User's Guide for COPL_LP with Python

September 29, 2017

1 Introduction

COPL_LP (Computational Optimization Programming Library: Linear Programming) is an optimization solver for linear programs. In particular, it can solve linear programs of the following form:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0 \end{aligned} \tag{1}$$

where $A \in \mathbb{R}^{m \times n}$,$x \in \mathbb{R}^n$,$c \in \mathbb{R}^n$,$b \in \mathbb{R}^m$. COPL_LP adopts homogeneous primal dual algorithm for implementation, which has been well studied and received great success in the last century. Hence it has the benefit that the algorithm will generate a solution when the problem is feasible, and will detect infeasibility and unboundedness. COPL_LP was firstly coded in C and can be downloaded from https://web.stanford.edu/ yyye/Col.html. **This is the Python version of COPL_LP, which is developed and maintained by SHUFE solver team.**

2 Call COPL_LP in Python

Find the proper version `.pyd` file for your computer copy it to `PythonRoot/Lib/site-packages`.
If you want to compile the code and create the `.pyd` file yourself, find `cython/copllp.pyx`, and run cython code setup.py as follows:

```
python setup.py install
```

Then `.pyd` file and `.egg-info` file will be added to `PythonRoot/Lib/site-packages`.
To call COPL_LP, first import copllp

```
import copllp as lp
```

Then linprg() is the only function, and 'res' is the object returned by linprg(), under which
'fv' : represents the primal optimal value;
'dfv' : represents the dual optimal value;
'optx' : represents the optimal primal solution;
'opty' : represents the optimal dual solution.
We can solve problem in either mps format or numpy format. To solve a mps format file, we can use

```
linprg(mpsfilepath) or  linprg(path = mpsfilepath).
```

While to solve a problem in numpy format(datatype: numpy.array, nmupy.matrix, scipy.matrix, s-cipy.sparse), we need to call linprg() in the following manner.

```
linprg(obj = c,Aineq = A1,bineq = b1,Aeq = A2,beq = b2,lb = L,ub = U).
```

There are four parameters in linprg() can be set: prelevel, docross, showiter and showtime with default values 5, 1, 1, 1 respectively. The meaning of these parameters are explained as follows:

```
    prelevel--------the level of presolve
                0:no presolving is performed.
                1:dependent rows,null and fixed columns.
                2:singleton rows,forcing rows,doubleton rows,dominated rows.
                3:dominated columns.
                4:duplicate rows.
                5:duplicate columns.
    docross --------whether to do 'cross over'
                0  -not
                1  -yes
    showiter--------whether to show information of each iteration
                0  -not
                1  -yes
    showtime--------whether to show information of time
                0  -not
                1  -yes
```

For instance, if we want to exclude the information of each iteration, simply use

```
linprg(mpsfilepath,showiter =0).
```

We can use lp.INF to express infinity and help() to get more information about COPL_LP.

3 Examples in Different Problem Format

3.1 MPS format

The parameter we need to define is the path of mps file. To learn more about mps format, refer to the `C/lpguide.pdf` file or https://en.wikipedia.org/wiki/MPS_(format).
For instance, to solve the problem "25FV47" in netlib, we use variable mpsfile to define the path and then call linprg(). The command lines are shown below:

```
mpsfile = b'25FV47.mps'
    # Since in the same directory, just type the name of the problem
result = lp.linprg(path = mpsfile, showiter = 0)
    # not show information of each iteration
result['fv']
    # optimal primal value
result['optx'][:5]
    # show the first five variables' value of primal optimal solution
```

Consequently, we obtain the information returned by COPL_LP as follows.

Information Returned by COPL_LP for solving problem "25FV47"

```

 *************************** COPL STARTS ***************************


 Exit -- 0: optimal solution found.

 time for initial data input ............................. 0.02 sec
 time for presolve process ............................... 0.01 sec
 time for symbolic computation ........................... 0.00 sec
 time for numerical computation .......................... 0.07 sec
 time for cross-over computation ......................... 0.01 sec
 time for the whole process .............................. 0.10 sec

 **************************** COPL ENDS ****************************

```

```
 Optimal objective value : 5501.8459

Out[8]: 5501.845888300157

Out[9]: [53.13886510118521, 0.0, 0.0, 34.22576854795362, 0.0]
```

3.2 Numpy format

The problem should be formulated as following:

$$
\begin{aligned}
\underset{x}{\text{minimize}} \quad & c^T x \\
\text{subject to} \quad & Aineq * x \leq bineq, \\
& Aeq * x = beq, \\
& l \leq x \leq u
\end{aligned}
\tag{2}
$$

where $Aineq \in \mathbb{R}^{m1 \times n}$, $Aeq \in \mathbb{R}^{m2 \times n}$, $bineq \in \mathbb{R}^{m1}$, $beq \in \mathbb{R}^{m2}$, $x \in \mathbb{R}^n$, $c \in \mathbb{R}^n$. The command line is like:

```
linprg(obj = c,Aineq = A1,bineq = b1,Aeq = A2,beq = b2,lb = L,ub = U).
```

In terms of the datatype of Input :
  Matrix – support scipy.sparse.find() function, like numpy.array or sparsematrix of scipy
  Vector – numpy.array or list.

The following Python code shows how to generate a random LP example in numpy format and then call linpro() to get the solution.

Python code for randomly generating and solving a LP in numpy format

```python
MP = lp.INF
density = 0.1
m1 = int(50)                             # decide the rows of Aineq

m2 = int(ceil(m1 * (rd.random())))   # generate the rows of Aeq
n = int(m1 + m2)                         # generate the dimension of x
x0 = ny.zeros((n, 1))

L = ny.zeros((n, 1))                     # default lower bound(not must)
U = ny.ones((n, 1)) * lp.INF             # default upper bound(not must)

# randomly give variables lowef bound and upper bound
for i in range(n):
    if rd.random() > 0.5:
        idd = ceil(rd.random() * 6)
        if idd == 1:
            L[i] = -MP
        if idd == 1:
            L[i] = rd.uniform(-1000, 1000)
            x0[i] = L[i]+10
        if idd == 3:
            L[i] = -MP
            U[i] = rd.uniform(-1000, 1000)
            x0[i] = U[i] - 10.
        if idd == 4:
            U[i] = rd.uniform(-1000, 1000)
            L[i] = U[i]
            x0[i] = L[i]
        if idd == 5:
```

```
            tem = ny.random.random([2, 1]) * 2000 - 1000
            L[i] = min(tem)
            U[i] = max(tem)
            x0[i] = (L[i]+U[i])/2


A1 = sparse.rand(m1, n, density, 'coo', ny.float64)      # Aineq
A2 = sparse.rand(m2, n, density, 'coo', ny.float64)      # Aeq
b1 = A1 * x0 + ny.random.random(size=(m1, 1))*5         # bineq
b2 = A2 * x0                                             # beq

y1 = ny.random.random(size=(m1, 1))*2 - 2               # dual variables for Aineq
y2 = ny.random.random(size=(m2, 1))*2-1                 # dual variables for Aeq

c = A1.T * y1 + A2.T * y2                                # cost function
# to make this problem feasible
for i in range(n):
    if L[i] <= -MP and U[i] < MP:
        c[i] = c[i] - 10
    if L[i] > -MP and U[i] >= MP:
        c[i] = c[i] + 10

print('SPARSE (%d+%d) * %d, ok' % (m1, m2, n))


res1 = lp.linprg(c,A1,b1,A2,b2,L,U)
res2 = lp.linprg(obj = c,Aineq = A1,bineq = b1,
                 Aeq = A2,beq = b2,lb = L,ub = U)
res3 = lp.linprg(obj = c,Aineq = A1,bineq = b1,lb = L,ub = U)
res4 = lp.linprg(obj = c,Aeq = A2,beq = b2,lb = L,ub = U)
```

4 Numerical Results for Netlib Problems

In this section, we apply both COPL_LP and SDPT3_4.0 to solve 60 Netlib problems, and compare their solution quality and running time. The results are presented in the following table, and the meaning of each column is explained below:

Data_set — problems in netlib
Py_Time — COPL_LP runtime on python
LP_Obj — COPL_LP optimal primal value
LP_Ver — Verdict COPL_LP
S_Time — SDPT3_4.0 runtime
S_Obj — SDPT3_4.0 optimal primal value
SDPT_Ver — Verdict of SDPT3_4.0

From this table, we can see that of COPL_LP passes all instances in Netlib while SDPT3_4.0 fails to converge in many cases. In addition, the running time of COPL_LP is much faster than SDPT3_4.0. This indicates the robustness and efficiency of COPL_LP.

Comparisons of COPL_LP and SDPT3_4.0

Data_set	Py_Time	LP_Obj	LP_Ver	S_Time	S_Obj	SDPT_Ver
25FV47	0.13	5.50e+03	ok	17.91	1.50e−01	diverge
80BAU3B	0.33	9.87e+05	ok	0.64	−1.58e+13	diverge
ADLITTLE	0.03	2.25e+05	ok	0.21	2.25e+05	ok
AFIRO	0.02	−4.65e+02	ok	0.07	−4.65e+02	ok
AGG	0.05	−3.60e+07	ok	0.26	−3.60e+07	ok
AGG2	0.06	−2.02e+07	ok	0.21	−2.02e+07	ok

8	AGG3	0.08	1.03e+07	ok	0.21	1.03e+07	ok
9	BANDM	0.05	−1.59e+02	ok	0.17	−1.59e+02	ok
10	BEACONFD	0.13	3.36e+04	ok	0.08	3.36e+04	ok
11	BLEND	0.11	−3.08e+01	ok	0.07	−3.08e+01	ok
12	BNL1	0.1	1.98e+03	ok	9.0	3.70e−01	diverge
13	BNL2	0.41	1.81e+03	ok	2.21	1.81e+03	ok
14	BOEING1	0.05	−3.35e+02	ok	0.14	4.96e+03	diverge
15	BOEING2	0.02	−3.15e+02	ok	0.12	−2.33e+02	diverge
16	BORE3D	0.02	1.37e+03	ok	0.12	0.00e+00	wrong
17	BRANDY	0.03	1.52e+03	ok	2.14	0.00e+00	diverge
18	CAPRI	0.05	2.69e+03	ok	0.3	1.91e+03	wrong
19	CYCLE	0.25	−5.23e+00	ok	74.04	−1.10e−01	diverge
20	CZPROB	0.1	2.19e+06	ok	0.36	2.18e+06	ok
21	D2Q06C	0.7	1.23e+05	ok	1.6	1.23e+05	ok
22	D6CUBE	0.2	3.15e+02	inaccu	3.83	9.86e+06	diverge
23	DEGEN2	0.06	−1.44e+03	ok	0.14	−1.44e+03	ok
24	DEGEN3	0.31	−9.87e+02	ok	0.63	−9.87e+02	ok
25	DFL001	17.53	1.13e+07	ok	68.39	1.10e+07	wrong
26	E226	0.03	−1.88e+01	ok	0.22	−1.88e+01	ok
27	ETAMACRO	0.05	−7.56e+02	ok	0.07	−1.60e+12	wrong
28	FFFFF800	0.0	5.56e+05	ok	0.39	5.56e+05	ok
29	FINNIS	0.05	1.73e+05	ok	0.29	−7.92e+11	diverge
30	FIT1D	0.06	−9.15e+03	ok	0.06	−3.68e+11	diverge
31	FIT1P	0.33	9.15e+03	ok	0.4	9.15e+03	ok
32	FIT2D	0.66	−6.85e+04	ok	0.17	−8.49e+13	diverge
33	FIT2P	30.32	6.85e+04	ok	1.95	0.00e+00	wrong
34	FORPLAN	0.05	−6.64e+02	wrong	0.13	−1.36e+03	wrong
35	GANGES	0.06	−1.10e+05	ok	0.28	−2.58e+05	wrong
36	GFRD–PNC	0.05	6.90e+06	ok	0.13	0.00e+00	wrong
37	GREENBEA	0.34	−7.26e+07	inaccu	39.28	−1.30e+06	diverge
38	GREENBEB	0.28	−4.30e+06	ok	39.1	−1.31e+06	diverge
39	GROW15	0.05	−1.07e+08	ok	0.05	−7.12e+07	diverge
40	GROW22	0.06	−1.61e+08	ok	0.07	−1.19e+08	diverge
41	GROW7	0.03	−4.78e+07	ok	0.04	−3.36e+07	diverge
42	ISRAEL	0.05	−8.97e+05	ok	0.15	−8.97e+05	ok
43	KB2	0.05	−1.75e+03	ok	0.03	−8.51e+07	diverge
44	LOTFI	0.05	−2.53e+01	ok	0.18	−2.53e+01	ok
45	MAROS–R7	1.19	1.50e+06	ok	2.99	1.50e+06	ok
46	MAROS	0.09	−5.81e+04	ok	0.4	−2.68e+11	diverge
47	MODSZK1	0.0	3.21e+02	inaccu	8.93	5.10e−01	diverge
48	NESM	0.16	1.41e+07	ok	0.33	0.00e+00	wrong
49	PEROLD	0.14	−9.38e+03	ok	1.49	−1.16e+04	diverge
50	PILOT–JA	0.22	−6.11e+03	ok	2.85	−4.00e+06	diverge
51	PILOT–WE	0.16	−2.72e+06	ok	1.65	−2.69e+10	diverge
52	PILOT	1.12	−5.57e+02	ok	3.5	−1.74e+03	diverge
53	PILOT4	0.09	−2.58e+03	ok	1.79	−1.03e+02	diverge
54	PILOT87	4.72	3.02e+02	ok	13.41	1.03e+02	wrong
55	PILOTNOV	0.17	−4.50e+03	ok	2.78	−1.41e+07	wrong
56	QAP12	15.07	5.23e+02	ok	5.73	5.23e+02	ok
57	QAP15	163.87	1.04e+03	ok	30.14	1.04e+03	ok
58	QAP8	0.33	2.04e+02	ok	0.31	2.04e+02	ok
59	RECIPELP	0.0	−2.67e+02	ok	0.03	−2.34e+09	diverge
60	SC105	0.03	−5.22e+01	ok	0.05	−5.22e+01	ok