Provably Optimal Sparse Solutions to Overdetermined Linear Systems by Implicit Enumeration

F.S. AKTAŞ, M.Ç. PINAR
Bilkent University

Computing sparse solutions to linear systems is an ubiquitous problem in several fields such as regression analysis, signal and image processing, information theory. The objective of the present project is to develop a Python code implementing an implicit enumeration algorithm to find provably optimal solutions to the NP-hard problem

$$\min_{x} \ ||Ax-b||_2^2$$
s.t
$$\text{number of nonzeroes(x)} \leq s,$$

for a given sparsity level s. Here we assume A is mxn real matrix where $m > n$, i.e., the system is overdetermined. Therefore, a solution in the least squares sense is sought. Most previous research efforts concentrated on approximating the above problem, and/or finding local solutions by means of descent algorithms. We incorporate some of these local algorithms into our implicit enumeration software as well to find good upper bounds. Different enumeration strategies will be tested.

Keywords: Overdetermined Linear Systems, Sparse Solutions, Branch and Bound, Implicit Enumeration

1 Introduction

This paper describes a Python package, which is referred to as LSSPAR, that implements an implicit enumeration algorithm to find sparse solutions to overdetermined linear systems in a provably optimal fashion. Hence, the algorithm solves the following problem;

$$\min_{x \in R^n} \ ||Ax-b||_2^2$$
s.t
$$||x||_0 \leq s$$

where $A \in \mathbb{R}^{mxn}$ matrix where $m > n$, $b \in \mathbb{R}^n$ vector, $||v||_q$ is the ℓ_q-norm of vector v, $(\sum_{i=1}^n |v_i|^q)^{\frac{1}{q}}$, $||v||_0$ is the ℓ_0-norm of vector v, which is simply the number of nonzero elements of v, and s is called sparsity level. A good reference on the problem addressed in the present paper is by Amir Beck & Yonina C. Eldar, Sparsity Constrained Nonlinear Optimization: Optimality Conditions and Algorithms [1]. The interested reader can consult that reference for theoretical properties and for iterative descent algorithms that compute L-Stationary points or PCW-Minima (Partial Coordinate-wise optimum). The motivation for L-Stationarity and PCW-Minimality is that these are necessary conditions for global optimality. However, those algorithms do not guarantee optimal solutions, and hence can be viewed as heuristics.

The sub-problem solved at each step of the algorithm underlying LSSPAR is standard least squares problem. In a statistical context, in particular in regression analysis, this problem occurs naturally when one wants to solve an estimation problem related to the linear model:

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

where Y is the variable to be predicted, also called dependent variable, and $X_1, X_2, \ldots, X_k$ are the predictors, also called independent variables, and the residuals ϵ have zero mean and are independently sampled from a Gaussian distribution with finite variance.

The values of the coefficients $\beta = (\beta_0, \beta_1, \ldots, \beta_k)^T$ are found from the solution of the least squares problem

$$\min_{\beta} \|y - X\beta\|_2^2$$

where $\beta \in \mathbb{R}^k, y \in \mathbb{R}^n$ vectors, $X \in \mathbb{R}^{nxk}$, matrix. LSSPAR makes use of implicit enumeration and least squares to evaluate goodness of a solution. The package allows the user to choose different kinds of enumeration strategies such as e.g., "the lexicographically oriented search" or "m-st-lsc search" (see section 4) and search strategies like depth or breadth first search. It 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 $\binom{n}{s}$. On the other hand, when the solution is "simple" enough, branching only s times may be enough to obtain it.

In a nutshell, the algorithm of LSSPAR proceeds as follows. Each node is searched according to user's preferred search rule. Given that algorithm is at any node of the search tree, the first thing is to check if the current solution is feasible. If affirmative it is recorded; coefficients and objective function value are computed along with some other operations according to different search rules. If infeasible then the algorithm chooses a variable to branch on according to a rule chosen by user, and creates two new nodes, one where the chosen variable is in solution and the second where the variable is erased from the problem. These two new nodes are added to a search tree. Then, the algorithm goes back to the first step and explores other nodes according to the search rule.

The rest of the paper is organized as follows. Section 2 is a brief reminder about the method of least squares. Section 3 describes the input, output and different choices that can be deployed for search and enumeration rules. Section 4 explains default parameters, support variables, hidden variables and how the problem instance must be defined. Section 5 refers to implementation details. Section 6 shows a summary of numerical results we obtained using LSSPAR.

2 Recalls on Least Squares

In each node of the search tree, a least squares sub-problem is solved to evaluate the objective value to find a lower bound for the current solution. Solution can be obtained by solving the following "normal equations".

$$A^T Ax = A^T b.$$

Since $A^T A$ symmetric positive definite, this linear system can be solved by using Cholesky Decomposition. Orthogonalization methods like QR factorization or Singular Value Decomposition (SVD) are also used to solve this problem. If the condition number of the matrix $A, K_2(A)$, is large Cholesky Decomposition is known to give less accurate solutions compared to QR or SVD [2]. Our algorithm uses QR factorization and Singular Value Decomposition (SVD) to find the least squares solution at each node.

2.1 QR Factorization

The following properties of QR factorization are well-known:

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

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

- 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

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

2.2 Singular Value Decomposition

The Singular Value Decomposition of a matrix $A \in \mathbb{R}^{mxn}$ is given as follows:

$$A = U\Sigma V^T \ ,$$

where $U \in \mathbb{R}^{mxm}, V \in \mathbb{R}^{nxn}$ are both orthogonal matrices and $\Sigma \in \mathbb{R}^{mxn}$ is a diagonal matrix with only non-negative entries which are the singular values of the matrix A. Columns of U and V are the left and right singular vectors corresponding to the singular values Σ of A. U and V are generated by finding the eigenvalues and eigenvectors of AA^T and A^TA. Singular values of A are square roots of eigenvalues of A^TA.

The pseudoinverse of matrix A is

$$A^+ = V\Sigma^+U^T \ .$$

The solution to least squares problem can also be expressed using the pseudoinverse as

$$x = A^+b.$$

In fact, while solving the east squares problem with SVD algorithm, the matrix U^T is not necessarily computed explicitly, but rather applied to b. In general, the QR based method is faster than the one based on SVD. However, SVD is more reliable especially when the matrix A is ill conditioned [2].

2.3 Accuracy and Precision

The condition number of a 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

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

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 $\rho_{ls} = ||Ax_{LS} - b||_2$ the norm of the residual. Then the relative error is approximately as follows

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

[2].

3 Description of LSSPAR

LSSPAR requires as input the matrix A storing the values of independent variables, the vector b storing the values of the dependent variable and s, the sparsity level, that is, the maximum number of nonzero elements allowed to be in the solution vector x. The user has the option to also select different methods to deploy in enumeration and search, but these are not necessary. When they are not specified, the algorithm uses by default a best-first-search strategy and “m-st-lsc“ strategy (see section 4) for enumeration.

At the end of execution, the code will print the memory usage and CPU time of the algorithm. The algorithm works as in Figure 1.

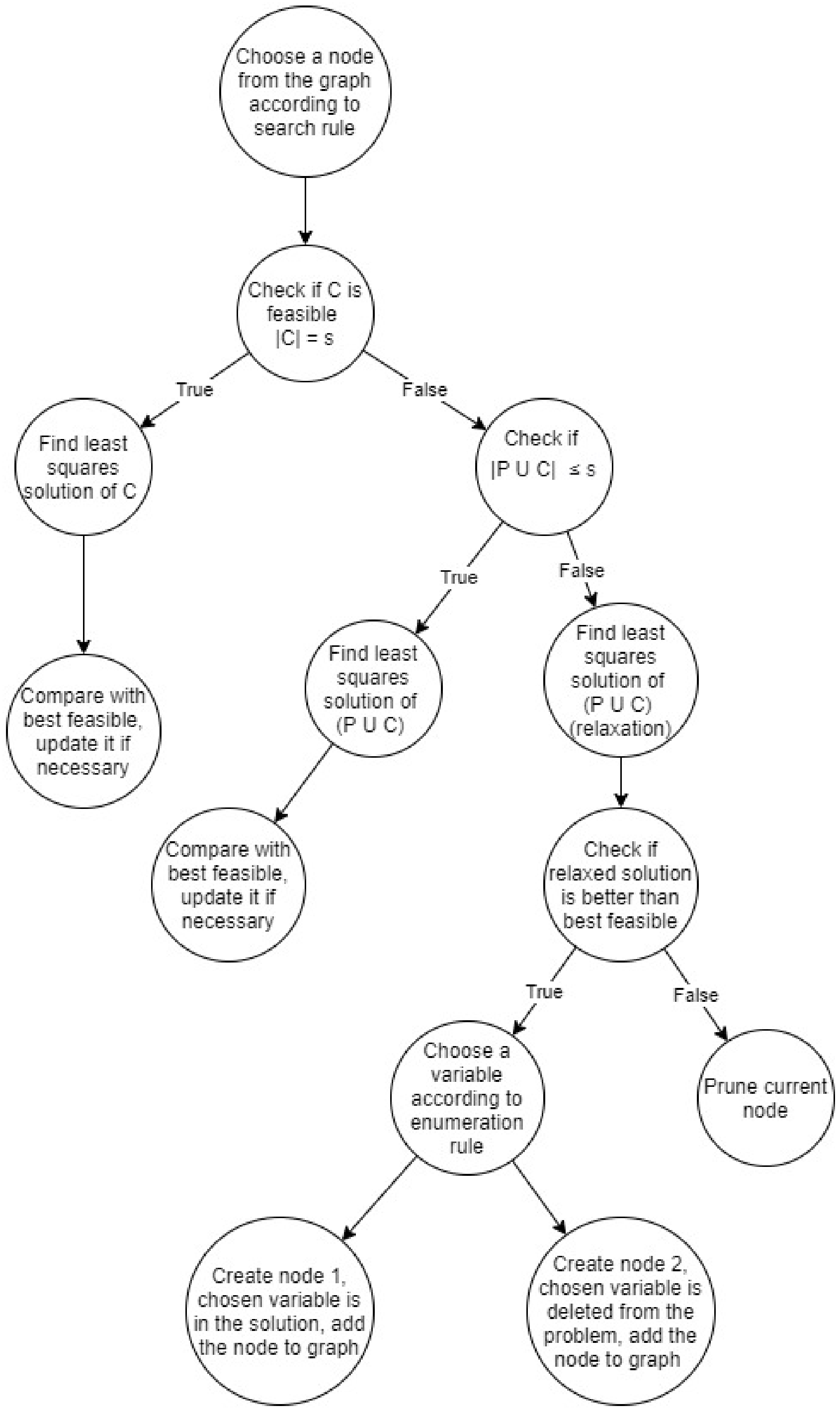
Two lists, P and C, are kept. Initially, P = [1,2, .. ,n] where n is the number of independent variables and C = []. P holds the indices of the independent variables that are candidates to be included in the solution. C holds the indices of the variables that are chosen to be in the solution. In each 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 recursive calls for depth-first search, first-in-first-out queue for breadth first search and priority queue for best first search, respectively. 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 s many variables. In other words, $|C| \leq s$ holds for any C.

In Figure 1, “best feasible” refers to the best feasible solution found thus far. Also, comparison of two solutions are made based on sum of squared errors.

Additionally, LSSPAR provides routines to find optimal subsets of all sizes up to sparsity level s specified, and best d subsets for each problem instance. These routines are considerably faster than using the code in a brute force manner and solving s separate problems. The use of this option is explained in the User’s Guide to LSSPAR [3].

The user may choose to print out various details on the progress of the algorithm at any search point as explained in the next section using the parameter OUT.

The user can specify the parameter C as input to the algorithm to force a variable to be in the solution set. Then LSSPAR will compute an optimal solution where the given index(es) are in the solution.



Choose a node from the graph according to search rule
Check if C is feasible |C| = s
True
False
Find least squares solution of C
Check if |P U C| ≤ s
Compare with best feasible, update it if necessary
True
False
Find least squares solution of (P U C)
Find least squares solution of (P U C) (relaxation)
Compare with best feasible, update it if necessary
Check if relaxed solution is better than best feasible
True
False
Choose a variable according to enumeration rule
Prune current node
Create node 1, chosen variable is in the solution, add the node to graph
Create node 2, chosen variable is deleted from the problem, add the node to graph

Figure 1: Generic flow of the algorithm

Algorithm 1: Flow of LSSPAR Computations

```
Input: Independent variable matrix $A$, dependent variable vector $b$,
    sparsity level $s$.
Output: Best subset indexes, least squares coefficients, residual of the
    solution, number of nodes searched
Step 1: (Initialization) Begin with node 0, solve least squares problem
  with all independent variables, Set P = [1,2 ... , n ] , C = [], lower
  bound = inf and go to Step 5.
Step 2: (Feasibility condition 1) Check if C is feasible (if number of
  variables in C = $s$ then it is feasible), if C is feasible, go to Step 3
  otherwise go to Step 4.
Step 3: (Solving the Least Squares Problem) Find the least squares
  solution with variables only in C if prior step is step 2, $P \cup C$ if prior
  step is step 4. Update the lower bound, prune the node. Go to step 7.
Step 4: (Feasibility condition 2) Check if $P \cup C$ is feasible (if number
  of variables in $P \cup C = s$ then it is feasible), if $P \cup C$ is feasible, go to
  step 3 otherwise go to Step 5.
Step 5: (Search Condition) the number of variables in $P \cup C$ must be
  greater than s. Find the the least squares solution with variables in
  $P \cup C$. This is a relaxation of the problem. If the residual value is
  greater than the lower bound, prune the node and go to step 7,
  otherwise go to step 6.
Step 6: (Branching) Choose a variable from P according to the
  enumeration rule, and delete it from P. Create two new nodes, one
  where the variable is forced to be in least squares solution ( the
  variable is added to C), other where it is removed from the problem (
  the variable is not added to C). Go to Step 7.
Step 7: (Searching) Choose a node according to search rule, if there
  are no nodes the algorithm is terminated, otherwise go to step 2.
```

4 Details of Parameters and Variables

In this section, we explain the parameters of LSSPAR.

A (input) = $m\,x\,n$ matrix storing the values of the independent variables where m must be greater than n

b (input) = $m\,x\,1$ vector storing the values of the dependent variables

s (input) = An integer value indicating sparsity level (maximum number of nonzero elements in the solution)

out (input, by default it is 1) = An integer value, that is a parameter controlling the detail level of output:

- out = 0, Nothing will be printed
- out = 1, The results and the hardware usage of the algorithm will be printed
- out = 2, At every iteration, lower bound of the current node and the number of nodes searched so far will be printed. After the algorithm terminates LSSPAR 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.

iter (input, by default it is 1000) = A positive integer value that specifies maximum allowed iterations that will be performed by heuristic algorithms.

C (input, by default it is $\emptyset$) = Array of integers, storing the indexes of chosen variables

enumerate (input, by default it is "m-st-lsc") = A string specifying which enumeration rule to be used in the algorithm

- enumerate = "m-st-lsc", the algorithm will calculate the quantity
  $|(|\bar{X}_i| + S_{x_i})a_{x_i}|$ for $i \in$ C
  where $|\bar{X}_i|$ is the absolute value of the mean of i'th variable and S_{x_i}) is the standard deviation of i'th variable and a_{x_i} is the coefficient of i'th variable in the least squares solution. Then, the algorithm will choose the index i so that $|(|\bar{X}_i| + S_{x_i})a_{x_i}|$ is maximized
- enumerate = "m-lsc", the algorithm will calculate the quantity
  $|\bar{X}_i a_{x_i}|$ for $i \in$ C
  Then, the algorithm will choose the index i so that $|\bar{X}_i a_{x_i}|$ is maximized
- enumerate = "lexi", the algorithm will enumerate lexicographically
- enumerate = "stat", the algorithm will calculate statistical significance of each variable's coefficient in the least squares solution, $Ea_{x_i}/Vara_{x_i}$, and will choose the index i so that $|Ea_{x_i}/Vara_{x_i}|$ is maximized

search (input, by default it is "best") = A string specifying which search rule to be used in the algorithm.

- search = "best", best-first-search will be done.
- search = "depth", depth-first-search will be performed.
- search = "breadth", breadth-first-search is activated.

solver (input, by default it is "qr") = A string specifying which solver,orthogonalization method, to be used in the algorithm

- solver = "svd", Singular Value Decomposition is used.

- solver = "qr", QR Factorization is used.

many (input, by default it is 4) = An integer specifying number of best subsets to be found

solcoef (output) = a vector of length s, storing the least squares solution coefficients of the variables in the order given by solset

x_vector (output) = an array storing the least squares solution coefficients of the variables in the vector form of x, i.e, it is a $n\,x\,1$ vector with least squares solution coefficients if they are in the solset, 0 otherwise

best_feasible (output) = a float, showing the sum of squared errors in the least squares solution

solset (output) = a vector of length n, storing the indexes of the variables in the solution in the order the variables are chosen. Also, their coefficients are given in the same order in the solcoef

node (output) = an integer showing the number of branchings done, equal to number of nodes at the end of the search, actual number of nodes created is $2*$ node $+1$

residual_squared (output) = a list of length $s * many$ containing residual corresponding to subset found for some sparsity level, Will give None unless problem is solved for multiple subsets.

indexes (output) = a list of length $s * many$ containing indexes of the independent variables that make the subsets for some sparsity level. Will give None unless problem is solved for multiple subsets.

coefficients (output) = a list of length $s * many$ containing coefficients of the least squares solution of each subset for some sparsity level. Will give None unless problem is solved for multiple subsets.

cpu (hidden) = a float showing the CPU time of the algorithm. CPU time of the algorithm is print if out is not 0

memory (hidden) = a float showing the total memory usage of the algorithm. Detailed table of memory usage is print if out is not 0

x_vector (hidden) = an array storing the least squares solution coefficients of the variables in the vector form of x, i.e, it is a n x 1 vector with least squares solution coefficients if they are in the solset, 0 otherwise best_heuristic (hidden) = a float, showing the sum of squared error in the least squares solution obtained by heuristic algorithms. It stores only the best one between two heuristics. Returns None if heuristic algorithms are not used

coef_heuristic (hidden) = an array, showing the coefficients of the least squares solution obtained by heuristic algorithms. It stores only the best one between two heuristics. Returns None if heuristics algorithms are not used

check (hidden) = an integer showing the number of nodes visited.

mean (hidden) = an array showing the mean of each independent variable, i.e, column means of matrix A

sterror (hidden) = an array showing the standard error of each independent variable, i.e, standard error of columns of matrix A

variances (hidden) = an array showing the variance of each independent variable, i.e, variance of columns of matrix A

rem_qsize (hidden) = an integer showing the number of unvisited nodes in the graph

ill (hidden) = a boolean checking if the matrix A is ill conditioned or has linearly dependent columns.

- ill = "True", a warning will be printed. If the A has linearly dependent columns, solution accuracy and precision is not lost. However, if A does not have linearly dependent columns but is ill-conditioned, problem will still be solved but accuracy and precision of the solution is not guaranteed.
- ill = "False", Nothing will be print. Problem will be solved to the precision given in Section 2.3.

4.1 An Example

Assume we are trying to solve an instance of a problem where A, b and s are specified for input into LSSPAR 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 $x = (3, 2, -1, 1)^T$.

We create instance of an object:

```
u = LSSPAR(A,b,s)
```
Call the solve function
```
sol = u.solve()
```

Reassuringly, the output does not surprise

```
sol = [array([[ 1.], [-1.], [ 2.], [ 3.]]), 3.4070674139423715e-29, [7, 4, 3, 0], 4]
```

We can also perturb the data

```
b_p = A.dot(x) + random.normal(0,1,[20,1])
sol_p = LSSPAR(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.

```
[array([[ 1.06663347], [-0.87915142], [ 2.67771575], [ 2.18227863]]), 11.963547042406926,
[7, 4, 0, 3], 4]
```

Here, the first array shows the coefficients of the variables, in the order given by the second array. Hence, 1.41109344 is the coefficient of 7'th independent variable, -1.0655792 is the coefficient of 4'th independent variable, and so on. The integer number at the end of the output shows the number of nodes visited by the algorithm.

4.2 Special cases

If solution of multiple subsets is desired for a given sparsity level, the following command should be used

```
u.solve_multiple().
```

If solution of multiple subsets for each sparsity level $i = 1, 2, \ldots, s-1, s$ is desired (this is the default of A. Miller's Fortran code [4] as we shall discuss below), then the following function should be called

```
u.solve_allsubsets().
```

5 Implementation Details

Both matrix A and vector b should be registered as numpy arrays and specify the data type as float64 even if all the values are integer to avoid potential numerical issues. While registering and working with matrices is not problematic, in general in Python vectors require caution. Python has two different definitions of vector. If the user registers an 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).

LSSPAR uses lstsq, the least squares algorithm of numpy library. The lstsq algorithm uses dgelsy routine to solve the problem and this routine is based on singular value decomposition and the pseudo inverse. This numerical method is justified in the least squares method section, where otherwise the solution will end up having poor numerical precision due to round offs. LSSPAR also uses qr routine, i.e., QR factorization, of the same library. This routine implements another orthogonalization method for computing an accurate least squares solution. In general QR is known to provide a solution faster than SVD. Also, using QR, the structure of the problem can be exploited, which makes QR factorization from scratch at each node unnecessary. The economic version of QR factorization can be used instead, by deleting columns of initial R and doing QR on the new R matrix, which works faster than SVD which must be computed from scratch at each node. However, using SVD near rank deficiency cannot go undetected, which is not always the case for QR [2] and hence SVD is more reliable especially when the problem is ill-conditioned. Therefore, if the problem is well-conditioned we suggest that QR should be used. Otherwise, SVD should be used in the interest of numerical accuracy.

Let A be the independent data set and assume that we are trying to find the best fitting subset with 2 elements.

$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,1} & a_{4,2} & a_{4,3} \end{pmatrix} Q_A = \begin{pmatrix} q_{1,1} & q_{1,2} & q_{1,3} & q_{1,4} \\ q_{2,1} & q_{2,2} & q_{2,3} & q_{2,4} \\ q_{3,1} & q_{3,2} & q_{3,3} & q_{3,4} \\ q_{4,1} & q_{4,2} & q_{4,3} & q_{4,4} \end{pmatrix} R_A = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ 0 & r_{2,2} & r_{2,3} \\ 0 & 0 & r_{3,3} \\ 0 & 0 & 0 \end{pmatrix}$$

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

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

We do not have to perform a QR factorization from scratch as we can delete columns of R:

$$R_{AB} = \begin{pmatrix} r_{1,2} & r_{1,3} \\ r_{2,2} & r_{2,3} \\ 0 & r_{3,3} \\ 0 & 0 \end{pmatrix}$$

Now we can work with a 3 x 2 matrix instead of a 4 x 3 matrix

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

LSSPAR calls the guppy3 package by YiFei Zhu and Sverker Nilsson for tracking the memory usage of the algorithm [5]. As a result, guppy3 package should be installed to be able to use LSSPAR.

6 Tests and Results

We first tested the algorithm with some random data we created. We solved problems in different sizes and different sparsity levels and then collected some performance measures. Collected performance measures are CPU time, memory use and number of nodes used by the algorithm. Each case is solved 20 times for reliability of the results.

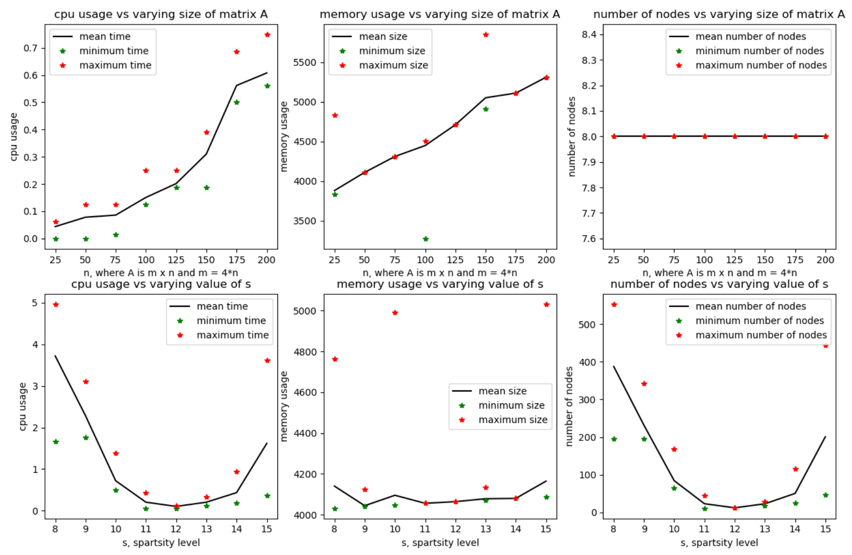


Figure 2: Test cases

Results summarized in Figure 2 suggest that the number of observations in the matrix A, i.e., m, has no impact over the number of nodes that will be searched. Number of nodes searched is determined by dominant elements of vector x and number of independent variables n. Also, CPU usage and memory usage increases as size of matrix A increases because it requires more space to register matrix A and more CPU time to calculate least squares solution. When the sparsity level changes, it requires the least amount of node search and CPU time when the "dominant" elements of x is equal to sparsity level s. CPU usage increases not because increased difficulty in least squares solution but because of the nodes searched. Hence, memory usage does not change since registered objects are of equal size. These results are obtained with "svd" solver, "m-st-lsc" enumeration and best first search.

We also tested the algorithm on the OZONE data set by Breiman. This data set is about ozone concentration, which is the independent variable, in Upland CA, east of Los Angeles and some predictor variables, (8 independent variables), which are meteorological measurements based on 330 observations in 1976 [6]. In his book entitled *Subset Selection in Regression*, Alan Miller lists some applications of the heuristic algorithms for subset selection as well as best solutions obtained by exhaustive search [7]. In addition to 8 independent variables, Miller creates 36 artificial variables from those 8 variables. First 8 original variables are $= X_1X_2..X_8$. The artificially created variables are interactions in the order $= X_1^2, X_1X_2, X_2^2, X_1X_3, X_2X_3, X_3^2, ..., X_8^2$. We compared the Alan Miller's Fortran software package with LSSPAR [4]. The table below shows the elapsed CPU and Wall clock time in seconds for different level of sparsity. Alan Miller's package is specifically written to find all subsets up to sparsity level specified. Hence, both algorithms find the best 5 subsets for each sparsity level $1, .., s$ in each s value in the table.

s	Miller CPU	LSSPAR CPU	Miller Real	LSSPAR Real
6	2.48	3.83	2.72	3.95
7	10.40	8.74	10.53	8.87
8	38.45	17.36	38.89	17.67
9	93.07	29.70	93.94	29.83
10	134.17	44.1	134.76	44.45

6.1 Differences Between LSSPAR and Alan Miller Fortran Package

Alan Miller's Fortran Package uses "Leaps and Bounds" algorithm [4], which generates and computes the regressions both from bottom to top and top to bottom in parallel [8]. Subsets are generated and computed, selecting variables lexicographically in the so called "product tree" and in the inverse tree, branching symmetrically according to product tree which also allows them to exploit the structure of the problem in the computation. LSSPAR does only inverse tree search with implicit enumeration, meaning that at each step it creates at most two nodes, and this search scheme is closer to depth first search. Leaps and Bounds algorithm on the other hand creates up to as many variables as the node has in inverse tree, and that search scheme is closer to breadth first search. However, Leaps and Bounds algorithm in the product tree does implicit enumeration similar to LSSPAR (c.f Figures 3 and 4).

Leaps and Bounds algorithm is pessimistic in its choice of branching scheme. It branches lexicographically. LSSPAR is more optimistic with its choice of branching scheme where there is a simple heuristic choice at each node to choose the next variable to branch on. In fact LSSPAR supports lexicographical search too, but we use "m-st-lsc" enumeration scheme because in our experience it proves to be much faster than lexicographical search. Furthermore, Leaps and Bounds algorithm is designed to find all best subsets of size from 1 to s, it finds bounds and subsets of all size at any step of the algorithm. LSSPAR is designed to find the best subset of size s. LSSPAR solves the all best subsets problem sequentially, starting with sparsity level $s = 1$ to find the bounds and subsets at each step with best first search (again, LSSPAR allows breadth or depth first search too, but they are found to be inferior compared to best first search in our computational experience).

We also used some data from machine learning benchmarks that are publicly available [9]. All times in subsequent tables are in seconds.

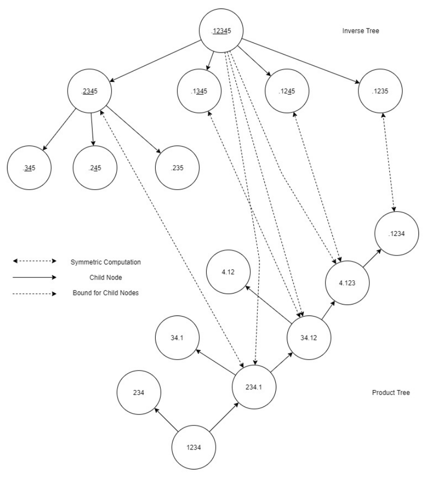


Figure 3: Leaps and Bounds

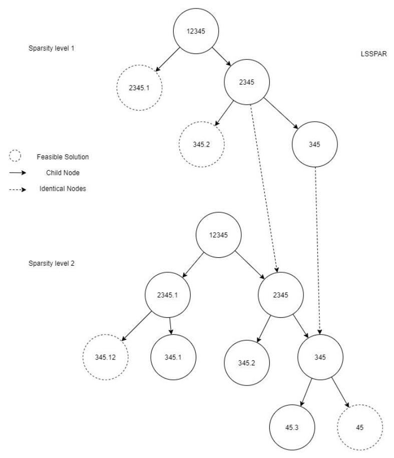


Figure 4: LSSPAR

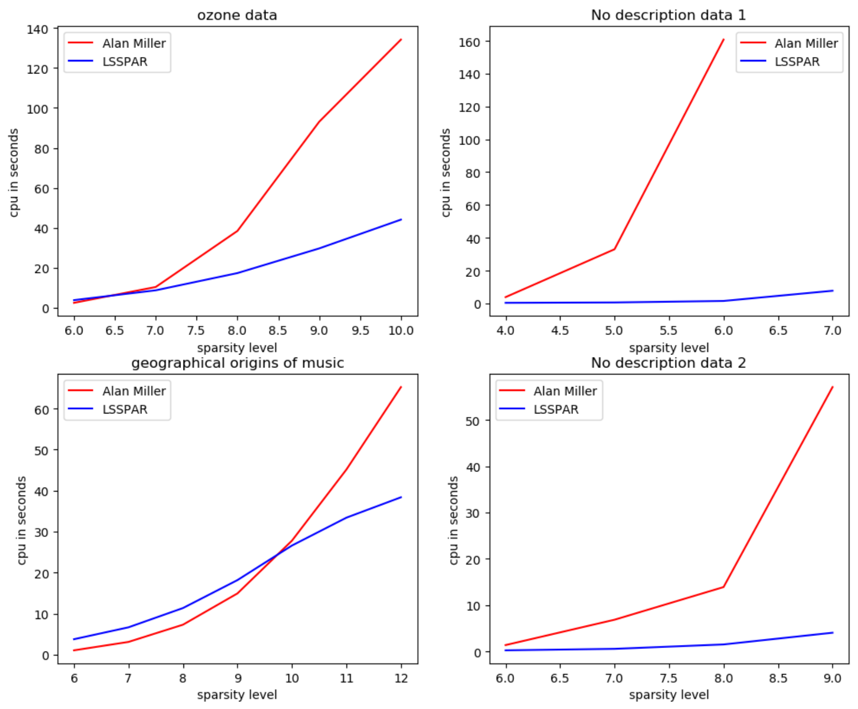


Figure 5: Comparison of Alan Miller Software and LSSPAR

No description data[9], 1000 observations 100 independent variable

s	Miller CPU	LSSPAR CPU	Miller Real	LSSPAR Real
4	3.82	0.34	3.95	0.35
5	33	0.57	33.34	0.59
6	160.78	1.48	164.78	1.51
7	-	7.70	-	7.75

"-" means that after 15 minutes of wall clock time, the algorithm still did not respond, and execution was terminated.

Geographical Origins of music [9], 6435 observations 36 independent variables

s	Miller CPU	LSSPAR CPU	Miller Real	LSSPAR Real
6	1.04	3.75	1.20	3.81
7	3.09	6.65	3.38	6.68
8	7.29	11.35	7.35	11.47
9	14.92	18.17	15.49	18.57
10	27.79	26.60	28.28	26.84
11	45.15	33.40	45.48	33.69
12	65.25	38.37	65.8	38.61

No description data [9], 1000 observations 50 independent variable

s	Miller CPU	LSSPAR CPU	Miller Real	LSSPAR Real
6	1.375	0.24	1.49	0.26
7	6.85	0.56	7.30	0.60
8	13.90	1.51	14.21	1.69
9	57.09	4.04	58.78	4.68

Finally, using Ozone data, we compare a general purpose state-of-the-art Quadratic Integer Program solver (QIP of MOSEK [10]) and LSSPAR, for a given sparsity level (i.e., we do not search for the best solution for each value of sparsity up to the specified sparsity level as done in the comparison with the Fortran code of Miller).

s	QIP CPU	LSSPAR CPU	QIP Real	LSSPAR Real
4	100.03	0.64	30.57	0.63
5	290.48	1.17	83.06	1.18
6	612.48	2.92	172.2	2.92
7	1751.125	7.23	480.16	7.23

LSSPAR is competitive with Miller's Fortran package when there are many independent variables and especially as the sparsity level s increases. Even if LSSPAR is slower in the beginning, it catches up with Miller's code quickly.

It is also (as expected) much faster than general purpose quadratic integer programming solver option of MOSEK.

We advocate the use of QR decomposition for the solver choice, best-first-search for the search type and "m-st-lsq" for the enumeration rule which are the default choices, respectively, because they are the fastest among all other possibilities. Finally, the difficulty of the problem does not scale with m, but with n and s which determine the number of possible subsets. It is the authors' hope that LSSPAR becomes a new benchmark in subset selection and sparse recovery research.

References

[1] Amir Beck and Yonina C. Eldar. Sparsity constrained nonlinear optimization: Optimality conditions and algorithms. *SIAM Journal on Optimization*, 23(3):1480–1509, July 2013.

[2] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. John Hopkins University, Department of Computer Science, Stanford University; Department of Computer Science, Cornell University, 3rd. edition, 1996.

[3] F.S. Aktaş and M.Ç. Pınar. User's guide to lsspar, 2020.

[4] Alan Miller. Alan miller's fortran software: Subset selection in regression, feb 2004.

[5] YiFei Zhu and Sverker Nilsson. Guppy 3: A python programming environment & heap analysis toolset, github repository, November 2019.

[6] Leo Breiman and Jerome H. Friedma. Estimating optimal transformations for multiple regression and correlation. *Journal of the American Statistical Association*, 80(391), September 1985.

[7] Alan Miller. *Subset Selection in Regression*. CRC, Melbourne, 2002.

[8] George M. Furnival and Jr Robert W. Wilson. Regressions by leaps and bounds. *Technometrics*, 16(4):499–511, November 1974.

[9] Patryk Orzechowski Ryan J. Urbanowicz Randal S. Olson, William La Cava and Jason H. Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10, December 2017.

[10] Erling Andersen, Cees Roos, and Tamas Terlaky. On implementing a primal-dual interior-point method for conic quadratic optimization. *Mathematical Programming Ser. B*, 95(2):249–277, 2003.