shortPath nb updating values

April 29, 2018

In [1]:
```
# Austin Griffith
# Python 3.6.5
# 4/25/2018

import pandas as pd
import numpy as np
from gurobipy import *
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import networkx as nx
```

In [2]:
```
# set up plotting parameters
params = {'legend.fontsize': 20,
          'figure.figsize': (13,9),
         'axes.labelsize': 20,
         'axes.titlesize':20,
         'xtick.labelsize':15,
         'ytick.labelsize':15}
pylab.rcParams.update(params)
```

In [3]:
```
# graph all nodes and paths
def networkCompletePlot(solution,maxNode):
    G = nx.DiGraph()
    G.add_nodes_from(range(0,maxNode+1))
    for i,j in nodes:
        G.add_edge(i,j)

    # get solution nodes
    sp = [i for i,j in solution[1]]
    sp.append(end)

    colorNode = ['white' if not node in sp else 'red' for node in G.nodes()]
    title = 'Complete Network: Gamma = '+str(int(solution[0]))+', Opt Obj = '+str(round(
    nx.draw_networkx(G,node_color=colorNode,node_size=200)
    plt.axis('off')
    plt.title(title)
    plt.show()
```

```
# graph path, with costs on edges
def networkPathPlot(solution,maxNode,cost):
    # get solution nodes
    sp = [i for i,j in solution[1]]
    sp.append(end)

    # set up random position values
    a = np.arange(maxNode+1)
    b = np.arange(maxNode+1)
    np.random.shuffle(a)
    posArray = np.array([a,b]).transpose()

    positions = {}
    for p in range(0,len(sp)):
        L = posArray[p]
        positions[sp[p]] = (L[0],L[1])

    # set up network graph
    G = nx.DiGraph()
    G.add_nodes_from(sp)

    for i,j in tuplelist(solution[1]):
        G.add_edge(i,j)

    labels = {}
    for i in solution[1]:
        labels[i] = round(c[i],3)

    title = 'Optimal Path: Gamma = '+str(int(solution[0]))+', Opt Obj = '+str(round(solu
    nx.draw_networkx(G,positions,node_size=350)
    nx.draw_networkx_edge_labels(G,positions,edge_labels=labels)
    plt.axis('off')
    plt.title(title)
    plt.show()
```

In [4]:
```
# pull data
edges = pd.read_csv('edge_values.csv')
edges['i'] = np.int64(edges['i'])
edges['j'] = np.int64(edges['j'])

# create dictionaries of edge values
c = {}
d = {}
nodes = tuplelist()
for i in edges.index:
    c[edges['i'][i],edges['j'][i]] = edges['c(ij)'][i]
    d[edges['i'][i],edges['j'][i]] = edges['d(ij)'][i]
```

```
        nodes.append((edges['i'][i],edges['j'][i]))

    maxNodes = max(edges['j'])
    minNodes = min(edges['i'])
```

In [5]:
```
# choose start and end nodes
start = 0
end = int(maxNodes)

# allowed edge congestions
gend = 4
gammas = np.linspace(0,gend,gend+1)
print('Allowed Congestions:')
print(gammas)
```

```
Allowed Congestions:
[ 0.  1.  2.  3.  4.]
```

In [6]:
```
# initialize model
model = Model('Shortest_Path')

# set up x binary variables, set to each location/movement
xVars = model.addVars(nodes, vtype=GRB.BINARY, name='move')
y0 = model.addVar(vtype=GRB.CONTINUOUS, name='y0')
zVars = model.addVars(nodes, lb=0.0, vtype=GRB.CONTINUOUS, name='cong')
model.update()
```

In [7]:
```
# constrain all entrance and exit nodes
enterStart = []
leaveStart = []
enterEnd = []
leaveEnd = []
for n in nodes:
    # for start nodes
    if n[0] == start:
        leaveStart.append(xVars[n])
    elif n[1] == start:
        enterStart.append(xVars[n])
    # for end nodes
    if n[0] == end:
        leaveEnd.append(xVars[n])
    elif n[1] == end:
        enterEnd.append(xVars[n])

model.addConstr(quicksum(leaveStart) == 1)
model.addConstr(quicksum(enterStart) == 0)
model.addConstr(quicksum(leaveEnd) == 0)
```

```
    model.addConstr(quicksum(enterEnd) == 1)
    model.update()
```

In [8]:
```
# gather all paths
paths = []
for i in range(minNodes+1,maxNodes):
    pathFrom = []
    pathTo = []
    for n in nodes:
        if n[0] == i:
            pathFrom.append(xVars[n])
        elif n[1] == i:
            pathTo.append(xVars[n])
    paths.append([pathFrom,pathTo])
model.update()

for p in paths:
    model.addConstr(quicksum(p[0]) - quicksum(p[1]) == 0.0)
model.update()

print('Example of Path Constraint for a Given Node:')
print(quicksum(p[0]) - quicksum(p[1]))
```

```
Example of Path Constraint for a Given Node:
<gurobi.LinExpr: move[28,22] + move[28,11] + move[28,15] + move[28,28] + move[28,9] + move[28,19
```

In [9]:
```
# objective function
costObj = []
for n in nodes:
    costObj.append(xVars[n]*c[n])
    model.addConstr(zVars[n] >= xVars[n]*d[n] - y0)
model.update()

print('Example of Congestion Constraint:')
print(zVars[n],' >= ',xVars[n]*d[n] - y0)
```

```
Example of Congestion Constraint:
<gurobi.Var cong[30,6]>  >=  <gurobi.LinExpr: 4.577964888409282 move[30,6] + -1.0 y0>
```

In [10]:
```
# iterate optimization through various gammas (congestions)
output = []
for g in gammas:
    # optimize
    objective = quicksum(costObj) + g*y0 + quicksum(zVars)
    model.setObjective(objective, GRB.MINIMIZE)

    model.optimize()
```

```
            # order the printout of optimal edges
            moves = []
            for m in xVars:
                if xVars[m].x != 0:
                    moves.append(m)
            order = [moves[0]]
            for i in range(len(moves)):
                for m in moves:
                    if order[i][1] == m[0]:
                        order.append(m)
            output.append([g,order,model.objVal])
```

```
Optimize a model with 795 rows, 1527 columns and 3761 nonzeros
Variable types: 764 continuous, 763 integer (763 binary)
Coefficient statistics:
  Matrix range     [9e-04, 5e+00]
  Objective range  [7e-03, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 0.8726727
Presolve removed 765 rows and 895 columns
Presolve time: 0.00s
Presolved: 30 rows, 632 columns, 1215 nonzeros
Variable types: 0 continuous, 632 integer (632 binary)

Root relaxation: objective 2.816806e-01, 13 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

*    0     0               0       0.2816806    0.28168  0.00%     -    0s

Explored 0 nodes (13 simplex iterations) in 0.02 seconds
Thread count was 8 (of 8 available processors)

Solution count 2: 0.281681 0.872673

Optimal solution found (tolerance 1.00e-04)
Best objective 2.816805681065e-01, best bound 2.816805681065e-01, gap 0.0000%
Optimize a model with 795 rows, 1527 columns and 3761 nonzeros
Variable types: 764 continuous, 763 integer (763 binary)
Coefficient statistics:
  Matrix range     [9e-04, 5e+00]
  Objective range  [7e-03, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
```

```
Loaded MIP start with objective 5.28168

Presolve removed 71 rows and 832 columns
Presolve time: 0.02s
Presolved: 724 rows, 695 columns, 2726 nonzeros
Found heuristic solution: objective 5.2811299
Variable types: 1 continuous, 694 integer (694 binary)

Root relaxation: objective 9.033342e-01, 73 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    0.90333    0   40    5.28113    0.90333  82.9%     -    0s
H    0     0                       4.0651681    0.90333  77.8%     -    0s
H    0     0                       1.5399132    0.90333  41.3%     -    0s
     0     0    1.38375    0    9    1.53991    1.38375  10.1%     -    0s
     0     0    1.44475    0    5    1.53991    1.44475  6.18%     -    0s

Explored 1 nodes (124 simplex iterations) in 0.06 seconds
Thread count was 8 (of 8 available processors)

Solution count 3: 1.53991 4.06517 5.28113

Optimal solution found (tolerance 1.00e-04)
Best objective 1.539913208932e+00, best bound 1.539913208932e+00, gap 0.0000%
Optimize a model with 795 rows, 1527 columns and 3761 nonzeros
Variable types: 764 continuous, 763 integer (763 binary)
Coefficient statistics:
  Matrix range     [9e-04, 5e+00]
  Objective range  [7e-03, 2e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]

Loaded MIP start with objective 2.20427

Presolve removed 29 rows and 54 columns
Presolve time: 0.01s
Presolved: 766 rows, 1473 columns, 3628 nonzeros
Variable types: 737 continuous, 736 integer (736 binary)

Root relaxation: objective 1.281366e+00, 61 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    1.28137    0   33    2.20427    1.28137  41.9%     -    0s
     0     0    1.57175    0   22    2.20427    1.57175  28.7%     -    0s
```

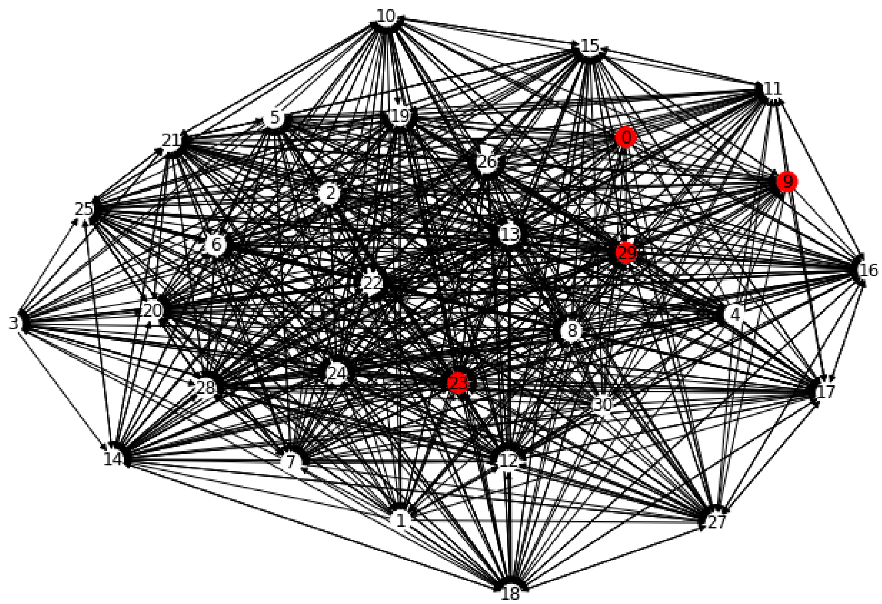
```
    0     0    1.57912    0   15    2.20427    1.57912  28.4%     -    0s
H   0     0                       1.7697391    1.57912  10.8%     -    0s
    0     0    1.71081    0   12    1.76974    1.71081  3.33%     -    0s
    0     0     cutoff    0         1.76974    1.76974  0.00%     -    0s

Explored 1 nodes (156 simplex iterations) in 0.07 seconds
Thread count was 8 (of 8 available processors)

Solution count 2: 1.76974 2.20427

Optimal solution found (tolerance 1.00e-04)
Best objective 1.769739091273e+00, best bound 1.769739091273e+00, gap 0.0000%
Optimize a model with 795 rows, 1527 columns and 3761 nonzeros
Variable types: 764 continuous, 763 integer (763 binary)
Coefficient statistics:
  Matrix range     [9e-04, 5e+00]
  Objective range  [7e-03, 3e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]

Loaded MIP start with objective 1.99956

Presolve removed 29 rows and 54 columns
Presolve time: 0.00s
Presolved: 766 rows, 1473 columns, 3628 nonzeros
Variable types: 737 continuous, 736 integer (736 binary)

Root relaxation: objective 1.515815e+00, 51 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

    0     0    1.51582    0   26    1.99956    1.51582  24.2%     -    0s
    0     0    1.81369    0   12    1.99956    1.81369  9.30%     -    0s
*   0     0               0       1.8554621    1.85546  0.00%     -    0s

Explored 1 nodes (78 simplex iterations) in 0.05 seconds
Thread count was 8 (of 8 available processors)

Solution count 2: 1.85546 1.99956

Optimal solution found (tolerance 1.00e-04)
Best objective 1.855462126819e+00, best bound 1.855462126819e+00, gap 0.0000%
Optimize a model with 795 rows, 1527 columns and 3761 nonzeros
Variable types: 764 continuous, 763 integer (763 binary)
Coefficient statistics:
  Matrix range     [9e-04, 5e+00]
  Objective range  [7e-03, 4e+00]
```

```
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]

Loaded MIP start with objective 1.85546

Presolve removed 29 rows and 54 columns
Presolve time: 0.00s
Presolved: 766 rows, 1473 columns, 3628 nonzeros
Variable types: 737 continuous, 736 integer (736 binary)

Root relaxation: objective 1.687283e+00, 52 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    1.68728    0   33    1.85546    1.68728  9.06%     -    0s

Cutting planes:
  Gomory: 1

Explored 1 nodes (52 simplex iterations) in 0.04 seconds
Thread count was 8 (of 8 available processors)

Solution count 1: 1.85546

Optimal solution found (tolerance 1.00e-04)
Best objective 1.855462126819e+00, best bound 1.855462126819e+00, gap 0.0000%
```

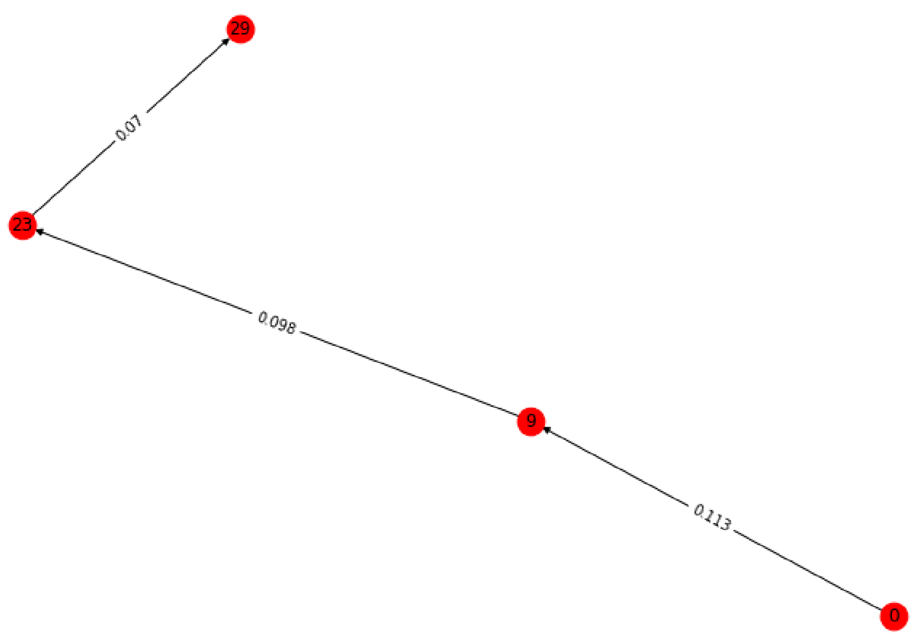
In [11]:
```python
# print optimal values and paths, plot network
for o in output:
    print('\nFor Gamma: '+str(o[0]))
    print('Path:')
    print(o[1])
    print('Cost of Movement (Objective):')
    print(o[2])
    networkCompletePlot(o,maxNodes)
    networkPathPlot(o,maxNodes,c)
```

```
For Gamma: 0.0
Path:
[(0, 9), (9, 23), (23, 29)]
Cost of Movement (Objective):
0.281680568106491
```

Complete Network: Gamma = 0, Opt Obj = 0.28168

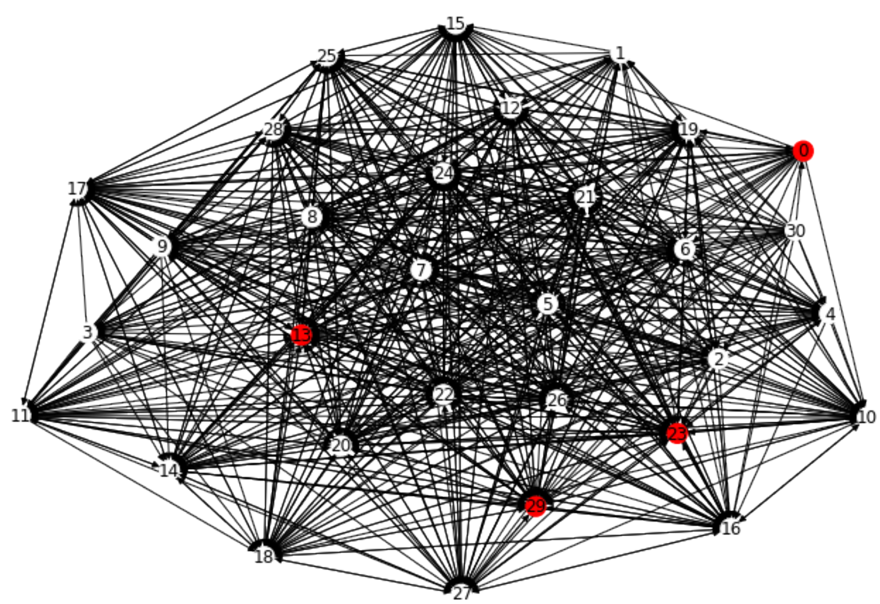


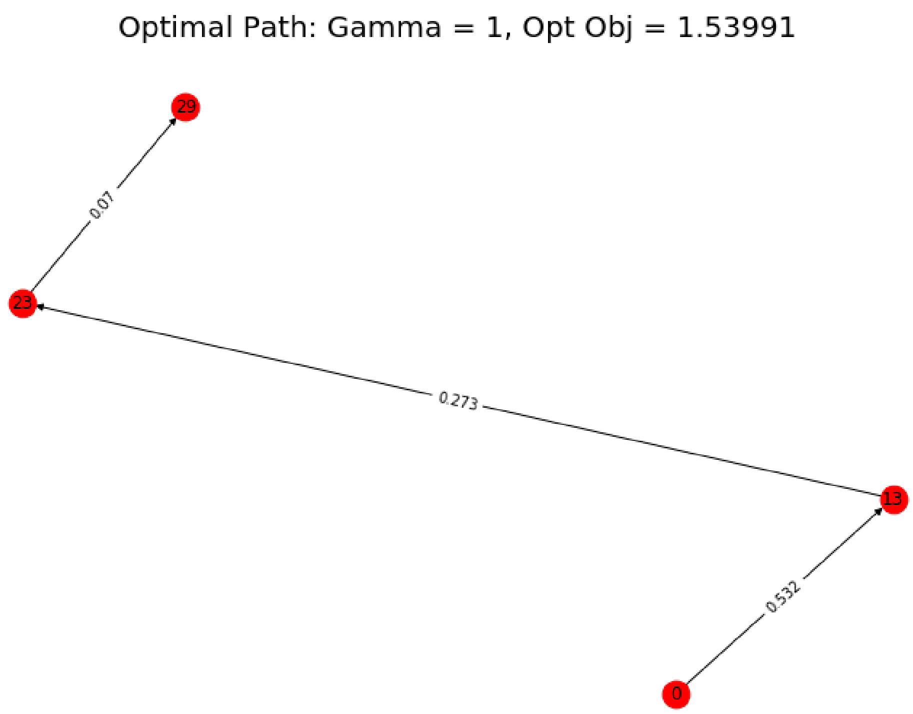
Optimal Path: Gamma = 0, Opt Obj = 0.28168



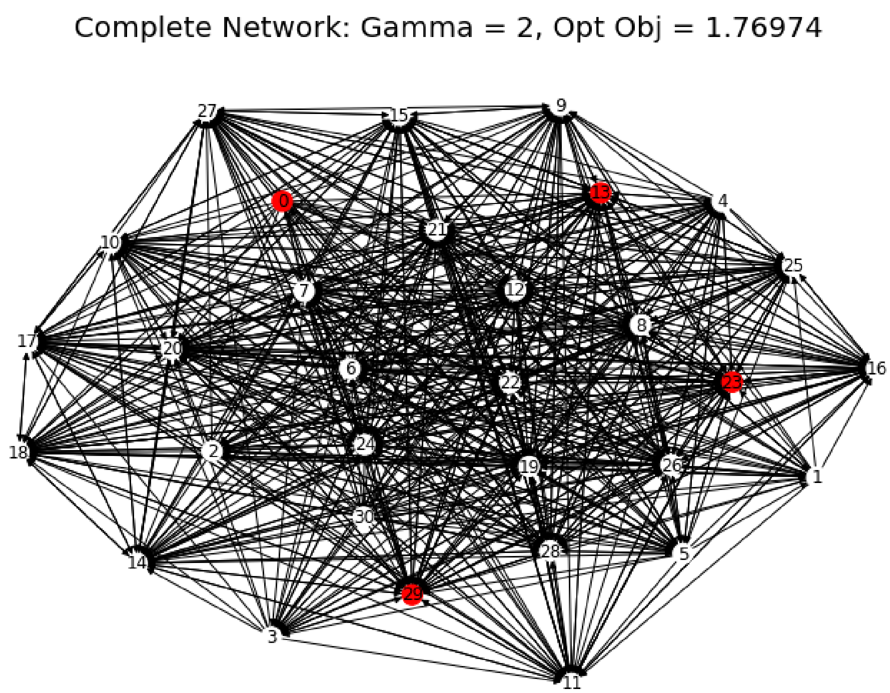
```
For Gamma: 1.0
Path:
[(0, 13), (13, 23), (23, 29)]
Cost of Movement (Objective):
1.5399132089321064
```

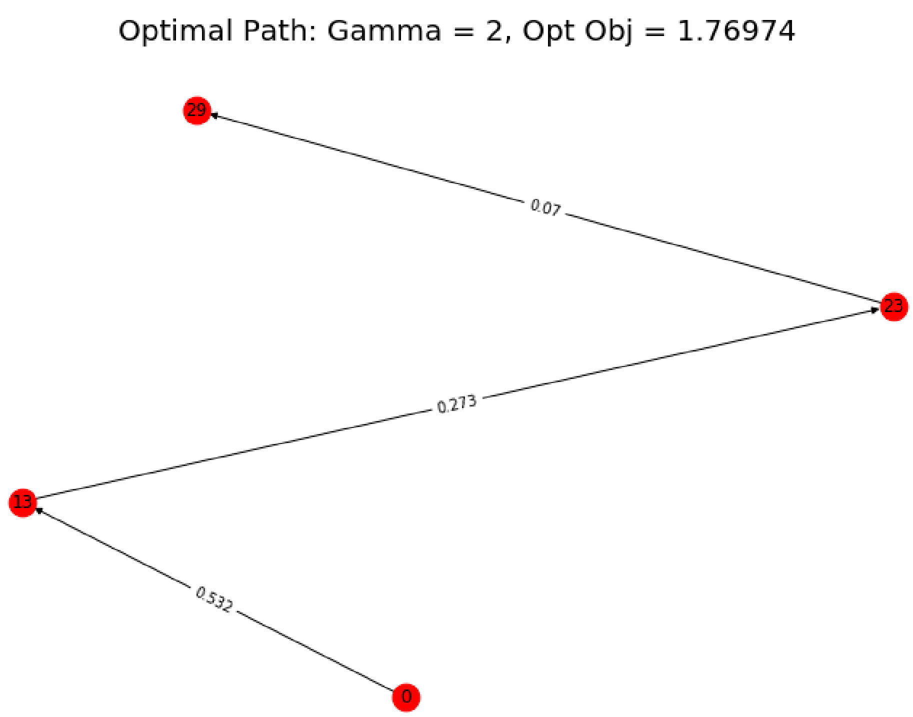
Complete Network: Gamma = 1, Opt Obj = 1.53991



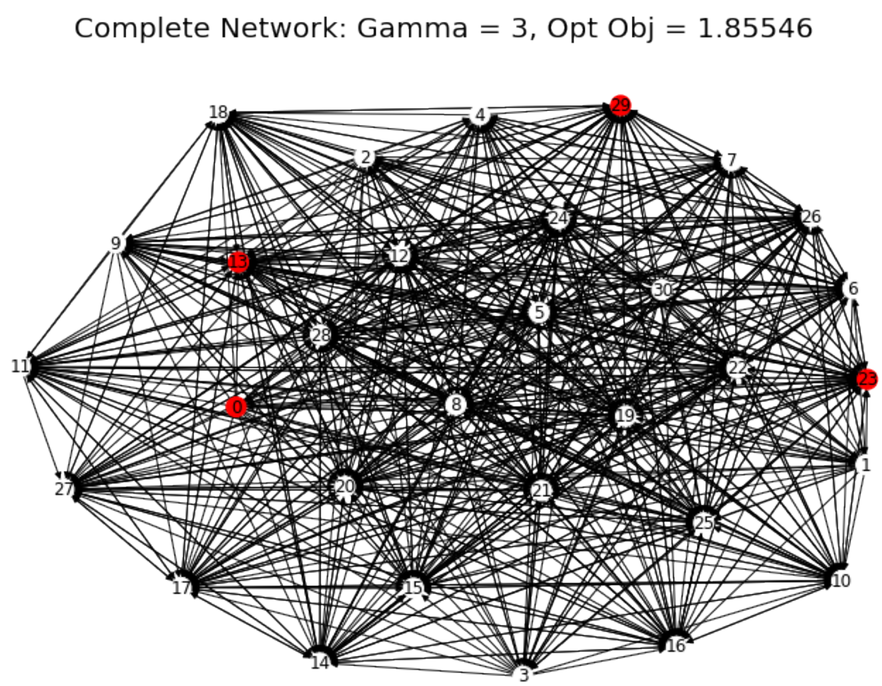


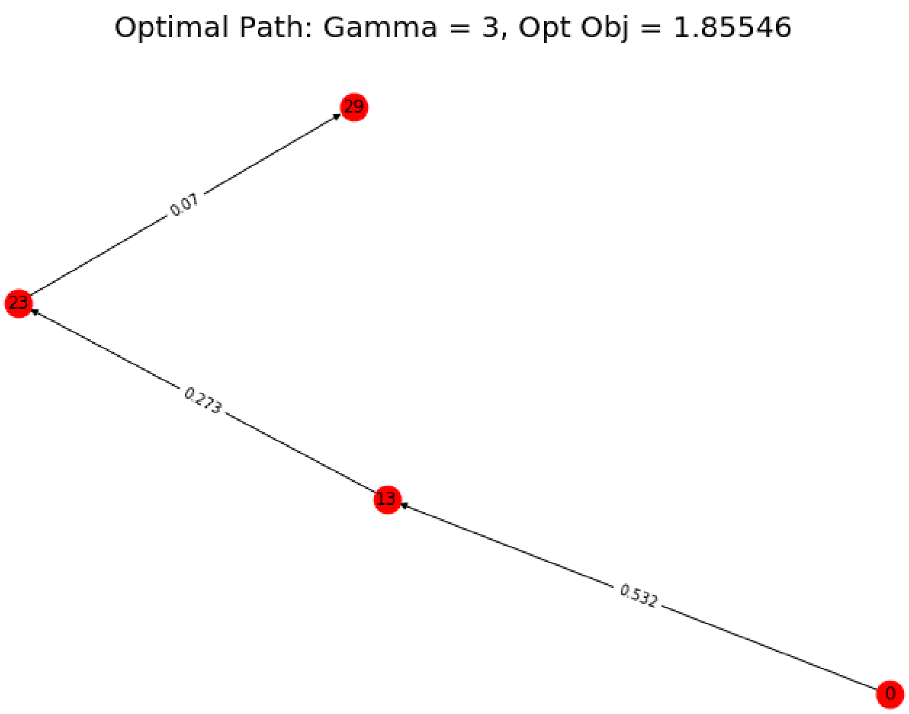

```
For Gamma: 2.0
Path:
[(0, 13), (13, 23), (23, 29)]
Cost of Movement (Objective):
1.7697390912734972
```


Complete Network: Gamma = 2, Opt Obj = 1.76974
27
15
9
13
0
4
21
10
25
7
12
8
17
20
6
22
23
16
18
2
24
19
26
1
30
28
5
14
29
3
11


Optimal Path: Gamma = 2, Opt Obj = 1.76974
29
0.07
23
0.273
13
0.532
0

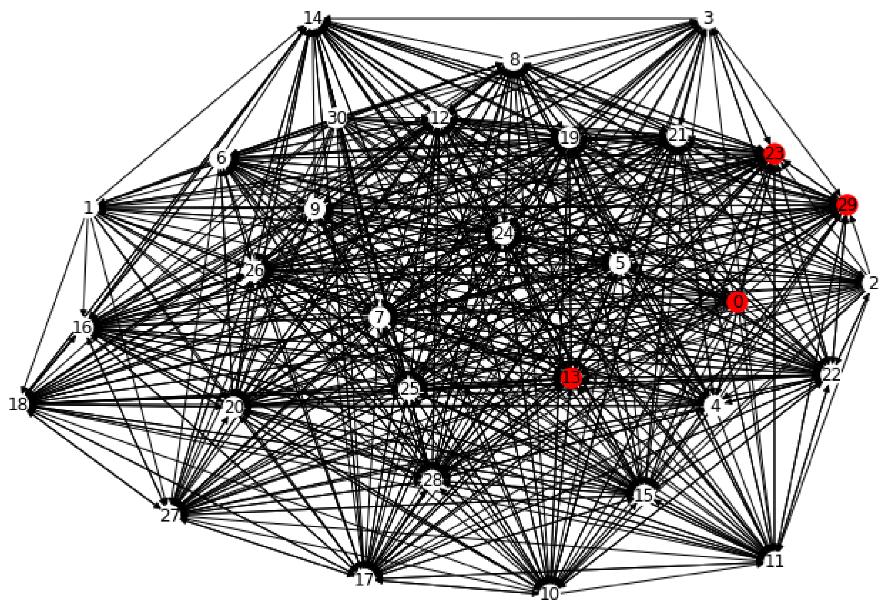
```
For Gamma: 3.0
Path:
[(0, 13), (13, 23), (23, 29)]
Cost of Movement (Objective):
1.85546212681852
```






```
For Gamma: 4.0
Path:
[(0, 13), (13, 23), (23, 29)]
Cost of Movement (Objective):
1.85546212681852
```

Complete Network: Gamma = 4, Opt Obj = 1.85546



Optimal Path: Gamma = 4, Opt Obj = 1.85546