TRANSPORT PROBLEMS

CS4491 Introduction to Computational Models with Python

José M. Garrido
Department of Computer Science

January 2016

College of Computing and Software Engineering
Kennesaw State University

1 Model of a Transportation Problem

A transportation problem is formulated as a standard linear optimization problem. The objective function is defined to minimize the cost of transportation, subject to demand and supply constraints.

Assume there are m *supply points* and n *demand points* in a problem. Let $c_{i,j}$ denote the given unit cost of transportation from supply point i to demand (destination) point j. Let $x_{i,j}$ denote the amount of product to be transported from supply point i to demand (destination) point j. The objective function can then be expressed as follows:

Minimize z,

$$z = \sum_{i=1}^{i=m} \sum_{j=1}^{j=n} x_{i,j} c_{i,j}$$

This equation can be expanded and written with equations in which each row of the right-hand side of equation represents the cost of transportation from a supply point. For example, row 1 of the right-hand side of the equation represents the cost of transportation from supply point 1; row 2 represents the cost of transportation from supply point 2, the last row represents the cost of transportation from supply point m.

$$z = \begin{array}{llll} c_{1,1}x_{1,1} & + c_{1,2}x_{1,2} & + \cdots & + c_{1,n}x_{1,n} + \\ c_{2,1}x_{2,1} & + c_{2,2}x_{2,2} & + \cdots & + c_{2,n}x_{2,n} + \\ \vdots & \vdots & \ddots & \vdots \ + \\ c_{m,1}x_{m,1} & + c_{m,2}x_{m,2} & + \cdots & + c_{m,n}x_{m,n} \end{array}$$

In transportation problems there are two types of constraints: *supply constraints* and *demand constraints*. Let s_i denote the amount of product at the supply point i. Let d_j denote the amount of product at the demand point j.

The supply constraints have the right-hand side as an upper bound. There are m supply constraints, the constraint of supply point i is expressed as follows:

$$\sum_{j=1}^{j=n} x_{i,j} \leq s_i$$

This equation can be expanded to show all the quantities of the product to be shipped from supply point i. There are m supply constraints, with each row representing the total quantity of product transported from an indicated supply

point. For example, the first row (1) represents the quantities of product shipped from supply point 1. The second row (2) represents the quantities of product shipped from supply point 2. The last row (m) represents the quantities of product shipped from supply point m.

$$\begin{array}{lllll} x_{1,1} & + x_{1,2} & + \cdots & + x_{1,n} & \leq s_1 \\ x_{2,1} & + x_{2,2} & + \cdots & + x_{2,n} & \leq s_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{m,1} & + x_{m,2} & + \cdots & + x_{m,n} & \leq s_m \end{array}$$

The demand constraints have the right-hand side as a lower bound. There are n demand constraints, the constraint of supply point j is expressed as follows:

$$\sum_{i=1}^{i=m} x_{i,j} \geq d_j$$

This equation can be expanded to show all the quantities of the product to be received at demand point j. There are n demand constraints, with each row representing the total quantity of product to be transported and received by the indicated demand point. For example, the first row (1) represents the quantities of product at demand point 1. The second row (2) represents the quantities of product at demand point 2. The last row (n) represents the quantities of product at demand point n.

$$\begin{array}{lllll} x_{1,1} & + x_{2,1} & + \cdots & + x_{m,1} & \geq d_1 \\ x_{1,2} & + x_{2,2} & + \cdots & + x_{m,2} & \geq d_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{1,n} & + x_{2,n} & + \cdots & + x_{m,n} & \geq d_m \end{array}$$

The decision variables $x_{i,j}$ have sign constraint: $x_{i,j} \geq 0$, for $i = 1, 2, \ldots m$ and $j = 1, 2, \ldots n$.

The following three case studies help to illustrate the modeling of transportation problems.

2 Transportation Case Study 1

The distribution manager of a company needs to minimize global transport costs between a set of three factories (supply points) S1, S2, and S3, and a set of four distributors (demand points) D1, D2, D3, and D4. The following table shows the transportation cost from each supply point to every demand point, the supply of

the product at the supply points, and the demand of the product at the demand points.

	D1	D2	D3	D4	Supply
S1	20	40	70	50	400
S2	100	60	90	80	1500
S3	10	110	30	200	900
Demand	700	600	1000	500	

The transportation unit costs for every supply point are shown from columns 2 to 5. The transportation unit cost from supply point S1 to demand point D1 is 20. The transportation unit cost from supply point S1 to demand point D2 is 40. The transportation unit cost from supply point S2 to demand point D3 is 90, and so on.

The last column in the table shows the supply capacity of the supply point, in quantity of the product. The capacity of supply point S1 is 400. The summation of the values in the last column is the total supply in the system; this value is 2800. The last row of the table shows the demand of each demand point, in quantity of the product. The demand of demand point D1 is 700, of demand point D2 is 600, and so on. The summation of the values in the last row is the total demand in the system; this value is 2800.

Note that the total supply is 2800, and the total demand is also 2800. This is calculated by summing the values of the last column and the last row. Because the value for the amount of product of total supply and the amount of total demand is the same, this transportation problem is said to be *balanced*. The objective function can be expressed as follows:

Minimize z,

$$z = \sum_{i=1}^{1=m} \sum_{j=1}^{j=n} x_{i,j} c_{i,j}$$

This problem has $m = 3$ supply points and $n = 4$ demand points. The objective function can be completely written with the unit cost values of the product to be transported, given in the table shown previously. The objective function is expressed as follows:

$$\begin{array}{rllll} & 20x_{1,1} & + 40x_{1,2} & + 70x_{1,3} & + 50x_{1,4} + \\ z = & 100x_{2,1} & + 60x_{2,2} & + 90x_{2,3} & + 80x_{2,4} + \\ & 10x_{3,1} & + 110x_{3,2} & + 30x_{3,3} & + 200x_{3,4} \end{array}$$

The supply constraints are:

$$\begin{array}{llllll} x_{1,1} & + x_{1,2} & + x_{1,3} & + x_{1,4} & \leq & 400 \\ x_{2,1} & + x_{2,2} & + x_{2,3} & + x_{2,4} & \leq & 1500 \\ x_{3,1} & + x_{3,2} & + x_{3,3} & + x_{3,4} & \leq & 900 \end{array}$$

The demand constraints are:

$$\begin{array}{llll} x_{1,1} & + x_{2,1} & + x_{3,1} & \geq 700 \\ x_{1,2} & + x_{2,2} & + x_{3,2} & \geq 600 \\ x_{1,3} & + x_{2,3} & + x_{3,3} & \geq 1000 \\ x_{1,4} & + x_{2,4} & + x_{3,4} & \geq 500 \end{array}$$

2.1 Formulation Using the Pyomo Modeler

The formulation of this problem using the Pyomo modeler is shown in the following listing and stored in file `transport1.py`. It defines a two dimensional index list, *xindx*, using list comprehension and is shown in line 19. This allows creating two-dimensional decision variables in line 26 and using these variables for specifying the objective function (lines 28–32) and the constraints (lines 35–40 and lines 43–50.

```
 1 """
 2 Python Formulation for the Pyomo Modeler
 3 Example transportation problem. File: transport1.py
 4 J M Garrido, September  2014
 5 usage: pyomo transport1.py --summary
 6 """
 7 print "Transportation Problem 1"
 8 # Import
 9 from coopr.pyomo import *
10
11 # Data for Linear Optimization Problem
12 M = 3  # Supply points
13 N = 4  # Demand points
14 a = range(1, M+1)
15 al = range(M)
16 b = range(1,N+1)
17 bl = range(N)
18 # Index list for decision variables x
19 xindx = [(a[i],b[j]) for j in bl for i in al]
20
21 #Concrete Model
22 model = ConcreteModel(name="Transportation Problem 1")
23
24 #Decision Variables
25 model.x = Var(xindx, within=NonNegativeReals)
26
27 # The objective function
28 model.obj = Objective(expr=
29 20.0 * model.x[1,1] + 40.0 * model.x[1,2] + 70.0 * model.x[1,3] +
```

```
       50.0*model.x[1,4]
30  + 100*model.x[2,1] + 60.0*model.x[2,2] + 90.0*model.x[2,3] +
      80.0*model.x[2,4]
31  +10.0*model.x[3,1] + 110.0*model.x[3,2] + 30.0*model.x[3,3] +
      200*model.x[3,4],
32  sense = minimize)
33
34 # Supply Constraints
35 model.SConstraint1 = Constraint(expr=
36   model.x[1,1] + model.x[1,2] +  model.x[1,3] + model.x[1,4]
      <= 400.0)
37 model.SConstraint2 = Constraint(expr=
38   model.x[2,1] + model.x[2,2] + model.x[2,3] + model.x[2,4]
     <= 1500.0)
39 model.SConstraint3 = Constraint(expr=
40   model.x[3,1] + model.x[3,2] + model.x[3,3] + model.x[3,4]
     <=  900.0)
41
42 # Demand Constraints
43 model.DConst1 = Constraint(expr=
44   model.x[1,1]  +  model.x[2,1] + model.x[3,1]  >= 700.0)
45 model.DConst2 = Constraint(expr=
46   model.x[1,2]  +  model.x[2,2] + model.x[3,2]  >= 600.0)
47 model.DConst3 = Constraint(expr=
48   model.x[1,3]  +  model.x[2,3] + model.x[3,3]  >= 1000)
49 model.DConst4 = Constraint(expr=
50   model.x[1,4] + model.x[2,4] + model.x[3,4]   >= 500.0)
```

Running the model, produces the following output listing. Note that the optimal total transportation cost is $141,000$ and some of the values of x are zero. If $x_{i,j} = 0$, then the amount of the product to be transported from supply point i to demand point j is zero. In this problem, the total demand of demand point D1 is satisfied by the amount 400 from supply point S1, and the amount 300 from supply point S3. There was no supply from supply point D2 to demand point D1, therefore $x_{2,1} = 0$.

```
$ pyomo transport1.py --summary
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
Transportation Problem 1
[    0.00] Creating model
[    0.01] Applying solver
[    0.05] Processing results
```

```
    Number of solutions: 1
    Solution Information
      Gap: 0.0
      Status: feasible
      Function Value: 141000.0
    Solver results file: results.json

============================================================
Solution Summary
============================================================

Model Transportation Problem 1

  Variables:
    x : Size=12, Index=x_index, Domain=NonNegativeReals
        Key    : Lower : Value : Upper : Initial : Fixed : Stale
        (1, 1) :     0 : 400.0 :  None :    None : False : False
        (1, 2) :     0 :   0.0 :  None :    None : False : False
        (1, 3) :     0 :   0.0 :  None :    None : False : False
        (1, 4) :     0 :   0.0 :  None :    None : False : False
        (2, 1) :     0 :   0.0 :  None :    None : False : False
        (2, 2) :     0 : 600.0 :  None :    None : False : False
        (2, 3) :     0 : 400.0 :  None :    None : False : False
        (2, 4) :     0 : 500.0 :  None :    None : False : False
        (3, 1) :     0 : 300.0 :  None :    None : False : False
        (3, 2) :     0 :   0.0 :  None :    None : False : False
        (3, 3) :     0 : 600.0 :  None :    None : False : False
        (3, 4) :     0 :   0.0 :  None :    None : False : False

  Objectives:
    obj : Size=1, Index=None, Active=True
        Key  : Active : Value
        None :   True : 141000.0

  Constraints:
    SConstraint1 : Size=1
        Key  : Lower : Body  : Upper
        None :  None : 400.0 : 400.0
    SConstraint2 : Size=1
        Key  : Lower : Body   : Upper
        None :  None : 1500.0 : 1500.0
    SConstraint3 : Size=1
        Key  : Lower : Body  : Upper
        None :  None : 900.0 : 900.0
    DConst1 : Size=1
```

```
        Key  : Lower : Body  : Upper
        None : 700.0 : 700.0 :  None
    DConst2 : Size=1
        Key  : Lower : Body  : Upper
        None : 600.0 : 600.0 :  None
    DConst3 : Size=1
        Key  : Lower  : Body   : Upper
        None : 1000.0 : 1000.0 :  None
    DConst4 : Size=1
        Key  : Lower : Body  : Upper
        None : 500.0 : 500.0 :  None

[    0.14] Applying Pyomo postprocessing actions
[    0.14] Pyomo Finished
```

2.2 Formulation Using the Pulp Modeler

The following listing shows the formulation of the transportation problem with Pulp and is stored in file `transort1.py` in the directory `pulp_models`. Note that setting the data is similar to the model that was formulated with Pyomo.

```
"""
Python Formulation for the Pulp Modeler
Example transportation problem. File: transport1.py
J M Garrido, September  2014
usage: python transport1.py
"""
print "Transportation Problem 1"
# Import PuLP modeler functions
from pulp import *

# Data for Linear Optimization Problem
M = 3  # Supply points
N = 4  # Demand points
a = range(1, M+1)
al = range(M)
b = range(1,N+1)
bl = range(N)
# Index list for decision variables x
xindx = [(a[i],b[j]) for j in bl for i in al]

# Create the model to contain the problem data
model = LpProblem("Transportation Problem 1",LpMinimize)
```

```
# Decision variables
x = LpVariable.dicts("X", xindx,0,None)

# The Pulp objective function
model += 20.0 * x[1,1] + 40.0 * x[1,2] + 70.0 * x[1,3] + 50.0*x[1,4] \
  + 100*x[2,1] + 60.0*x[2,2] + 90.0*x[2,3] + 80.0*x[2,4] \
  + 10.0*x[3,1] + 110.0*x[3,2] + 30.0*x[3,3] + 200*x[3,4], \
    "Transportation cost"

# Supply Constraints
model += x[1,1] + x[1,2] + x[1,3] + x[1,4] <= 400.0, "Supply Pt 1"
model += x[2,1] + x[2,2] + x[2,3] + x[2,4] <= 1500.0, "Supply Pt 2"
model += x[3,1] + x[3,2] + x[3,3] + x[3,4] <=  900.0, "Supply Pt 3"

# Demand Constraints
model += x[1,1]  +  x[2,1] + x[3,1]  >= 700.0, "Demand Pt 1"
model += x[1,2]  +  x[2,2] + x[3,2]  >= 600.0, "Demand Pt 2"
model += x[1,3]  +  x[2,3] + x[3,3]  >= 1000, "Demand Pt 3"
model += x[1,4] + x[2,4] + x[3,4]   >= 500.0, "Demand Pt 4"

# Solve the optimization problem using the specified PuLP Solver
model.solve(GLPK())

# Print the status of the solution
print "Status:", LpStatus[model.status]

# Print each of the variables with it's resolved optimum value
for v in model.variables():
    print v.name, "=", v.varValue

# Print the optimised value of the objective function
print "Objective Function", value(model.objective)
```

3 Unbalanced Problem: Case Study 2

The transportation problem discussed in the previous section is an example of a balanced problem. In this case, the total supply is equal to the total demand and is expressed mathematically as:

$$\sum_{i=1}^{i=m} s_i = \sum_{j=1}^{j=n} d_j$$

If the transportation problem is not balanced, the total supply may be less than the total demand, or the total supply may be greater than the total demand. When total supply is greater than the total demand, the problem is *unbalanced* and its formulation must include a *dummy demand* point to balance the problem, and the transportation costs to this demand point are zero.

The following problem has a small variation to the one discussed in the previous section. The distribution manager of a company needs to minimize global transport costs between a set of three factories (supply points) S1, S2, and S3, and a set of four distributors (demand points) D1, D2, D3, and D4.

The following table shows the transportation cost from each supply point to every demand point, the supply of the product at the supply points, and the demand of the product at the demand points.

	D1	D2	D3	D4	D5	Supply
S1	20	40	70	50	0	600
S2	100	60	90	80	0	1500
S3	10	110	30	200	0	900
Demand	700	600	1000	500	200	

The last column in the table shows the supply capacity of the supply point, in quantity of the product. The summation of the values in the last column is the total supply in the system, this value is 3000. The last row of the table shows the demand of each demand point, in quantity of the product. The demand of demand point D1 is 700, of demand point D2 is 600, and so on. The summation of the values in the last row is the total demand in the system, this value is 2800.

Note that the total supply is 3000, and the total demand is 2800. This is calculated by summing the values of the last column and the last row. Because the value for the amount of product of total supply is greater than the amount of total demand, this transportation problem is said to be *unbalanced*.

The transportation unit costs for every supply point are shown from columns 2 to 5. The transportation unit cost from supply point S1 to demand point D1 is 20. The transportation unit cost from supply point S1 to demand point D2 is 40. The transportation unit cost from supply point S2 to demand point D3 is 90, and so on.

Because there is an *excess supply* of 200, the formulation of the problem must include a *dummy demand* point, D5, with a demand of 200. The transportation costs to demand point D5 are zero, and this is expressed as follows:

$$c_{i,5} = 0, \quad i = 1, \ldots, 3$$

This problem now has $m = 3$ supply points and $n = 5$ demand points. The objective function can be completely written with the unit cost values of the product

to be transported, given in the table shown previously. The objective function is expressed as follows:

$$\begin{array}{rrrrr} & 20x_{1,1} & + 40x_{1,2} & + 70x_{1,3} & + 50x_{1,4} + \\ z = & 100x_{2,1} & + 60x_{2,2} & + 90x_{2,3} & + 80x_{2,4} + \\ & 10x_{3,1} & + 110x_{3,2} & + 30x_{3,3} & + 200x_{3,4} \end{array}$$

The *supply constraints* have the right-hand side as an upper bound. There are 3 supply constraints, the constraint of supply point i is expressed as follows:

$$\sum_{j=1}^{j=n} x_{i,j} \leq s_i$$

There are 3 supply constraints, with each row representing the total quantity of product transported from an indicated supply point. For example, the first row (1) represents the quantities of product shipped from supply point 1. The second row (2) represents the quantities of product shipped from supply point 2. The last row (3) represents the quantities of product shipped from supply point 3.

$$\begin{array}{rrrrrr} x_{1,1} & + x_{1,2} & + x_{1,3} & + x_{1,4} & + x_{1,5} & \leq 600 \\ x_{2,1} & + x_{2,2} & + x_{2,3} & + x_{2,4} & + x_{2,5} & \leq 1500 \\ x_{3,1} & + x_{3,2} & + x_{3,3} & + x_{3,4} & + x_{3,5} & \leq 900 \end{array}$$

The *demand constraints* have the right-hand side as a lower bound. There are 5 demand constraints, the constraint of supply point j is expressed as follows:

$$\sum_{i=1}^{i=m} x_{i,j} \geq d_j$$

There are 5 demand constraints, with each row representing the total quantity of product to be transported and received by the indicated demand point.

$$\begin{array}{rrrrr} x_{1,1} & + x_{2,1} & + x_{3,1} & \geq & 700 \\ x_{1,2} & + x_{2,2} & + x_{3,2} & \geq & 600 \\ x_{1,3} & + x_{2,3} & + x_{3,3} & \geq & 1000 \\ x_{1,4} & + x_{2,4} & + x_{3,4} & \geq & 500 \\ x_{1,5} & + x_{2,5} & + x_{3,5} & \geq & 200 \end{array}$$

The decision variables $x_{i,j}$ have the sign constraint: $x_{i,j} \geq 0$, for $i = 1, 2, 3$ and $j = 1, 2, \ldots, 5$.

3.1 Formulation with the Pyomo Modeler

The following listing shows the Python script that contains the formulation of the model using Pyomo. The main difference with the previous model is that now there are five demand points (line 13, three additional decision variables, and the index list is now a (3×5) list (line 19. There is also an additional demand constraint (lines 51–52).

```
 1 """
 2 Python Formulation for the Pyomo Modeler
 3 Example unbalanced transportation problem.
     File: transport1u.py
 4 J M Garrido, September  2014
 5 usage: pyomo transport1u.py --summary
 6 """
 7 print "Unbalanced Transportation Problem"
 8 # Import
 9 from coopr.pyomo import *
10
11 # Data for Linear Optimization Problem
12 M = 3  # Supply points
13 N = 5  # Demand points
14 a = range(1, M+1)
15 al = range(M)
16 b = range(1,N+1)
17 bl = range(N)
18 # Index list for decision variables x
19 xindx = [(a[i],b[j]) for j in bl for i in al]
20
21 #Concrete Model
22 model = ConcreteModel(name="Unbalanced Transportation Problem 1")
23
24 #Decision Variables
25 model.x = Var(xindx, within=NonNegativeReals)
26
27 # The objective function
28 model.obj = Objective(expr=
29 20.0 * model.x[1,1] + 40.0 * model.x[1,2] + 70.0 * model.x[1,3] +
       50.0*model.x[1,4]
30  + 100*model.x[2,1] + 60.0*model.x[2,2] + 90.0*model.x[2,3] +
       80.0*model.x[2,4]
31  +10.0*model.x[3,1] + 110.0*model.x[3,2] + 30.0*model.x[3,3] +
       200*model.x[3,4],
32  sense = minimize)
```

```
 33
 34 # Supply Constraints
 35 model.SConstraint1 = Constraint(expr=
 36   model.x[1,1] + model.x[1,2] +  model.x[1,3] + model.x[1,4] +
       model.x[1,5] <= 600.0)
 37 model.SConstraint2 = Constraint(expr=
 38   model.x[2,1] + model.x[2,2] + model.x[2,3] + model.x[2,4] +
       model.x[2,5] <= 1500.0)
 39 model.SConstraint3 = Constraint(expr=
 40   model.x[3,1] + model.x[3,2] + model.x[3,3] + model.x[3,4] +
       model.x[3,5] <=  900.0)
 41
 42 # Demand Constraints
 43 model.DConst1 = Constraint(expr=
 44   model.x[1,1]  +  model.x[2,1] + model.x[3,1]  >= 700.0)
 45 model.DConst2 = Constraint(expr=
 46   model.x[1,2]  +  model.x[2,2] + model.x[3,2]  >= 600.0)
 47 model.DConst3 = Constraint(expr=
 48   model.x[1,3]  +  model.x[2,3] + model.x[3,3]  >= 1000)
 49 model.DConst4 = Constraint(expr=
 50   model.x[1,4] + model.x[2,4] + model.x[3,4]   >= 500.0)
 51 model.DConst5 = Constraint(expr=
 52   model.x[1,5] + model.x[2,5] + model.x[3,5]   >= 200.0)
```

Running the model with Pyomo produces the following output listing.

```
$ pyomo transport1u.py --summary
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
Unbalanced Transportation Problem
[    0.02] Creating model
[    0.02] Applying solver
[    0.05] Processing results
    Number of solutions: 1
    Solution Information
      Gap: 0.0
      Status: feasible
      Function Value: 131000.0
    Solver results file: results.json

============================================================
Solution Summary
============================================================
```

```
Model Unbalanced Transportation Problem 1

  Variables:
    x : Size=15, Index=x_index, Domain=NonNegativeReals
        Key    : Lower : Value : Upper : Initial : Fixed : Stale
        (1, 1) :     0 : 600.0 :  None :    None : False : False
        (1, 2) :     0 :   0.0 :  None :    None : False : False
        (1, 3) :     0 :   0.0 :  None :    None : False : False
        (1, 4) :     0 :   0.0 :  None :    None : False : False
        (1, 5) :     0 :   0.0 :  None :    None : False : False
        (2, 1) :     0 :   0.0 :  None :    None : False : False
        (2, 2) :     0 : 600.0 :  None :    None : False : False
        (2, 3) :     0 : 200.0 :  None :    None : False : False
        (2, 4) :     0 : 500.0 :  None :    None : False : False
        (2, 5) :     0 : 200.0 :  None :    None : False : False
        (3, 1) :     0 : 100.0 :  None :    None : False : False
        (3, 2) :     0 :   0.0 :  None :    None : False : False
        (3, 3) :     0 : 800.0 :  None :    None : False : False
        (3, 4) :     0 :   0.0 :  None :    None : False : False
        (3, 5) :     0 :   0.0 :  None :    None : False : False

  Objectives:
    obj : Size=1, Index=None, Active=True
        Key  : Active : Value
        None :   True : 131000.0

  Constraints:
    SConstraint1 : Size=1
        Key  : Lower : Body  : Upper
        None :  None : 600.0 : 600.0
    SConstraint2 : Size=1
        Key  : Lower : Body   : Upper
        None :  None : 1500.0 : 1500.0
    SConstraint3 : Size=1
        Key  : Lower : Body  : Upper
        None :  None : 900.0 : 900.0
    DConst1 : Size=1
        Key  : Lower : Body  : Upper
        None : 700.0 : 700.0 :  None
    DConst2 : Size=1
        Key  : Lower : Body  : Upper
        None : 600.0 : 600.0 :  None
    DConst3 : Size=1
        Key  : Lower  : Body   : Upper
```

```
        None : 1000.0 : 1000.0 :  None
    DConst4 : Size=1
        Key  : Lower : Body  : Upper
        None : 500.0 : 500.0 :  None
    DConst5 : Size=1
        Key  : Lower : Body  : Upper
        None : 200.0 : 200.0 :  None

[    0.41] Applying Pyomo postprocessing actions
[    0.41] Pyomo Finished
```

3.2 Formulation with the Pulp Modeler

The Python script with the model formulated for Pulp is shown in the following listing. The output listing produced when running the model shows the same results as with the model using the Pyomo modeler.

```
"""
Python Formulation for the Pulp Modeler
Example transport problem. File: transport1u.py
J M Garrido, September  2014
usage: python transport1u.py
"""
print "Unbalanced Transportation Problem 1"
# Import PuLP modeler functions
from pulp import *

# Data for Linear Optimization Problem
M = 3  # Supply points
N = 5  # Demand points
a = range(1, M+1)
al = range(M)
b = range(1,N+1)
bl = range(N)
# Index list for decision variables x
xindx = [(a[i],b[j]) for j in bl for i in al]

# Create the model to contain the problem data
model = LpProblem("Unbalanced Transportation Problem 1",LpMinimize)

# Decision variables
x = LpVariable.dicts("X", xindx,0,None)

# The Pulp objective function
```

```
model += 20.0 * x[1,1] + 40.0 * x[1,2] + 70.0 * x[1,3] + 50.0*x[1,4] \
+ 100*x[2,1] + 60.0*x[2,2] + 90.0*x[2,3] + 80.0*x[2,4] \
+10.0*x[3,1] + 110.0*x[3,2] + 30.0*x[3,3] + 200*x[3,4], \
"Transportation cost"

# Supply Constraints
model += x[1,1] + x[1,2] + x[1,3] + x[1,4] + x[1,5] <= 600.0,
  "Supply Pt 1"
model += x[2,1] + x[2,2] + x[2,3] + x[2,4] + x[2,5] <= 1500.0,
  "Supply Pt 2"
model += x[3,1] + x[3,2] + x[3,3] + x[3,4] + x[3,5] <=  900.0,
  "Supply Pt 3"

# Demand Constraints
model += x[1,1]  +  x[2,1] + x[3,1]  >= 700.0, "Demand Pt 1"
model += x[1,2]  +  x[2,2] + x[3,2]  >= 600.0, "Demand Pt 2"
model += x[1,3]  +  x[2,3] + x[3,3]  >= 1000.0, "Demand Pt 3"
model += x[1,4] + x[2,4] + x[3,4]   >= 500.0, "Demand Pt 4"
model += x[1,5] + x[2,5] + x[3,5]   >= 200.0, "Dummy Demand Pt"

# Solve the optimization problem using the specified PuLP Solver
model.solve(GLPK())

# Print the status of the solution
print "Status:", LpStatus[model.status]

# Print each of the variables with it's resolved optimum value
for v in model.variables():
    print v.name, "=", v.varValue

# Print the optimised value of the objective function
print "Objective Function", value(model.objective)
```

4 Unbalanced Problem: Case Study 3

When the demand exceeds the supply, the problem formulation includes a *penalty* associated with the *unmet demand*. Suppose that in the original problem discussed previously, supply point S1 produces 300 units of the product (instead of 400). This problem is now unbalanced, with an unmet demand of 100.

A dummy supply point, S4 with a supply of 100, is added to the problem formulation. The penalty for unmet demand at demand point D1 is 125; at demand point D2 is 147; at demand point D3 is 95; and at demand point D4 is 255.

The following table shows the transportation cost from each supply point to every

demand point, the supply of the product at the supply points, and the demand of the product at the demand points.

	D1	D2	D3	D4	Supply
S1	20	40	70	50	300
S2	100	60	90	80	1500
S3	10	110	30	200	900
S4	125	147	95	255	100
Demand	700	600	1000	500	

Note that the dummy supply point S4 has been included in the table. The penalty amounts have also been included for this supply point. The total supply and demand is now 2800.

This problem has $m = 4$ supply points and $n = 4$ demand points. The objective function can be completely written with the unit cost values of the product to be transported, given in the table shown previously. The objective function is expressed as follows:

$$z = \begin{array}{rrrr} 20x_{1,1} & + 40x_{1,2} & + 70x_{1,3} & + 50x_{1,4} + \\ 100x_{2,1} & + 60x_{2,2} & + 90x_{2,3} & + 80x_{2,4} + \\ 10x_{3,1} & + 110x_{3,2} & + 30x_{3,3} & + 200x_{3,4} + \\ 125x_{4,1} & + 147x_{4,2} & + 95x_{4,3} & + 255x_{4,4} \end{array}$$

There are 4 supply constraints, with each row representing the total quantity of product transported from an indicated supply point. For example, the first row (1) represents the quantities of product shipped from supply point 1. The second row (2) represents the quantities of product shipped from supply point 2. The last row (4) represents the quantities of product shipped from supply point 4.

$$\begin{array}{rrrrl} x_{1,1} & + x_{1,2} & + x_{1,3} & + x_{1,4} & \leq 300 \\ x_{2,1} & + x_{2,2} & + x_{2,3} & + x_{2,4} & \leq 1500 \\ x_{3,1} & + x_{3,2} & + x_{3,3} & + x_{3,4} & \leq 900 \\ x_{4,1} & + x_{4,2} & + x_{4,3} & + x_{4,4} & \leq 100 \end{array}$$

The demand constraints have the right-hand side as a lower bound. There are 4 demand constraints, with each row representing the total quantity of product to be transported and received by the indicated demand point.

$$\begin{array}{rrrrl} x_{1,1} & + x_{2,1} & + x_{3,1} + & x_{4,1} & \geq 700 \\ x_{1,2} & + x_{2,2} & + x_{3,2} + & x_{4,2} & \geq 600 \\ x_{1,3} & + x_{2,3} & + x_{3,3} + & x_{4,3} & \geq 1000 \\ x_{1,4} & + x_{2,4} & + x_{3,4} + & x_{4,4} & \geq 500 \end{array}$$

The decision variables $x_{i,j}$ have the sign constraint: $x_{i,j} \geq 0$, for $i = 1, \ldots, 4$ and $j = 1, \ldots, 4$.

4.1 Formulation with the Pyomo Modeler

The following listing shows the Python script that contains the formulation of the model using Pyomo and is stored in file `transport1ub.py` (in folder `pyomo`.

```
"""
Python Formulation for the Pyomo Modeler
Example unbalanced transportation problem with excess supply.
File: transport1u.py J M Garrido, September  2014
usage: pyomo transport1ub.py --summary
"""
print "Unbalanced Transportation Problem 2"
# Import
from coopr.pyomo import *

# Data for Linear Optimization Problem
M = 4  # Supply points
N = 4  # Demand points
a = range(1, M+1)
al = range(M)
b = range(1,N+1)
bl = range(N)
# Index list for decision variables x
xindx = [(a[i],b[j]) for j in bl for i in al]

#Concrete Model
model = ConcreteModel(name="Unbalanced Transportation Problem 2")

#Decision Variables
model.x = Var(xindx, within=NonNegativeReals)

# The objective function
model.obj = Objective(expr=
20.0 * model.x[1,1] + 40.0 * model.x[1,2] + 70.0 * model.x[1,3]
    + 50.0*model.x[1,4]
 + 100*model.x[2,1] + 60.0*model.x[2,2] + 90.0*model.x[2,3]
    + 80.0*model.x[2,4]
 +10.0*model.x[3,1] + 110.0*model.x[3,2] + 30.0*model.x[3,3]
    + 200*model.x[3,4]
 +125.0*model.x[4,1] + 147.0*model.x[4,2] + 95.0*model.x[4,3]
    + 255.0*model.x[4,4],
 sense = minimize)

# Supply Constraints
model.SConstraint1 = Constraint(expr=
```

```
  model.x[1,1] + model.x[1,2] +  model.x[1,3] + model.x[1,4] <= 300.0)
model.SConstraint2 = Constraint(expr=
  model.x[2,1] + model.x[2,2] + model.x[2,3] + model.x[2,4] <= 1500.0)
model.SConstraint3 = Constraint(expr=
  model.x[3,1] + model.x[3,2] + model.x[3,3] + model.x[3,4] <=  900.0)
model.SConstraint4 = Constraint(expr=
  model.x[4,1] + model.x[4,2] + model.x[4,3] + model.x[4,4] <=  100.0)

# Demand Constraints
model.DConst1 = Constraint(expr=
  model.x[1,1]  +  model.x[2,1] + model.x[3,1] + model.x[4,1] >= 700.0)
model.DConst2 = Constraint(expr=
  model.x[1,2]  +  model.x[2,2] + model.x[3,2] + model.x[4,2] >= 600.0)
model.DConst3 = Constraint(expr=
  model.x[1,3]  +  model.x[2,3] + model.x[3,3] + model.x[4,3] >= 1000)
model.DConst4 = Constraint(expr=
  model.x[1,4] + model.x[2,4] + model.x[3,4] +model.x[4,4]  >= 500.0)
```

The following listing is produced after running the model with the Pyomo modeler.

```
$ pyomo transport1ub.py --summary
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
Unbalanced Transportation Problem 2
[    0.02] Creating model
[    0.02] Applying solver
[    0.06] Processing results
    Number of solutions: 1
    Solution Information
      Gap: 0.0
      Status: feasible
      Function Value: 146500.0
    Solver results file: results.json

============================================================
Solution Summary
============================================================

Model Unbalanced Transportation Problem 2

  Variables:
    x : Size=16, Index=x_index, Domain=NonNegativeReals
```

```
      Key    : Lower : Value : Upper : Initial : Fixed : Stale
      (1, 1) :     0 : 300.0 :  None :    None : False : False
      (1, 2) :     0 :   0.0 :  None :    None : False : False
      (1, 3) :     0 :   0.0 :  None :    None : False : False
      (1, 4) :     0 :   0.0 :  None :    None : False : False
      (2, 1) :     0 :   0.0 :  None :    None : False : False
      (2, 2) :     0 : 600.0 :  None :    None : False : False
      (2, 3) :     0 : 400.0 :  None :    None : False : False
      (2, 4) :     0 : 500.0 :  None :    None : False : False
      (3, 1) :     0 : 400.0 :  None :    None : False : False
      (3, 2) :     0 :   0.0 :  None :    None : False : False
      (3, 3) :     0 : 500.0 :  None :    None : False : False
      (3, 4) :     0 :   0.0 :  None :    None : False : False
      (4, 1) :     0 :   0.0 :  None :    None : False : False
      (4, 2) :     0 :   0.0 :  None :    None : False : False
      (4, 3) :     0 : 100.0 :  None :    None : False : False
      (4, 4) :     0 :   0.0 :  None :    None : False : False

Objectives:
  obj : Size=1, Index=None, Active=True
      Key  : Active : Value
      None :   True : 146500.0

Constraints:
  SConstraint1 : Size=1
      Key  : Lower : Body  : Upper
      None :  None : 300.0 : 300.0
  SConstraint2 : Size=1
      Key  : Lower : Body   : Upper
      None :  None : 1500.0 : 1500.0
  SConstraint3 : Size=1
      Key  : Lower : Body  : Upper
      None :  None : 900.0 : 900.0
  SConstraint4 : Size=1
      Key  : Lower : Body  : Upper
      None :  None : 100.0 : 100.0
  DConst1 : Size=1
      Key  : Lower : Body  : Upper
      None : 700.0 : 700.0 :  None
  DConst2 : Size=1
      Key  : Lower : Body  : Upper
      None : 600.0 : 600.0 :  None
  DConst3 : Size=1
      Key  : Lower  : Body   : Upper
      None : 1000.0 : 1000.0 :  None
```

```
    DConst4 : Size=1
        Key  : Lower : Body  : Upper
        None : 500.0 : 500.0 :  None

[    0.08] Applying Pyomo postprocessing actions
[    0.08] Pyomo Finished
```

4.2 Formulation with the Pulp Modeler

The following listing shows the Python script with the formulation of the model using Pulp modeler and is stored in file `transport1ub.py` in directory `pulp_models`.

```
"""
Python Formulation for the Pulp Modeler
Example transport problem with excess demand.
File: transport1ub.py
J M Garrido, September  2014
usage: python transport1ub.py
"""
print "Unbalanced Transportation Problem 2"
# Import PuLP modeler functions
from pulp import *

# Data for Linear Optimization Problem
M = 4  # Supply points
N = 4  # Demand points
a = range(1, M+1)
al = range(M)
b = range(1,N+1)
bl = range(N)
# Index list for decision variables x
xindx = [(a[i],b[j]) for j in bl for i in al]

# Create the model to contain the problem data
model = LpProblem("Unbalanced Transportation Problem 2",LpMinimize)

# Decision variables
x = LpVariable.dicts("X", xindx,0,None)

# The Pulp objective function
model += 20.0 * x[1,1] + 40.0 * x[1,2] + 70.0 * x[1,3] + 50.0*x[1,4] \
+ 100*x[2,1] + 60.0*x[2,2] + 90.0*x[2,3] + 80.0*x[2,4] \
+10.0*x[3,1] + 110.0*x[3,2] + 30.0*x[3,3] + 200*x[3,4] \
```

```
+125.0*x[4,1] + 147.0*x[4,2] + 95.0*x[4,3] + 255.0*x[4,4], \
"Transportation cost"

# Supply Constraints
model += x[1,1] + x[1,2] + x[1,3] + x[1,4] <= 300.0,"Supply Pt 1"
model += x[2,1] + x[2,2] + x[2,3] + x[2,4] <= 1500.0, "Supply Pt 2"
model += x[3,1] + x[3,2] + x[3,3] + x[3,4] <=  900.0, "Supply Pt 3"
model += x[4,1] + x[4,2] + x[4,3] + x[4,4] <=  100.0, "Dummy Supply Pt"

# Demand Constraints
model += x[1,1] + x[2,1] + x[3,1] + x[4,1] >= 700.0, "Demand Pt 1"
model += x[1,2] + x[2,2] + x[3,2] + x[4,2] >= 600.0, "Demand Pt 2"
model += x[1,3] + x[2,3] + x[3,3] + x[4,3] >= 1000.0, "Demand Pt 3"
model += x[1,4] + x[2,4] + x[3,4] + x[4,4] >= 500.0, "Demand Pt 4"

# Solve the optimization problem using the specified PuLP Solver
model.solve(GLPK())

# Print the status of the solution
print "Status:", LpStatus[model.status]

# Print each of the variables with it's resolved optimum value
for v in model.variables():
    print v.name, "=", v.varValue

# Print the optimised value of the objective function
print "Objective Function", value(model.objective)
```

5 Transshipment Models

A transshipment model includes *intermediate* or *transshipment points* in a transportation model. A transshipment point is an intermediate point between one or more supply points and one or more demand points. Quantities of a product can be sent from a supply point directly to a demand point or via a transshipment point.

A transshipment point can be considered both, a supply point and a demand point. At a transshipment point, k, the total quantity of product shipped to this point must equal the total quantity of the product shipped from this intermediate point and can be expressed as follows:

$$\sum_{i=1}^{i=m} x_{i,k} = \sum_{j=1}^{j=n} x_{k,j}$$

Where m is the number of supply points, n is the number of demand points, and $x_{i,j}$ is the amount of the product shipped from supply point i to demand point j. From the previous equation, the *transshipment constraint* of a point, k, is expressed as follows:

$$\sum_{i=1}^{i=m} x_{i,k} - \sum_{j=1}^{j=n} x_{k,j} = 0 \tag{1}$$

6 Transshipment Problem: Case Study 4

Various quantities of a product are shipped from two cities (supply points), S1 and S2, to three destinations (demand points), D1, D2, and D3. The products are first shipped to three warehouses (transshipment points), T1, T2, and T3, then shipped to their final destinations.

The following table shows the transportation cost from each supply point to the intermediate points and to every demand point. The table also includes the supply of the product at the supply points, and the demand of the product at the demand points.

	T1	T2	T3	D1	D2	D3	Supply
S1	16	10	12	0	0	0	300
S2	15	14	17	0	0	0	300
T1	0	0	0	6	8	10	0
T2	0	0	0	7	11	11	0
T3	0	0	0	4	5	12	0
Demand	0	0	0	200	100	300	

Note that the total supply of the product is 600 and the total demand is also 600. Therefore, this a balanced problem.

The problem can be formulated directly as a standard transportation problem, using the data in the previous table. The conventional notation for the quantity of product is used and $x_{i,j}$ denotes the quantity of product shipped from point i to point j. For example: $x_{2,3}$ denotes the quantity of product shipped from supply point S2 to transshipment point T3.

This problem has $m = 2$ supply points, $n = 3$ demand points, and 3 transshipment points. Because a transshipment point can be a supply point and a demand point, the problem can be formulated with a total of 5 supply points and 6 demand points.

The objective function can be completely written with the unit cost values of the product to be transported given in the table. The objective function is expressed as follows:

$$
\begin{array}{rrrrrr}
 & 16x_{1,1} & +\,10x_{1,2} & +\,12x_{1,3} & + \\
 & 15x_{2,1} & +\,14x_{2,2} & +\,17x_{2,3} & + \\
z = & 6x_{3,4} & +\,8x_{3,5} & +\,10x_{3,6} & + \\
 & 7x_{4,4} & +\,11x_{4,5} & +\,11x_{4,6} & + \\
 & 4x_{5,4} & +\,5x_{5,5} & +\,12x_{5,6} &
\end{array}
$$

There are three transshipment constraints, one for each transshipment point. These constraints are:

$$
\begin{array}{rcrcrcrcll}
x_{1,1} & + & x_{2,1} & - & x_{3,4} & - & x_{3,5} & - & x_{3,6} & = 0 \quad \text{(T1)} \\
x_{1,2} & + & x_{2,2} & - & x_{4,4} & - & x_{4,5} & - & x_{4,6} & = 0 \quad \text{(T2)} \\
x_{1,3} & + & x_{2,3} & - & x_{5,4} & - & x_{5,5} & - & x_{5,6} & = 0 \quad \text{(T3)}
\end{array}
$$

There are two supply constraints, each representing the total quantity of product transported from an indicated supply point.

$$
\begin{array}{rrrrl}
x_{1,1} & + x_{1,2} & + x_{1,3} & \leq & 300 \\
x_{2,1} & + x_{2,2} & + x_{2,3} & \leq & 300
\end{array}
$$

There are three demand constraints, each representing the total quantity of product transported to the indicated demand point.

$$
\begin{array}{rrrrl}
x_{3,6} & + x_{4,6} & + x_{5,6} & \geq & 300 \\
x_{3,5} & + x_{4,5} & + x_{5,5} & \geq & 100 \\
x_{3,4} & + x_{4,4} & + x_{5,4} & \geq & 200
\end{array}
$$

There are three transshipment constraints applying the balance equations (Equation 1) at each one.

$$
\begin{array}{rrrrrl}
x_{1,1} & + x_{2,1} & - x_{3,4} & - x_{3,5} & - x_{3,6} & = 0 \\
x_{1,2} & + x_{2,2} & - x_{4,4} & - x_{4,5} & - x_{4,6} & = 0 \\
x_{1,3} & + x_{2,3} & - x_{5,4} & - x_{5,5} & - x_{5,6} & = 0
\end{array}
$$

The decision variables $x_{i,j}$ have the sign constraint: $x_{i,j} \geq 0$, for $i = 1, \ldots, 5$ and $j = 1, \ldots, 6$.

6.1 Formulation with the Pyomo Modeler

The following listing shows the Python script with the model using Pyomo and is stored in file `transship1.py`. Note that a list of artificial variables were included in the formulation because Pyomo does not accept constraints with an equal sign. The size of this list, y, is the number of transshipment points. With this list, the right-hand side of the transshipment constraints were written with ≥ 0.0 in lines 55-60.

```
 1 """
 2 Python Formulation for the Pyomo Modeler
 3 Example a transshipment problem.
 4 File: transship1.py - J M Garrido, September  2014
 5 usage: pyomo transship1.py --summary
 6 """
 7 print "Transshipment Problem 1"
 8 # Import
 9 from coopr.pyomo import *
10
11 # Data for Linear Optimization Problem
12 M = 5  # Supply points
13 N = 6  # Demand points
14 NT = 3 # Number of transshipment points
15 a = range(1, M+1)
16 al = range(M)
17 b = range(1,N+1)
18 bl = range(N)
19 # Index list for decision variables x
20 xindx = [(a[i],b[j]) for j in bl for i in al]
21 tindx = range(1, NT+1) # index list for y variables
22
23 #Concrete Model
24 model = ConcreteModel(name="Transshipment Problem 1")
25
26 # Decision Variables
27 model.x = Var(xindx, within=NonNegativeReals)
28 # Artificial variables
29 model.y = Var(tindx, within=NonNegativeReals)
30
31 # The objective function
32 model.obj = Objective(expr=
33 16.0 * model.x[1,1] + 10.0 * model.x[1,2] + 12.0 * model.x[1,3]
34  + 15*model.x[2,1] + 14.0*model.x[2,2] + 17.0*model.x[2,3]
35  +6.0*model.x[3,4] + 8.0*model.x[3,5] + 10.0*model.x[3,6]
36  +7.0*model.x[4,4] + 11.0*model.x[4,5] + 11.0*model.x[4,6]
37  +4.0*model.x[5,4] + 5.0*model.x[5,5] + 12.0*model.x[5,6],
38  sense = minimize)
39
40 # Supply Constraints
41 model.SConstraint1 = Constraint(expr=
42   model.x[1,1] + model.x[1,2] +  model.x[1,3] <= 300.0)
43 model.SConstraint2 = Constraint(expr=
44   model.x[2,1] + model.x[2,2] + model.x[2,3]  <= 300.0)
45
```

```
46 # Demand Constraints
47 model.DConst1 = Constraint(expr=
48   model.x[3,4] + model.x[4,4] + model.x[5,4] >= 200.0)
49 model.DConst2 = Constraint(expr=
50   model.x[3,5] + model.x[4,5] + model.x[5,5] >= 100.0)
51 model.DConst3 = Constraint(expr=
52   model.x[3,6] + model.x[4,6] + model.x[5,6] >= 300.0)
53
54 # Transshipment Constraints
55 model.TConst1 = Constraint(expr=
56   model.x[1,1] + model.x[2,1] - model.x[3,4] - model.x[3,5]
   - model.x[3,6] - model.y[1] >= 0.0)
57 model.TConstraint2 = Constraint(expr=
58   model.x[1,2] + model.x[2,2] - model.x[4,4] - model.x[4,5]
   - model.x[4,6] - model.y[2] >= 0.0)
59 model.TConstraint3 = Constraint(expr=
60   model.x[1,3] + model.x[2,3] - model.x[5,4] - model.x[5,5]
   - model.x[5,6] - model.y[3] >= 0.0)
```

After running the model with Pyomo, the following listing is produced. Note that the value of the objective function is 12400.0, which is the optimal value.

```
$ pyomo transship1.py --summary
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
Transshipment Problem 1
[    0.02] Creating model
[    0.02] Applying solver
[    0.05] Processing results
    Number of solutions: 1
    Solution Information
      Gap: 0.0
      Status: feasible
      Function Value: 12400.0
    Solver results file: results.json

============================================================
Solution Summary
============================================================

Model Transshipment Problem 1

  Variables:
```

```
  x : Size=30, Index=x_index, Domain=NonNegativeReals
      Key    : Lower : Value : Upper : Initial : Fixed : Stale
      (1, 1) :     0 :   0.0 :  None :    None : False : False
      (1, 2) :     0 :   0.0 :  None :    None : False : False
      (1, 3) :     0 : 300.0 :  None :    None : False : False
      (1, 4) :     0 :  None :  None :    None : False :  True
      (1, 5) :     0 :  None :  None :    None : False :  True
      (1, 6) :     0 :  None :  None :    None : False :  True
      (2, 1) :     0 : 300.0 :  None :    None : False : False
      (2, 2) :     0 :  -0.0 :  None :    None : False : False
      (2, 3) :     0 :   0.0 :  None :    None : False : False
      (2, 4) :     0 :  None :  None :    None : False :  True
      (2, 5) :     0 :  None :  None :    None : False :  True
      (2, 6) :     0 :  None :  None :    None : False :  True
      (3, 1) :     0 :  None :  None :    None : False :  True
      (3, 2) :     0 :  None :  None :    None : False :  True
      (3, 3) :     0 :  None :  None :    None : False :  True
      (3, 4) :     0 :   0.0 :  None :    None : False : False
      (3, 5) :     0 :   0.0 :  None :    None : False : False
      (3, 6) :     0 : 300.0 :  None :    None : False : False
      (4, 1) :     0 :  None :  None :    None : False :  True
      (4, 2) :     0 :  None :  None :    None : False :  True
      (4, 3) :     0 :  None :  None :    None : False :  True
      (4, 4) :     0 :   0.0 :  None :    None : False : False
      (4, 5) :     0 :   0.0 :  None :    None : False : False
      (4, 6) :     0 :   0.0 :  None :    None : False : False
      (5, 1) :     0 :  None :  None :    None : False :  True
      (5, 2) :     0 :  None :  None :    None : False :  True
      (5, 3) :     0 :  None :  None :    None : False :  True
      (5, 4) :     0 : 200.0 :  None :    None : False : False
      (5, 5) :     0 : 100.0 :  None :    None : False : False
      (5, 6) :     0 :   0.0 :  None :    None : False : False
  y : Size=3, Index=y_index, Domain=NonNegativeReals
      Key : Lower : Value : Upper : Initial : Fixed : Stale
        1 :     0 :   0.0 :  None :    None : False : False
        2 :     0 :   0.0 :  None :    None : False : False
        3 :     0 :   0.0 :  None :    None : False : False

Objectives:
  obj : Size=1, Index=None, Active=True
      Key  : Active : Value
      None :   True : 12400.0

Constraints:
  SConstraint1 : Size=1
```

```
        Key  : Lower : Body  : Upper
        None :  None : 300.0 : 300.0
    SConstraint2 : Size=1
        Key  : Lower : Body  : Upper
        None :  None : 300.0 : 300.0
    DConst1 : Size=1
        Key  : Lower : Body  : Upper
        None : 200.0 : 200.0 :  None
    DConst2 : Size=1
        Key  : Lower : Body  : Upper
        None : 100.0 : 100.0 :  None
    DConst3 : Size=1
        Key  : Lower : Body  : Upper
        None : 300.0 : 300.0 :  None
    TConst1 : Size=1
        Key  : Lower : Body : Upper
        None :   0.0 :  0.0 :  None
    TConstraint2 : Size=1
        Key  : Lower : Body : Upper
        None :   0.0 :  0.0 :  None
    TConstraint3 : Size=1
        Key  : Lower : Body : Upper
        None :   0.0 :  0.0 :  None

[    0.08] Applying Pyomo postprocessing actions
[    0.08] Pyomo Finished
```

6.2 Formulation with the Pulp Modeler

The following listing shows the Python script with the model using Pulp and is stored in file `transship1.py`. Note that a list of artificial variables were included in the formulation because Pulp does not accept constraints with an equal sign. The size of this list, y, is the number of transshipment points. With this list, the right-hand side of the transshipment constraints were written with ≥ 0.0.

```
"""
Python Formulation for the Pulp Modeler
Transshipment problem. File: transship1.py
J M Garrido, September  2014
usage: python transship1.py
"""
print "Transshipment Problem"
# Import PuLP modeler functions
```

```
from pulp import *

# Data for Linear Optimization Problem
M = 5  # Supply points
N = 6  # Demand points
NT = 3 # Number of transshipment points
a = range(1, M+1)
al = range(M)
b = range(1,N+1)
bl = range(N)
# Index list for decision variables x
xindx = [(a[i],b[j]) for j in bl for i in al]
tindx = range(1, NT+1) # index list for y variables

# Create the model to contain the problem data
model = LpProblem("Transshipment Problem",LpMinimize)

# Decision variables
x = LpVariable.dicts("X", xindx,0,None)
y = LpVariable.dicts("Y", tindx,0, None)

# The Pulp objective function
model += 16.0 * x[1,1] + 10.0 * x[1,2] + 12.0 * x[1,3] \
+ 15.0*x[2,1] + 14.0*x[2,2] + 17.0*x[2,3]  \
+6.0*x[3,4] + 8.0*x[3,5] + 10.0*x[3,6] \
+7.0*x[4,4] + 11.0*x[4,5] + 11.0*x[4,6] \
+4.0*x[5,4] + 5.0*x[5,5] + 12.0*x[5,6], \
"Transportation cost"

# Supply Constraints
model += x[1,1] + x[1,2] + x[1,3] <= 300.0,"Supply Pt 1"
model += x[2,1] + x[2,2] + x[2,3] <= 300.0, "Supply Pt 2"

# Demand Constraints
model += x[3,4] + x[4,4] + x[5,4] >= 200.0, "Demand Pt 1"
model += x[3,5] + x[4,5] + x[5,5] >= 100.0, "Demand Pt 2"
model += x[3,6] + x[4,6] + x[5,6] >= 300.0, "Demand Pt 3"

# Transshipment Constraints
model += x[1,1] + x[2,1] - x[3,4] - x[3,5] -x[3,6] - y[1] >= 0.0,
   "Transshipment Pt 1"
model += x[1,2] + x[2,2] - x[4,4] - x[4,5] -x[4,6] - y[2] >= 0.0,
   "Transshipment Pt 2"
model += x[1,3] + x[2,3] - x[5,4] - x[5,5] -x[5,6] - y[3] >= 0.0,
   "Transshipment Pt 3"
```

```
# Solve the optimization problem using the specified PuLP Solver
model.solve(GLPK())

# Print the status of the solution
print "Status:", LpStatus[model.status]

# Print each of the variables with it's resolved optimum value
for v in model.variables():
    print v.name, "=", v.varValue

# Print the optimized value of the objective function
print "Objective Function", value(model.objective)
```

7 Assignment Problems

An *assignment problem* is a special case of a transportation problem that is formulated as a linear optimization model. The basic goal is to find an optimal assignment of *resources* to tasks. The typical objective is to minimize the total time to complete a task or to minimize the cost of the assignment.

A simple description of the general assignment problem is: minimize the total cost of a set of workers assigned to a set of tasks. The constraints are: each worker is assigned no more that a specified number of jobs, each job requires no more than a specified number of workers.

An assignment of a resource i to a job j is denoted by $x_{i,j}$. Let m be the number of resources and n the number of jobs. The decision variables $x_{i,j}$ have the constraint: $x_{i,j} = 1$ or $x_{i,j} = 0$, for $i = 1 \ldots m$ and $j = 1 \ldots n$. The cost of resource i to complete the job j is denoted by $c_{i,j}$. The total cost of the resource allocation is:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} c_{i,j} x_{i,j}$$

Let P_j denote the maximum number of resources to be assigned to job j. Similarly, let Q_i denote the maximum number of jobs that resource i can be assigned to. In simple problems, all these parameters are equal to 1. The two types of constraints are m *resource constraints* and n *job constraints*. The job constraints are expressed as:

$$\sum_{i=1}^{m} x_{i,j} \leq P_j, \quad j = 1 \ldots n$$

The resource constraints are expressed as:

$$\sum_{j=1}^{n} x_{i,j} \leq Q_i, \quad i = 1 \ldots m$$

8 Assignment Problem: Case Study 5

A factory has 3 machines: M1, M2, and M3. These are to be assigned to four jobs: T1, T2, T3, and T4. The following table is the cost matrix that gives the expected costs when a specific machine is assigned a specific job. The goal of the problem is to optimize the assignment of machines to jobs. In this problem, each machine can only be assigned to one job, and each job can only receive one machine.

	T1	T2	T3	T4
M1	13	16	12	11
M2	15	2	13	20
M3	5	7	10	6

Note that the total number of machines is 3 and the total number of jobs is 4, so this is an unbalanced problem. A dummy machine, M4, is included with zero cost when assigned to a job.

The problem can be formulated directly using the data in the previous table. The conventional notation for an assignment of resource i to job j is used and denoted by $x_{i,j}$. For example: $x_{2,3}$ denotes the assignment of machine M2 to job T3.

The goal of the problem is to minimize the objective function, which is the cost of the assignment of the various machines to the jobs and expressed as follows:

$$\begin{array}{rrrrrr} & 13x_{1,1} & + 16x_{1,2} & + 12x_{1,3} & + 11x_{1,4} & + \\ z = & 15x_{2,1} & + 2x_{2,2} & + 13x_{2,3} & + 20x_{2,4} & + \\ & 5x_{3,1} & + 7x_{3,2} & + 10x_{3,3} & + 6x_{3,4} & + \end{array}$$

The dummy machine assignments are not included in the objective function because their costs are zero. There are four resource constraints, each representing the possible assignments of a machine.

$$\begin{array}{rrrrrrr} x_{1,1} & + x_{1,2} & + x_{1,3} & + x_{1,4} & = 1 & \text{(M1)} \\ x_{2,1} & + x_{2,2} & + x_{2,3} & + x_{2,4} & = 1 & \text{(M2)} \\ x_{3,1} & + x_{3,2} & + x_{3,3} & + x_{3,4} & = 1 & \text{(M3)} \\ x_{4,1} & + x_{4,2} & + x_{4,3} & + x_{4,4} & = 1 & \text{(M4)} \end{array}$$

There are four job constraints, each representing the possible assignments of the machines to it.

$$\begin{array}{llllll} x_{1,1} & + x_{2,1} & + x_{3,1} & + x_{4,1} & = 1 & \text{(T1)} \\ x_{1,2} & + x_{2,2} & + x_{3,2} & + x_{4,2} & = 1 & \text{(T2)} \\ x_{1,3} & + x_{2,3} & + x_{3,3} & + x_{4,3} & = 1 & \text{(T3)} \\ x_{1,4} & + x_{2,4} & + x_{3,4} & + x_{4,4} & = 1 & \text{(T4)} \end{array}$$

The decision variables $x_{i,j}$ have the constraint: $x_{i,j} = 0$ or $x_{i,j} = 1$, for $i = 1, \ldots, 4$ and $j = 1, \ldots, 4$.

8.1 Formulation with the Pyomo Modeler

The following listing shows the Python script with the model using Pyomo and stored in file `assign1.py`. As with the previous models, it was necessary to introduce artificial variables.

```
"""
Python Formulation for the Pyomo Modeler
Assignment problem. File: assign1.py
J M Garrido, September  2014
usage: pyomo assign1.py --summary
"""
print "Assignment Problem 1"
# Import
from coopr.pyomo import *

# Data for Linear Optimization Problem
M = 4  # Number of Jobs
N = 6  # Number of Machines
NAV = M+N # Number of artificial variables
a = range(1, M+1)
al = range(M)
b = range(1,N+1)
bl = range(N)
# Index list for decision variables x
xindx = [(a[i],b[j]) for j in bl for i in al]
tindx = range(1, NAV+1) # index list for y variables

#Concrete Model
model = ConcreteModel(name="Assignment Problem 1")

# Decision Variables
model.x = Var(xindx, within=NonNegativeReals)
```

```
# Artificial variables
model.y = Var(tindx, within=NonNegativeReals)

# The objective function
model.obj = Objective(expr=
13.0 * model.x[1,1] + 16.0 * model.x[1,2] + 12.0 * model.x[1,3]
    + 11.0*model.x[1,4]
 + 15*model.x[2,1] + 2.0*model.x[2,2] + 13.0*model.x[2,3]
    + 20.0*model.x[2,4]
 +5.0*model.x[3,1] + 7.0*model.x[3,2] + 10.0*model.x[3,3]
    + 6.0*model.x[3,4],
 sense = minimize)

# Job Constraints
model.JConstraint1 = Constraint(expr=
  model.x[1,1] + model.x[2,1] +  model.x[3,1] + model.x[4,1]
     - model.y[1] >= 1)
model.JConstraint2 = Constraint(expr=
  model.x[1,2] + model.x[2,2] + model.x[3,2] + model.x[4,2]
     - model.y[2] >= 1)
model.JConstraint3 = Constraint(expr=
  model.x[1,3] + model.x[2,3] + model.x[3,3] + model.x[4,3]
     - model.y[3] >= 1)
model.JConstraint4 = Constraint(expr=
  model.x[1,4] + model.x[2,4] + model.x[3,4] + model.x[4,4]
     - model.y[4] >= 1)

# Machine Constraints
model.MConst1 = Constraint(expr=
  model.x[1,1] + model.x[1,2] + model.x[1,3] + model.x[1,4]
      - model.y[5] >= 1)
model.MConst2 = Constraint(expr=
  model.x[2,1] + model.x[2,2] + model.x[2,3] + model.x[2,4]
      - model.y[6] >= 1)
model.MConst3 = Constraint(expr=
  model.x[3,1] + model.x[3,2] + model.x[3,3] + model.x[3,4]
      - model.y[7] >= 1)
model.MConst4 = Constraint(expr=
  model.x[4,1] + model.x[4,2] + model.x[4,3] + model.x[4,4]
      - model.y[8] >= 1)
```

After running the model with Pyomo, the following listing is products. Note that the optimum value of the objective function is 18.

```
$ pyomo assign1.py --summary
```

```
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
Assignment Problem 1
[    0.00] Creating model
[    0.00] Applying solver
[    0.05] Processing results
    Number of solutions: 1
    Solution Information
      Gap: 0.0
      Status: feasible
      Function Value: 18.0
    Solver results file: results.json

============================================================
Solution Summary
============================================================

Model Assignment Problem 1

  Variables:
    x : Size=24, Index=x_index, Domain=NonNegativeReals
        Key    : Lower : Value : Upper : Initial : Fixed : Stale
        (1, 1) :     0 :   0.0 :  None :    None : False : False
        (1, 2) :     0 :   0.0 :  None :    None : False : False
        (1, 3) :     0 :   0.0 :  None :    None : False : False
        (1, 4) :     0 :   1.0 :  None :    None : False : False
        (1, 5) :     0 :  None :  None :    None : False :  True
        (1, 6) :     0 :  None :  None :    None : False :  True
        (2, 1) :     0 :   0.0 :  None :    None : False : False
        (2, 2) :     0 :   1.0 :  None :    None : False : False
        (2, 3) :     0 :   0.0 :  None :    None : False : False
        (2, 4) :     0 :   0.0 :  None :    None : False : False
        (2, 5) :     0 :  None :  None :    None : False :  True
        (2, 6) :     0 :  None :  None :    None : False :  True
        (3, 1) :     0 :   1.0 :  None :    None : False : False
        (3, 2) :     0 :   0.0 :  None :    None : False : False
        (3, 3) :     0 :   0.0 :  None :    None : False : False
        (3, 4) :     0 :   0.0 :  None :    None : False : False
        (3, 5) :     0 :  None :  None :    None : False :  True
        (3, 6) :     0 :  None :  None :    None : False :  True
        (4, 1) :     0 :   0.0 :  None :    None : False : False
        (4, 2) :     0 :   0.0 :  None :    None : False : False
        (4, 3) :     0 :   1.0 :  None :    None : False : False
        (4, 4) :     0 :   0.0 :  None :    None : False : False
        (4, 5) :     0 :  None :  None :    None : False :  True
```

```
      (4, 6) :     0 :  None :  None :    None : False :  True
  y : Size=10, Index=y_index, Domain=NonNegativeReals
      Key : Lower : Value : Upper : Initial : Fixed : Stale
        1 :     0 :   0.0 :  None :    None : False : False
        2 :     0 :   0.0 :  None :    None : False : False
        3 :     0 :   0.0 :  None :    None : False : False
        4 :     0 :   0.0 :  None :    None : False : False
        5 :     0 :   0.0 :  None :    None : False : False
        6 :     0 :   0.0 :  None :    None : False : False
        7 :     0 :   0.0 :  None :    None : False : False
        8 :     0 :   0.0 :  None :    None : False : False
        9 :     0 :  None :  None :    None : False :  True
       10 :     0 :  None :  None :    None : False :  True

Objectives:
  obj : Size=1, Index=None, Active=True
      Key  : Active : Value
      None :   True :  18.0

Constraints:
  JConstraint1 : Size=1
      Key  : Lower : Body : Upper
      None :   1.0 :  1.0 :  None
  JConstraint2 : Size=1
      Key  : Lower : Body : Upper
      None :   1.0 :  1.0 :  None
  JConstraint3 : Size=1
      Key  : Lower : Body : Upper
      None :   1.0 :  1.0 :  None
  JConstraint4 : Size=1
      Key  : Lower : Body : Upper
      None :   1.0 :  1.0 :  None
  MConst1 : Size=1
      Key  : Lower : Body : Upper
      None :   1.0 :  1.0 :  None
  MConst2 : Size=1
      Key  : Lower : Body : Upper
      None :   1.0 :  1.0 :  None
  MConst3 : Size=1
      Key  : Lower : Body : Upper
      None :   1.0 :  1.0 :  None
  MConst4 : Size=1
      Key  : Lower : Body : Upper
      None :   1.0 :  1.0 :  None
```

```
[    0.09] Applying Pyomo postprocessing actions
[    0.09] Pyomo Finished
```

8.2 Formulation with the Pulp Modeler

The following listing shows the Python script with the model using Pulp and stored in file `assign1.py`. As with the previous models, it was necessary to introduce artificial variables.

```
"""
Python Formulation for the Pulp Modeler
Assignment problem. File: assign1.py
J M Garrido, September  2014
usage: python assign1.py
"""
print "Transshipment Problem"
# Import PuLP modeler functions
from pulp import *

# Data for Linear Optimization Problem
M = 4  # Number of jobs
N = 4  # Number of resources
NAV = M+N # Number of artificial variables
a = range(1, M+1)
al = range(M)
b = range(1,N+1)
bl = range(N)
# Index list for decision variables x
xindx = [(a[i],b[j]) for j in bl for i in al]
tindx = range(1, NAV+1)

# Create the model to contain the problem data
model = LpProblem("Assignment Problem",LpMinimize)

# Decision variables
x = LpVariable.dicts("X", xindx,0,None)
y = LpVariable.dicts("Y", tindx,0,None)

# The Pulp objective function
model += 13.0 * x[1,1] + 16.0 * x[1,2] + 12.0 * x[1,3] + 11.0*x[1,4] \
+ 15.0*x[2,1] + 2.0*x[2,2] + 13.0*x[2,3] + 20.0*x[2,4]  \
+5.0*x[3,1] + 7.0*x[3,2] + 10.0*x[3,3] + 6.0*x[3,4], \
"Assignment cost"
```

```
# Job Constraints
model += x[1,1] + x[2,1] + x[3,1] + x[4,1] -y[1] >= 1,"Job 1"
model += x[1,2] + x[2,2] + x[3,2] + x[4,2] -y[2] >= 1,"Job 2"
model += x[1,3] + x[2,3] + x[3,3] + x[4,3] -y[3] >= 1,"Job 3"
model += x[1,4] + x[2,4] + x[3,4] + x[4,4] -y[4] >= 1,"Job 4"

# Machine Constraints
model += x[1,1] + x[1,2] + x[1,3] + x[1,4] -y[5] >= 1, "Machine 1"
model += x[2,1] + x[2,2] + x[2,3] + x[2,4] -y[6] >= 1, "Machine 2"
model += x[3,1] + x[3,2] + x[3,3] + x[3,4] -y[7] >= 1, "Machine 3"
model += x[4,1] + x[4,2] + x[4,3] + x[4,4] -y[8] >= 1, "Machine 4"

# Solve the optimization problem using the specified PuLP Solver
model.solve(GLPK())

# Print the status of the solution
print "Status:", LpStatus[model.status]

# Print each of the variables with it's resolved optimum value
for v in model.variables():
    print v.name, "=", v.varValue

# Print the optimized value of the objective function
print "Objective Function", value(model.objective)
```