NETWORK 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 Graphs

A *graph* is used as a visual representation of a network. A graph consists of a finite set of *nodes* (also known as vertices) and a finite set of *arcs* that connect pairs of nodes. A directed arc connects an ordered pair of vertices; if an arc starts at node P (head) and ends at node Q (tail), it is denoted as (P, Q). An arc will typically have an associated weight or length.

A *path* is a sequence of arcs with the property that for every arc its tail vertex is the head vertex of the next arc. The length (or cost) of the path is the summation of the lengths of the individual arcs in it.

2 Shortest Path Problem

A shortest path problem consists of finding the path from an initial node in the graph to a final node with the shortest length. Several algorithms have been developed to solve this general problem, the most important of which is Dijkstra's algorithm.

This section discusses the formulation of shortest path problems as transshipment problems. For this, the techniques of the previous chapters are applied. As discussed previously, 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.

The general problem consists of transporting one unit of a product from a source point P to a destination point Q. The intermediate nodes are the transshipment points. Units of the product can be sent from the source point to the destination point using one of several possible paths.

When in a graph there is no arc between two points, an *artificial arc* is included and its length is given a relatively large value, H. An arc from a node to the same node will have zero length.

The *cost* of sending 1 unit of product from node i to node j is the length of the arc and denoted by $c_{i,j}$. For example: $c_{2,3}$ denotes the cost of sending 1 unit of product shipped from node 2 to node 3. The shipment of 1 unit of the product from node i to node j is denoted by $x_{i,j}$. Therefore, the value of $x_{i,j}$ is 1 or zero.

The objective function indicates the total cost of transporting 1 unit of the product from the source node to the destination node and can be expressed as follows:

Minimize: z,

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

The number of nodes is n and there is one supply point and one demand point. A transshipment point can be considered both a supply point and a demand point. At a transshipment point, k, the total inputs to this point must equal the total outputs from this intermediate point and can be expressed as follows:

$$\sum_{i=1}^{i=p} x_{i,k} = \sum_{j=1}^{j=q} x_{k,j}$$

Where p is the number of supply points, q is the number of demand points, and $x_{i,j}$ denotes a unit of (1) product or no product (0) shipped from point i to point j. From the previous equation, the transshipment constraint of a point, k, is expressed as follows:

$$\sum_{i=1}^{i=p} x_{i,k} - \sum_{j=1}^{j=q} x_{k,j} = 0$$

3 Shortest Path: Case Study 1

A product is shipped from a supply city (supply point), represented by node 1, to a destination city (demand point), represented by node 6. The product is first shipped to one or more cities that are represented by intermediate nodes 2–5. The goal of the problem is to find the shortest path from the supply city to the destination city. The distance between the cities is shown by the arcs between nodes in the graph of Figure 1.

The problem can be formulated directly as a standard transportation problem, using the data in the previous figure. This problem has one supply point and one demand point. The objective function can be completely written with the unit cost values of the product to be transported. The objective function is to minimize the following expression:

$$z = \begin{array}{rrrrrrr} 0x_{1,1} & +\ 4x_{1,2} & +\ 2x_{1,3} & +\ Hx_{1,4} & +\ Hx_{1,5} & +\ Hx_{1,6} & + \\ Hx_{2,1} & +\ 0x_{2,2} & +\ Hx_{2,3} & +\ 5x_{2,4} & +\ Hx_{2,5} & +\ Hx_{2,6} & + \\ Hx_{3,1} & +\ 1x_{3,2} & +\ 0x_{3,3} & +\ 8x_{3,4} & +\ 10x_{3,5} & +\ Hx_{3,6} & + \\ Hx_{4,1} & +\ Hx_{4,2} & +\ Hx_{4,3} & +\ 0x_{4,4} & +\ 2x_{4,5} & +\ 6x_{4,6} & + \\ Hx_{5,1} & +\ Hx_{5,2} & +\ Hx_{5,3} & +\ Hx_{5,4} & +\ 0x_{5,5} & +\ 2x_{5,6} & + \\ Hx_{6,1} & +\ Hx_{6,2} & +\ Hx_{6,3} & +\ Hx_{6,4} & +\ Hx_{6,5} & +\ 0x_{6,6} & \end{array}$$

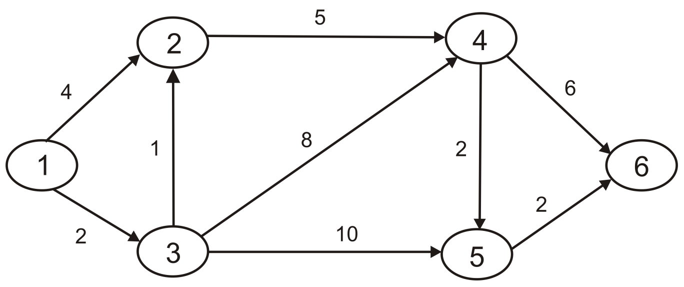

Figure 1: Graph of shortest path problem.

There are two source-destination constraints, each representing 1 unit of product transported from the indicated source node to the indicated destination node.

$$\begin{array}{llll} x_{1,2} & + x_{1,3} & = & 1 \quad \text{(Source node)} \\ x_{4,6} & + x_{5,6} & = & 1 \quad \text{(Destination node)} \end{array}$$

There are four intermediate nodes, so there are four transshipment constraints, one for each intermediate node. These constraints are:

$$\begin{array}{llllllllll} x_{1,2} & + & x_{3,2} & - & x_{4,2} & & & = & 0 & \text{(Node 2)} \\ x_{1,3} & - & x_{3,2} & - & x_{3,4} & - & x_{3,5} & = & 0 & \text{(Node 3)} \\ x_{2,4} & + & x_{3,4} & - & x_{4,5} & - & x_{4,6} & = & 0 & \text{(Node 4)} \\ x_{3,5} & + & x_{4,5} & - & x_{5,6} & & & = & 0 & \text{(Node 5)} \end{array}$$

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

3.1 Formulation Using the Pyomo Modeler

The following listing shows the formulation of the model using the Pyomo modeler. The model is stored in file `network1.py`

```
"""
Python Formulation using Pyomo Modeler
Network problem, shortest path. File: network1.py
J M Garrido, September  2014
usage: pyomo network1.py --summary
"""
print "Assignment Problem 1"
# Import
```

```
from coopr.pyomo import *

# Data for Linear Optimization Problem
N = 6  # Number of nodes in network
M = 2  # number of end nodes (source and destination)
INT = 4  # Number of intermediate nodes
H = 10000.0 # A very high cost constant
a = range(1, N+1)
al = range(N)
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]
T = INT + M # number of artificial variables (y)
tindx = range(1, T+1) # index list for y variables

#Concrete Model
model = ConcreteModel(name="Shortest Path 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=
0.0 * model.x[1,1] + 4.0 * model.x[1,2] + 2.0 * model.x[1,3]
      + H*model.x[1,4] +H*model.x[1,5] + H*model.x[1,6]
 + H*model.x[2,1] + 0.0*model.x[2,2] + H*model.x[2,3]
      + 5.0*model.x[2,4] + H*model.x[2,5] + H*model.x[2,6]
 + H*model.x[3,1] + 1.0*model.x[3,2] + 0.0*model.x[3,3]
      + 8.0*model.x[3,4] + 10.0*model.x[3,5] + H*model.x[3,6]
 + H*model.x[4,1] + H*model.x[4,2] + H*model.x[4,3]
      + 0.0*model.x[4,4] + 2.0*model.x[4,5] + 6.0*model.x[4,6]
 + H*model.x[5,1] + H*model.x[5,2] + H*model.x[5,3]
      + H*model.x[5,4] + 0.0*model.x[5,5] + 2.0*model.x[5,6]
 + H*model.x[6,1] + H*model.x[6,2] + H*model.x[6,3]
      + H*model.x[6,4] + H*model.x[6,5] + 0.0*model.x[6,6]
 , sense = minimize)

# Source and Destination Constraints
model.SConstraint1 = Constraint(expr=
  model.x[1,2] + model.x[1,3] -model.y[1] >= 1)
model.DConstraint1 = Constraint(expr=
  model.x[4,6] + model.x[5,6] -model.y[2] >= 1)
```

```
# Intermediate Node Constraints
model.IntConst1 = Constraint(expr=
  model.x[1,2] + model.x[3,2] - model.x[2,4] - model.y[3] >= 0)
model.IntConst2 = Constraint(expr=
  model.x[1,3] - model.x[3,2] - model.x[3,4] - model.x[3,5]
       - model.y[4] >= 0)
model.IntConst3 = Constraint(expr=
  model.x[2,4] + model.x[3,4] - model.x[4,5] - model.x[4,6]
      - model.y[5] >= 0)
model.IntConst4 = Constraint(expr=
  model.x[3,5] + model.x[4,5] - model.x[5,6] - model.y[6] >= 0)
```

After running the model with Pyomo, the following listing is produced. Note that the optimal value of the objective function is 12.0. The results show that the minimum path has length = 12. The path selected consists of the following arcs: $x_{1,3}$, $x_{3,2}$, $x_{2,4}$, $x_{4,5}$, and $x_{5,6}$.

```
$ pyomo network1.py --summary
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
Assignment Problem 1
[    0.01] Creating model
[    0.01] Applying solver
[    0.05] Processing results
    Number of solutions: 1
    Solution Information
      Gap: 0.0
      Status: feasible
      Function Value: 12.0
    Solver results file: results.json

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

Model Shortest Path Problem 1

  Variables:
    x : Size=36, Index=x_index, Domain=NonNegativeReals
        Key    : Lower : Value : Upper : Initial : Fixed : Stale
        (1, 1) :     0 :  None :  None :    None : False :  True
        (1, 2) :     0 :   0.0 :  None :    None : False : False
        (1, 3) :     0 :   1.0 :  None :    None : False : False
        (1, 4) :     0 :   0.0 :  None :    None : False : False
        (1, 5) :     0 :   0.0 :  None :    None : False : False
        (1, 6) :     0 :   0.0 :  None :    None : False : False
        (2, 1) :     0 :   0.0 :  None :    None : False : False
        (2, 2) :     0 :  None :  None :    None : False :  True
        (2, 3) :     0 :   0.0 :  None :    None : False : False
        (2, 4) :     0 :   1.0 :  None :    None : False : False
        (2, 5) :     0 :   0.0 :  None :    None : False : False
        (2, 6) :     0 :   0.0 :  None :    None : False : False
        (3, 1) :     0 :   0.0 :  None :    None : False : False
        (3, 2) :     0 :   1.0 :  None :    None : False : False
        (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 :   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 :   0.0 :  None :    None : False : False
        (4, 4) :     0 :  None :  None :    None : False :  True
        (4, 5) :     0 :   1.0 :  None :    None : False : False
        (4, 6) :     0 :   0.0 :  None :    None : False : False
        (5, 1) :     0 :   0.0 :  None :    None : False : False
        (5, 2) :     0 :   0.0 :  None :    None : False : False
        (5, 3) :     0 :   0.0 :  None :    None : False : False
        (5, 4) :     0 :   0.0 :  None :    None : False : False
        (5, 5) :     0 :  None :  None :    None : False :  True
        (5, 6) :     0 :   1.0 :  None :    None : False : False
        (6, 1) :     0 :   0.0 :  None :    None : False : False
        (6, 2) :     0 :   0.0 :  None :    None : False : False
        (6, 3) :     0 :   0.0 :  None :    None : False : False
        (6, 4) :     0 :   0.0 :  None :    None : False : False
        (6, 5) :     0 :   0.0 :  None :    None : False : False
        (6, 6) :     0 :  None :  None :    None : False :  True
    y : Size=6, 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

  Objectives:
    obj : Size=1, Index=None, Active=True
```

```
        Key  : Active : Value
        None :   True :  12.0

  Constraints:
    SConstraint1 : Size=1
        Key  : Lower : Body : Upper
        None :   1.0 :  1.0 :  None
    DConstraint1 : Size=1
        Key  : Lower : Body : Upper
        None :   1.0 :  1.0 :  None
    IntConst1 : Size=1
        Key  : Lower : Body : Upper
        None :   0.0 :  0.0 :  None
    IntConst2 : Size=1
        Key  : Lower : Body : Upper
        None :   0.0 :  0.0 :  None
    IntConst3 : Size=1
        Key  : Lower : Body : Upper
        None :   0.0 :  0.0 :  None
    IntConst4 : Size=1
        Key  : Lower : Body : Upper
        None :   0.0 :  0.0 :  None

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

3.2 Formulation Using the Pulp Modeler

The following listing shows the formulation of the model using the Pulp modeler. The model is stored in file `network1.py`

```
"""
Python Formulation using the Pulp Modeler
Network problem, shortest path. File: network1.py
J M Garrido, September  2014
usage: python network1.py
"""
print "Network Problem, shortest path"
# Import PuLP modeler functions
from pulp import *

# Data for Linear Optimization Problem
```

```
N = 6  # Number of nodes in network
M = 2  # number of end nodes (source and destination)
INT = 4 # Number of intermediate nodes
H = 10000.0 # A very high cost constant
a = range(1, N+1)
al = range(N)
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]
T = INT + M # number of artificial variables (y)
tindx = range(1, T+1)

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

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

# The Pulp objective function
model += \
0.0*x[1,1] + 4.0*x[1,2] + 2.0*x[1,3] + H*x[1,4] +H*x[1,5] + H*x[1,6] \
 + H*x[2,1] + 0.0*x[2,2] + H*x[2,3] + 5.0*x[2,4] + H*x[2,5]
      + H*x[2,6]  \
+ H*x[3,1] + 1.0*x[3,2] + 0.0*x[3,3] + 8.0*x[3,4] + 10.0*x[3,5]
      + H*x[3,6] \
+ H*x[4,1] + H*x[4,2] + H*x[4,3] + 0.0*x[4,4] + 2.0*x[4,5]
      + 6.0*x[4,6] \
+ H*x[5,1] + H*x[5,2] + H*x[5,3] + H*x[5,4] + 0.0*x[5,5] + 2.0*x[5,6] \
+ H*x[6,1] + H*x[6,2] + H*x[6,3] + H*x[6,4] + H*x[6,5] + 0.0*x[6,6] \
, "Transportation cost"

# Source and Constraints
model += x[1,2] + x[1,3] -y[1] >= 1,"Source node"
model += x[4,6] + x[5,6] -y[2] >= 1,"Destination node"

# Intermediate Node Constraints
model += x[1,2] + x[3,2] - x[2,4] - y[3] >= 0,"Node 2"
model += x[1,3] - x[3,2] - x[3,4] - x[3,5] - y[4] >= 0,"Node 3"
model += x[2,4] + x[3,4] - x[4,5] - x[4,6] - y[5] >= 0, "Node 4"
model += x[3,5] + x[4,5] - x[5,6] - y[6] >= 0, "Node 5"

# Solve the optimization problem using the 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)
```

4 Maximum Flow Problems

The are many problems in which the goal is to send the *maximum quantity* of a product from a source node to a destination node of a network. The main limitation is the *capacity* of each segment of the network represented by the arcs. For these problems, the Ford-Fulkerson algorithm was developed. In this section, the linear optimization formulation is discussed.

The following maximum flow problem illustrates the basic approach to formulate a linear optimization problem. An airline company needs to plan and setup an optimal number of flights from Chicago to Rio de Janeiro. The intermediate stops that need to be included are: first Atlanta, then Bogota and/or Caracas. Table 1 shows the routes, the corresponding arcs in Figure 2, and the capacity (maximum number of flights allowed) of each route.

Table 1: Airline routes from Chicago to Rio.

Route	Arc	Max number of flights
Chicago to Atlanta	$x_{1,2}$	7
Atlanta to Bogota	$x_{2,3}$	4
Atlanta to Caracas	$x_{2,4}$	3
Bogota to Caracas	$x_{3,4}$	6
Bogota to Rio	$x_{3,5}$	3
Caracas to Rio	$x_{4,5}$	4

The problem can be formulated directly as a standard transportation problem, using the data in Table 1 and Figure 2.

Let $x_{i,j}$ denote the number of flights from node i to node j, and $c_{i,j}$ the capacity between nodes i and j. This problem has one supply point (source) and one demand

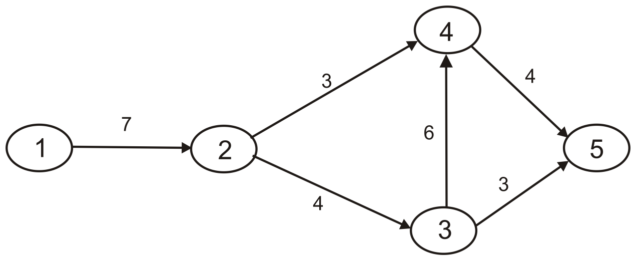

Figure 2: Graph of airline routes to Rio.

point (destination). The objective function can be written by observing that the maximum flow from node 1 to node 5 is actually the flow from node 1 to node 2, which is represented by the variable $x_{1,2}$. The objective function is then to maximize the expression: $z = x_{1,2}$.

To simplify the formulation of the problem, three types of constraints are considered:

- Source-destination constraints
- Transshipment constraints
- Capacity constraints

There is one source-destination constraint, which represents the balance in the flow from the source node and the flow into the destination node in Figure 2. This constraint is expressed as follows:

$$x_{1,2} = x_{3,5} + x_{4,5}$$

There are three intermediate nodes (2, 3, and 4), so there are three transshipment constraints, one for each intermediate node. These constraints are:

$$\begin{array}{ccccccl} x_{1,2} & - & x_{2,3} & - & x_{2,4} & = & 0 \quad \text{(Node 2)} \\ x_{2,3} & - & x_{3,4} & - & x_{3,5} & = & 0 \quad \text{(Node 3)} \\ x_{2,4} & + & x_{3,4} & - & x_{4,5} & = & 0 \quad \text{(Node 4)} \end{array}$$

The capacity constraints represent the maximum number of flights possible on the indicated route:

$$x_{i,j} \leq \; c_{i,j}$$

These are shown as the value of the arcs in Figure 2. There are six arcs in the graph, therefore there are six capacity constraints.

$$\begin{aligned} x_{1,2} &\leq 7 \\ x_{2,3} &\leq 4 \\ x_{2,4} &\leq 3 \\ x_{3,4} &\leq 6 \\ x_{3,5} &\leq 3 \\ x_{4,5} &\leq 4 \end{aligned}$$

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

4.1 Formulation Using the Pyomo Modeler

The following listing shows the formulation of the model using the Pyomo modeler and is stored in file `maxflow1.py`. Note that this is a maximization model and the artificial variables are used with a + sign and the right-hand side of the respective constraints are `<= 0`. This is shown in line 36 and in lines 54–58.

```
 1 """
 2 Python Formulation using Pyomo Modeler
 3 Network problem, maximum flow. File: network1.py
 4 J M Garrido, September  2014
 5 usage: pyomo maxflow1.py --summary
 6 """
 7 print "Maximum Flow Problem"
 8 # Import
 9 from coopr.pyomo import *
10
11 # Data for Linear Optimization Problem
12 N = 5  # Number of nodes in network
13 INT = 3 # Number of intermediate nodes
14 a = range(1, N+1)
15 al = range(N)
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 T = INT + 1 # number of artificial variables (y)
21 tindx = range(1, T+1) # index list for y variables
22
23 #Concrete Model
```

```
model = ConcreteModel(name="Maximum Flow Problem")

# Decision Variables
model.x = Var(xindx, within=NonNegativeReals)
# Artificial variables
model.y = Var(tindx, within=NonNegativeReals)

# The objective function
model.obj = Objective(expr= model.x[1,2], sense = maximize)

# Source and Destination Constraint
model.SDConstraint1 = Constraint(expr=
  model.x[1,2] - model.x[3,5] - model.x[4,5] + model.y[1] <= 0)

# Arc Capacity Constraints
model.ArcConstraint1 = Constraint(expr=
  model.x[1,2] <= 7)
model.ArcConstraint2 = Constraint(expr=
  model.x[2,3] <= 4)
model.ArcConstraint3 = Constraint(expr=
  model.x[2,4] <= 3)
model.ArcConstraint4 = Constraint(expr=
  model.x[3,4] <= 6)
model.ArcConstraint5 = Constraint(expr=
  model.x[3,5] <= 3)
model.ArcConstraint6 = Constraint(expr=
  model.x[4,5] <= 4)

# Intermediate Node Constraints
model.IntConst1 = Constraint(expr=
  model.x[1,2] - model.x[2,3] - model.x[2,4] + model.y[2] <= 0)
model.IntConst2 = Constraint(expr=
  model.x[2,3] - model.x[3,4] - model.x[3,5] + model.y[3] <= 0)
model.IntConst3 = Constraint(expr=
  model.x[2,4] + model.x[3,4] - model.x[4,5] + model.y[4] <= 0)
```

After running the model with the Pyomo modeler, the following listing is produced. Note that optimum number of flights is 7 and the value of variable $x_{3,4}$ is 1, which means that the maximum number of flights from node 3 to node 4 is 1.

```
$ pyomo maxflow1.py --summary
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
```

```
Maximum Flow
[    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: 7.0
    Solver results file: results.json

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

Model Maximum Flow Problem

  Variables:
    x : Size=25, Index=x_index, Domain=NonNegativeReals
        Key    : Lower : Value : Upper : Initial : Fixed : Stale
        (1, 1) :     0 :  None :  None :    None : False :  True
        (1, 2) :     0 :   7.0 :  None :    None : False : False
        (1, 3) :     0 :  None :  None :    None : False :  True
        (1, 4) :     0 :  None :  None :    None : False :  True
        (1, 5) :     0 :  None :  None :    None : False :  True
        (2, 1) :     0 :  None :  None :    None : False :  True
        (2, 2) :     0 :  None :  None :    None : False :  True
        (2, 3) :     0 :   4.0 :  None :    None : False : False
        (2, 4) :     0 :   3.0 :  None :    None : False : False
        (2, 5) :     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 :   1.0 :  None :    None : False : False
        (3, 5) :     0 :   3.0 :  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 :  None :  None :    None : False :  True
        (4, 5) :     0 :   4.0 :  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 :  None :  None :    None : False :  True
        (5, 5) :     0 :  None :  None :    None : False :  True
    y : Size=4, 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

  Objectives:
    obj : Size=1, Index=None, Active=True
        Key  : Active : Value
        None :   True :   7.0

  Constraints:
    SDConstraint1 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  0.0 :   0.0
    ArcConstraint1 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  7.0 :   7.0
    ArcConstraint2 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  4.0 :   4.0
    ArcConstraint3 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  3.0 :   3.0
    ArcConstraint4 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  1.0 :   6.0
    ArcConstraint5 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  3.0 :   3.0
    ArcConstraint6 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  4.0 :   4.0
    IntConst1 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  0.0 :   0.0
    IntConst2 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  0.0 :   0.0
    IntConst3 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  0.0 :   0.0

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

4.2 Formulation Using the Pulp Modeler

The following listing shows the formulation of the model using the Pulp modeler and is stored in file `maxflow1.py`.

```
"""
Python Formulation using the Pulp Modeler
Network problem, maximum flow. File: maxflow1.py
J M Garrido, September  2014
usage: python maxflow1.py
"""
print "Network Problem, shortest path"
# Import PuLP modeler functions
from pulp import *

# Data for Linear Optimization Problem
N = 5  # Number of nodes in network
INT = 3 # Number of intermediate nodes
a = range(1, N+1)
al = range(N)
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]
T = INT + 1 # number of artificial variables (y)
tindx = range(1, T+1)

# Create the model to contain the problem data
model = LpProblem("Maximum Flow Problem",LpMaximize)

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

# The Pulp objective function
model += x[1,2], "Maximum Flow"

# Source and Destination Constraints
model += x[1,2] - x[4,5] - x[3,5] + y[1] <= 0,"Source to destination"

# Arc Capacity Constraints
```

```
model += x[1,2] <= 7,"Arc 1-2"
model += x[2,3] <= 4,"Arc 2-3"
model += x[2,4] <= 3,"Arc 2-4"
model += x[3,4] <= 6,"Arc 3-4"
model += x[3,5] <= 3,"Arc 3-5"
model += x[4,5] <= 4,"Arc 4-5"

# Intermediate Node Constraints
model += x[1,2] - x[2,3] - x[2,4] + y[2] <= 0,"Node 2"
model += x[2,3] - x[3,4] - x[3,5] + y[3] <= 0,"Node 3"
model += x[2,4] + x[3,4] - x[4,5] + y[4] <= 0,"Node 4"

# Solve the optimization problem using the 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)
```

5 Critical Path Method

The critical path method (CPM) is a network model that can help in the *scheduling* of large projects. The important computations are the total time to complete the project and the interval that represents how long an activity of the project can be delayed without causing delays to the project. This method calculates the minimum completion time for a project and the possible start and finish times for the project activities.

A network model of a project typically consists of a sequence of the various activities that need to be performed, and the duration of each activity. A directed arc represents an *activity*, a node represents a *start* or *finish event* of an activity. A special initial node represents the start of the project, and a special end node represents the completion of the project.

In any project there are cases in which more than one activity needs to be completed before the next activity can start. Figure 3 illustrates this situation. Activity *Aj* and activity *Ak* have to be completed before activity *Al* can start.

Another situation is an activity that needs to be completed before two or more activities can start. Figure 4 illustrates this by showing activity Al that needs to be completed before activities Am, Ak, and Ap can start.

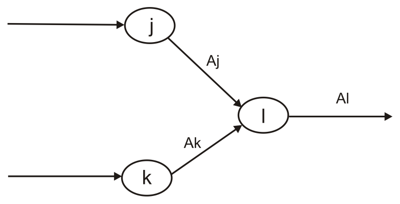

Figure 3: Several activities completed before next activity.

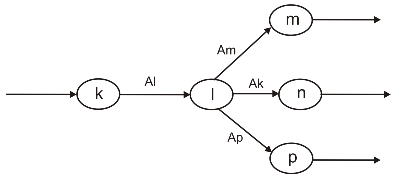

Figure 4: Sequencing of activities.

The following terms are used in project scheduling using the duration of the activities in the project. The *early event* time of node i, denoted by e_i, is the earliest time at which the event can occur. The *late event* time of node i, denoted by l_i, is the latest time at which the event can occur.

The *total float*, denoted by $f_{i,j}$ of an activity $A_{i,j}$ is the time interval by which the starting time of the activity can be delayed and not cause delay in the completion time of the project.

An activity that has a total float equal to zero is known as a *critical activity.* A critical path consists of a sequence of critical activities. Delays in the activities in a critical path will delay the completion of the project.

Recall that for any activity $A_{i,j}$, the start time of the activity is the event represented by node i and the completion time of the activity is the event represented by node j. Let x_k denote the time occurrence of event k and the duration of activity $A_{i,j}$ is denoted by $\Delta_{i,j}$. For every activity $A_{i,j}$, the completion time of the activity is given by the expression $x_j \geq x_i + \Delta_{i,j}$.

Let f denote the finish node of the project, the event time of the completion of the project is denoted by x_f. Similarly, node 1 is the start node of the project and the event time of the start of the project is denoted by x_1. The total time interval or duration of the entire project is given by the expression $x_f - x_1$. Let n denote the total number of nodes in the project network, this implies that $x_f = x_n$. The formulation of the linear optimization problem that finds the critical path of a project is given by the following expressions:

Minimize: $z = x_f - x_1$
Subject to:

$$x_j \geq x_i + \Delta_{i,j}, \quad i = 1, \ldots, n-1, \quad j = 2, \ldots, n$$

The variables $x_i, i = 1, \ldots, n$ are unrestricted in sign.

5.1 Critical Path Method: Case Study

A project has been defined with the activities and their duration given in Table 2. The goal of the problem is to find the critical path of the project. The various activities and their predecessors are shown by the arcs between nodes in the graph of Figure 5.

Table 2: Project data.

Activity	Predecessor	Duration
$A_{1,2}$	-	6
$A_{2,3}$	$A_{1,2}$	9
$A_{3,5}$	$A_{2,3}$	13
$A_{3,6}$	$A_{2,3}$	6
$A_{3,4}$	$A_{2,3}$	5
$A_{4,5}$	$A_{3,4}$	7
$A_{5,6}$	$A_{4,5}$, $A_{3,5}$	4

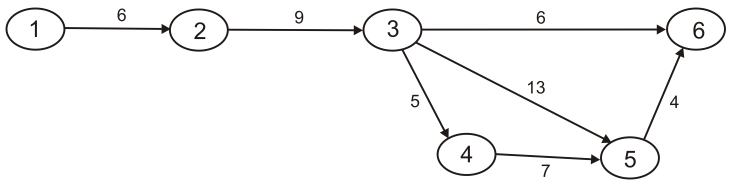

Figure 5: Graph of project activities.

The problem can be formulated directly as a standard transportation problem, using the data in Table 2 and in Figure 5.

The final node of the project in this problem is node 6. The objective function is then to minimize the expression: $z = x_6 - x_1$. The constraints are expressed as follows:

$$\begin{array}{lcll} x_2 & \geq & x_1 + 6 & \text{Activity } A_{1,2} \\ x_3 & \geq & x_2 + 9 & \text{Activity } A_{2,3} \\ x_4 & \geq & x_3 + 5 & \text{Activity } A_{3,4} \\ x_5 & \geq & x_4 + 7 & \text{Activity } A_{4,5} \\ x_5 & \geq & x_3 + 13 & \text{Activity } A_{3,5} \\ x_6 & \geq & x_3 + 6 & \text{Activity } A_{3,6} \\ x_6 & \geq & x_5 + 4 & \text{Activity } A_{5,6} \end{array}$$

The decision variables x_i have unrestricted sign constraint for $i = 1, \ldots, 6$.

5.2 Formulation Using the Pyomo Modeler

The following listing shows the Python script of the model written for the Pyomo modeler, and is stored in file `cpm1.py`.

```
"""
Python Formulation using Pyomo Modeler
Network problem, critical path method. File: cpm1.py
J M Garrido, September  2014
usage: pyomo cpm1.py --summary
"""
print "Maximum Flow"
# Import
from coopr.pyomo import *

# Data for Linear Optimization Problem
N = 6  # Number of nodes in network
# Index list for decision variables x
xindx =  range(1, N+1)

#Concrete Model
model = ConcreteModel(name="Critical Path Problem")

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

# The objective function
model.obj = Objective(expr= model.x[6] - model.x[1], sense = minimize)

# Activity Constraint
model.ActConstraint1 = Constraint(expr=
  model.x[2] - model.x[1] >= 6)
model.ActConstraint2 = Constraint(expr=
  model.x[3] - model.x[2] >= 9)
model.ActConstraint3 = Constraint(expr=
  model.x[4] - model.x[3] >= 5)
model.ActConstraint4 = Constraint(expr=
  model.x[5] - model.x[4] >= 7)
model.ActConstraint5 = Constraint(expr=
  model.x[5] - model.x[3] >= 13)
model.ActConstraint6 = Constraint(expr=
  model.x[6] - model.x[3] >= 6)
model.ActConstraint7 = Constraint(expr=
  model.x[6] - model.x[5] >= 4)
```

The following listing shows the results after running the model with the Pyomo modeler. Note that the total time interval for completion of the project is 32 days and the start time of every activity is shown in the listing.

```
$ pyomo cpm1.py --summary
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
Maximum Flow
[    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: 32.0
    Solver results file: results.json

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

Model Critical Path Problem

  Variables:
    x : Size=6, Index=x_index, Domain=NonNegativeReals
        Key : Lower : Value : Upper : Initial : Fixed : Stale
          1 :     0 :   0.0 :  None :    None : False : False
          2 :     0 :   6.0 :  None :    None : False : False
          3 :     0 :  15.0 :  None :    None : False : False
          4 :     0 :  20.0 :  None :    None : False : False
          5 :     0 :  28.0 :  None :    None : False : False
          6 :     0 :  32.0 :  None :    None : False : False

  Objectives:
    obj : Size=1, Index=None, Active=True
        Key  : Active : Value
        None :   True :  32.0

  Constraints:
    ActConstraint1 : Size=1
        Key  : Lower : Body : Upper
        None :   6.0 :  6.0 :  None
    ActConstraint2 : Size=1
```

```
        Key  : Lower : Body : Upper
        None :   9.0 :  9.0 :  None
    ActConstraint3 : Size=1
        Key  : Lower : Body : Upper
        None :   5.0 :  5.0 :  None
    ActConstraint4 : Size=1
        Key  : Lower : Body : Upper
        None :   7.0 :  8.0 :  None
    ActConstraint5 : Size=1
        Key  : Lower : Body : Upper
        None :  13.0 : 13.0 :  None
    ActConstraint6 : Size=1
        Key  : Lower : Body : Upper
        None :   6.0 : 17.0 :  None
    ActConstraint7 : Size=1
        Key  : Lower : Body : Upper
        None :   4.0 :  4.0 :  None

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

5.3 Formulation Using the Pulp Modeler

The following listing shows the Python script of the model written for the Pyomo modeler, and is stored in file `cpm1.py`.

```
"""
Python Formulation using the Pulp Modeler
Network problem, critical path. File: cpm1.py
J M Garrido, September  2014
usage: python cpm1.py
"""
print "Network Problem, critical path method"
# Import PuLP modeler functions
from pulp import *

# Data for Linear Optimization Problem
N = 6  # Number of nodes in network
# Index list for decision variables x
xindx = range(1, N+1)

# Create the model to contain the problem data
model = LpProblem("Critical path Method",LpMinimize)
```

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

# The Pulp objective function
model += x[6] - x[1], "Duration of project"

# Activity Constraints
model += x[2] - x[1] >= 6,"Activity 1-2"
model += x[3] - x[2] >= 9,"Acivity  2-3"
model += x[4] - x[3] >= 5,"Activity 3-4"
model += x[5] - x[4] >= 7,"Activity 4-5"
model += x[5] - x[3] >= 13,"Activity 3-5"
model += x[6] - x[3] >= 6,"Activity 3-6"
model += x[6] - x[5] >= 4,"Activity 5-6"

# Solve the optimization problem using the 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)
```

6 Reducing the Time to Complete a Project

When a decision is taken to *reduce* the total time to complete a project, additional resources must be allocated to the various activities. Linear optimization is used to minimize the total cost of allocating the additional resources to the project activities.

Let $r_{i,j}$ denote the number of days that the duration of activity $A_{i,j}$ is reduced, and $c_{i,j}$ denote the cost per day of allocating additional resources to activity $A_{i,j}$. Let R denote the time (in days) that an activity can be reduced, and let T denote the new total time (in days) of the project completion. The objective function to minimize is:

$$z = \sum_{i=1}^{n-1} \sum_{j=2}^{n} c_{i,j} r_{i,j}, \quad \text{for all activities } A_{i,j}$$

The time-reduction constraints are expressed as:

$$r_{i,j} \leq R, \quad \text{for all activities } A_{i,j}$$

The activity constraints are expressed as:

$$x_j = x_i + \Delta_{i,j} - r_{i,j}, \quad \text{for all activities } A_{i,j}$$

The total time constraint is:

$$x_n - x_1 \leq T$$

.

6.1 Reducing Time Case Study

Consider a reduction of 4 days in the total time to complete the project described in the previous problem. The completion time is now 28 days. The activity completion time can be reduced up to 2 days. The following table shows the cost per day of reducing each activity of the project.

Table 3: Project additional cost.

Activity	Cost
$A_{1,2}$	22.50
$A_{2,3}$	15.75
$A_{3,5}$	13.25
$A_{3,6}$	16.50
$A_{3,4}$	25.30
$A_{4,5}$	17.50
$A_{5,6}$	14.75

The various activities and their predecessors are shown by the arcs between nodes in the graph of Figure 5. This problem has node 6 as the final node of the project. The goal of the problem is to find the minimum cost of reducing the total completion time of the project. The objective function is then to minimize the expression:

$$\begin{aligned} z = \; & 22.50\, r_{1,2} + 15.75\, r_{2,3} + 13.25\, r_{3,5} + 16.50\, r_{3,6} + 25.30\, r_{3,4} + \\ & 17.50\, r_{4,5} + 14.75\, r_{5,6} \end{aligned}$$

.

The time-reduction constraints are expressed as follows:

$$\begin{array}{lcll} r_{1,2} & \leq & 2, & \text{Activity } A_{1,2} \\ r_{2,3} & \leq & 2, & \text{Activity } A_{2,3} \\ r_{3,4} & \leq & 2, & \text{Activity } A_{3,4} \\ r_{4,5} & \leq & 2, & \text{Activity } A_{4,5} \\ r_{3,5} & \leq & 2, & \text{Activity } A_{3,5} \\ r_{3,6} & \leq & 2, & \text{Activity } A_{3,6} \\ r_{5,6} & \leq & 2, & \text{Activity } A_{5,6} \end{array}$$

The activity constraints are expressed as follows:

$$\begin{array}{lcll} x_2 & \geq & x_1 + 6 - r_{1,2}, & \text{Activity } A_{1,2} \\ x_3 & \geq & x_2 + 9 - r_{2,3}, & \text{Activity } A_{2,3} \\ x_4 & \geq & x_3 + 5 - r_{3,4}, & \text{Activity } A_{3,4} \\ x_5 & \geq & x_4 + 7 - r_{4,5}, & \text{Activity } A_{4,5} \\ x_5 & \geq & x_3 + 13 - r_{3,5}, & \text{Activity } A_{3,5} \\ x_6 & \geq & x_3 + 6 - r_{3,6}, & \text{Activity } A_{3,6} \\ x_6 & \geq & x_5 + 4 - r_{5,6}, & \text{Activity } A_{5,6} \end{array}$$

The decision variables x_i have unrestricted sign constraint for $i = 1, \ldots, 6$.

6.2 Formulation Using the Pyomo Modeler

The following listing shows the formulation of the model using the Pyomo modeler and is stored in file `cpm1b.py`

```
"""
Python Formulation using Pyomo Modeler
Network problem, reducing time of project. File: cpm1b.py
J M Garrido, September  2014
usage: pyomo cpm1b.py --summary
"""
print "Reducing Time of Project"
# Import
from coopr.pyomo import *

# Data for Linear Optimization Problem
N = 6  # Number of nodes in project network
```

```
# Index list for decision variables x
xindx = range(1, N+1)
a = range(1, N+1)
al = range(N)
b = range(1,N+1)
bl = range(N)
# Index list for decision variables x
rindx = [(a[i],b[j]) for j in bl for i in al]

#Concrete Model
model = ConcreteModel(name="Project Time Reduction")

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

# The objective function
model.obj = Objective(expr= 22.5*model.r[1,2] + 15.75*model.r[2,3]
+ 13.25*model.r[3,5] + 16.5*model.r[3,6] + 25.3*model.r[3,4]
+ 17.5*model.r[4,5] + 14.75*model.r[5,6]
, sense = minimize)

# Time Reduction Constraints
model.TRConstraint1 = Constraint(expr=
  model.r[1,2] <= 2)
model.TRConstraint2 = Constraint(expr=
  model.r[2,3] <= 2)
model.TRConstraint3 = Constraint(expr=
  model.r[3,4] <= 2)
model.TRConstraint4 = Constraint(expr=
  model.r[4,5] <= 2)
model.TRConstraint5 = Constraint(expr=
  model.r[3,5] <= 2)
model.TRConstraint6 = Constraint(expr=
  model.r[3,6] <= 2)
model.TRConstraint7 = Constraint(expr=
  model.r[5,6] <= 2)

# Activity Constraint
model.ActConstraint1 = Constraint(expr=
  model.x[2] - model.x[1] + model.r[1,2] >= 6)
model.ActConstraint2 = Constraint(expr=
  model.x[3] - model.x[2] + model.r[2,3] >= 9)
model.ActConstraint3 = Constraint(expr=
  model.x[4] - model.x[3] + model.r[3,4] >= 5)
model.ActConstraint4 = Constraint(expr=
  model.x[5] - model.x[4] + model.r[4,5] >= 7)
model.ActConstraint5 = Constraint(expr=
  model.x[5] - model.x[3] + model.r[3,5] >= 13)
model.ActConstraint6 = Constraint(expr=
  model.x[6] - model.x[3] + model.r[3,6] >= 6)
model.ActConstraint7 = Constraint(expr=
  model.x[6] - model.x[5] + model.r[5,6] >= 4)

# Total Time of Project
model.TConstraint = Constraint(expr=
    model.x[6] - model.x[1] <= 28)
```

The following listing shows the resuly of running the model with the Pyomo modeler. The total cost of reducing time interval for completion of the project is \$58.50. Note that the completion time of activities $A_{1,2}$, $A_{3,6}$, $A_{3,4}$, and $A_{4,5}$, were not reduced. However, the completion time of activity $A_{5,6}$ was reduced in 2 days.

```
[    0.00] Setting up Pyomo environment
[    0.00] Applying Pyomo preprocessing actions
Reducing Time of Project
[    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: 58.5
    Solver results file: results.json

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

Model Project Time Reduction

  Variables:
    x : Size=6, Index=x_index, Domain=NonNegativeReals
        Key : Lower : Value : Upper : Initial : Fixed : Stale
          1 :     0 :   0.0 :  None :    None : False : False
          2 :     0 :   6.0 :  None :    None : False : False
          3 :     0 :  14.0 :  None :    None : False : False
          4 :     0 :  19.0 :  None :    None : False : False
          5 :     0 :  26.0 :  None :    None : False : False
          6 :     0 :  28.0 :  None :    None : False : False
    r : Size=36, Index=r_index, Domain=NonNegativeReals
        Key    : Lower : Value : Upper : Initial : Fixed : Stale
        (1, 1) :     0 :  None :  None :    None : False :  True
        (1, 2) :     0 :   0.0 :  None :    None : False : False
        (1, 3) :     0 :  None :  None :    None : False :  True
        (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 :  None :  None :    None : False :  True
        (2, 2) :     0 :  None :  None :    None : False :  True
        (2, 3) :     0 :   1.0 :  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 :   1.0 :  None :    None : False : False
        (3, 6) :     0 :   0.0 :  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 :  None :  None :    None : False :  True
        (4, 5) :     0 :   0.0 :  None :    None : False : False
        (4, 6) :     0 :  None :  None :    None : False :  True
        (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 :  None :  None :    None : False :  True
        (5, 5) :     0 :  None :  None :    None : False :  True
        (5, 6) :     0 :   2.0 :  None :    None : False : False
        (6, 1) :     0 :  None :  None :    None : False :  True
        (6, 2) :     0 :  None :  None :    None : False :  True
        (6, 3) :     0 :  None :  None :    None : False :  True
        (6, 4) :     0 :  None :  None :    None : False :  True
        (6, 5) :     0 :  None :  None :    None : False :  True
        (6, 6) :     0 :  None :  None :    None : False :  True

  Objectives:
    obj : Size=1, Index=None, Active=True
        Key  : Active : Value
        None :   True :  58.5
  Constraints:
    TRConstraint1 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  0.0 :   2.0
    TRConstraint2 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  1.0 :   2.0
    TRConstraint3 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  0.0 :   2.0
    TRConstraint4 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  0.0 :   2.0
    TRConstraint5 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  1.0 :   2.0
    TRConstraint6 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  0.0 :   2.0
    TRConstraint7 : Size=1
        Key  : Lower : Body : Upper
        None :  None :  2.0 :   2.0
    ActConstraint1 : Size=1
        Key  : Lower : Body : Upper
        None :   6.0 :  6.0 :  None
    ActConstraint2 : Size=1
        Key  : Lower : Body : Upper
        None :   9.0 :  9.0 :  None
    ActConstraint3 : Size=1
        Key  : Lower : Body : Upper
        None :   5.0 :  5.0 :  None
    ActConstraint4 : Size=1
        Key  : Lower : Body : Upper
        None :   7.0 :  7.0 :  None
    ActConstraint5 : Size=1
        Key  : Lower : Body : Upper
        None :  13.0 : 13.0 :  None
    ActConstraint6 : Size=1
        Key  : Lower : Body : Upper
        None :   6.0 : 14.0 :  None
    ActConstraint7 : Size=1
        Key  : Lower : Body : Upper
        None :   4.0 :  4.0 :  None
    TConstraint : Size=1
```

```
        Key  : Lower : Body : Upper
        None :  None : 28.0 :  28.0

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

6.3 Formulation Using the Pulp Modeler

The following listing shows the formulation of the model using the Pulp modeler and is stored in file `cpm1b.py`

```
"""
Python Formulation using the Pulp Modeler
Network problem, project time reduction. File: cpm1.py
J M Garrido, September  2014
usage: python cpm1b.py
"""
print "Network Problem, project time reduction"
# Import PuLP modeler functions
from pulp import *

# Data for Linear Optimization Problem
N = 6  # Number of nodes in network
# Index list for decision variables x
xindx = range(1, N+1)
a = range(1, N+1)
al = range(N)
b = range(1,N+1)
bl = range(N)
# Index list for decision variables x
rindx = [(a[i],b[j]) for j in bl for i in al]

# Create the model to contain the problem data
model = LpProblem("Reducing Time of Project",LpMinimize)

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

# The Pulp objective function
model += 22.5*r[1,2] + 15.75*r[2,3] + 13.25*r[3,5] \
+ 16.5*r[3,6] + 25.3*r[3,4] + 17.5*r[4,5] + 14.75*r[5,6], \
"Time duration of project"
```

```
# Time Reduction Constraints
model += r[1,2] <= 2, "Reduc time Act 1-2"
model += r[2,3] <= 2, "Reduc time Act 2-3"
model += r[3,4] <= 2, "Reduc time Act 3-4"
model += r[4,5] <= 2, "Reduc time Act 4-5"
model += r[3,5] <= 2, "Reduc time Act 3-5"
model += r[3,6] <= 2, "Reduc time Act 3-6"
model += r[5,6] <= 2, "Reduc time Act 5-6"

# Activity Constraints
model += x[2] - x[1] + r[1,2] >= 6,"Activity 1-2"
model += x[3] - x[2] + r[2,3] >= 9,"Activity 2-3"
model += x[4] - x[3] + r[3,4] >= 5,"Activity 3-4"
model += x[5] - x[4] + r[4,5] >= 7,"Activity 4-5"
model += x[5] - x[3] + r[3,5] >= 13,"Activity 3-5"
model += x[6] - x[3] + r[3,6] >= 6,"Activity 3-6"
model += x[6] - x[5] + r[5,6] >= 4,"Activity 5-6"

# Total Time of Project
model += x[6] - x[1] <= 28, "Project Total Time"

# Solve the optimization problem using the 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)
```