User's Guide for COPL_LP with Python

October 19, 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, scipy.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"

```
1
2  ************************** COPL STARTS **************************
3
4
5  Exit -- 0: optimal solution found.
6
7  time for initial data input ..............................  0.02 sec
8  time for presolve process ................................  0.01 sec
9  time for symbolic computation ............................  0.00 sec
10 time for numerical computation ...........................  0.07 sec
11 time for cross-over computation ..........................  0.01 sec
12 time for the whole process ...............................  0.10 sec
13
14 ************************** COPL ENDS **************************
15
```

```
 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

```
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 ,Coin-LP1.16 and Gurobi7.5.1 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
CLP_Time — Coin-LP runtime
Gr_Time — Gurobi runtime
Py_Obj — COPL_LP optimal primal value
CLP _Obj — Coin-LP optimal primal value
Gr_Obj — Gurobi optimal primal value
Py-Gr — difference between COPL_LP and Gurobi, 0 means same, 1 means not same

We can see from this table that COPL_LP has a good performance comparing with CLP and Gurobi.

Comparisons of COPL_LP and SDPT3_4.0

Data_set	Py_Time	CLP_Time	Gr_Time	Py_Obj	CLP_Obj	Gr_Obj	Py−Gr
25FV47	0.13	0.22	0.17	5.50e+03	5.50e+03	5.50e+03	0
ADLITTLE	0.03	0.0	0.02	2.25e+05	2.25e+05	2.25e+05	0
AFIRO	0.02	0.0	0.02	−4.65e+02	−4.65e+02	−4.65e+02	0
AGG	0.05	0.01	0.02	2.25e+05	−3.60e+07	−3.60e+07	1
AGG2	0.06	0.01	0.02	−2.02e+07	−2.02e+07	−2.02e+07	0
AGG3	0.08	0.01	0.02	−2.02e+07	1.03e+07	1.03e+07	1
BANDM	0.05	0.01	0.03	−1.59e+02	−1.59e+02	−1.59e+02	0

9	BEACONFD	0.13	0.0	0.02	3.36e+04	3.36e+04	3.36e+04	0
10	BLEND	0.11	0.0	0.02	−3.08e+01	−3.08e+01	−3.08e+01	0
11	BNL1	0.1	0.05	0.07	1.98e+03	1.98e+03	1.98e+03	0
12	BNL2	0.41	0.26	0.1	1.81e+03	1.81e+03	1.81e+03	0
13	BOEING1	0.05	0.02	0.03	−3.35e+02	−2.73e+02	−3.35e+02	0
14	BOEING2	0.02	0.0	0.02	−3.15e+02	−3.17e+02	−3.15e+02	0
15	BORE3D	0.02	0.0	0.02	1.37e+03	7.30e+02	1.37e+03	0
16	BRANDY	0.03	0.01	0.02	1.52e+03	1.52e+03	1.52e+03	0
17	CAPRI	0.05	0.01	0.02	2.69e+03	2.69e+03	2.69e+03	0
18	CYCLE	0.25	0.18	0.05	−5.23e+00	−5.23e+00	−5.23e+00	0
19	CZPROB	0.1	0.1	0.04	2.19e+06	2.19e+06	2.19e+06	0
20	D2Q06C	0.7	1.74	0.48	2.19e+06	1.23e+05	1.23e+05	1
21	D6CUBE	0.2	1.54	0.16	3.15e+02	3.14e+02	3.15e+02	0
22	DEGEN2	0.06	0.02	0.03	−1.44e+03	−1.44e+03	−1.44e+03	0
23	DEGEN3	0.31	0.26	0.17	−9.87e+02	−9.87e+02	−9.87e+02	0
24	DFL001	17.53	8.96	3.05	1.13e+07	1.13e+07	1.13e+07	0
25	E226	0.03	0.01	0.04	−1.88e+01	−1.88e+01	−1.16e+01	1
26	ETAMACRO	0.05	0.02	0.03	−7.56e+02	−3.36e+02	−7.56e+02	0
27	FINNIS	0.05	0.02	0.02	1.73e+05	−6.75e+04	1.73e+05	0
28	FIT1D	0.06	0.04	0.03	−9.15e+03	−9.15e+03	−9.15e+03	0
29	FIT1P	0.33	0.1	0.06	9.15e+03	9.15e+03	9.15e+03	0
30	FIT2D	0.66	4.92	0.18	−6.85e+04	−6.85e+04	−6.85e+04	0
31	FIT2P	30.32	7.38	0.41	−6.85e+04	6.85e+04	6.85e+04	1
32	FORPLAN	0.05	0.01	0.04	−6.64e+02	−6.05e+02	−1.16e+03	1
33	GANGES	0.06	0.08	0.03	−1.10e+05	−1.10e+05	−1.10e+05	0
34	GFRD–PNC	0.05	0.04	0.09	6.90e+06	6.90e+06	6.90e+06	0
35	GREENBEB	0.28	0.79	0.27	−4.30e+06	−4.42e+06	−4.30e+06	0
36	GROW15	0.05	0.06	0.07	−1.07e+08	−1.07e+08	−1.07e+08	0
37	GROW22	0.06	0.15	0.28	−1.61e+08	−1.61e+08	−1.61e+08	0
38	GROW7	0.03	0.01	0.03	−4.78e+07	−4.78e+07	−4.78e+07	0
39	ISRAEL	0.05	0.0	0.02	−8.97e+05	−8.97e+05	−8.97e+05	0
40	KB2	0.05	0.0	0.02	−1.75e+03	−1.75e+03	−1.75e+03	0
41	LOTFI	0.05	0.0	0.02	−2.53e+01	−2.53e+01	−2.53e+01	0
42	MAROS–R7	1.19	2.35	0.85	1.50e+06	1.50e+06	1.50e+06	0
43	MAROS	0.09	0.08	0.05	−5.81e+04	−9.35e+04	−5.81e+04	0
44	NESM	0.16	0.31	0.1	1.50e+06	6.38e+04	1.41e+07	1
45	PEROLD	0.14	0.12	0.11	−9.38e+03	−9.23e+03	−9.38e+03	0
46	PILOT–WE	0.16	0.27	0.21	−2.72e+06	−2.72e+06	−2.72e+06	0
47	PILOT	1.12	1.53	0.66	−5.57e+02	−1.47e+02	−5.57e+02	0
48	PILOT4	0.09	0.07	0.06	−2.58e+03	−2.58e+03	−2.58e+03	0
49	PILOT87	4.72	6.51	1.77	3.02e+02	3.01e+02	3.02e+02	0
50	QAP12	15.07	27.09	1.04	5.23e+02	5.23e+02	5.23e+02	0
51	QAP15	163.87	366.32	5.22	1.04e+03	1.04e+03	1.04e+03	0
52	QAP8	0.33	0.56	0.1	2.04e+02	2.04e+02	2.04e+02	0
53	RECIPELP	0.01	0.0	0.01	−2.67e+02	−2.66e+02	−2.67e+02	0
54	SC105	0.03	0.0	0.01	−5.22e+01	−5.22e+01	−5.22e+01	0
55	SC205	0.04	0.0	0.01	−5.22e+01	−5.22e+01	−5.22e+01	0
56	SC50A	0.02	0.0	0.01	−6.46e+01	−6.46e+01	−6.46e+01	0
57	SC50B	0.01	0.0	0.01	−7.00e+01	−7.00e+01	−7.00e+01	0
58	SCAGR25	0.03	0.01	0.02	−1.48e+07	−1.48e+07	−1.48e+07	0
59	SCAGR7	0.03	0.0	0.01	−2.33e+06	−2.33e+06	−2.33e+06	0
60	SCFXM1	0.05	0.01	0.02	1.84e+04	1.84e+04	1.84e+04	0