector composed of the entries of x indexed by $\mathcal{K}$. The value $f^*(\mathcal{K})$ is the error associated to the node corresponding to $\mathcal{K}$. We solve the NNLS subproblems using an active-set method [12]. On top of solving the subproblems exactly, this method supports a *warm start*, that is, it can be initialized at a given node with the solution from a previous node. This significantly speeds up the computation as the initial guess at each node is close to the optimal solution.

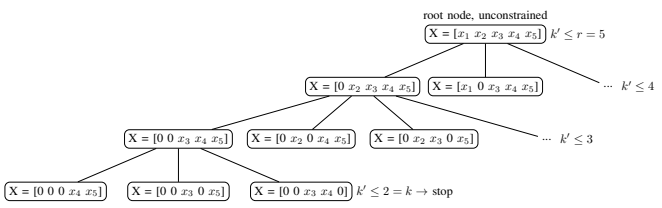


Fig. 2. Example of the Arbo search tree, for $r = 5$ and $k = 2$.

The root node represents the NNLS problem with no sparsity constraint, and every descending node represents this problem with one entry constrained to be zero. This is done recursively, until reaching the nodes with k unconstrained

entries. The nodes at this depth are leafs of the search tree, and represent feasible solutions to problem (2). To prune this tree, we use the fact that in any optimization problem, when adding constraints, the solution cannot improve. By construction, a given node will always have an error greater than (or equal to) the error of its parent node. When we reach a leaf, we obtain a feasible solution whose error is an upper bound for problem (2). Therefore, if a given node $\mathcal{N}$ has an error greater than this bound, then all children nodes descending from $\mathcal{N}$ will also have a error greater than the bound, and thus cannot be optimal solutions; $\mathcal{N}$ can be pruned safely. Moreover, by ordering the entries in the root node by ascending order and then exploring depth-first and "left-first", we first constrain to zero the entries that are already close to zero in the standard NNLS problem, and therefore that are more likely to be zero in the constrained problem. This strategy leads quickly to good feasible solutions and allows to prune efficiently large parts of the search space. Other technical choices that are key in the performance of the algorithm are detailed in [11].

IV. The biobjective extension

Although Arbo was designed to solve the k-sparse NNLS problem, it can be easily extended to compute the whole Pareto front. Indeed, while exploring the search tree and computing intermediary nodes, it also computes automatically the optimal k'-sparse solutions for all $k' \in \{k, \dots, r\}$. Our extension (that we call Arbo-Pareto) therefore consists in maintaining a list of the optimal k'-sparse solutions, making a comparison for every node explored, and updating the list when a better solution is found. This almost does not affect the computational cost of the algorithm as the cost of comparison and update is negligible compared to the cost of the exploration of a node.

To show that Arbo indeed computes these solutions, we prove by contradiction that it cannot prune an optimal k'-sparse solution. Suppose the node γ is the optimal k'-sparse solution for a given $k' > k$, and that it is pruned by Arbo. Because it is pruned, its error must be larger than the error of some feasible k-sparse solution α, $f^*(\gamma) > f^*(\alpha)$. However, there exists necessarily a parent node of α that is k'-sparse; we call it β. By construction, $f^*(\alpha) > f^*(\beta)$, so we have $f^*(\gamma) > f^*(\beta)$, meaning that γ is not the optimal k'-sparse solution. This contradicts the hypothesis.

Arbo-Pareto is detailed in Algorithm 1. NNLS refers to the active-set method described above. The set P is the pool of nodes, initialized with the root node (with no entry constrained) on line 4. A node is selected from P on line 8, and removed from P on line 9. On line 10, the NNLS subproblem restricted to the over-support $\mathcal{K}$ is solved using the parent solution as initialization. If the error at the current node is worse than the current best feasible solution, then no descending node can be optimal, and we prune the current node (line 13). Otherwise, we continue the exploration. If the sparsity target k is not reached, we generate one node for every entry of the over-support (lines 16 and 17). We then compare the error of the current node with the error of the current best k'-sparse solution, on line 18. If it is lower, we update this error (line 19) and the Pareto front (line 20).

Algorithm 1: Arbo-Pareto

Input: $A \in \mathbb{R}_+^{m \times r}$, $b \in \mathbb{R}_+^m$, $k \in \{1, 2, \dots, n\}$
Output: Pareto front $\mathcal{S}$

1 Init $\mathcal{K}_0 \leftarrow \{1, ..., r\}$
2 Init $x_0 \leftarrow \text{NNLS}(A, b)$
3 Sort the entries in x_0 in ascending order
4 Init $P \leftarrow \{(\mathcal{K}_0, x_0)\}$
5 Init $\mathcal{E}_i \leftarrow +\infty$ for all $i \in \{k, \dots, r\}$
6 Init $\mathcal{S}_i \leftarrow \vec{0}$ for all $i \in \{k, \dots, r\}$
7 **while** $P \neq \emptyset$ **do**
8 $\quad (\mathcal{K}, x_{parent}) = P.\text{select}()$
9 $\quad P \leftarrow P \setminus \{(\mathcal{K}, x_{parent})\}$
10 $\quad x, error \leftarrow \text{NNLS}(A(:, \mathcal{K}), b, x_{parent}(\mathcal{K}))$
11 $\quad k' = size(\mathcal{K})$
12 $\quad$ **if** $error > \mathcal{E}_k$ **then**
13 $\quad\quad$ prune (do nothing)
14 $\quad$ **else**
15 $\quad\quad$ **if** $k' > k$ **then**
16 $\quad\quad\quad$ **foreach** $i \in \mathcal{K}$ **do**
17 $\quad\quad\quad\quad P \leftarrow P \cup \{(\mathcal{K} \setminus \{i\}, x)\ \}$
18 $\quad\quad$ **if** $error < \mathcal{E}_{k'}$ **then**
19 $\quad\quad\quad \mathcal{E}_{k'} \leftarrow error$
20 $\quad\quad\quad S_{k'} \leftarrow x$

Note that, if $k = 1$, Arbo computes the whole Pareto front. Otherwise, it only computes the part with $k' \in \{k, \dots, r\}$. The extended algorithm Arbo-Pareto can be used to solve sparse NNLS problems in a biobjective way, but it can also be used as a subroutine in a MNNLS algorithm with a matrix-wise sparsity constraint, as described in the next section.

V. Matrix-wise sparsity constraint in MNNLS

In MNNLS problems, that occur for example in alternating algorithms for NMF, sparsity is usually enforced column-wise, as follows,

$$\min_{X \geq 0} \|AX - B\|_F^2 \text{ such that } \forall j,\ \|X(:, j)\|_0 \leq k \qquad (4)$$

where $B \in \mathbb{R}^{m \times n}$ is a given data matrix, $A \in \mathbb{R}^{m \times r}$ is a given dictionary, $X \in \mathbb{R}_+^{r \times n}$ is the solution matrix we compute and $X(:, j)$ denotes the jth column of X. Problem (4) can be decomposed into n independent k-sparse NNLS problems of the form (2).

Although this formulation is intuitive (a data point is composed of at most k basis components), it can be limiting in some contexts. Notably, when sparsity varies between columns, setting the right k can be tricky. This is often the case in hyperspectral images where pixels contain different numbers of materials. To overcome this issue, we consider a matrix-wise sparsity constraint,

$$\min_{X \geq 0} \|AX - B\|_F^2 \text{ such that } \|X\|_0 \leq q, \qquad (5)$$

where q is a matrix-wise sparsity parameter, thus enforcing an *average* sparsity q/n on the columns of X.

Theoretically, we could solve problem (5) with any algorithm for k-sparse NNLS (such as greedy algorithms, or even Arbo), by vectorizing the problem. However, this would not be tractable computationally for large instances, as the resulting NNLS problem would have dimensions mn by rn. To the best of our knowledge, only one previous work [13] considered problem (5) in its matrix nature. It proposed an algorithm to solve it approximately in two steps. First, it applies a homotopy method to generate a regularization path for every column, that is, a set of solutions representing different tradeoffs between error and sparsity. This first step is only approximate because the homotopy method relies on an ℓ_1-penalized formulation, so there is no guarantee that the solutions computed correspond to the real Pareto front of the biobjective k-sparse NNLS problem. Second, it selects one solution per column to build a solution matrix X that minimizes the error while respecting a matrix-wise sparsity constraint; this is done with a greedy-like algorithm, that is very cheap but was shown to optimally solve the selection subproblem.

Here, we use a similar approach, but we replace the homotopy method of the first step by our algorithm Arbo-Pareto. We call this new approach Arbo+sel. After computing the Pareto front for every column, we build a cost matrix $\mathcal{C}$ where every entry $\mathcal{C}(k', j)$ is the error of the k'-sparse solution (with k' between k and r) of the jth column of X. Then, we select one solution per column to build an optimal solution matrix X, following the greedy-like method from [13]. In a nutshell, we consider one cursor per column of $\mathcal{C}$, and begin with all cursors at zero, meaning that the zero vector is selected for each column. Then, at each iteration, we choose one cursor to increment such that the error decrease is maximized. We stop when the sparsity target q is reached. This greedy selection is globally optimal because the squared Frobenius norm is separable by columns.

If Arbo-Pareto is run for each column with $k = 1$, then the proposed algorithm Arbo+sel provides the globally optimal solution for (5), because the selection subproblem is also solved exactly.

VI. Experiments

In this section, we study the performance of the proposed approach Arbo+sel on the unmixing of 4 hyperspectral images.

A hyperspectral image can be represented as a matrix B where each column corresponds to a pixel and each row to a different wavelength. The r columns of the dictionary A represent the spectral signature of the pure materials (also called endmembers) present in the image [2]. Given B and A, we compute X, whose columns represent the abundance of materials in each pixel. Most pixels contain only a few endmembers [14], therefore it makes sense to enforce sparsity on X. We consider the 4 widely used hyperspectral images[1] Samson, Jasper, Urban, and Cuprite, and we use as dictionaries (that is, for the matrix A) the ground truths from [15]. The characteristics of the data are summarized in Table I. The number m corresponds to the number of wavelengths, n to the number of pixels, and r to the number of endmembers in the ground truth; $B \in \mathbb{R}^{m \times n}$ and $A \in \mathbb{R}^{m \times r}$.

TABLE I
Summary of the datasets studied

Dataset	m	n	r
Samson	156	$95 \times 95 = 9025$	3
Jasper	198	$100 \times 100 = 1000$	4
Urban	162	$307 \times 307 = 94249$	6
Cuprite	188	$250 \times 191 = 47750$	12

Our method, described in section V, is noted Arbo+sel. We run Arbo-Pareto with $k = 1$ to compute the whole Pareto front, and then apply the selection strategy to build X with a matrix-wise sparsity constraint. We compare the performance of Arbo+sel with 3 other methods:

- An active-set algorithm that solves the NNLS problem with no sparsity constraint, noted AS. This is equivalent to exploring only the root node in Arbo.
- The original Arbo algorithm with a column-wise k-sparsity constraint, noted Arbo k-s.
- The algorithm from [13], that solves problem (5) approximately with a homotopy method followed by a matrix-wise selection. It is noted Ht+sel.

All algorithms are implemented in Julia. They are mono-threaded, and executed on a computer with a processor Intel Core i5-8350U @1.70GHz. Source code and scripts are provided in an online repository[2].

For every dataset, we run the 4 algorithms and measure the average column sparsity of the solutions (number of entries larger than 10^{-3} divided by the number of columns, after a normalization of the columns so that the maximum per column is 1), the relative error $\frac{\|AX-B\|_F}{\|B\|_F}$, and the running time (median over 10 runs). Jasper and Urban are processed once with all algorithms for $k = q/n = 2$, and once with Ht+sel and Arbo+sel for $q/n = 1.8$, which is not possible with other algorithms.

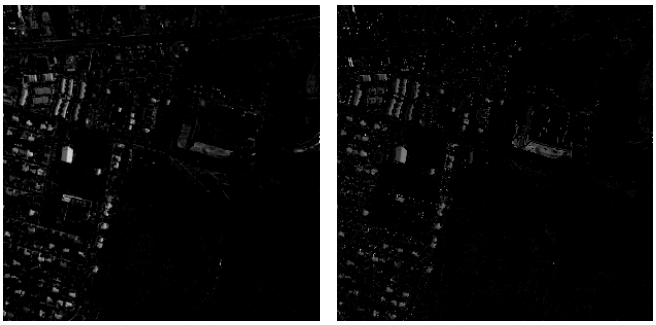
The results of the experiments for the unmixing of hyperspectral images are shown on Table II. Time is in seconds and the relative error is in percent. Without sparsity constraint, we observe that the results are already quite sparse. The column-wise k-sparse method Arbo produces solutions with an average sparsity below the target k, meaning that many columns are actually sparser than the target. Logically, all methods enforcing sparsity increase the reconstruction error. However, this loss is limited for Ht+sel and even smaller for Arbo+sel. Arbo+sel is always better than the other sparsity-enforcing methods, which is expected as it is the only one that solves problem (5) exactly. Arbo-based methods show an increase in computing time, but it is reasonable for the datasets with small r. The overcost of Arbo-sel compared to Arbo k-s

[1] Downloaded from http://lesun.weebly.com/hyperspectral-data-set.html

[2] https://gitlab.com/nnadisic/giant.jl

TABLE II
RESULTS OF THE EXPERIMENTS

		AS	Arbo k-s	Ht+sel	Arbo+sel
Samson	Time	0.10	0.19	0.25	0.43
$r = 3$	Rel error	3.30	3.40	3.30	3.30
$k = 2$	Sparsity	2.19	1.83	2.0	2.0
Jasper	Time	0.13	0.42	0.40	0.80
$r = 4$	Rel error	5.71	6.18	5.72	5.71
$k = 2$	Sparsity	2.23	1.78	2.0	2.0
Jasper	Time			0.39	0.78
$r = 4$	Rel error			5.95	5.74
$q/n = 1.8$	Sparsity			1.8	1.8
Urban	Time	2.19	13.26	6.66	29.63
$r = 6$	Rel error	7.67	8.27	7.83	7.71
$k = 2$	Sparsity	2.62	1.83	2.0	2.0
Urban	Time			6.52	29.22
$r = 6$	Rel error			8.22	7.80
$q/n = 1.8$	Sparsity			1.8	1.8
Cuprite	Time	1.53	224.23	6.82	1408.5
$r = 12$	Rel error	1.74	1.94	2.01	1.83
$k = 4$	Sparsity	6.60	3.81	4.0	4.0



(a) Active-set (no sparsity constraint) (b) Arbo with $k = 2$

(c) Ht+sel with $q/n = 1.8$ (d) Arbo+sel with $q/n = 1.8$

Fig. 3. Abundance maps of the sixth endmember from the unmixing of the Urban hyperspectral image (that is, sixth row of X reshaped) by several algorithms.

is due to the fact that, for the latter, we set $k = 1$ to generate the whole Pareto front. When r grows, the computing time grows exponentially; this is the main limitation of our method. However, in applications such as hyperspectral unmixing, r is typically small. Also, these applications are generally not in real-time, thus the computing time is not critical, and the overcost of our method can be accepted when an exact result is needed.

Some abundance maps of one endmember of the Urban image are shown on Figure 3. It corresponds to pixels from rooftops. Visually, with no sparsity constraint, the image is pretty noisy and it includes pixels from other materials. The column-wise Arbo reduces a little noise from other endmembers, but it also adds noise to the rooftop pixels (some zones are blurry and pixelated). Ht+sel removes most of the noise, but it also loses a lot of information from the rooftop pixels (some relevant zones are blacked out). Arbo+sel reduces significantly the noise while preserving most of the rooftop pixels, with a more distinct separation.

VII. CONCLUSION

We proposed Arbo-Pareto, an extension to the algorithm Arborescent to compute the Pareto front of the biobjective k-sparse NNLS problem, that is, the set of optimal k'-sparse solutions for different values of k'. We also proposed Arbo+sel, a way to leverage this extension to solve exactly multiple right-hand sides NNLS with a matrix-wise sparsity constraint, by computing a Pareto front for every column and then applying an optimal selection strategy. We showed that, for a modest increase in computing cost, Arbo+sel brings improvement over existing methods in the unmixing of hyperspectral images. It scales well and is applicable to large datasets, as long as the rank r is small.

REFERENCES

[1] D. D. Lee and H. S. Seung, "Unsupervised learning by convex and conic coding," in *Advances in neural information processing systems*, 1997, pp. 515–521.

[2] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, and J. Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 2, pp. 354–379, 2012.

[3] N. Gillis, *Nonnegative Matrix Factorization*. SIAM, 2020.

[4] C. L. Byrne, *Applied Iterative Methods*. AK Peters, 2008.

[5] P. O. Hoyer, "Non-negative sparse coding," in *IEEE Workshop on Neural Networks for Signal Processing*, 2002, pp. 557–565.

[6] J. E. Cohen and N. Gillis, "Nonnegative Low-rank Sparse Component Analysis," in *ICASSP*, 2019, pp. 8226–8230.

[7] T. T. Nguyen, J. Idier, C. Soussen, and E.-H. Djermoune, "Non-Negative Orthogonal Greedy Algorithms," *IEEE Transactions on Signal Processing*, pp. 1–16, 2019.

[8] D. Bienstock, "Computational study of a family of mixed-integer quadratic programming problems," *Mathematical programming*, vol. 74, no. 2, pp. 121–140, 1996.

[9] 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*, vol. 64, no. 6, pp. 1405–1419, 2016.

[10] R. B. Mhenni, S. Bourguignon, and J. Ninin, "Global Optimization for Sparse Solution of Least Squares Problems," *Preprint hal-02066368*, 2019.

[11] N. Nadisic, A. Vandaele, N. Gillis, and J. E. Cohen, "Exact Sparse Nonnegative Least Squares," in *ICASSP*, 2020, pp. 5395 – 5399.

[12] L. F. Portugal, J. J. Judice, and L. N. Vicente, "A comparison of block pivoting and interior-point algorithms for linear least squares problems with nonnegative variables," *Mathematics of Computation*, vol. 63, no. 208, pp. 625–643, 1994.

[13] N. Nadisic, A. Vandaele, and N. Gillis, "A homotopy-based algorithm for sparse multiple right-hand sides nonnegative least squares," *Preprint arXiv:2011.11066*, 2020.

[14] W.-K. Ma, J. M. Bioucas-Dias, T.-H. Chan, N. Gillis, P. Gader, A. J. Plaza, A. Ambikapathi, and C.-Y. Chi, "A signal processing perspective on hyperspectral unmixing: Insights from remote sensing," *IEEE Signal Processing Magazine*, vol. 31, no. 1, pp. 67–81, 2013.

[15] F. Zhu, "Hyperspectral unmixing: ground truth labeling, datasets, benchmark performances and survey," *Preprint arXiv:1708.05125*, 2017.