Optimization with puLp in Python

Dr. Mauricio Acuna, USC

Korean study tour in Australia
Sippy downs, August 2022

What is puLP

Forest Industries
Research Centre

- `PuLP` is a modeling framework for Linear (LP) and Integer Programing (IP) problems written in Python
- Maintained by COIN-OR Foundation (Computational Infrastructure for Operations Research)
- `PuLP` interfaces with Solvers
  - `CPLEX`
  - `COIN`
  - `Gurobi`
  - etc...

puLP example – resource scheduling

USC

Forest Industries
Research Centre

- Consultant for boutique cake bakery that sell 2 types of cakes
- 30 day month
- There is:
  - 1 oven
  - 2 bakers
  - 1 packaging packer – only works 22 days

	Cake A	Cake B
Oven	0.5 days	1 day
Bakers	1 day	2.5 days
Packers	1 day	2 days

	Cake A	Cake B
Profit	$20.00	$40.00

How many cakes A and B to produce to maximize profits?

puLP example – resource scheduling

Forest Industries
Research Centre

- Objective is to Maximize Profit
  - Profit = 20*A + 40*B
- Subject to:
  - $A \geq 0$
  - $B \geq 0$
  - $0.5A + 1B \leq 30$
  - $1A + 2.5B \leq 60$
  - $1A + 2B \leq 22$

Common modeling process for puLP

Forest Industries
Research Centre

1. Initialize Model
2. Define Decision Variables
3. Define the Objective Function
4. Define the Constraints
5. Solve Model

Initiliazing model – LpProblem()

Forest Industries
Research Centre

```
LpProblem(name='NoName', sense=LpMinimize)
```

puLP example – resource scheduling

Forest Industries Research Centre

1. **Initialize Model**

```
from pulp import *

# Initialize Class
model = LpProblem("Maximize Bakery Profits", LpMaximize)
```

Define decision variables – LpVariable()

Forest Industries Research Centre

```
LpVariable(name, lowBound=None, upBound=None, cat='Continuous', e=None)
```

- `name` = Name of the variable used in the output .lp file
- `lowBound` = Lower bound
- `upBound` = Upper bound
- `cat` = The type of variable this is
  - Integer
  - Binary
  - Continuous *(default)*
- `e` = Used for column based modeling

puLP example – resource scheduling

Forest Industries
Research Centre

1. Initialize Class
2. **Define Variables**

```
# Define Decision Variables
A = LpVariable('A', lowBound=0, cat='Integer')
B = LpVariable('B', lowBound=0, cat='Integer')
```

puLP example – resource scheduling

Forest Industries
Research Centre

1. Initialize Class
2. Define Variables
3. **Define Objective Function**

```
# Define Objective Function
model += 20 * A + 40 * B
```

puLP example – resource scheduling

Forest Industries
Research Centre

1. Initialize Class
2. Define Variables
3. Define Objective Function
4. **Define Constraints**

```
# Define Constraints
model += 0.5 * A + 1 * B <= 30
model += 1 * A + 2.5 * B <= 60
model += 1 * A + 2 * B <= 22
```

puLP example – resource scheduling

Forest Industries
Research Centre

1. Initialize Class
2. Define Variables
3. Define Objective Function
4. Define Constraints
5. **Solve Model**

```
# Solve Model
model.solve()
print("Produce {} Cake A".format(A.varValue))
print("Produce {} Cake B".format(B.varValue))
```

puLP example – resource scheduling

USC
Forest Industries
Research Centre

```
from pulp import *

# Initialize Class
model = LpProblem("Maximize Bakery Profits",
                   LpMaximize)

# Define Decision Variables
A = LpVariable('A', lowBound=0,
                cat='Integer')
B = LpVariable('B', lowBound=0,
                cat='Integer')

# Define Objective Function
model += 20 * A + 40 * B
```

```
# Define Constraints
model += 0.5 * A + 1 * B <= 30
model += 1 * A + 2.5 * B <= 60
model += 1 * A + 2 * B <= 22

# Solve Model
model.solve()
print("Produce {} Cake A".format(A.varValue))
print("Produce {} Cake B".format(B.varValue))
```

Using lpSum

Forest Industries Research Centre

Moving from simple to complex

Simple Bakery Example

```
# Define Decision Variables
A = LpVariable('A', lowBound=0, cat='Integer')
B = LpVariable('B', lowBound=0, cat='Integer')
```

More Complex Bakery Example

```
# Define Decision Variables
A = LpVariable('A', lowBound=0, cat='Integer')
B = LpVariable('B', lowBound=0, cat='Integer')
C = LpVariable('C', lowBound=0, cat='Integer')
D = LpVariable('D', lowBound=0, cat='Integer')
E = LpVariable('E', lowBound=0, cat='Integer')
F = LpVariable('F', lowBound=0, cat='Integer')
```

Using lpSum

Forest Industries
Research Centre

```
lpSum(vector)
```

- `vector` = A list of linear expressions

Therefore ...

```
# Define Objective Function
model += 20*A + 40*B + 33*C + 14*D + 6*E + 60*F
```

Equivalent to ...

```
# Define Objective Function
var_list = [20*A, 40*B, 33*C, 14*D, 6*E, 60*F]
model += lpSum(var_list)
```

Using lpSum with list comprehension

Forest Industries
Research Centre

```
# Define Objective Function
cake_types = ["A", "B", "C", "D", "E", "F"]
profit_by_cake = {"A":20, "B":40, "C":33, "D":14, "E":6, "F":60}
var_dict = {"A":A, "B":B, "C":C, "D":D, "E":E, "F":F}

model += lpSum([profit_by_cake[type] * var_dict[type]
                for type in cake_types])
```

lpVariable dictionary function

Forest Industries
Research Centre

Moving from simple to complex

Complex Bakery Example

```
# Define Decision Variables
A = LpVariable('A', lowBound=0, cat='Integer')
B = LpVariable('B', lowBound=0, cat='Integer')
C = LpVariable('C', lowBound=0, cat='Integer')
D = LpVariable('D', lowBound=0, cat='Integer')
E = LpVariable('E', lowBound=0, cat='Integer')
F = LpVariable('F', lowBound=0, cat='Integer')
```

```
# Define Objective Function
var_dict = {"A":A, "B":B, "C":C, "D":D, "E":E, "F":F}

# Define Objective Function
model += lpSum([profit_by_cake[type] * var_dict[type] for type in cake_types])
```

Using LpVariable.dicts()


```
LpVariable(name, indexs, lowBound=None, upBound=None, cat='Continuous')
```

- `name` = The prefix to the name of each LP variable created
- `indexs` = A list of strings of the keys to the dictionary of LP variables
- `lowBound` = Lower bound
- `upBound` = Upper bound
- `cat` = The type of variable this is
  - Integer
  - Binary
  - Continuous *(default)*

LpVariable.dicts() with list comprehension

- `LpVariable.dicts()` often used with Python's list comprehension

Transportation Optimization

```
# Define Decision Variables
customers = ['East','South','Midwest','West']
warehouse = ['New York','Atlanta']
transport = LpVariable.dicts("route", [(w,c) for w in warehouse for c in customers],
                            lowBound=0, cat='Integer')
# Define Objective
model += lpSum([cost[(w,c)]*transport[(w,c)] for w in warehouse for c in customers])
```

Common modeling process for puLP

Forest Industries Research Centre

1. ~~Initialize Model~~
2. ~~Define Decision Variables~~
3. ~~Define the Objective Function~~
4. ~~Define the Constraints~~
5. Solve Model
   - call the `solve()` method
   - check the status of the solution
   - print optimized decision variables
   - print optimized objective function

Solve model – solve method


```
.solve(solver=None)
```

- `solver` = Optional: the specific solver to be used, defaults to the default solver.

Solve model – status of the solution


```
LpStatus[model.status]
```

- **Not Solved:** The status prior to solving the problem.
- **Optimal:** An optimal solution has been found.
- **Infeasible:** There are no feasible solutions (e.g. if you set the constraints x ≤ 1 and x ≥ 2).
- **Unbounded:** The object function is not bounded, maximizing or minimizing the objective will tend towards infinity (e.g. if the only constraint was x ≥ 3).
- **Undefined:** The optimal solution may exist but may not have been found.

Shadow price – Sensitive analysis

Modeling in issues:

- Input for model constraints are often estimates
- Will changes to input change our solution?

Shadow Prices:

- *The change in optimal value of the objective function per unit increase in the right-hand-side for a constraint, given everything else remain unchanged.*

Print shadow price

Python Code:

```
for name, c in list(model.constraints.items()):
        print(c.pi())
```

Shadows price explained

Forest Industries
Research Centre

Output:

```
name   shadow price
 _C1      78.148148
 _C2       2.962963
 _C3      -0.000000
```

Remember the Constraints:

1. *limited production capacity*
2. *limited warehouse capacity*
3. *max production of A*

Constrain slack

`slack` :

- The amount of a resource that is unused.

Code Python:

```
for name, c in list(model.constraints.items()):
        print(c.slack())
```

Constraint slack explained

Output:

```
 name  shadow price     slack
  _C1     78.148148 -0.000000
  _C2      2.962963 -0.000000
  _C3     -0.000000  1.333333
```

More About Binding

- `slack` = 0, then ***binding***
- Changing ***binding*** constraint, ***changes*** solution

Remember the Constraints:

1. *limited production capacity*
2. *limited warehouse capacity*
3. *max production of A*

Reduced cost (opportunity cost)

Forest Industries
Research Centre

- It is the amount by which an objective function coefficient would have to improve (so increase for maximization problem, decrease for minimization problem) before it would be possible for a corresponding variable to assume a positive value in the optimal solution

Code Python:

```
for v in model.variables():
        print(v.dj)
```