Optimization and Engineering (2021) 22:2505–2535
https://doi.org/10.1007/s11081-021-09676-2

RESEARCH ARTICLE

Provably optimal sparse solutions to overdetermined linear systems with non-negativity constraints in a least-squares sense by implicit enumeration

Fatih S. Aktaş[1] · Ömer Ekmekcioglu[1] · Mustafa Ç. Pinar[1]

Received: 15 January 2021 / Revised: 10 August 2021 / Accepted: 10 August 2021 /
Published online: 28 August 2021

Abstract

Computing sparse solutions to overdetermined linear systems is a ubiquitous problem in several fields such as regression analysis, signal and image processing, information theory and machine learning. Additional non-negativity constraints in the solution are useful for interpretability. Most of the previous research efforts aimed at approximating the sparsity constrained linear least squares problem, and/or finding local solutions by means of descent algorithms. The objective of the present paper is to report on an efficient and modular implicit enumeration algorithm to find provably optimal solutions to the NP-hard problem of sparsity-constrained non-negative least squares. We focus on the problem where the system is assumed to be overdetermined where the matrix has full column rank. Numerical results with real test data as well as comparisons of competing methods and an application to hyperspectral imaging are reported. Finally, we present a Python library implementation of our algorithm.

Keywords Inverse problems · Sparse approximation · Overdetermined linear systems · Sparse solutions · Branch and bound · Implicit enumeration · Non-negative least squares

Mathematics Subject Classification 65F20 · 65F22 · 65K05 · 90C26

Submitted to the editors DATE.

✉ Mustafa Ç. Pinar
mustafap@bilkent.edu.tr

Fatih S. Aktaş
selim.aktas@ug.bilkent.edu.tr

Ömer Ekmekcioglu
omer.ekmekcioglu@bilkent.edu.tr

[1] Department of Industrial Engineering, Bilkent University, Ankara, Turkey

1 Introduction

One of the main challenges in inverse problems research is finding a sparse representation of data b in a dictionary A. The problem is therefore equivalent to finding a solution to the system $Ax = b$ with the smallest number of non-zero components. In sparse approximation, due to the presence of noise and model errors, one usually seeks a solution minimizing a data misfit measure commonly based on the Euclidean norm. In the literature, the non-negatively constrained sparse approximation problem is encountered in many applications such as best subset selection in regression analysis, signal and image processing, hyperspectral imaging, sparse recovery and machine learning. The formulation of the problem (that we refer to as SNNLS) is as follows

$$\begin{aligned} \min_{x \in R^n} \ & \|Ax - b\|_2^2 \\ \text{s.t.} \ & x \geq 0 \\ & \|x\|_0 \leq s \end{aligned} \tag{1.1}$$

where the matrix $A \in \mathbb{R}^{m \times n}$ with $m > n$, $b \in \mathbb{R}^n$, $\|v\|_0$ is the ℓ_0-norm of vector v, which counts the number of nonzero elements of v, and s is called the sparsity level. SNNLS is NP-hard due to the combinatorial nature of the ℓ_0-norm Natarajan (1995). The sparsity constraint is widely used in the compressed sensing literature Vidyasagar (2019). Beck and Eldar (2013) provide a good discussion of the problem. They review theoretical properties that enable the computation of points satisfying necessary conditions for global optimality (L-Stationary points and/or Partial Coordinatewise optimal points). However, their algorithms do not guarantee optimal solutions. Such methods which at best compute a local optimum without guarantee of global optimality generally appear in the compressed sensing literature where sparsity is induced by ℓ_1 regularization, iterative hard thresholding or basis pursuit methods on least squares estimation problems Slawski and Hein (2013). On the other hand, an optimal solution to the SNNLS problem could be obtained using Mixed Integer Programming (MIP); see e.g. Bertsimas et al. (2016), Bourguignon et al. (2016). However, off-the-shelf solvers might not always perform well with Quadratic Mixed Integer Programs (QMIP) as opposed to their significantly better performance with linear mixed integer programs (MIP)s. Furthermore, the MIP formulations may fail to return an optimal sparse solution because of numerical difficulties that arise from the utilization of a "big-M" constant in implementing the sparsity constraint Donne et al. (2020).

The literature on solution methods for the SNLSS problem is divided into two categories. The first category consists of the sub-optimal (local) solution methods mentioned above, using thresholding or regularization techniques. These methods are more commonly used in compressed sensing problems as explained in Slawski and Hein (2011), Slawski and Hein (2013) due to the properties inherent in those problems. The second category of research, which is encountered less commonly in the literature, is concerned with finding optimal solutions. Here, MIP solutions arise as the most obvious reformulation and solution method for the problem. However, these solutions are not always scalable as we demonstrate in

the present paper. Related problems are also addressed by Atamturk and Gomez (2020) using MIP formulations. Analysis of similar sparsity constrained problems using gauge function is given in Chandrasekaran et al. (2012) which also proposes a MIP-based solution. In a recent study Nadisic et al. (2020), it was also proposed to adapt the Branch and Bound algorithm to solve this problem.

The thesis of the present paper is that one can solve to optimality a large number of instances by using a judicious enumeration strategy. We propose a branch and bound framework along with a novel implicit enumeration rule for the SNNLS problems to find optimal sparse support. This allows our algorithm to find optimal solutions to the combinatorial SNNLS problem efficiently. Furthermore, we present a Python implementation of our algorithm, which is referred to as NNLSSPAR, that is available as a library for computing optimal sparse solutions to the SNNLS problem.

The rest of the paper is organized as follows. Section 2 briefly outlines the core ingredients used in the algorithm. Section 3 describes the proposed algorithm and explains the NNLSSPAR library. Section 4 provides a summary of numerical results obtained using NNLSSPAR. Section 5 reports an application of the algorithm on a hyperspectral imaging application to demonstrate the effectiveness of the algorithm even further. Section 6 describes an application of the algorithm to a sparse deconvolution problem.

2 Background

2.1 Problem description

In a statistical context, the SNNLS problem occurs naturally when one solves an estimation problem related to the linear model:

$$Y = \beta_0 + \sum_{i=1}^{k} \beta_i X_i + \epsilon, \tag{2.1}$$

where Y is the variable to be predicted, and $X_1 X_2, \ldots, X_k$ are the predictors and the residuals ϵ have zero mean and are independently sampled from a Gaussian distribution with finite variance. Additionally, if X_i's are known to have non-negative values, as it may be the case in the applications of time series or pixel intensities, it makes sense to add non-negativity constraints for the β_i's. As a result, the values of coefficients β are calculated using problem 1.1.

In statistical applications, the non-negativity constraint is shown to be very useful for regularization purposes since it allows the solution of a regular least squares problem to enjoy two desirable properties: sparsity and resistance to overfitting Slawski and Hein (2013). This approach is further extended to applications such as hyperspectral imaging with the addition of the sparsity constraint. These features motivate the search for efficiently computable solutions to the SNNLS problem.

2.2 Branch & Bound method

The Branch and Bound (B & B) technique is a mainstay of the integer programming literature. For the integer programming case, the B & B algorithm creates a tree structure to solve the problem optimally. The linear relaxation of the optimization problem is solved at the root node of the tree, and in each branch (possibly adding some cutting planes), a subproblem is further investigated. This investigation appears e.g., in the form of fixing the value of a variable and searching for the best integer solution under this restriction. Several elimination rules are implemented based on the bound information gathered from branches. Integer programming technology has indeed reached an advanced stage where several commercial and academic solvers compete with one another, using very sophisticated and fast implementations.

The situation is different in the case of SNNLS where research on B&B algorithms is yet at an early stage. In SNNLS, we concentrate on the combinatorial nature of the sparsity constraint where the focus is on the inclusion or exclusion of variables in the solution. This allows our algorithm to find a sparse support for the problem.

2.3 QR factorization

The following properties of QR factorization are well-known:

- The Euclidean norm of a vector is preserved under orthogonal transforms: $\|Qr\|_2^2 = r^T Q^T Q r = r^T r = \|r\|_2^2$ since $Q^T Q = I$. As a result, we have $\min_x \|Ax - b\|_2^2 = \min_x \left\|(Q^T A)x - (Q^T b)\right\|_2^2$ and

$$\left\| \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ 0 & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{n,n} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} - \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \\ c_{n+1} \\ \vdots \\ c_m \end{bmatrix} \right\|_2^2$$

- Minimum norm is achieved by solving the system:

$$\begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ 0 & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{n,n} \end{pmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

- Norm of the solution is given as

$$\left\| \begin{pmatrix} c_{n+1} & c_{n+2} & \cdots & c_m \end{pmatrix} \right\|_2^2.$$

As an illustration, let A be the data matrix, and assume that a best fitting subset with two elements is desired:

$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,1} & a_{4,2} & a_{4,3} \end{pmatrix}.$$

Let $A = QR$ be a QR decomposition of matrix A

$$Q_A = \begin{pmatrix} q_{1,1} & q_{1,2} & q_{1,3} & q_{1,4} \\ q_{2,1} & q_{2,2} & q_{2,3} & q_{2,4} \\ q_{3,1} & q_{3,2} & q_{3,3} & q_{3,4} \\ q_{4,1} & q_{4,2} & q_{4,3} & q_{4,4} \end{pmatrix} R_A = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ 0 & r_{2,2} & r_{2,3} \\ 0 & 0 & r_{3,3} \\ 0 & 0 & 0 \end{pmatrix}.$$

Let B be the matrix composed of the last two columns of A

$$B = \begin{pmatrix} a_{1,2} & a_{1,3} \\ a_{2,2} & a_{2,3} \\ a_{3,2} & a_{3,3} \\ a_{4,2} & a_{4,3} \end{pmatrix}.$$

Instead of performing a QR factorization from scratch, columns of R can be deleted:

$$R_B = \begin{pmatrix} r_{1,2} & r_{1,3} \\ r_{2,2} & r_{2,3} \\ 0 & r_{3,3} \\ 0 & 0 \end{pmatrix}.$$

Now the system is reduced to a 3×2 matrix instead of a 4×2 matrix

$$Q_B : \begin{pmatrix} p_{1,1} & p_{1,2} & p_{1,3} \\ p_{2,1} & p_{2,2} & p_{2,3} \\ p_{3,1} & p_{3,2} & p_{3,3} \end{pmatrix} R_B : \begin{pmatrix} s_{1,2} & s_{1,3} \\ 0 & s_{2,3} \\ 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

Based on this observation we use a pre-processing step on the initial system, and thus our algorithm can solve much larger problems. This exploitation of orthogonal transforms is especially useful when the design matrix A has many rows.

2.4 Recalls on least squares

In each node of the search tree of the algorithm, a non-negative least squares subproblem is solved to evaluate the objective value to find a lower bound for the current solution. Lawson & Hanson's active set algorithm is used for that purpose

Lawson and Hanson (1995). The KKT optimality conditions for the non-negative least squares subproblem are as follows

$$A^T Ax - A^T b \geq 0 \tag{2.2}$$

$$\lambda^T x = 0 \tag{2.3}$$

$$\lambda, x \geq 0. \tag{2.4}$$

Optimality conditions show that the problem is to find a non-negative x, for which the so-called "normal equations" are satisfied. Let C denote the set of indices for which x is allowed to have non-negative values. Hence, one has

$$x = \begin{cases} x_j \geq 0 & j \in C \\ x_j = 0 & j \notin C. \end{cases} \tag{2.5}$$

Then, the problem is reduced to finding C that satisfies the following system

$$A_C^T A_C x_C = A_C^T b \tag{2.6}$$

$$A^T Ax - A^T b \geq 0, \tag{2.7}$$

where A_C denotes the sub-matrix whose columns correspond to the columns of A corresponding to the indices in C, and $x_C \in \mathbb{R}^{|C|}$ is a vector consisting of values of x corresponding to indices in C.

Some remarks concerning the impact of non-negativity constraints on the sparsity level are in order here since optimality conditions indicate that non-negativity constraints might induce some sparsity. If the true x is non-negative and sufficiently sparse, or there are sufficiently many observations, and data matrix A satisfies certain conditions like Gaussian Windows, then the sparsity constraint may be removed from the problem Eftekhari et al. (2021). However, in this paper, we develop an algorithm for arbitrary matrices and sparsity values. Furthermore, in the branch and bound search procedure, variables that are set to zero by the solution of the unconstrained problem will be given lower priority in the branching step because these variables will not improve the search. The children nodes constructed by branching on variables that are already at zero will have the same residual as the mother node, which makes it more difficult to understand the effect of the variables, therefore slowing down the search. Moreover, for certain types of problems such as the sparse deconvolution application discussed in Sect. 6, the unconstrained solution at node 0 sometimes sets the true non-zero variables, variables that are in support of optimal s sparse x, to zero. This is mainly caused by noise and the presence of correlated variables in the design matrix A. Then, the variables in the support of optimal x are not given positive values by non-negative least squares solution when solving a subproblem until the enumeration algorithm progresses and some of the correlated variables, with the variables in the optimal

support, are set to zero. Following this step, a non-negative least squares solution assigns positive values to the variables that are in the optimal support.

2.5 Accuracy and precision

The condition number of matrix A is given by:

$$K(A) = \frac{\sigma_1}{\sigma_n}$$

where σ_1 is the largest singular value, and σ_n is the smallest singular value, and the least squares problem represented on a finite precision computer is given by

$$\min_x \|(A + \Delta A)x - (b + \Delta b)\|_2^2$$

where $\|\Delta A\|_2^2 \approx u\|A\|_2^2$, $\|\Delta b\|_2^2 \approx u\|b\|_2^2$ represent the errors incurred in registering matrix A and vector b to a computer, u is unit round-off and $\rho_{ls} = \left\|Ax_{LS} - b\right\|_2$ the norm of the residual. Then the relative error is approximately as given in Golub and Loan (1996).

$$\frac{\left\|\tilde{x_{LS}} - x_{LS}\right\|}{\left\|x_{LS}\right\|} \approx u(K(A) + \rho_{LS}K(A)^2). \tag{2.8}$$

However, since we are only using a subset of columns of A, relative error for the final solution of non-negative least squares is given instead by

$$\frac{\left\|\tilde{x_{NNLS}} - x_{NNLS}\right\|}{\left\|x_{NNLS}\right\|} \approx u(K(A_C) + \rho_{NNLS}K(A_C)^2),$$

which is obtained by plugging in the smaller matrix to the formula given in (2.8).

3 Description of algorithm and NNLSSPAR library

3.1 Algorithm description

The cardinality constraint is the central challenge because it lies at the heart of the NP-Hardness result Natarajan (1995). To handle the cardinality constraint, our algorithm uses a branch and bound framework. The original problem is reduced to small sized subproblems in each iteration, and variables are selected according to a particular rule to ensure efficiency. As a result, subproblems solved at each step of the algorithm are reduced to the standard least squares problem with non-negativity constraints. Therefore, the subproblem solved at each step is the standard non-negative least-squares problem:

$$\begin{aligned} \min_{\hat{x}} \ & \left\| \hat{A}\hat{x} - \hat{b} \right\|_2^2 \\ \text{s.t.} \ & \hat{x} \geq 0 \end{aligned} \tag{3.1}$$

where $\hat{x} \in \mathbb{R}^k, \hat{b} \in \mathbb{R}^n, \hat{A} \in \mathbb{R}^{n \times k}$.[1] Since the number of potential subproblems grows exponentially, subset selection is the most critical part of the algorithm. NNLSSPAR performs implicit enumeration to generate unique subsets and uses the solution of subproblems to evaluate the quality of a subset. The information about the quality of a subset is instrumental for branching on high quality subsets for fast convergence. The enumeration rule described in Sect. 4 utilizes the so-called "scaled feature impact in the least squares". NNLSSPAR can be deployed in any problem instance without a restriction. However, as linear dependencies between independent variables and the dependent variable get more pronounced, the search may move closer to performing an exhaustive search that will require $\binom{n}{s}$ trials. On the other hand, when the solution is "simple" enough, branching only s times may be enough to obtain it. The motivation behind this approach comes from the recent advances in best subset selection algorithms and the mixed integer programming approaches presented in e.g., Bertsimas et al. (2016). Furthermore, we do not rely on the off-the-shelf solvers for the computation of the sparse NNLS problem. We also provide an effective branching method.

In a nutshell, the algorithm of NNLSSPAR can be described as follows. Each node is searched according to a best first search with respect to the residual of the system. Given that the algorithm is at any node of the search tree, the first step is to check if the current point is feasible. If the answer is affirmative, it is optimal because of the search rule. If the current point is infeasible, then the algorithm chooses a variable to branch on according to an enumeration rule and creates two new nodes, one where the chosen variable is in the solution and the second where the variable is erased from the problem. These two new nodes are added to the search tree. Then, the algorithm goes back to the first step and explores the other nodes.

3.2 Description of NNLSSPAR library

NNLSSPAR requires as inputs matrix A storing the values of independent variables, vector b storing the values of the dependent variable, and sparsity level s, (the maximum number of nonzero elements allowed to be in the solution vector x).

The flow of the algorithm is presented in Fig. 1. Two disjoint sets, P and C, are kept. Initially, $P = \{1, 2, .., n\}$ where n is the number of independent variables and

[1] As alluded to in Sect. 2, in machine learning and statistics an alternative notation is commonly used:

$$\begin{aligned} \min_{\beta} \ & \|y - X\beta\|_2^2 \\ \text{s.t.} \ & \beta \geq 0 \end{aligned} \tag{3.2}$$

where $\beta \in \mathbb{R}^k, y \in \mathbb{R}^n$, $X \in \mathbb{R}^{n \times k}$ and βs are referred to as coefficients. Unless otherwise indicated, we prefer to stick to the usual A, b-notation of linear systems of equations.

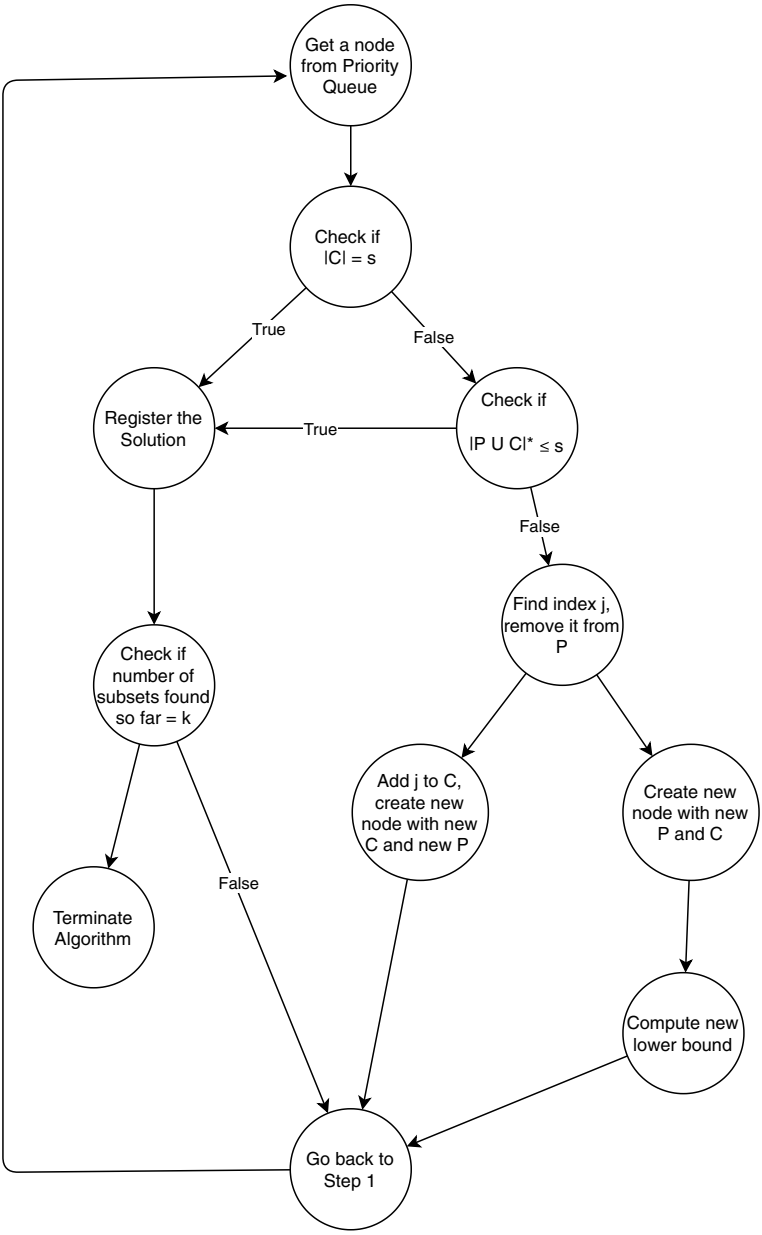

Fig. 1 Generic flow of the algorithm; *Cardinality of nonzero elements in the solution with variables whose indices are given by P and C is checked

$C = \emptyset$. P holds the indices of the independent variables that are candidates to be included in the solution. C holds the indices of the variables that are chosen to be in the solution. In each step of the algorithm we enumerate implicitly, and remove a

variable from P and create two new nodes where in the first one the removed variable is in C and in the other it is not. Implicit enumeration will always find unique subsets so, P and C together identify a unique node in the search graph. The search tree is constructed by priority queue for best first search. In the flow of the algorithm, the fact that the sum of squared errors is monotonically decreasing with the number of elements included in the prediction is utilized. Hence, the algorithm will always pick at most s many variables. In other words, $|C| \leq s$ holds for any C in the search tree.

Algorithm 3.1 Flow of NNLSSPAR Computations

Input: Input: Independent variable matrix A, dependent variable vector b, sparsity level s, number of subsets desired k.

Output: Output: Best subset indices, solution values, residual of the solution.

Step 0: (Preprocessing) Do a QR Decomposition on the linear system to shrink the size of the problem. $A = QR$ and $d = Q^T b$

Step 1: (Initialization) Begin with node 0, solve the non-negative least squares problem with all independent variables, Set $P = \{1, 2..., n\}$, $C = \varnothing$, lower bound = inf and go to Step 2.

Step 2: (Feasibility condition 1) Check if the node has a feasible solution (if number of variables in C is equal to s i.e., if $|C| = s$ holds, then it is feasible), if affirmative, go to Step 7; otherwise go to Step 3.

Step 3: (Feasibility condition 2) Check if $P \cup C$ is feasible (number of nonzero elements in the solution of system support $P \cup C$ is less than s i.e., if $||x_{P \cup C}||_0 \leq s$ holds, then it is feasible), if affirmative, go to step 7 otherwise go to Step 4.

Step 4: (Branching) Choose a variable from P according to enumeration rule and delete it from P. Create two new nodes, one where the variable is forced to be in solution (the variable is added to C, call it C_1), the other where it is removed from the problem (the variable is not added to C, call it C_2). Go to step 5.

Step 5: Compute lower bound for non-negative least squares problem for the new node. (Solve non-negative least squares problem with only support $P \cup C_2$ of the matrix R and vector d since $P \cup C_1$ is the same as parent node). Add the two new child nodes to the current node. Go to Step 6.

Step 6: (Searching) Choose the node with the smallest residual. Go to step 2.

Step 7: (Termination Condition) Optimal solution is C if prior step is step 2, $P \cup C$ if prior step is step 3. Solution is optimal because best first search is done. Terminate the algorithm if number of subsets found is equal to k, otherwise go back to Step 6.

Additionally, NNLSSPAR provides a routine to find optimal subsets of all sizes up to the specified sparsity level s. That routine is considerably faster than using

a brute force approach and solving s separate problems. The use of this option is explained in the User's Guide to NNLSSPAR Aktaş et al. (2020).

The user can specify the parameter C as input to the algorithm to force a variable to be in the solution set. Then, NNLSSPAR will compute an optimal solution where the given index/indices are in the solution although these variables are not guaranteed to have a positive value in the solution.

NNLSSPAR employs nnls, the non-negative least squares function of the scipy library. The nnls algorithm is based on the active set method described by Lawson and Hanson (1995); Virtanen et al. (2020). NNLSSPAR also uses the QR routine, i.e., QR factorization, of the same library. This routine implements the orthogonalization method for computing an accurate least squares solution. However, NNLSSPAR utilizes QR factorization to shrink the size of the system. Moreover, the algorithm does not report the actual residual of the system but rather only the residual of the reduced system after orthogonal transforms. Hence, the true squared residuals of the systems are obtained by residual squared reported plus the initial residual incurred after the orthogonal transform.

3.3 Relation to ℓ_2 norm regularization (ridge)

A modified version of the problem we study 1.1 is of special interest and is given in the following form

$$\begin{aligned} \min_{x \in R^n} \ & \|Ax - b\|_2^2 + \mu \|x\|_2^2 \\ \text{s.t.} \ & x \geq 0 \\ & \|x\|_0 \leq s, \end{aligned} \tag{3.3}$$

where parameter μ controls the ℓ_2 norm penalty term. This is commonly known as Tikhonov Regularization Tikhonov et al. (1995) in the literature. The ℓ_2 norm penalty is often used to account for noise in the data matrix Ghaoui and Lebret (1997) or when the data matrix is ill conditioned Engl et al. (2000). Although we do not address this problem, it can still be solved easily with NNLSSPAR by changing the formulation to transform it into the format given in 1.1. Let A,b and μ be the original problem parameters. Then we would treat the following problem with our algorithm:

$$\hat{A} = \begin{bmatrix} A \\ \sqrt{\mu} I_n \end{bmatrix}, \ \hat{b} = \begin{bmatrix} b \\ \mathbf{0_n} \end{bmatrix},$$

where $I_n \in \mathbb{R}^{n \times n}$ is identity matrix, $\mathbf{0_n} \in \mathbb{R}^{n \times 1}$ is a vector of zeros.

3.4 Novel heuristic enumeration rule

The enumeration rule of NNLSSPAR is referred to as "scaled feature impact in the least squares". It approximates the scaled non-negative least squares value without actually scaling the data and solving a non-negative least squares problem. The algorithm solves the following problem at each node

$$j = \arg\max_{i \in P} \left| (|\bar{x}_i| + \sigma_{x_i}) \hat{x}_i \right|,$$

where $|\bar{x}_i|$ denotes the absolute value of the mean of i'th variable, σ_{x_i} denotes the standard deviation of i'th variable and $\hat{x}_i$ is the value of i'th variable in the non-negative least squares solution. The main novel idea behind NNLSSPAR is this clever enumeration rule which chooses the variable that has the highest impact in the least squares problem at each node. Furthermore, at any given node, the non-negative least squares problem is solved to find lower bounds. Therefore, values are known and the cost of applying our impact rule requires $2k$ flops and $O(k)$ for linear search over the list of size k. This is computationally cheap since $k = |P|$. As described in section 2.2, the decision to keep the i'th variable is taken with the Branch and Bound algorithm. This creates two sub-problems depending on whether the index i is included in the support or not. To ensure that the problem remains tractable, a selection rule is required. Otherwise, the problem would simply become a naive enumeration algorithm. The selection of the i'th variable is performed in an efficient manner with the proposed method using scaled feature impact.

Although it is a heuristic search method, it is very fast, and in general able to identify the important variables. Moreover, being able to identify variables with a high impact on the residual leads to quick elimination of nodes due to the high residuals at the nodes without those variables. As a result, the search algorithm quickly finds the optimal solution, an observation that is substantiated in subsequent sections with numerical results.

In Atamturk and Gomez (2020), a preprocessing method to eliminate some variables from the problem prior to the enumeration step is suggested. However, this method relies on a ℓ_2 norm regularization term. The problem we have focused on does not have an explicit ℓ_2 regularization term. Moreover, since we have a non-negativity constraint, the optimal solution is not given in the closed form. Hence, the screening rules described do not apply to our problem. A small problem instance that demonstrates this phenomenon can be given as follows;

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 1 & -1 \end{pmatrix}, b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, s = 2, \mu = 1$$

where μ corresponds to the ℓ_2 regularization term. The preprocessing method suggests that variable x_3 should be in the solution yet the non-negative least squares solution to the problem returns $x = [\frac{2}{3}, \frac{2}{3}, 0]^T$, and with regularization $x = [\frac{1}{2}, \frac{1}{2}, 0]^T$ which is 2-sparse and therefore optimal. As seen, the framework in Atamturk and Gomez (2020) is not directly applicable to our problem.

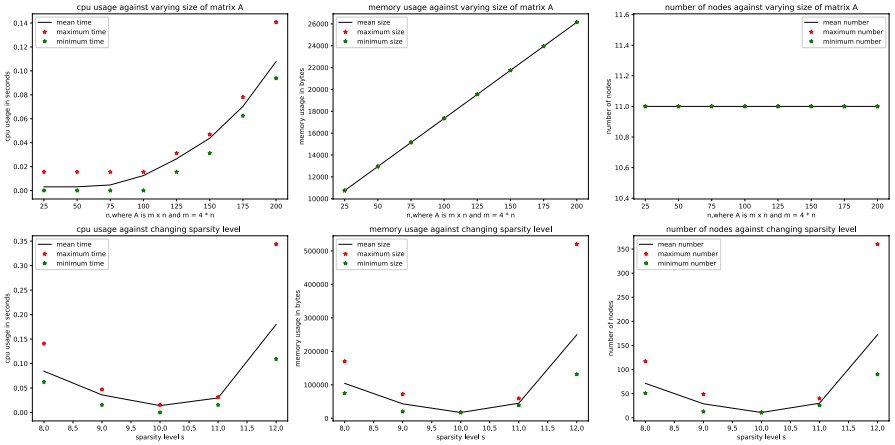

Fig. 2 Test cases

4 Tests and results

We first tested the algorithm with some randomly generated data. We solved problems of different sizes and different sparsity levels and collected different performance measures: CPU time, memory use, and the number of nodes searched by the algorithm. Each case is solved ten times for the reliability of the results. All experiments were performed on a computer with i7 processor with clock frequency 2.6 GHz and 16 GB of RAM.

The results summarized in Fig. 2 suggest that the number of observations in the matrix A, i.e., m, has no impact on the number of nodes examined. The number of nodes searched is determined by dominant elements of vector x and the number of independent variables n. Also, CPU and memory usage increase as the size of matrix A increases because it requires more space to register matrix A and longer CPU time to calculate the non-negative least squares solution. When the sparsity level changes, the algorithm requires the least amount of node search and CPU time when the number of "dominant" elements of x is equal to sparsity level s. CPU usage increases not as a result of increased difficulty in the solution of the subproblem but because of the number of nodes searched. Similarly, since a larger number of nodes are checked and a larger number of subproblems are solved the memory usage also increases.

We also tested the algorithm on some of the publicly available real data sets. For all real data sets, the design matrix A is centered and scaled so that each column of A has zero mean and unit standard deviation.

Figure 3 shows performance measures obtained using the OZONE data set in Breiman and Friedma (1985). Since the CVXPY values dominate others we also separately plotted the performance of methods excluding CVXPY (Fig. 4) for clarification. This data set includes ozone concentration which is the dependent variable, in Upland CA, east of Los Angeles and 8 independent variables, which are meteorological measurements based on 330 observations in 1976. In addition to the 8 independent variables, we created 36 artificial variables from those

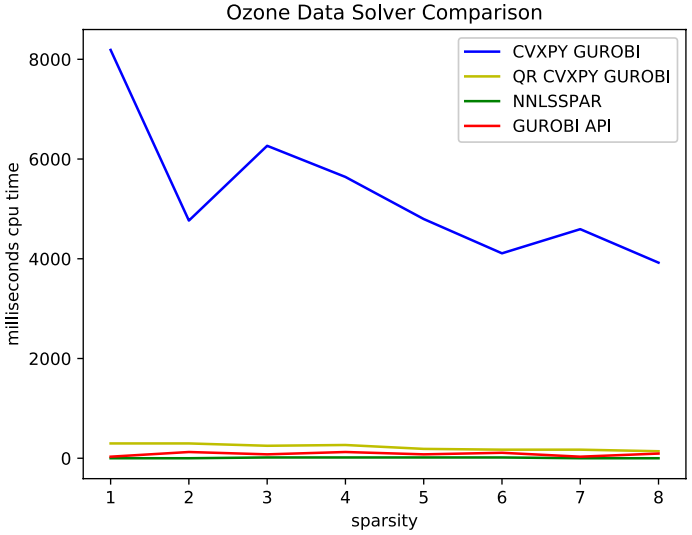

Fig. 3 Ozone data set solver comparisons

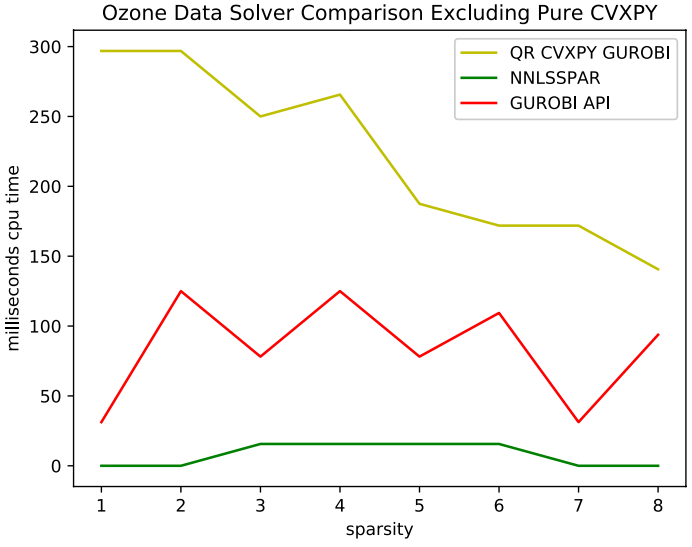

Fig. 4 Ozone data set solver comparison, CVXPY is excluded

8 variables. The first 8 original variables are $= X_1 X_2 .. X_8$. The artificially created variables are interactions in the order $= X_1^2, X_1 X_2, X_2^2, X_1 X_3, X_2 X_3, X_3^2, \ldots, X_8^2$ as in Miller's subset selection experiments Miller (2002). We compared GUROBI version 9.0.2 MIP solver with NNLSSPAR Gurobi (2020). The table below shows the elapsed CPU and wall clock time in seconds for different levels of sparsity.

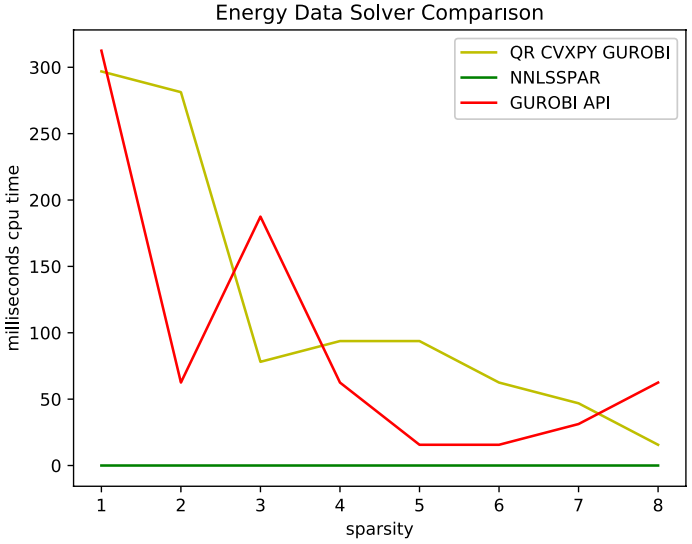

Fig. 5 Energy data set solver comparisons

GUROBI refers to the performance of the integer programming algorithm when the full system is given. QR GUROBI refers to the performance of the integer programming algorithm after the system is shrunk using QR decomposition. The results show that QR decomposition improves the performance of the MIP solver GUROBI, although QR overhead is ignored in the values given in the figure. For the other data sets, a pure MIP algorithm was not used as MIP algorithms dramatically slow down when data sets have many observations.

Energy data set by Candanedo et al. (2017). This data set was used for predicting energy consumption in houses. There are 19,735 observations in total although not all of the observations are from unique houses. Independent variables are temperature and humidity levels in different parts of a house, some other meteorological measurements and a random variable to detect the non-predictive variables. Results are presented in Fig. 5.

Conduct data set in Hamidieh (2018). This data set was used to predict the critical temperature of a superconductor. Predictor variables consist of physical characteristics of the material used in a superconductor (81 independent variables). The data set includes observations from 21,263 superconductors. Results on this data set is summarized in Fig. 6.

Online data set by Fernandes and Cortez (2015). This data set contains popularity levels of online news that was published on Mashable. There are 39,644 observations and 58 independent variables which characterize the news content. However, there are categorical features and they result in rank deficiency of matrix A. Hence, we removed two features from the data set that identified whether a news item was published on Saturday and Sunday. Results for this data set are reported in Fig. 7.

The MIP solver for the last data set performs slower compared to the other two data sets because CPLEX was used as the solver for CVXPY Cplex (2009);

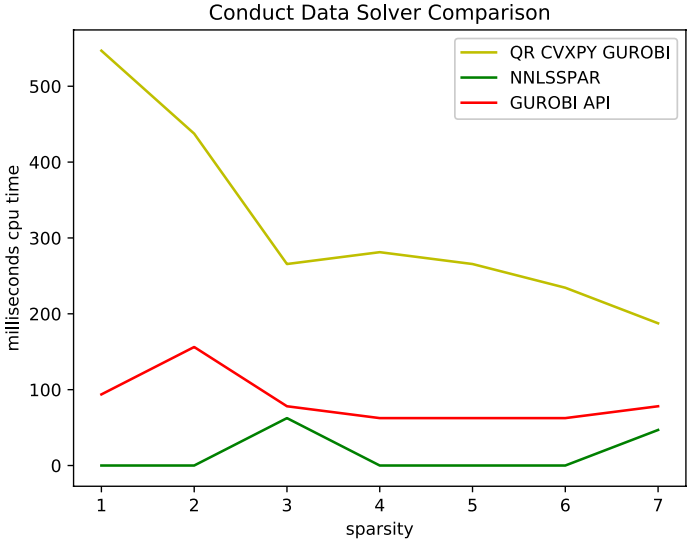

Fig. 6 Conduct data set solver comparisons

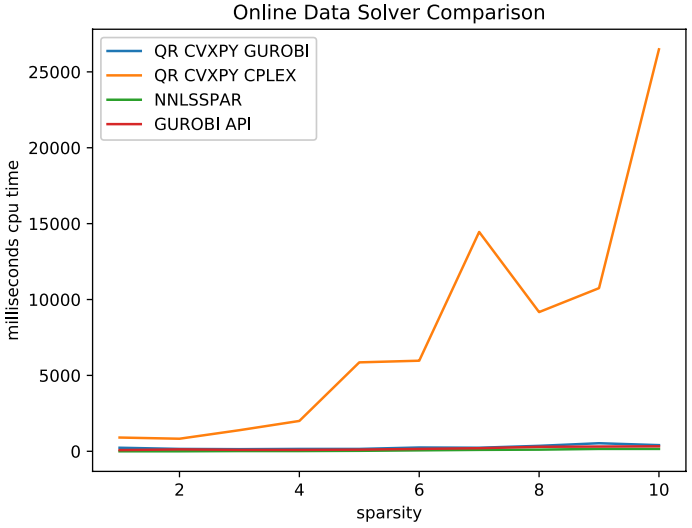

Fig. 7 Online data set solver comparisons

Diamonda and Boyd (2016). For the other data sets, GUROBI was used as the solver because in our experience it was the fastest among CVXPY solvers that support Quadratic Integer Programming. GUROBI was not able to find the correct solution on this data set. Therefore we reported CPLEX results. Although it should be noted that this data set is more ill-conditioned than the others due to a large condition number of A and large residual of the system.

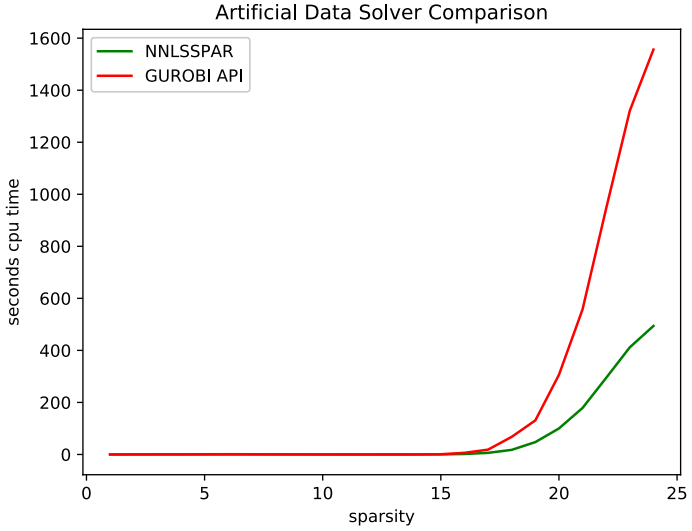

Fig. 8 Artificially difficult problem solver comparisons

In the previous data sets, the non-negative sparse least squares problem was not challenging given that the CPU times of the algorithms are in milliseconds. Hence, to see how well the algorithms perform, in particular, to compare GUROBI and NNLSSPAR, we created an artificial problem where $A \in \mathbb{R}^{400\times 40}$, x has 12 dominant values and the rest of the entries are equal to 1, and b is created as $Ax + \epsilon$ where ϵ denotes Gaussian noise.

Figure 8 shows that NNLSSPAR seems to be faster especially as sparsity reaches higher levels.

The results reported in the figures above do not include the initial QR step to shrink the system. The next table shows the QR overhead in seconds for each data set:

Time\Data	Ozone	Energy	Conduct	Online
CPU	0.0000	0.0625	0.2500	0.3125
Real	0.0020	0.0160	0.0608	0.0848

Bertsimas et al. (2016) propose a warm start procedure using Iterative Hard Thresholding. This procedure helps Mixed Integer Programming solvers find an optimal solution faster. Hence, Iterative Hard Thresholding with projection to non-negative orthant was used to find an initial good approximation of an optimal point for MIP solvers. Figure 9 shows the performance figures of MIP solver Gurobi (2020) with warm start when we use the heuristic point as warm start and optimal point as a warm start, respectively. Here, the time measurements do not include the time to find the (optimal or heuristic) warm start point. We also added cold start results for comparison.

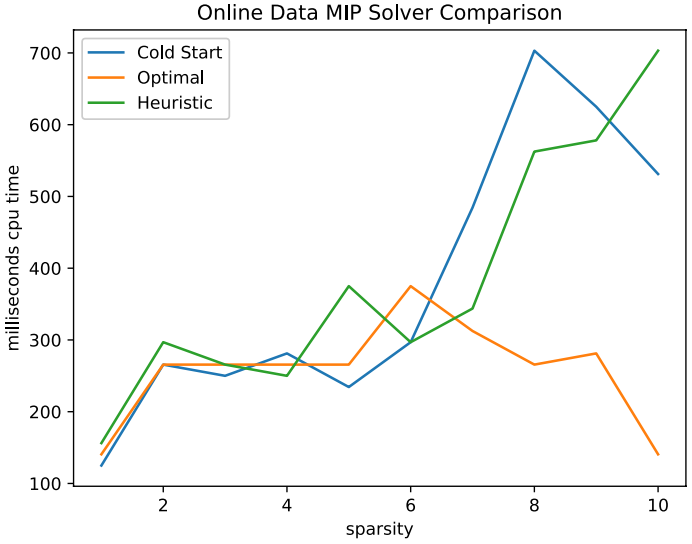

Fig. 9 Online data set MIP start comparisons

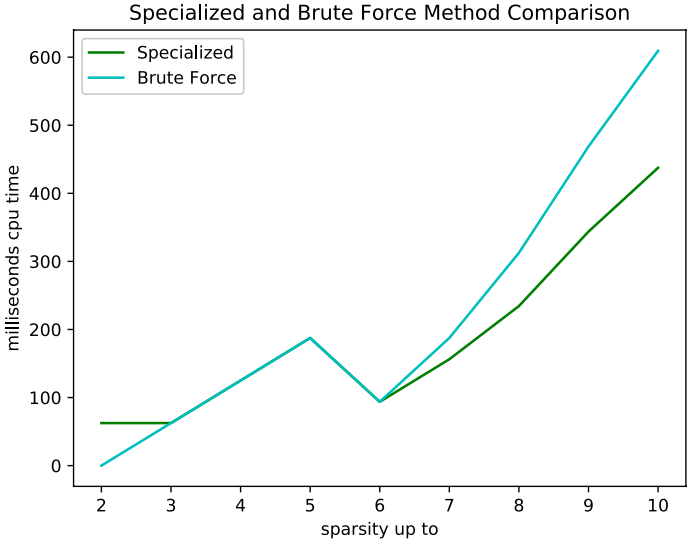

Fig. 10 Online data set special and brute force method comparisons

MIP solvers seem to be so fast that even giving an optimal initial point does not change the CPU time when the sparsity level is low. However, as the sparsity level increases, CPU usages of different options varies remarkably. Moreover, giving the heuristic solution as the initial input, in our case, using non-negative IHT, did not speed up the algorithm and overall results for both cases are very similar.

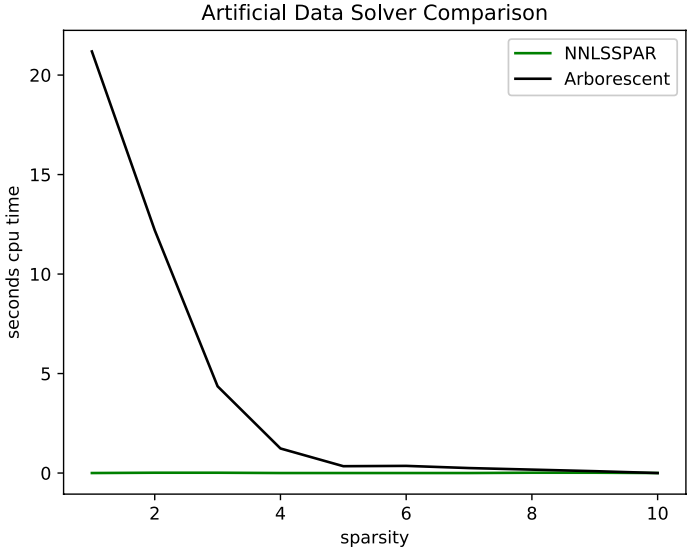

Fig. 11 Arborescent and NNLSSPAR

In addition, a comparison of the brute force approach (when the algorithm is called once for each sparsity level up to the upper bound s) and the specialized function to all subset problems (a routine to find optimal subsets of all sizes up to the sparsity level s) on the Online data set is shown in Fig. 10.

Comparison with Arborescent One of the main alternatives to our algorithm is another enumeration algorithm called Arborescent Nadisic et al. (2020). That approach uses a branch and bound idea to solve the sparse NNLS problem. However, there are significant differences between our method and Arborescent. The first difference is that we use a forward selection approach in the branching whereas they employ a backward elimination approach. Due to the sparsity assumption, our convergence is theoretically faster due to the reduced number of branching our algorithm undertakes. A second major difference is that our algorithm branches based on our decision rule which makes the algorithm search for the strong candidate nodes first. Arborescent on the other hand uses lexicographical search after sorting the variables according to their magnitude in the initial non-negative least squares problem. The ambiguity in their application in terms of the node generation and node priorities makes the comparison hard. Thirdly, Arborescent uses depth-first-search and NNLSSPAR proceeds by best-first-search. Furthermore, Arborescent's branching scheme enumerates over all possible variables at one node, and consequently creates redundant nodes. However, NNLSSPAR uses implicit enumeration and hence enumerates one variable at a time and so all nodes are unique. Their implementation of the algorithm is based on C++. In order to facilitate a meaningful and fair comparison, we have implemented the Arborescent algorithm in Python. Comparisons and results can be found in Figs. 11, 12, 13.

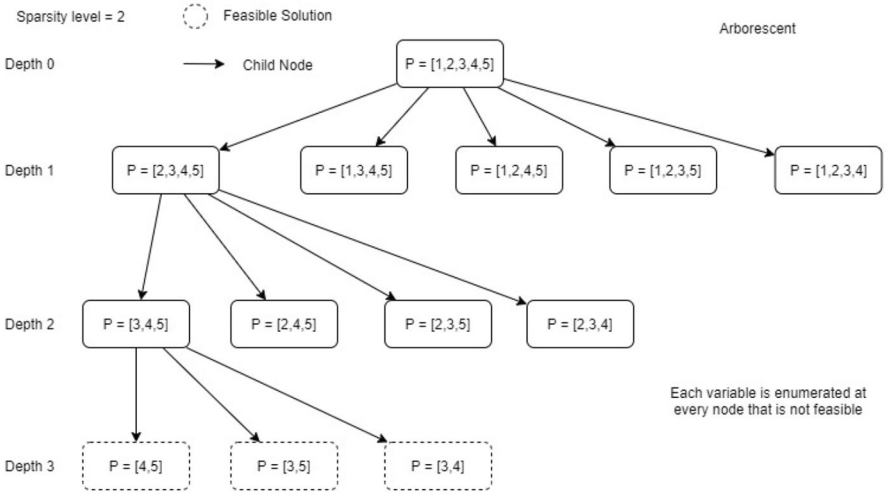

Fig. 12 Arborescent

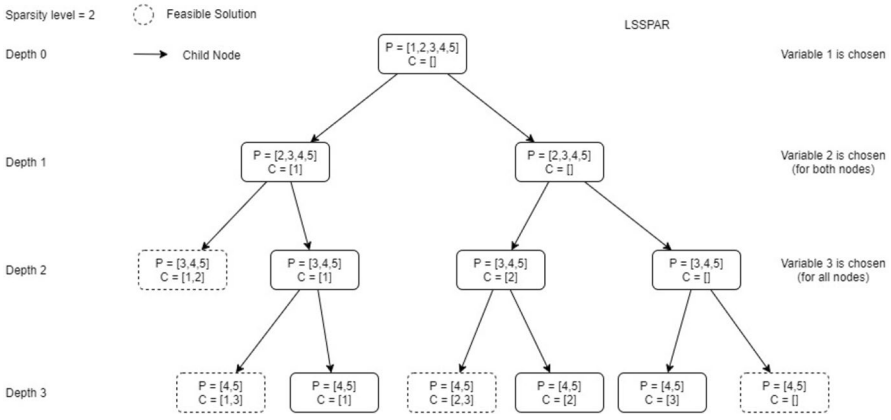

Fig. 13 NLSSPAR

The Arborescent algorithm does not perform well when there are many variables, and the sparsity level is not close to the number of variables as the algorithm uses a backward elimination process to induce sparsity. Figure 11 shows the results on artificial data (with 100 observations and 14 variables) specifically generated for comparison with Arborescent. On the other datasets that we compared NNLSSPAR and MIP solvers, Arborescent fails to solve the problem in a reasonable time. Moreover, it creates redundant nodes because of its branching scheme, which also stalls the algorithm. Additionally, Arborescent uses a depth first search and so examines a larger number of nodes compared to NNLSSPAR's best first search.

Comparison with Nonconvex Gauge Function Approach As with the sparsity constraint, a constraint based on the convex Gauge function is defined as $\mathcal{G}_{\mathcal{B}}(y) = \inf\{\mathbf{1}^T x \mid y = \sum_{i=1}^{|\mathcal{B}|} B_i x_i,\ B_i \in \mathcal{B}\}$ Rockafellar (1970) is also often used to find sparse solutions Chandrasekaran et al. (2012). In Eftekhari and Esfahani (2021), a new non-convex $Gauge_s$ function is proposed for sparse recovery, and a numerical algorithm is described to solve a modified version of SNNLS where a linear constraint is added to the problem based on the newly defined non-convex $Gauge_s$ function. Hence one solves the following problem

$$\begin{aligned} \min_{x \in R^n} \ & \left\|A(B_i)x - b\right\|_2^2 \\ \text{s.t.} \ & x \geq 0 \\ & \mathbf{1}^T x \leq \mathcal{G}_{\mathcal{B},s}(y) \\ & \|x\|_0 \leq s \end{aligned} \tag{4.1}$$

where $\mathcal{G}_{\mathcal{B},s}(y)$ is the $Gauge_s$ function defined as

$$\mathcal{G}_{\mathcal{B},s}(y) = \inf\{\mathbf{1}^T x \mid y = \sum_{i=1}^{|\mathcal{B}|} B_i x_i,\ ||x||_0 \leq s,\ B_i \in \mathcal{B}\},$$

which is analogous to a sparsity constrained Gauge function (see Eftekhari and Esfahani (2021)). We note that in the literature the Gauge function approach focuses on y and its sparse decomposition on alphabet $\mathcal{B}$ given by x whereas we directly model and focus on the decomposition x itself. For comparison with NNLSSPAR, we use the same setup used in Eftekhari and Esfahani (2021) which can be described as follows

$$\begin{aligned} A_i(t) &= e^{-\frac{(t-0.05*i)^2}{0.35^2}},\ i = 1, .., 20 \\ x^T &= [\mathbf{1}_2^T, \mathbf{0}_{18}^T] \\ b &= Ax + \varepsilon \\ \varepsilon &\sim \mathcal{N}(0, \sigma_e) \end{aligned}$$

where $\mathcal{N}(0, \sigma_e)$ denotes the Gaussian distribution with zero mean and standard deviation σ_e. Values of $t \in \mathbb{R}^{40}$ were randomly generated to construct the dictionary A. We solved the same problem for different noise levels $\sigma_e = 0, 10^{-1}, 10^{-2}, 10^{-3}$ and repeated the experiment 40 times. We implemented the $Gauge_s$ subroutine using CVXPY Diamonda and Boyd (2016). Results are summarized in Figs. 14 and 15.

NNLSSPAR successfully recovers the true solution when the noise level is sufficiently low. When the noise level is high, the current problem formulation does not allow recovery of the true x, due to the sensitivity of the squared loss to outliers and the non-negativity constraint. We also observed that due to noise, even when we solve the non-negative least squares problem with the true subset of columns of A, not all variables obtain positive values. Some of them are assigned zero value, meaning that the noise significantly corrupts the underlying system, and we are unable to recover a true solution using SNNLS. We observe similar behavior in sparse deconvolution problems, which is discussed in the section 6. Moreover, we also

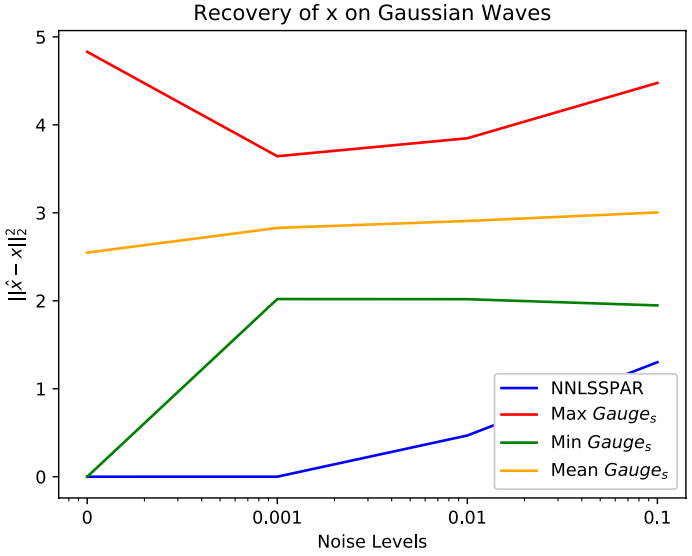

Fig. 14 Recovery comparison of NNLSSPAR and $Gauge_s$

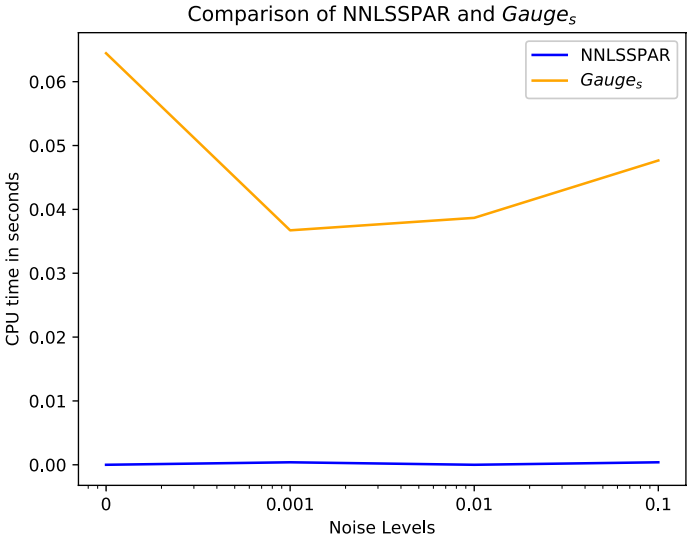

Fig. 15 CPU time comparison of NNLSSPAR and $Gauge_s$

observed that $Gauge_s$ recovers the true underlying variables in some of the experiments. Therefore, the true subset of variables is in the set of solutions obtained by $Gauge_s$. The $Gauge_s$ approach merits a more comprehensive numerical investigation in the future. Finally, both NNLSSPAR and $Gauge_s$ can solve small sized problems very quickly.

In summary, NNLSSPAR performs well in general since non-negativity constraints naturally induce sparsity. Furthermore, when implicit enumeration zeroes out one of the nonzero elements in a non-negative least squares solution, magnitudes of the residuals of the system increase so much that in general, very few nodes need to be searched. Finally, we observe that the difficulty of the problem does not depend on m, but on n, in particular with dominant elements of x in non-negative least squares solutions, and s, which determine the number of possible subsets.

5 Application to hyperspectral imaging

5.1 NNLS implementation

One of the immediate applications for the sparse NNLS problem is hyperspectral imaging which is encountered in areas such as chemistry, computer vision and aerial imaging. The application domain of the present paper is abundance detection. The abundance estimation framework can be described as follows. We start with the linear mixing model assumption and formulate the problem as a sparsity constrained NNLS problem. Let $X = [x_1, x_2, ..., x_b]^T$ be a mixing matrix of dimension $b \times p$, such that $x_i \in \mathbb{R}^p$ where b denotes the number of existing spectra in the image and p is the number of the end-members selected. The selection of the end-members and the construction of the mixing matrix is performed using the end-member extraction algorithm N-FINDR which is commonly used in the literature Heylen et al. (2011) and Winter (1999). In our model, $y_i \in \mathbb{R}^b$ denote the values of the pixel i as a vector in the dimension of the number of spectra. The mixing weights $w_i \in \mathbb{R}^p$ are calculated using our algorithm for each pixel:

$$y_i = Xw_i. \tag{5.1}$$

Since the mixing weights correspond to a sparse combination of non-negative values, our algorithm is a natural candidate for the solution to this problem. The formulation of the problem for pixel i is as follows;

$$\begin{array}{ll} \min & \|y_i - Xw_i\|_2^2 \\ \text{s.t.} & w_i \geq 0 \\ & \|w_i\|_0 \leq s. \end{array} \tag{5.2}$$

In the literature, it is common to solve a relaxation of this problem by dropping the cardinality constraint and using a surrogate function such as $\sum w_i = 1$ Menon et al. (2016). This formulation is also meaningful when the weights are allowed to have fractional values. Naturally, the present paper is more concerned with the precise selection of the materials. Therefore, the cardinality constraint is more pertinent for such problems. Furthermore, alternative approaches to obtain sparsity using regularization methods have been studied in the literature Drumetz et al. (2019).

The hyperspectral images from the Salinas A dataset have been used for tests. Parameter selection is made based on previous studies in the literature involving this dataset. There are 6 major materials scattered in the dataset where we

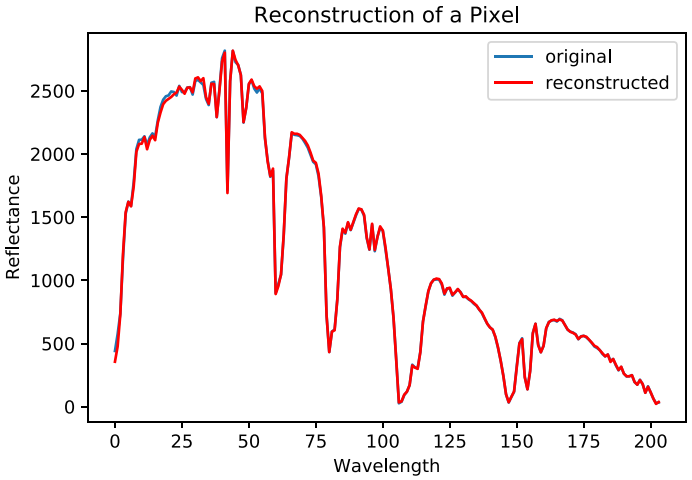

Fig. 16 Reconstruction of a Pixel

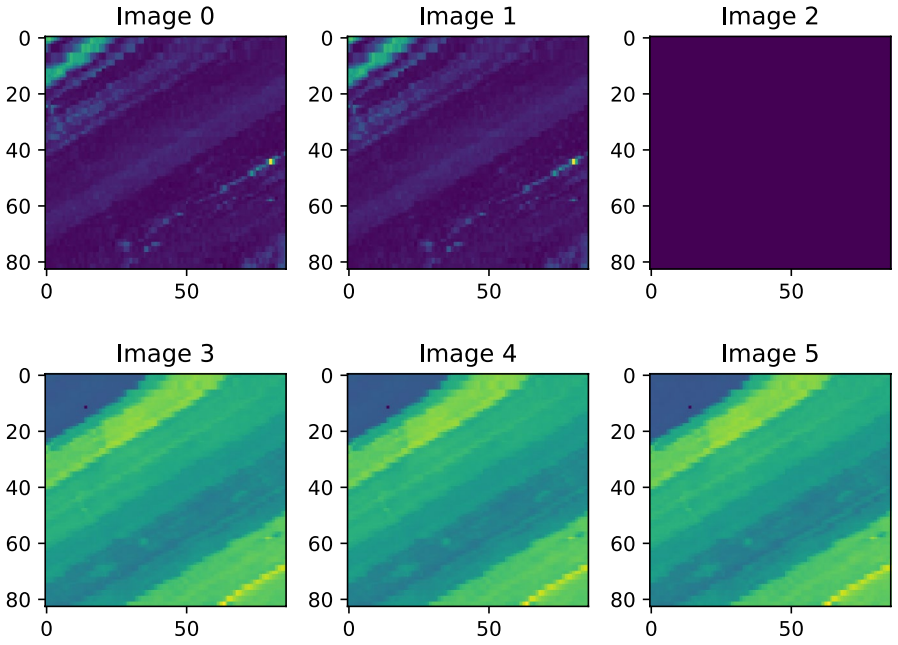

Fig. 17 Reconstruction comparison of our algorithm and Gurobi

have the assumption that the pixels contain only sparse combinations of these six materials. We generate the mixing matrix for the 12 most meaningful materials and solve the NNLS problem mentioned above for each pixel in the data. The problems are solved very efficiently with very minimal reconstruction errors. The maximum sparsity level equal to 6 has been used as oracle information during

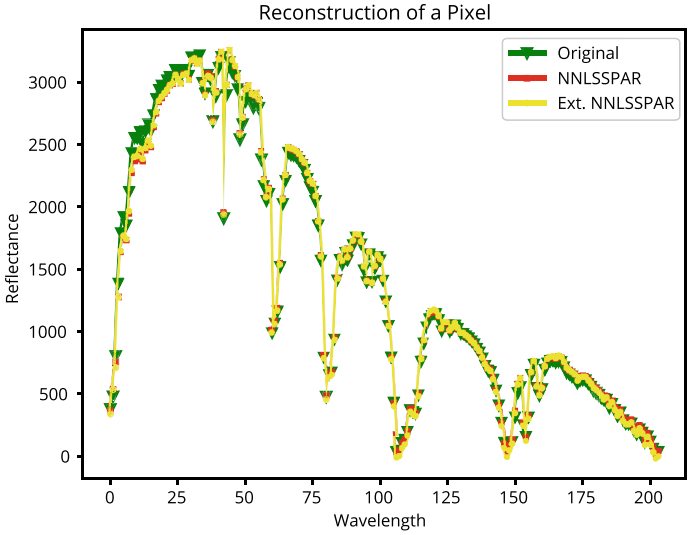

Fig. 18 Reconstruction of a Pixel using NNLS and extended NNLS

the tests. However, most of the pixels were even sparser than this level having $\|w_i\|_0 \leq 3$ for most i's (Fig. 16).

In Fig. 17, a color map of average MSE values for each pixel and reconstruction of a randomly selected band is displayed. Color maps of the average MSE errors of each pixel for all bands are shown in the first row, and the reconstructed images are shown in the second row. The last column represents the ground truth values, the first column represents the results obtained by our algorithm and the second column is obtained using the Gurobi MIP solver. The performance of our algorithm is slightly better as the overall MSE obtained is less than Gurobi's MIP solution and the overall execution time is also smaller than the MIP solution. This concludes the discussion of the performance of our algorithm for the exact sparse NNLS problem.

5.2 Extended least squares implementation

In the literature, there exist variants to the original NNLS problem. One of the most prominent extensions is the one reported in Guo and Berman (2012) regarding applications in spectroscopy. We also developed an addendum to our NNLS algorithm to be compatible with this specific extension. This appendage addresses the following problem

$$\begin{aligned} \min_{w_i, v_i \in R^p} \quad & \|Xw_i + Zv_i - y_i\|_2^2 \\ \text{s.t.} \quad & w_i \geq 0 \\ & \|w_i\|_0 \leq s. \end{aligned} \tag{5.3}$$

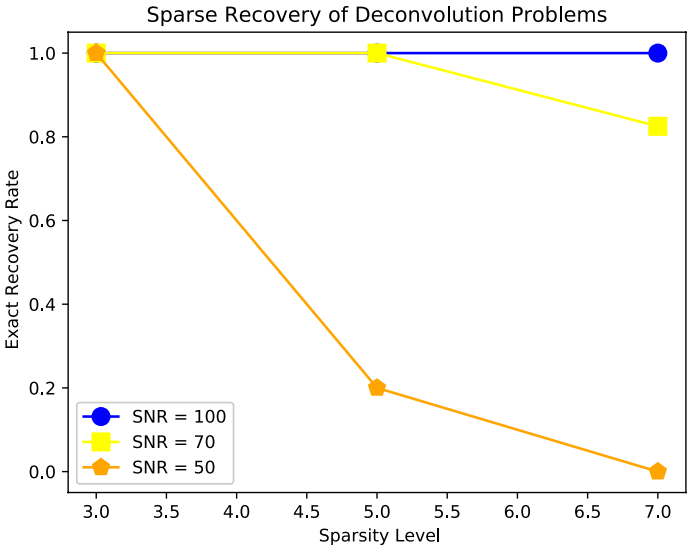

Fig. 19 Recovery Rate of NNLSSPAR in Sparse Deconvolution Problems

We re-formulate the hyperspectral imaging problem by taking the matrix Z as the bias term, i.e., every row of Z is a row of ones, $Z_i = \mathbf{1}_p$. This new matrix $Z \in \mathbf{R}^{b \times p}$ allows new variables to be introduced to the NNLS problem with no non-negativity or sparsity restriction, allowing us to effectively use it as a bias term for every variable w_i. We present the extended NNLS results on the same hyperspectral imaging dataset as the spectroscopy datasets have more restricted access. The comparison of the methods used is given in Fig. 18.

6 Sparse deconvolution problems

In the signal processing literature, an approach to formulate deconvolution problems as mixed integer problems were discussed in Bourguignon et al. (2016). This reformulation guarantees optimality of the solution whereas common heuristics such as IHT and COSAMP do not provide this guarantee. On the other hand, a recent study Donne et al. (2020) reports that well-known MIP formulations of sparse recovery problems may not always yield an exact recovery, and proposes an alternative formulation based on set covering. Our algorithm is applicable to the test problems of Bourguignon et al. (2016) since we provide an efficient solution for the problems that they have formulated for general MIP solvers.

We compared the results provided in Bourguignon et al. (2016) by solving the test problems using our algorithm. Since the comparisons of deconvolution methods for different cases are studied in their paper, we directly compare their results with our algorithm.

We have generated 40 different tests for each of the three different sparsity values and signal to noise ratios (SNR). Figure 19 presents the results obtained.

Our algorithm is not as successful in exact recovery for different SNR values, with very low SNR values in particular, when compared with the formulations in Bourguignon et al. (2016). Solving the non-negative sparse least squares problem on data with a low SNR value does not allow recovery of the original support. An explanation for this result is that the least squares loss function may not be as robust to noise as its counterpart in Bourguignon et al. (2016). Different loss functions could have been used such as Huber Loss, in order to enforce robustness under such heavy noise conditions. Moreover, without changing the loss function one can deploy different data denoising methods that are useful under highly additive noise such as the one proposed in Bhatia et al. (2015). This type of denoising is computationally attractive since it reduces the number of rows in the data matrix and could be easily implemented for our algorithm. We will pursue this approach in future work. In addition, the presence of noise contamination slowed down our algorithm remarkably. Positive variables in the optimal solution of the sparse problem tend to vanish in the unconstrained problem when the corruptive noise is coupled with the non-negativity constraint. As a result, the algorithm does not prioritize enumerating over those variables, thereby slowing down the search.

7 Conclusion

We studied the sparsity constrained non-negative least squares problem. As opposed to local solution methods, we proposed a novel implicit enumeration idea to solve the sparse NNLS problem optimally and efficiently. We conducted tests to compare our method with the most common off-the-shelf solvers as well as the only competitor of our algorithm in the literature. In general, our algorithm converges faster and gives more stable results compared to the quadratic mixed integer solvers. Furthermore, we provide a Python implementation of our algorithm as a library to be easily used to solve any sparse NNLS problems. It is the authors' hope that NNLSSPAR will become a new benchmark in subset selection and sparse recovery research.

Appendices

Details of parameters and variables

Both the matrix A and the vector b should be registered as numpy arrays and one should specify the data type as float64 even if all the values are integer to avoid potential numerical issues. While registering and working with matrices or vectors using Python, shape definition is very important. Shape of the vector leads to large changes in how linear algebra functions behave. Python has two different definitions of vector. If the user registers a one dimensional vector of ones of length n, Python registers it as a vector that has a shape of (n,). However, if instead numpy.ones([1,10]) is used, then it will mathematically express the same vector but in Python it will have the shape of (1,10). Using matrix operations

with these two different vectors will give different results. The vector b should be registered as a column vector of shape (m,1).

NNLSSPAR calls the guppy3 package by Zhu and Nilsson for tracking the memory usage of the algorithm Zhu and Nilsson (2019). As a result, the guppy3 package should be installed to be able to use NNLSSPAR.

A description of the parameters of NNLSSPAR library is given below.

Parameters:

1. A (input) = $m \times n$ matrix storing the values of the independent variables where m must be greater than n
2. b (input) = $m \times 1$ vector storing the values of the dependent variables
3. s (input) = An integer value indicating sparsity level (maximum number of nonzero elements in the solution)
4. out (input, by default it is 1) = An integer value, that is a parameter controlling the detail level of output:

   (a) out = 0, Nothing will be printed
   (b) out = 1, The results and the hardware usage of the algorithm will be printed
   (c) out = 2, At every iteration, the lower bound of the current node and the number of nodes searched so far will be printed. After the algorithm terminates NNLSSPAR will report the results and the hardware usage. Although it is good to observe the progress of the algorithm, it should be noted here that it slows down the algorithm significantly.

5. C (input, by default it is $\emptyset$) = Array of integers, storing the indices of chosen variables
6. many (input, by default it is 4) = An integer specifying number of best subsets to be found
7. residual_squared (output) = a list of length *many* ($s * many$ if all subsets function is used) containing residuals corresponding to subset found for some sparsity level. This only shows the residuals of the system after QR decomposition. True residual squared of the systems are residual_squared + permanent_residual
8. indexes (output) = a list of length *many* ($s * many$ if all subsets function is used) containing indices of the independent variables that make the subsets for some sparsity level
9. coefficients (output) = a list of length *many* ($s * many$ if all subsets function is used) containing values of the least squares solution of each subset for some sparsity level
10. permanent_residual (output) = a float that records the residual squared of the system after orthogonal transform using QR decomposition of the matrix A
11. cpu (output) = a float showing the CPU time of the algorithm in seconds. CPU time of the algorithm is print if out is not 0
12. memory (output) = a float showing the total memory usage of the algorithm in bytes. Detailed table of memory usage is print if out is not 0
13. real (output) = a float showing the wall time of the algorithm in seconds.

14. nodes (hidden) = a list of integers showing number of visited nodes in the graph. It is equal to the number of branchings done due to best first search if only one solution is found. For each sparsity level it will be reported separately.
15. rem_qsize (hidden) = a list of integers showing the number of unvisited nodes in the graph. This will be equal to nodes if only one solution is found. For each sparsity level it will be reported separately.

An example

Assume we are trying to solve an instance of a problem where A, b and s are specified for input into NNLSSPAR in Python syntax as follows:

```
A = numpy.random.normal(0,1,[20,10])
x = numpy.reshape([3,0,0,2,1,0,0,1,0,0],[10,1])
b = A.dot(x)
s = 4
#Clearly, the optimal solution is x = [3;2;1;1].
#We create instance of an object:
u = NNLSSPAR(A,b,s)
#Call the solve function
sol = u.solve()

#Reassuringly, the output does not surprise
indexes = [4, 0, 3, 7]
coefficients = [7. 4. 2. 1.]
residual_squared = 2.1938946325562114e-29

#We can also perturb the data
b_p = A.dot(x) + random.normal(0,1,[20,1])
sol_p = NNLSSPAR(A,b_p,s)
sol_p.solve()

#The expected output is obtained, with the noisy data
#weights which are no longer the exact original values.

indexes = [4, 0, 3, 7]
coefficients = [6.64783762 4.37502627 1.62440717 1.04722373]
residual_squared = 18.81327471448796
```

In the output, the list of values of the variables are in the order given by the indexes. Hence, 6.64783762 is the value of fourth independent variable, 1.62440717 is the value of third independent variable, and so on.

A special case

If the solution of multiple subsets for each sparsity level $i = 1, 2, \ldots, s-1, s$ is desired, then the following function should be called:

```
u.solve_allsubsets()
```

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s11081-021-09676-2.

References

Aktaş FS, Ekmekçioğlu O, Pınar M Ç (2020) User's guide to nnlsspar, https://github.com/Fatih-S-AKTAS/LSSPAR

Atamturk A, Gomez A (2020) Safe screening rules for l0-regression from perspective relaxations. In: Dauma H III, Singh A (eds) Proceedings of the 37th international conference on machine learning, vol 119 of Proceedings of machine learning research, PMLR, 13–18, pp. 421–430, http://proceedings.mlr.press/v119/atamturk20a.html

Beck A, Eldar YC (2013) Sparsity constrained nonlinear optimization: optimality conditions and algorithms. SIAM J Optim 23:1480–1509. https://doi.org/10.1137/120869778

Bertsimas D, King A, Mazumder R (2016) Best subset selection via a modern optimization lens. Ann Stat 44:813–852. https://doi.org/10.1214/15-aos1388

Bhatia K, Jain P, Kar P (2015) Robust regression via hard thresholding. arXiv:1506:02428

Bourguignon S, Ninin J, Carfantan H, Mongeau M (2016) Exact sparse approximation problems via mixed-integer programming: formulations and computational performance. IEEE Trans Signal Proc 64:1405–1419

Breiman L, Friedma JH (1985) Estimating optimal transformations for multiple regression and correlation. J Am Stat Assoc. https://doi.org/10.1186/s13040-017-0154-4

Chandrasekaran V, Recht B, Parrilo PA, Willsky AS (2012) The convex geometry of linear inverse problems. Found Comput Math 12:805–849. https://doi.org/10.1007/s10208-012-9135-7

Cplex II (2009) V12. 1: User's manual for cplex. In: International business machines corporation vol 46

Diamond S, Boyd S (2016) CVXPY: a Python-embedded modeling language for convex optimization. J Mach Learn Res 17:1–5

Donne DD, Kowalski M, Liberti L (2020) Mip and set covering approaches for sparse approximation. In: iTWIST annual conference

Drumetz L, Meyer TR, Chanussot J, Bertozzi AL, Jutten C (2019) Hyperspectral image unmixing with endmember bundles and group sparsity inducing mixed norms. IEEE Trans Image Process 28:3435–3450. https://doi.org/10.1109/tip.2019.2897254

Eftekhari A, Esfahani PM (2021) The nonconvex geometry of linear inverse problems. arXiv:2101.02776

Eftekhari A, Tanner J, Thompson A, Toader B, Tyagi H (2021) Sparse non-negative super-resolution–simplified and stabilised. Appl Comput Harmon Anal 50:216–280. https://doi.org/10.1016/j.acha.2019.08.004, https://www.sciencedirect.com/science/article/pii/S1063520319300193

Engl H, Hanke M, Neubauer A (2000) Regularization of inverse problems. Mathematics and its applications, Springer, Netherlands https://books.google.com.tr/books?id=VuEV-Gj1GZcC

Ghaoui LE, Lebret H (1997) Robust solutions to least-squares problems with uncertain data. SIAM J Matrix Anal Appl 18:1035–1064. https://doi.org/10.1137/S0895479896298130

Golub GH, Loan CFV (1996) Matrix computations, John Hopkins University, Department of Computer Science, Stanford University; Department of Computer Science, Cornell University, 3rd. ed.,

Guo Y, Berman M (2012) A comparison between subset selection and l1 regularisation with an application in spectroscopy. Chemometric Intell Lab Syst 118:127–138. https://doi.org/10.1016/j.chemolab.2012.08.010, http://www.sciencedirect.com/science/article/pii/S0169743912001657

Gurobi L Optimization (2020) Gurobi optimizer reference manual. http://www.gurobi.com

Hamidieh K (2018) A data-driven statistical model for predicting the critical temperature of a superconductor. Comput Mater Sci 154:346–354. https://doi.org/10.1016/j.commatsci.2018.07.052

Heylen R, Burazerovic D, Scheunders P (2011) Fully constrained least squares spectral unmixing by simplex projection. IEEE Trans Geosci Remote Sens 49:4112–4122. https://doi.org/10.1109/tgrs.2011.2155070

Kelwin Fernandes PV, Cortez P (2015) A proactive intelligent decision support system for predicting the popularity of online news. In: Proceedings of the 17th EPIA, Portuguese conference on artificial intelligence

Lawson CL, Hanson RJ (1995) Solving least squares problems, SIAM. San Clemente, California; Rice University Houston, Texas classics in applied mathematics ed

Candanedo V. F. Luis M, Deramaix D (2017) Data driven prediction models of energy use of appliances in a low-energy house. Energy Build 140:81–97. https://doi.org/10.1016/j.enbuild.2017.01.083

Menon V, Du Q, Fowler JE (2016) Random-projection-based nonnegative least squares for hyperspectral image unmixing. In: 2016 8th workshop on hyperspectral image and signal processing: evolution in remote sensing (WHISPERS), https://doi.org/10.1109/whispers.2016.8071796

Miller A (2002) Subset selection in regression. CRC Press, Chapman & Hall/CRC Monographs on Statistics & Applied Probability

Nadisic N, Vandaele A, Gillis N, Cohen JE (2020) Exact sparse nonnegative least squares. In: ICASSP 2020–2020 IEEE international conference on acoustics, speech and signal processing (ICASSP). https://doi.org/10.1109/icassp40776.2020.9053295

Natarajan B (1995) Sparse approximate solutions to linear systems. SIAM J Comp 24:227–234

Rockafellar RT (1970) Convex analysis. Princeton University Press, Princeton, Princeton Mathematical Series

Slawski M, Hein M (2011) Sparse recovery by thresholded non-negative least squares. In: Shawe-Taylor J, Zemel R, Bartlett P, Pereira F, Weinberger KQ (eds) Advances in neural information processing systems, vol 24, Curran Associates, Inc., https://proceedings.neurips.cc/paper/2011/file/d6723e7cd6735df68d1ce4c704c29a04-Paper.pdf

Slawski M, Hein M (2013) Non-negative least squares for high-dimensional linear models: Consistency and sparse recovery without regularization. Electron J Stat 7:3004–3056. https://doi.org/10.1214/13-ejs868

Tikhonov AN, Goncharsky AV, Stepanov VV, Yagola AG (1995) Numerical methods for the solution of ill-posed problems. Mathematics and its applications, Springer, Dordrecht. https://doi.org/10.1007/978-94-015-8480-7, https://cds.cern.ch/record/1620560

Vidyasagar M (2019) An introduction to compressed sensing. SIAM, Philadelphia

Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, van der Walt SJ, Brett M, Wilson J, Millman KJ, Mayorov N, Nelson ARJ, Jones E, Kern R, Larson E, Carey CJ, Polat İ, Feng Y, Moore EW, VanderPlas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, van Mulbregt P, SciPy 1.0 Contributors, (2020) SciPy 1.0: fundamental algorithms for scientific computing in python. Nature Methods 17:261–272. https://doi.org/10.1038/s41592-019-0686-2

Winter ME (1999) N-findr: an algorithm for fast autonomous spectral end-member determination in hyperspectral data. Imaging Spectrometry V. https://doi.org/10.1117/12.366289

Zhu Y, Nilsson S (Nov. 2019) *Guppy 3: A python programming environment & heap analysis toolset, github repository*, https://github.com/zhuyifei1999/guppy3