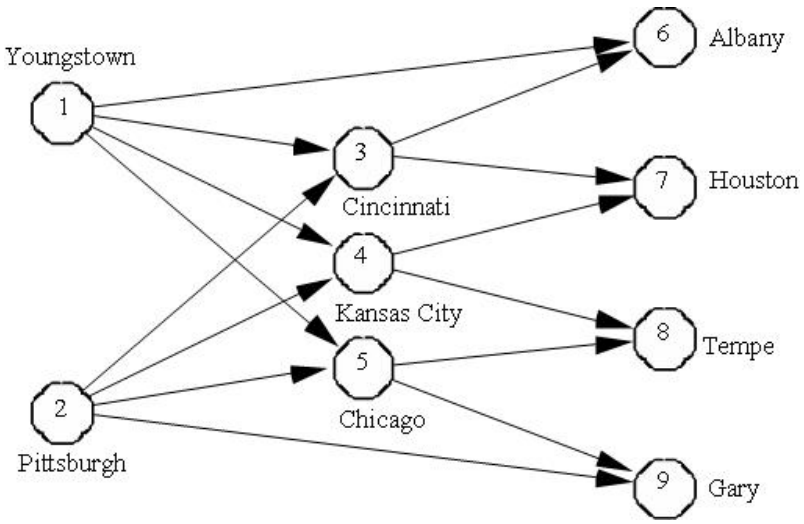
Previous: (5b) A Transportation Problem
Next: (5d) A Facility Location Problem

(5c) A Transshipment Problem

The American Steel Problem

Problem Description

American Steel, an Ohio-based steel manufacturing company, produces steel at its two steel mills located at Youngstown and Pittsburgh. The company distributes finished steel to its retail customers through the distribution network of regional and field warehouses shown below:



The network represents shipment of finished steel from American Steel's two steel mills located at Youngstown (node 1) and Pittsburgh (node 2) to their field warehouses at Albany, Houston, Tempe, and Gary (nodes 6, 7, 8 and 9) through three regional warehouses located at Cincinnati, Kansas City, and Chicago (nodes 3, 4 and 5). Also, some field warehouses can be directly supplied from the steel mills.

The table below presents the minimum and maximum flow amounts of steel that may be shipped between different cities along with the cost per 1000 ton/month of shipping the steel. For example, the shipment from Youngstown to Kansas City is contracted out to a railroad company with a minimal shipping clause of 1000 tons/month. However, the railroad cannot ship more then 5000 tons/month due the shortage of rail cars.

From node	*To node*	*Cost*	*Minimum*	*Maximum*
Youngstown	Albany	500	–	1000
Youngstown	Cincinnati	350	–	3000
Youngstown	Kansas City	450	1000	5000
Youngstown	Chicago	375	–	5000
Pittsburgh	Cincinnati	350	–	2000
Pittsburgh	Kansas City	450	2000	3000
Pittsburgh	Chicago	400	–	4000
Pittsburgh	Gary	450	–	2000
Cincinnati	Albany	350	1000	5000
Cincinnati	Houston	550	–	6000
Kansas City	Houston	375	–	4000
Kansas City	Tempe	650	–	4000
Chicago	Tempe	600	–	2000
Chicago	Gary	120	–	4000

The current monthly demand at American Steel's four field warehouses is as follows:

Field Warehouses	*Monthly Demand*

Albany, N.Y.	3000
Houston	7000
Tempe	4000
Gary	6000

The Youngstown and Pittsburgh mills can produce up to 10,000 tons and 15,000 tons of steel per month, respectively. The management wants to know the least cost monthly shipment plan.

Formulation

1. **Identify the Decision Variables**

   The decision variables for this problem are the same as for the transportation problem, the `Flow` of goods (cases of beer in The Beer Distribution Problem, tons of steels here) through the network. In A Transportation Problem all *arcs* from the supply nodes to the demand nodes existed (although in Forestry Management we used upper bounds to removes some arcs). In The American Steel Problem, the network has *transshipment nodes* and arcs don't exist between all nodes. To cater for this, we explicitly name all our arcs - these are our decision variables.

2. **Formulate the Objective Function**

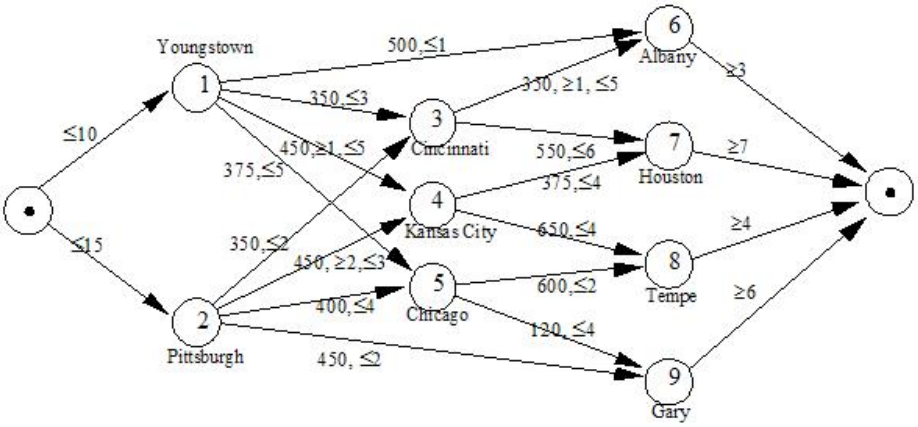
   The objective of transshipment problems in general and The American Steel Problem in particular is to minimise the cost of shipping goods through the network:

3. **Formulate the Constraints**

   All the nodes have supply and demand, demand = 0 for supply nodes, supply = 0 for demand nodes and supply = demand = 0 for transshipment nodes.

   The only constraints in the transshipment problem are *flow conservation* constraints. These constraints simply state that the flow of goods into a node must be greater than or equal to the flow of goods out of a node.

   Transshipment problems are often presented as a network formulation:



Solution

As always, your file will start with an introduction and the import statement.

```
"""
The American Steel Problem for the PuLP Modeller


Authors: Antony Phillips, Dr Stuart Mitchell   2007
"""


# Import PuLP modeller functions
from pulp import *
```

A list of all the node names is created. Whilst the nodes are not the problem variables, they are important in formulating the constraints. A dictionary with each node as the reference key is created, and a list containing the supply and demand of that node is the stored data.

```
# List of all the nodes
Nodes = ["Youngstown",
         "Pittsburgh",
         "Cincinatti",
         "Kansas City",
         "Chicago",
         "Albany",
         "Houston",
         "Tempe",
         "Gary"]



nodeData = {# NODE        Supply Demand
         "Youngstown":    [10000,0],
         "Pittsburgh":    [15000,0],
         "Cincinatti":    [0,0],
         "Kansas City":   [0,0],
         "Chicago":       [0,0],
         "Albany":        [0,3000],
         "Houston":       [0,7000],
         "Tempe":         [0,4000],
         "Gary":          [0,6000]
         }
```

A list of all the arc names is created, and a dictionary containing; the cost per ton of steel sent on that arc, and the bounds, is created. The cost values are in $ per ton, since the other quantities are in tonnes, not thousands of tonnes.

```
# List of all the arcs
Arcs = [("Youngstown","Albany"),
        ("Youngstown","Cincinatti"),
        ("Youngstown","Kansas City"),
        ("Youngstown","Chicago"),
        ("Pittsburgh","Cincinatti"),
        ("Pittsburgh","Kansas City"),
        ("Pittsburgh","Chicago"),
        ("Pittsburgh","Gary"),
        ("Cincinatti","Albany"),
        ("Cincinatti","Houston"),
        ("Kansas City","Houston"),
        ("Kansas City","Tempe"),
        ("Chicago","Tempe"),
        ("Chicago","Gary")]



arcData = { #      ARC                Cost Min Max
        ("Youngstown","Albany"):      [0.5,0,1000],
        ("Youngstown","Cincinatti"):  [0.35,0,3000],
        ("Youngstown","Kansas City"): [0.45,1000,5000],
        ("Youngstown","Chicago"):     [0.375,0,5000],
        ("Pittsburgh","Cincinatti"):  [0.35,0,2000],
        ("Pittsburgh","Kansas City"): [0.45,2000,3000],
        ("Pittsburgh","Chicago"):     [0.4,0,4000],
        ("Pittsburgh","Gary"):        [0.45,0,2000],
        ("Cincinatti","Albany"):      [0.35,1000,5000],
        ("Cincinatti","Houston"):     [0.55,0,6000],
        ("Kansas City","Houston"):    [0.375,0,4000],
        ("Kansas City","Tempe"):      [0.65,0,4000],
        ("Chicago","Tempe"):          [0.6,0,2000],
        ("Chicago","Gary"):           [0.12,0,4000]
        }
```

The `splitDict` function is used to create separate dictionaries from the dictionary inputs which had lists as the stored data. This splits up the lists so that (for example) 'Supply["Youngstown"]' will return `10000`, instead of having to use nodeData["Youngstown"][1] to get the value.

```
# Splits the dictionaries to be more understandable
(supply, demand) = splitDict(nodeData)
(costs, mins, maxs) = splitDict(arcData)
```

The problem variables are created with the usual parameters. "Route" is the arbitrary name for the category the problem variables fall into. The names in the list `Arcs` will form the main part of the problem variable names. It is important to note that both the lower and upper bounds have been set to `None`. This is because the bounds on each of the variables is different and so we cannot express what they individually are, in this line of code. Lastly, we specify that the solution must be integer, since it is fair to assume that each ton is inseparable. This is a certain degree of rounding which is appropriate.

```
# Creates the boundless Variables as Integers
vars = LpVariable.dicts("Route",Arcs,None,None,LpInteger)
```

The bounds on the variables are created by modifying the property `bounds` of the objects in `Vars`. There is no equals sign required for this. It could also have been done using 'Vars[a].lower([Mins[a])' and 'Vars[a].lower([Maxs[a])'

```
# Creates the upper and lower bounds on the variables
for a in Arcs:
    vars[a].bounds(mins[a], maxs[a])
```

The variable `prob` is created to store the problem data.

```
# Creates the prob variable to contain the problem data
prob = LpProblem("American Steel Problem",LpMinimize)
```

The objective function is added to prob. This is simply the cost of sending a tonne down each arc multiplied by the number of tonnes sent down.

```
# Creates the objective function
prob += lpSum([vars[a] * costs[a] for a in Arcs]), "Total Cost of Transport"
```

The constraints are all added in this statement. This ensures that the amount of steel flowing into a node is greater than or equal to the amount exiting. This caters for the fact that this problem is unbalanced - it has excess supply. There is no merit, and added cost, in sending steel to a node and not sending is out, so the excess supply will stay at the mills. The code should read as: The amount of steel the node is creating + the amount of steel on each of the arcs leading into the node must be greater than or equal to the amount of steel demanded by the node plus the amount of steel on each of the arcs exiting the node. Since the Arcs are stored as tuples with the "from" and "to" node names stored, when `for (i,j) in Arcs if j==n` is written, it says that we only look at the Arcs which have "n" (the node we are currently constraining) as the "to" node.

```
# Creates all problem constraints - this ensures the amount going into each node is at least equal to the amount leaving
for n in Nodes:
    prob += (supply[n]+ lpSum([vars[(i,j)] for (i,j) in Arcs if j == n]) >=
             demand[n]+ lpSum([vars[(i,j)] for (i,j) in Arcs if i == n])), "Steel Flow Conservation in Node:%s"%n
```

Now the formulation is complete, the standard end to the code can be written, starting at the prob.writeLP line. The full code is available here.

Post-Optimal Analysis

Validation

For our solution to be valid we need it to be integer. Observing the flow values shows that it is in fact integer. In fact, any network flow problem with integer supplies, demands and arc capacities has naturally integer solutions.

Presentation of Solution and Analysis

There is quite a bit of information to summarise and many ways to present it. Some suggestions include:

1. Summarise the problem as usual and list the shipments that American Steel should make (similar to the transportation problem);
2. Summarise the problem as usual and present a table of shipments that American Steel should make;
3. Draw the network formulation for the problem (being sure to specify what the labels mean). Then draw the actual solution on top of the network formulation. You could colour code flows long arcs to show if they are at their bounds.

Implementation and Ongoing Monitoring

Ongoing monitoring of the supply, demand and bounds will help American Steel to keep making good decisions.

Previous: (5b) A Transportation Problem

Next: (5d) A Facility Location Problem