5.Solve the following linear program using primal dual method

October 3, 2016

```
In [45]: from gurobipy import *

         # Model
         model = Model("prod")
         #model.setParam(GRB.param.Method, 0)

         # Create decision variables
         x1 = model.addVar(name="x1") # arguments by name
         x2 = model.addVar(name="x2") # arguments by position
         x3 = model.addVar(name="x3") # arguments by deafult
         x4 = model.addVar(name="x4") # arguments by deafult
         x5 = model.addVar(name="x5") # arguments by deafult

         # Update model to integrate new variables
         model.update()

         # The objective is to maximize (this is redundant now, but it will overwri
         model.setObjective(5*x1 + 2*x2 + x3 + 4*x4 + 6*x5, GRB.MINIMIZE)

         # Add constraints to the model
         model.addConstr(3*x1 + 5*x2 - 6*x3 + 2*x4 + 4*x5 ,GRB.EQUAL, 25, "c1")
         model.addConstr(x1 + 2*x2 + 3*x3 - 7*x4 + 6*x5 ,GRB.GREATER_EQUAL, 2, "c2'
         model.addConstr(9*x1 - 4*x2 + 2*x3 + 5*x4 - 2*x5 ,GRB.EQUAL, 16,  "c3")


         # Solve
         model.optimize()
         data = []
         # Let's print the solution
         for v in model.getVars():
             print v.varName, v.x
             data.append(v)
```

```
Optimize a model with 3 rows, 5 columns and 15 nonzeros
Coefficient statistics:
  Matrix range    [1e+00, 9e+00]
```

```
  Objective range [1e+00, 6e+00]
  Bounds range    [0e+00, 0e+00]
  RHS range       [2e+00, 2e+01]
Presolve time: 0.00s
Presolved: 3 rows, 5 columns, 15 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   4.375000e+00   0.000000e+00      0s
       2    2.2000000e+01   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.01 seconds
Optimal objective  2.200000000e+01
 x1 3.15789473684
x2 3.10526315789
x3 0.0
x4 0.0
x5 0.0
```

In [46]:
```python
# Let's print the dual variables
for c in model.getConstrs():
    print c.constrName, c.pi
```

```
c1 0.666666666667
c2 0.0
c3 0.333333333333
```

In [47]:
```python
data
```

Out[47]:
```
[<gurobi.Var x1 (value 3.15789473684)>,
 <gurobi.Var x2 (value 3.10526315789)>,
 <gurobi.Var x3 (value 0.0)>,
 <gurobi.Var x4 (value 0.0)>,
 <gurobi.Var x5 (value 0.0)>]
```

In [51]:
```python
from __future__ import print_function
from ortools.graph import pywrapgraph
import time

def main():
    """Solving Assignment Problem with MinCostFlow"""

    # Instantiate a SimpleMinCostFlow solver.
    min_cost_flow = pywrapgraph.SimpleMinCostFlow()
    # Define the directed graph for the flow.
    team_A = [1, 3, 5]
    team_B = [2, 4, 6]
```

```
start_nodes = ([0, 0]  + [11, 11, 11] + [12, 12, 12] +
              [1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5,
              [7, 8, 9, 10])
end_nodes =   ([11, 12] + team_A + team_B +
              [7, 8, 9, 10, 7, 8, 9, 10, 7, 8, 9, 10, 7, 8, 9, 10, 7, 8
              [13, 13, 13, 13])
capacities =  ([2, 2] + [1, 1, 1] + [1, 1, 1] +
              [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
              [1, 1, 1, 1])
costs      =  ([0, 0] + [0, 0, 0] + [0, 0, 0] +
              [90, 76, 75, 70, 35, 85, 55, 65, 125, 95, 90, 105, 45, 11
               80, 75, 45, 65, 110, 95] + [0, 0, 0, 0])

# Define an array of supplies at each node.

supplies = [4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -4]
source = 0
sink = 13

# Add each arc.
for i in range(0, len(start_nodes)):
    min_cost_flow.AddArcWithCapacityAndUnitCost(start_nodes[i], end_nc
                                                capacities[i], costs[i])

# Add node supplies.

for i in range(0, len(supplies)):
    min_cost_flow.SetNodeSupply(i, supplies[i])
# Find the minimum cost flow between node 0 and node 10.
if min_cost_flow.Solve() == min_cost_flow.OPTIMAL:

    min_cost_flow.Solve()
    print('Total cost = ', min_cost_flow.OptimalCost())
    print()

    for arc in range(min_cost_flow.NumArcs()):


    # Can ignore arcs leading out of source or intermediate nodes, or
        if (min_cost_flow.Tail(arc)!=0 and min_cost_flow.Tail(arc)!=11
            and min_cost_flow.Head(arc)!=13):


    # Arcs in the solution will have a flow value of 1. There start ar
    # give an assignment of worker to task.

            if min_cost_flow.Flow(arc) > 0:
                print('Worker %d assigned to task %d.  Cost = %d' % (
```

```
                                min_cost_flow.Tail(arc),
                                min_cost_flow.Head(arc),
                             min_cost_flow.UnitCost(arc)))
        else:
            print('There was an issue with the min cost flow input.')
    if __name__ == '__main__':
        start_time = time.clock()
        main()
        print()
        print("Time =", time.clock() - start_time, "seconds")
```

```
Total cost =  250

Worker 1 assigned to task 9.  Cost = 75
Worker 2 assigned to task 7.  Cost = 35
Worker 5 assigned to task 10.  Cost = 75
Worker 6 assigned to task 8.  Cost = 65

Time = 0.001774 seconds
```

In [52]:
```
from gurobipy import *

# Model data

commodities = ['Pencils', 'Pens']
nodes = ['Detroit', 'Denver', 'Boston', 'New York', 'Seattle']

arcs, capacity = multidict({
  ('Detroit', 'Boston'):   100,
  ('Detroit', 'New York'):  80,
  ('Detroit', 'Seattle'):  120,
  ('Denver',  'Boston'):   120,
  ('Denver',  'New York'): 120,
  ('Denver',  'Seattle'):  120 })
arcs = tuplelist(arcs)

cost = {
  ('Pencils', 'Detroit', 'Boston'):   10,
  ('Pencils', 'Detroit', 'New York'): 20,
  ('Pencils', 'Detroit', 'Seattle'):  60,
  ('Pencils', 'Denver',  'Boston'):   40,
  ('Pencils', 'Denver',  'New York'): 40,
  ('Pencils', 'Denver',  'Seattle'):  30,
  ('Pens',    'Detroit', 'Boston'):   20,
  ('Pens',    'Detroit', 'New York'): 20,
  ('Pens',    'Detroit', 'Seattle'):  80,
  ('Pens',    'Denver',  'Boston'):   60,
```

```
  ('Pens',    'Denver',  'New York'): 70,
  ('Pens',    'Denver',  'Seattle'):  30 }

inflow = {
  ('Pencils', 'Detroit'):   50,
  ('Pencils', 'Denver'):    60,
  ('Pencils', 'Boston'):   -50,
  ('Pencils', 'New York'): -50,
  ('Pencils', 'Seattle'):  -10,
  ('Pens',    'Detroit'):   60,
  ('Pens',    'Denver'):    40,
  ('Pens',    'Boston'):   -40,
  ('Pens',    'New York'): -30,
  ('Pens',    'Seattle'):  -30 }

# Create optimization model
m = Model('netflow')

# Create variables
flow = {}
for h in commodities:
    for i,j in arcs:
        flow[h,i,j] = m.addVar(ub=capacity[i,j], obj=cost[h,i,j],
                               name='flow_%s_%s_%s' % (h, i, j))
m.update()

# Arc capacity constraints
for i,j in arcs:
    m.addConstr(quicksum(flow[h,i,j] for h in commodities) <= capacity[i,j
                'cap_%s_%s' % (i, j))

# Flow conservation constraints
for h in commodities:
    for j in nodes:
        m.addConstr(
          quicksum(flow[h,i,j] for i,j in arcs.select('*',j)) +
              inflow[h,j] ==
          quicksum(flow[h,j,k] for j,k in arcs.select(j,'*')),
                   'node_%s_%s' % (h, j))

# Compute optimal solution
m.optimize()

# Print solution
if m.status == GRB.Status.OPTIMAL:
    solution = m.getAttr('x', flow)
    for h in commodities:
        print('\nOptimal flows for %s:' % h)
```

```
                for i,j in arcs:
                    if solution[h,i,j] > 0:
                        print('%s -> %s: %g' % (i, j, solution[h,i,j]))
```

```
Optimize a model with 16 rows, 12 columns and 36 nonzeros
Coefficient statistics:
  Matrix range    [1e+00, 1e+00]
  Objective range [1e+01, 8e+01]
  Bounds range    [8e+01, 1e+02]
  RHS range       [1e+01, 1e+02]
Presolve removed 16 rows and 12 columns
Presolve time: 0.01s
Presolve: All rows and columns removed
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    5.5000000e+03   0.000000e+00   0.000000e+00      0s

Solved in 0 iterations and 0.01 seconds
Optimal objective  5.500000000e+03

Optimal flows for Pencils:
Denver -> Seattle: 10
Denver -> New York: 50
Detroit -> Boston: 50

Optimal flows for Pens:
Denver -> Seattle: 30
Detroit -> New York: 30
Detroit -> Boston: 30
Denver -> Boston: 10
```

```
In [54]: import networkx as nx
         G = nx.DiGraph()
         G.add_node('a', demand = -5)
         G.add_node('d', demand = 5)
         G.add_edge('a', 'b', weight = 3, capacity = 4)
         G.add_edge('a', 'c', weight = 6, capacity = 10)
         G.add_edge('b', 'd', weight = 1, capacity = 9)
         G.add_edge('c', 'd', weight = 2, capacity = 5)
         flowDict = nx.min_cost_flow(G)
```

```
        ---------------------------------------------------------------------------

        ImportError                               Traceback (most recent call last)

        <ipython-input-54-6a042a2b4ba5> in <module>()
    ----> 1 import networkx as nx
```

```
  2 G = nx.DiGraph()
  3 G.add_node('a', demand = -5)
  4 G.add_node('d', demand = 5)
  5 G.add_edge('a', 'b', weight = 3, capacity = 4)


ImportError: No module named networkx
```

In [55]:
```
!pip install networkx
```

```
Requirement already satisfied (use --upgrade to upgrade): networkx in /Library/Fran
Requirement already satisfied (use --upgrade to upgrade): decorator>=3.4.0 in /Libr
```

In [56]:
```
import networkx as nx
```

```
---------------------------------------------------------------------------

ImportError                               Traceback (most recent call last)

<ipython-input-56-6002206e7c09> in <module>()
----> 1 import networkx as nx


ImportError: No module named networkx
```

In [ ]: