

PuLP

Expressing Math Programs in Python

Stuart Mitchell, Michael O'Sullivan
s.mitchell@auckland.ac.nz
michael.osullivan@auckland.ac.nz

Light Metals Research Centre
Auckland, NZ

INFORMS 2010

What is PuLP?

PuLP is a light weight package for Python that allows modelers to easily express Math Programming problems(Currently only LP and MILP).

What is PuLP?

PuLP is a light weight package for Python that allows modelers to easily express Math Programming problems(Currently only LP and MILP).

Aims

- Reduce the time needed to create, improve or experiment with models
- Leverage Python's syntax to increase readability
- Interface with a number of solvers
- Make Math Programming easy for Python programmers
- Allow Math Programmers to use Python to integrate their models into working software

History

2003 Created by J.S. Roy

History

2003 Created by J.S. Roy

2007 Maintained by Stuart Mitchell

History

- 2003 Created by J.S. Roy
- 2007 Maintained by Stuart Mitchell
- 2008 Certified for COIN-OR

History

- 2003 Created by J.S. Roy
- 2007 Maintained by Stuart Mitchell
- 2008 Certified for COIN-OR
- 2009 Officially included in COIN-OR

History

- 2003 Created by J.S. Roy
- 2007 Maintained by Stuart Mitchell
- 2008 Certified for COIN-OR
- 2009 Officially included in COIN-OR
- 2010 Dippy extension linking PuLP and DIP, created by Michael O'Sullivan and Qi-Shan Lim now uploaded into COIN-OR Bazaar

Why Python

- An interpreted language.
- Several language features that make constructing PuLP models easy

- List comprehensions

```
>>> [2**i for i in range(5)]  
[1, 2, 4, 8, 16]
```

- Dictionaries

```
>>> nz_maori['hello']  
'Kia ora'
```

- Classes
- Duck typing
- An extensive standard library and many third party extensions
 - scipy, numpy
 - matplotlib
 - web frameworks

A simple PuLP model

minimize: $z = y - 4x$

subject to:

$$x + y \leq 2$$

$$0 \leq x \leq 3, 0 \leq y \leq 1$$

```
import pulp

x = pulp.LpVariable("x", lowBound=0, upBound=3)
y = pulp.LpVariable("y", 0, 1)
prob = pulp.LpProblem("myProblem", LpMinimize)
prob += y - 4*x
prob += x + y <= 2
status = prob.solve(GLPK(msg=0))
```

Wedding seating - Set partitioning model

$$\text{minimize: } z = \sum_{j \in T} h(j)x_j$$

$$\text{subject to: } \sum_{j \in T} a_{i,j}x_j = 1, \forall i \in G$$

$$\sum_{j \in T} x_j \leq M_T$$

where:

$j \in T$ represents a feasible partition of G

$a_{i,j}$ is 1 if guest i is in table j

$x_j \in \{0, 1\}$ if the partition is chosen

$h(j)$ is the 'happiness' associated with partition j

M_T is the maximum number of tables at the wedding

Wedding seating - PuLP model (wedding.py)

```
#create list of all possible tables  
possible_tables = [tuple(c) for c in  
                    pulp.allcombinations(guests, max_table_size)]
```

Wedding seating - PuLP model (wedding.py)

```
#create list of all possible tables
possible_tables = [tuple(c) for c in
                    pulp.allcombinations(guests, max_table_size)]
```

```
#create a binary variable for each possible table
seating_model = pulp.LpProblem("Wedding Seating Model",
                               pulp.LpMinimize)
x = pulp.LpVariable.dicts('table', possible_tables,
                          lowBound=0, upBound=1, cat=pulp.LpInteger)
seating_model += sum([happiness(table) * x[table]
                      for table in possible_tables])
#specify the maximum number of tables
```

Wedding seating - PuLP model (wedding.py)

```
#create list of all possible tables
possible_tables = [tuple(c) for c in
                    pulp.allcombinations(guests, max_table_size)]
```

```
#create a binary variable for each possible table
seating_model = pulp.LpProblem("Wedding Seating Model",
                                pulp.LpMinimize)
x = pulp.LpVariable.dicts('table', possible_tables,
                           lowBound=0, upBound=1, cat=pulp.LpInteger)
seating_model += sum([happiness(table) * x[table]
                       for table in possible_tables])
#specify the maximum number of tables
```

```
seating_model += sum([x[table] for table in possible_tables]
                       ) <= max_tables, "Maximum_number_of_tables"
#A guest must seated at one and only one table
for guest in guests:
    seating_model += sum([x[table] for table in possible_tables
                           if guest in table]
                           ) == 1, "Must_seat_%s"%guest
```

Wedding seating objective

```
guests = 'A B C D E F G H I J K L M'.split()
max_table_size = 4
max_tables = len(guests)//max_table_size + 1

def happiness(table):
    """
    Find the happiness of the table
    by calculating the maximum distance between the letters
    """
    if len(table) > 1:
        result = max(ord(guest_b) - ord(guest_a)
                     for guest_a in table
                     for guest_b in table
                     if ord(guest_a) < ord(guest_b))
    else:
        result = 0
    return result
```

What is Dippy?

- A python wrapper around DIP
M. Galati and T.K. Ralphs,
A Framework for Decomposition in Integer Programming,
INFORMS 2009
- Allows the expression of DIP models in PuLP
- Allows modelers to create in PuLP
 - (customised) Branch and cut
 - (customised) Price and cut
 - (customised) Branch, cut and decompose
- Exposes the features of DIP without the agony of c++

Original variable formulation

$$\text{minimize: } z = \sum_{j \in T} h_j$$

$$\text{subject to: } \sum_{j \in T} x_{i,j} = 1, \forall i \in G$$

$$\sum_{i \in G} x_{i,j} \leq T_{max}, \forall j \in T$$

$$\sum_{j \in T} x_j \leq M_T$$

$$happy_{i,i'} * (x_{i,j} + x_{i',j} - 1) \leq h_j, \forall j \in T, i \in G, i' \in G$$

where:

T is the set of Tables

$x_{i,j}$ is 1 if guest i is in table j

$happy_{i,i'}$ pairwise happiness of i and i'

Dippy formulation (wedding_dip.py)

```
possible_seatings = [(g, t) for g in guests for t in tables]
seating_model = dippy.DipProblem("Wedding Seating Model (DIP)",
                                pulp.LpMinimize)

#create a binary variable to model a guest at a particular table
x = pulp.LpVariable.dicts('possible_seatings', possible_seatings,
                          lowBound=0, upBound=1, cat=pulp.LpInteger)

#create a set of variables to model the objective function
```

Dippy formulation (wedding_dip.py)

```
possible_seatings = [(g, t) for g in guests for t in tables]
seating_model = dippy.DipProblem("Wedding Seating Model (DIP)",
                                pulp.LpMinimize)

#create a binary variable to model a guest at a particular table
x = pulp.LpVariable.dicts('possible_seatings', possible_seatings,
                          lowBound=0, upBound=1, cat=pulp.LpInteger)
#create a set of variables to model the objective function
```

```
possible_pairs = [(a, b) for a in guests for b in guests
                  if ord(a) < ord(b)]
happy = pulp.LpVariable.dicts('table_happiness', tables,
                              lowBound = 0, upBound = None,
                              cat = pulp.LpContinuous)
seating_model += sum([happy[table] for table in tables])
#a guest must seated at one and only one table
```

Dippy formulation (wedding_dip.py)

```
possible_seatings = [(g, t) for g in guests for t in tables]
seating_model = dippy.DipProblem("Wedding Seating Model (DIP)",
                                pulp.LpMinimize)

#create a binary variable to model a guest at a particular table
x = pulp.LpVariable.dicts('possible_seatings', possible_seatings,
                          lowBound=0, upBound=1, cat=pulp.LpInteger)
#create a set of variables to model the objective function
```

```
possible_pairs = [(a, b) for a in guests for b in guests
                  if ord(a) < ord(b)]
happy = pulp.LpVariable.dicts('table_happiness', tables,
                              lowBound = 0, upBound = None,
                              cat = pulp.LpContinuous)
seating_model += sum([happy[table] for table in tables])
#a guest must seated at one and only one table
```

```
for guest in guests:
    seating_model += sum([x[(guest, table)] for table in tables]) \
                       == 1, "Must_seat_%s"%guest
#specify the maximum number of guests per table
```

Dippy formulation - Constraints for the objective

```
for table in tables:
    seating_model.relaxation[table] += \
        sum([x[(guest, table)] for guest in guests]) \
        <= max_table_size, "Maximum_table_size_%s"%table
#create constraints for each possible pair
for (a, b) in possible_pairs:
    for table in tables:
        seating_model.relaxation[table] += \
            happiness(a, b) * (x[(a, table)] + x[(b, table)] - 1) \
            <= happy[table]
dippy.Solve(seating_model, {'doPriceCut' : '1'})
```

DIP column generation (wedding_dip_solver.py)

```
def relaxed_solver(prob, table, redCosts, convexDual):
```

...

```
    neg_cost = sum(redCosts[x[(g, table)]] for g in possible_table)
    table_happiness = happiness(possible_table[0],
                                possible_table[-1])
    rc = (neg_cost + table_happiness * redCosts[happy[table]]
          - convexDual)
    if rc <= -0.000001:
        var_values = [(x[(g, table)], 1)
                       for g in possible_table]
        var_values.append((happy[table], table_happiness))
        #put the reduced cost and the variable in a list
        dv = dippy.DecompVar(var_values, rc, table_happiness)
        decomp_vars.append((rc, dv))
```

...

```
    return [dv for rc, dv in decomp_vars]
```

Results

Solution times for wedding problem formulations

