Getting things done in Python

Remigiusz Modrzejewski

February 28, 2012

Why bother?

Comparing to Java:

- Easier setup
- No compilation
- Programs are shorter
- Code resembles mathematical formulation

Comparing to Sage:

- Does not require Sage
- Easier to run remotely
- Easier to run distributed
- Easier to integrate with things, that Sage authors did not include yet

Classic problem of LP

My diet requires that all the food I eat come from one of the four "basic food groups": chocolate cake, ice cream, soda, and cheesecake. At present, the following four foods are available for consumption: brownies, chocolate ice cream, cola, and pineapple cheesecake. Each brownie costs $0.50, each scoop of chocolate ice cream costs $0.20, each bottle of cola costs $0.30, and each piece of pineapple cheesecake $0.80. Each day, I must ingest at least 500 calories, 6 oz of chocolate, 10 oz of sugar, and 8 oz of fat. The nutrition content per unit of each food is shown in the table below. Satisfy my daily nutritional requirements at minimum cost.

Operations Research: Applications and Algorithms, 4th Edition, by Wayne L. Winston

Classic LP

A linear program formulation for the crazy diet problem:

$$\begin{array}{lrrrrl}
\min & 0.5b+ & 0.2i+ & 0.3s+ & 0.8c & \\
 & & & & & \\
 & 400b+ & 200i+ & 150s+ & 500c & >= 500 \\
 & 3b+ & 2i & & & >= 6 \\
 & 2b+ & 2i+ & 4s+ & 4c & >= 10 \\
 & 2b+ & 4i+ & s+ & 5c & >= 8
\end{array}$$

Classic LP in Python

An implementation of the linear program:

```
model = LpProblem('Crazy diet', LpMinimize)
model += 0.5*b + 0.2*i + 0.3*s + 0.8*c, "Total cost"

model += 400*b + 200*i + 150*s + 500*c >= 500, "Calories"
model +=   3*b +   2*i                 >= 6, "Chocolate"
model +=   2*b +   2*i +   4*s +   4*c >= 10, "Sugar"
model +=   2*b +   4*i +     s +   5*c >= 8, "Fat"
```

Boilerplate code

```
from pulp import *

b = LpVariable('Brownies', lowBound = 0)
i = LpVariable('Ice cream', 0)
s = LpVariable('Soda', 0)
c = LpVariable('Cheesecake', 0)

# MODEL GOES HERE

model.writeLP('nutrition.lp')
pulp.CPLEX_CMD().solve(model)
print "I need to spend at least %.2f" % \
    model.objective.value()
print "Menu:", b, b.value(), i, i.value(), \
    s, s.value(), c, c.value()
```

Set up

Prerequisites: standard CPLEX (or Gurobi, or GLPK, or Coin) and Python installations. Then run:

```
$ easy_install pulp
# Some installation progress reports
$ python nutrition.py
# Some CPLEX progress reports
I need to spend at least 0.90
Menu: Brownies 0.0 Ice_cream 3.0 Soda 1.0 Cheesecake 0.0
```

Some more useful packages to install: **ipython**, **numpy**, **networkx** and **matplotlib**.

Set cover

Given a set of elements $\mathcal{E} = \{e_1, e_2, \cdots, e_m\}$ and a family of sets $\mathcal{F} = \{s_1, s_2, \cdots s_n\}$ s.t. $\bigcup \mathcal{F} = \mathcal{E}$, find a subset $\mathcal{X}$ of $\mathcal{F}$ s.t. $\bigcup \mathcal{X} = \mathcal{E}$ and $|\mathcal{X}|$ is as small as possible.

$$\min \sum_{i=1}^{m} x_i$$

$$\sum_{\{i|e_j \in s_i\}} x_i \geq 1 \quad ,\forall e_j \in \mathcal{E}$$

$$x_i \in \{0, 1\} \quad ,\forall i \in \{1, 2, \cdots n\}$$

Set Cover ILP in Python

```python
model = LpProblem('Set cover', LpMinimize)

chosen = LpVariable.dicts('Set chosen', sets,
                          0, 1, LpInteger)

model += lpSum(chosen), "Chosen sets"
for j in elements:
    model += lpSum(chosen[i] for i in sets
                   if j in setcontents[i]) >= 1, \
            "Element %s" % j
```

Running the Set Cover script

```
$ cat sets
a b   d
a b c
  b   d e
$ python setcover.py sets
# Some CPLEX progress reports
Need to use at least 2 sets:
['a', 'b', 'c']
['b', 'd', 'e']
```

Set Cover I/O code

```
import sys, itertools
from pulp import *

setcontents = [line.split() for line in
                open(sys.argv[1]).readlines()]
sets = range(len(setcontents))
elements = set(itertools.chain(*setcontents))

# MODEL GOES HERE

pulp.CPLEX_CMD().solve(model)
print "Need to use at least %d sets: " % \
        model.objective.value()
for i in sets:
    if chosen[i].value() == 1:
        print setcontents[i]
```

Integer Multiflow

Given a graph $G = (V, E)$ with edge capacities and a set of demands $\mathcal{D} \subset V \times V$, determine if it is possible to find a path for each $d_s^t \in \mathcal{D}$, from s to t, with no more than c_e leading through any $e \in E$.
We will use graph given by a capacity matrix. Demands will also be given by a matrix stating the numbers of demands between the vertices.

Integer Multiflow ILP

$$\sum_{v \in N_u} f_{v,u}^{s,t} - \sum_{z \in N_u} f_{u,z}^{s,t} = \begin{cases} -d_s^t & u = s \\ d_s^t & u = t \\ 0 & \text{otherwise} \end{cases} \quad , \forall s, t, u \in V$$

$$\sum_{s,t \in V} (f_{u,v}^{s,t} + f_{v,u}^{s,t}) \leq c_{uv} \quad , \forall u \in V, v \in N_u$$

$$f_{u,v}^{s,t} \in \mathbb{Z}_+ \quad , \forall s, t, u \in V, v \in N_u$$

Integer Multiflow ILP in Python: variables

```
model = LpProblem('Integer multi-flow')
flow = LpVariable.dicts('Flow', [(s, t, u, v)
    for s in xrange(n) for t in xrange(n)
    for u in xrange(n) for v in neighbours[u]],
    lowBound = 0, cat = LpInteger)
```

Integer Multiflow ILP in Python: constraints

```
for s in xrange(n):
 for t in xrange(n):
  for u in xrange(n):
   model += lpSum(flow[s, t, v, u] for v in neighbours[u])\
           -lpSum(flow[s, t, u, z] for z in neighbours[u])\
            ==  (-demands[s, t] if u == s \
                 else demands[s, t] if u == t \
                 else 0)
for u in xrange(n):
 for v in neighbours[u]:
  if v < u: # Constrain each edge only once
     model += lpSum(flow[s, t, v, u] + flow[s, t, u, v]
                    for s in xrange(n)
                    for t in xrange(n)) <= capacities[v, u]
```

Integer Multiflow ILP in Python: I/O

```
import sys, numpy
from pulp import *

capacities = numpy.loadtxt(sys.argv[1])
demands = numpy.loadtxt(sys.argv[2])
n = len(capacities)
neighbours = [[v for v in xrange(n)
                 if capacities[u, v] > 0]
              for u in xrange(n)]

# MODEL GOES HERE

pulp.CPLEX_CMD().solve(model)
print "Routing is%s feasible" % \
        ('' if LpStatus[model.status] ==
                  'Optimal' else ' not')
```

Integer Multiflow ILP in Python: visualization

```
import networkx as nx, matplotlib.pyplot as plt
from time import sleep

net = nx.Graph(capacities)
pos = nx.spring_layout(net)
nx.draw(net, pos = pos); plt.draw(); sleep(5)
for s in xrange(n):
  for t in xrange(n):
    if demands[s, t] > 0:
      nx.draw(net, pos = pos)
      route = nx.Graph([(u, v) for u in xrange(n)
                         for v in neighbours[u]
                         if flow[s, t, u, v].value() > 0])
      nx.draw(route, pos = pos,
                    edge_color = 'green', width = 3)
      plt.clf(); plt.draw(); sleep(2)
```

Homework

If you want to play with the scripts, they can be downloaded from: `http://www-sop.inria.fr/members/Remigiusz.Modrzejewski/LPSeminar` (or `http://u.42.pl/2I37`).
Note that the easy_install command in most distributions of Linux requires root priveleges by default. This can be circumvented in a dirty way by two lines in .bashrc (look out when copying, ’ sign must be the one on your keyboard):

```
echo >>~/.bashrc 'export PYTHONPATH=${HOME}/.pylibs'
echo >>~/.bashrc 'alias easy_install="easy_install -d ${PYTHONPATH}"'
source ~/.bashrc
mkdir ${PYTHONPATH}
```

Or in a proper way with virtualenv, what needs some reading.