User's Guide for LSSPAR

Fatih S. AKTAŞ and Mustafa Ç. Pınar

May 2020

This note explains how to use LSSPAR, the python package based on the paper;

Fatih S. Aktaş and Mustafa Ç. Pınar, "Provably Optimal Sparse Solutions to Overdetermined Linear Systems by Implicit Enumeration"

1 Problem

LSSPAR solves the following problem;

$$\begin{aligned} \min_{x} \quad & ||Ax - b||_2^2 \\ \text{s.t} \quad & \\ & ||x||_0 \leq s, \end{aligned}$$

2 Details of Parameters

The reader/user should skip to section 3 before looking through all of details.

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 it will print the results and the hardware usage. Although it is good to see the progress of the algorithm, it should be noted here that, it slows down the algorithm significantly.

heur (input, by default it is False) = Logical operator, True or False, that decides whether heuristics partial sparse simplex and greedy sparse simplex should be used to solve the problem

- heur = False, the package will not use heuristic algorithms.
- heur = True, the package will use both partial sparse simplex and greedy sparse simplex heuristic algorithms.

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 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 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 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 1) = 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 error 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 branching done, equal to number of nodes in the end of branches, 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.

3 How to use LSSPAR

Suppose A and b are generated as follows;

```
A = numpy.random.normal(0,1,[20,10])
x = numpy.reshape([3,0,0,2,-1,0,0,4,0,0],[10,1])
b = A.dot(x) + numpy.random.normal(0,1,[20,1])
s = 4
```

First initialize the LSSPAR object;

u = LSSPAR(A,b,s)

3.1 Original Problem

We can solve the problem described in section 1 by calling the function;

sol = u.solve()

Algorithm should output something like this;

CPU time of the algorithm 0.0 seconds
Partition of a set of 40 objects. Total size = 4012 bytes.

Index	Count	%	Size	%	Cumulative	%	Kind (class / dict of class)
0	3	8	720	18	720	18	dict (no owner)
1	2	5	480	12	1200	30	dict of function
2	1	2	432	11	1632	41	types.FrameType
3	3	8	408	10	2040	51	function
4	6	15	368	9	2408	60	tuple
5	5	12	240	6	2648	66	builtins.cell
6	3	8	240	6	2888	72	list
7	3	8	192	5	3080	77	types.MethodType
8	2	5	160	4	3240	81	builtins.weakref
9	2	5	160	4	3400	85	functools.partial

<7 more rows. Type e.g. '_.more' to view.>
Error bound on the norm of the x vector 1.674518517897563e-15 Number of correct digits of x 14.776110045391203

The answer is both outputted and also registered to the object.
sol =
[array([2.99837682, 1.96691624, 3.70750323, -0.66505351]),
27.218550874016955,
[0, 3, 7, 4],
4]

Output is given in the following way;

[coefficients, residual of the subset, indexes of the optimal subset, number of nodes searched]

Answers using the attributes;

u.solcoef
u.best_feasible
u.solset
u.rem_qsize

3.2 Multiple Subsets

solve function will find the optimal solution. However, if instead of only the best subset, best k subsets are desired, where $k \leq \binom{n}{s}$, the following function should be called.

u.many = k , the default value is 4
sol = u.solve_multiple()

Algorithm will again first output the cpu and memory usage.

CPU time of the algorithm 0.0 seconds
Partition of a set of 61 objects. Total size = 5644 bytes.

Index	Count	%	Size	%	Cumulative	%	Kind (class / dict of class)
0	7	11	728	13	728	13	list
1	3	5	720	13	1448	26	dict (no owner)
2	5	8	680	12	2128	38	function
3	4	7	512	9	2640	47	numpy.ndarray
4	8	13	496	9	3136	56	tuple
5	2	3	480	9	3616	64	dict of function
6	9	15	432	8	4048	72	builtins.cell
7	1	2	432	8	4480	79	types.FrameType
8	3	5	192	3	4672	83	types.MethodType
9	7	11	168	3	4840	86	float

<7 more rows. Type e.g. '_.more' to view.>

sol =
[[12.575805811707543, 24.27991597719219, 24.47371171944475, 24.6408217488149],
[[7, 0, 3, 4], [7, 0, 3, 9], [7, 0, 3, 1], [7, 0, 3, 2]],
[array([4.18336694, 2.70453523, 2.07150244, -1.03169592]),
array([4.23838793, 2.65954663, 2.73847658, 0.29313048]),
array([4.15389058, 2.80463675, 2.68713511, -0.31152941]),
array([4.22824262, 2.69683354, 2.69794202, -0.27029228])]]

Output is given in the following way;

[residuals , indexes , coefficients]

Solutions are sorted according to residual.

Answers using the attributes;

u.residual_squared
u.indexes
u.coefficients

No error bound on solution is given because there are multiple and bound depends on the solution. It can be calculated manually. For example, for the 2nd best subset;

```
K = numpy.linalg.cond(A[:,sol[1][1]])
error_bound = (K + (sol[0][1] ** 0.5) * K ** 2) * (finfo(float64).eps/2)
correct_digits = -1*log10(error_bound)
```

3.3 All Subsets

If best subsets for i = 1, 2 , .. , s-1, s and at each level best k subsets are desired, where $k \leq n$ instead of solving them separately, the following function can be called;

u.solve_allsubsets()

Which will again, first print out cpu and memory usage of the algorithm. Output will be very crowded but it should look something like this;

```
[[399.0038254062429,
.
.
31.84629390661033],
[[7],
.
.
[7, 0, 3, 2]],
[array([3.13018806]),

.
.
array([ 4.37672242, 3.54437539, 1.61157814, -0.29159787])]]
```

Using the attributes given in Section 3.2, answers can also be obtained.

4 Changing Defaults

4.1 Enforcing Variables

If prior to solving a problem, particular variables are known to be very likely to be in the solution set, or a solution where those particular variables are in the solution set is desired, they can be given as an input while calling the function.

```
enforce_indexes = [1,9]
u.solve(enforce_indexes)

sol =
[array([0.18205294, 0.46104965, 3.92001067, 2.78418166]),
86.26336557193149,
[1, 9, 7, 0],
2]
```

If an invalid or illogical set of indexes are given, they will be ignored and error message will be print. Enforcing variables will also work with "multiple" function but not "all_subsets" function.

4.2 Other Parameters

Make sure to overwrite any default parameters prior to calling any solving function. We suggest that, most of the default parameters are kept for better performance like enumeration and search rules. However, heuristic method, number of solutions to be found and number of enforced variable etc. are completely up to user.