03Optimization

December 19, 2019

[1]:

```
import networkx as nx
from copy import deepcopy
import queue
import matplotlib.pyplot as plt
import csv
import numpy as np
import pandas as pd
nodes_file = csv.reader(open('data/nodes.csv','r'));
links_file = csv.reader(open('data/links.csv','r'));
```

[6]:

```
G_network=nx.Graph()
G_risk_logit= nx.Graph()
G_risk_poisson= nx.Graph()
G_risk_xgboost= nx.Graph()
G_risk_ANN= nx.Graph()

tmp = 0
for row in nodes_file:
    if (tmp > 0):
        G_network.add_node(row[0])
        G_risk_logit.add_node(row[0])
        G_risk_poisson.add_node(row[0])
        G_risk_xgboost.add_node(row[0])
        G_risk_ANN.add_node(row[0])

    tmp=+1


tmp=0
for row in links_file:
    if (tmp>0): # Ignores the first line in the file
        G_network.add_edge(row[0],row[1]);
        G_network[row[0]][row[1]]['weight']=float(row[2]);
#         first risk model
        G_risk_logit.add_edge(row[0], row[1]);
        G_risk_logit[row[0]][row[1]]['weight']=float(row[3]);
#         second risk model
```

```
        G_risk_poisson.add_edge(row[0], row[1]);
        G_risk_poisson[row[0]][row[1]]['weight']=float(row[4]);
        G_risk_xgboost.add_edge(row[0], row[1]);
        G_risk_xgboost[row[0]][row[1]]['weight']=float(row[5]);
        G_risk_ANN.add_edge(row[0], row[1]);
        G_risk_ANN[row[0]][row[1]]['weight']=float(row[6]);

    tmp += 1;
```

```
      ␣
↪---------------------------------------------------------------------------

      IndexError                                Traceback (most recent call␣
↪last)

      <ipython-input-6-cf710fbd30b1> in <module>
       25 #         first risk model
       26           G_risk_logit.add_edge(row[0],row[1]);
  ---> 27           G_risk_logit[row[0]][row[1]]['weight']=float(row[3]);
       28 #         second risk model
       29           G_risk_poisson.add_edge(row[0],row[1]);


      IndexError: list index out of range
```

[ ]:

```
#redefine the network and find the total risk for each path



#get the risk for a certain path

# getEdge and drawP are for drawing
def getEdge(p):
   draw_edge = []
   for i in range(len(p)-1):
       a = (p[i],p[i+1])
       draw_edge.append(a)
   return draw_edge

def drawP(G_network,p,pos):
#    c,p = nx.single_source_dijkstra(G_network,"Ann_Arbor","Seattle")
    #print(p)
#    pos = nx.spring_layout(G_network)
    draw_edge = getEdge(p)
    nx.draw_networkx_nodes(G_network, pos,
                           node_size=100,
                           alpha=0.8)
    nx.draw_networkx_nodes(G_network, pos,nodelist=p,
                           node_size=100,
                           alpha=0.8)
    nx.draw_networkx_edges(G_network, pos, width=1, alpha=0.5)
    nx.draw_networkx_edges(G_network, pos, edgelist=draw_edge, \
                           width=5, alpha=0.5,edge_color='r')
    nx.draw_networkx_labels(G_network, pos,font_size=8)
```

[3]:

```
# Yen's algorithm for K-shortest paths in an edge-weighted graph G (undirected
# or directed)

# Cost/weight of path p in graph G
def pweight(G,p):
    w = 0;
    for i in range(len(p)-1):
#         print(p[i])
        w += G[p[i]][p[i+1]]['weight'];
    return w
```

[4]:

```
# Copy edge (a,z) of G, remove it, and return the copy.
# This can become expensive!
def cprm(G,a,z):
    ec = G[a][z]['weight'];
    G.remove_edge(a,z);
    return (a,z,ec)

# Copy node n of G, remove it, and return the copy.
# This can become expensive!
def cprmnode(G,n):
    ec = deepcopy(G[n]);
    G.remove_node(n);
    return (n,ec)

# K shortest paths in G from 'source' to 'target'
def yen(G,source,target,K):
    # Determine the shortest path from the source to the sink.
    (c,p) = nx.single_source_dijkstra(G,source,target);
    A = [p];  A_cost = [c];
    # Initialize the set to store the potential kth shortest path.
    B = queue.PriorityQueue();

    for k in range(1,K):
        # The spur node ranges from the first node to the next to last node in␣
 ↪the previous k-shortest path.
        for i in range(len(A[k-1])-1):
            # Spur node is retrieved from the previous k-shortest path, k - 1.
            sn = A[k-1][i];
            # The sequence of nodes from the source to the spur node of the␣
↪previous k-shortest path.
            rp = A[k-1][:i];

            # We store the removed edges
            removed_edges = [];  removed_root_edges = []; ␣
↪removed_root_nodes=[];
            # Remove the root paths

            # for each node rootPathNode in rootPath except spurNode:
            #   remove rootPathNode from Graph;
            for j in range(len(rp)):


                extra_edges = deepcopy(G.edges(rp[j]));

                for eg in extra_edges:

                    src=eg[0];
                    tgt=eg[1];
                    removed_root_edges.append(cprm(G,src,tgt));

                removed_root_nodes.append(cprmnode(G,rp[j]));



            erp = A[k-1][:i+1];  # extended root path
            for p in A:
                if erp == p[:i+1] and G.has_edge(p[i],p[i+1]):
                    removed_edges.append(cprm(G,p[i],p[i+1]));
            # The spur path
            DONE = 0
            try:
                (csp,sp) = nx.single_source_dijkstra(G,sn,target)
            except:
                # there is no spur path if sn is not connected to the target
                sp = [];  csp = None; DONE = 1;
                #return (A, A_cost)
            # Add back the edges that were removed
            for nd in removed_root_nodes: G.add_node(nd[0]);
            for re in removed_root_edges: G.add_edge(re[0],re[1],weight=re[2]);
            for re in removed_edges: G.add_edge(re[0],re[1],weight=re[2]);
            if len(sp) > 0:
                # The potential k-th shortest path (the root path may be empty)
                pk = rp + sp;
```

```
#                print(pk)
                cpk = pweight(G,pk);
                # Add the potential k-shortest path to the heap
                B.put((cpk,pk));



        if B.empty():
            print ('There are only', k, 'shortest paths for this pair');
            break;
        # The shortest path in B that is not already in A is the new k-th shortest path
        while not B.empty():
            cost, path = B.get();
            if path not in A:
                A.append(path);
                A_cost.append(cost);
                break;

    return (A, A_cost)
```

[5]:

```
src='node 1';
tgt='node 14';
k=4;

k_path, path_costs = yen(G_network,src,tgt,k);
pos = nx.spring_layout(G_network)
result = []


for i in range(k):
#    print(k_path[i],path_costs[i])
    rank = i+1
    plt.figure(i+1)
    t = "This is the path ranked as {}"
    plt.suptitle(t.format(rank))

    drawP(G_network,k_path[i],pos)

    if i == 0:
        plt.savefig('first.png', dpi = 1200)
    if i == 1:
        plt.savefig('second.png', dpi = 1200)
    if i == 2:
        plt.savefig('third.png', dpi = 1200)
    if i == 3:
        plt.savefig('four.png', dpi = 1200)
    if i == 4:
        plt.savefig('five.png', dpi = 1200)

    r1 = pweight(G_risk_logit,k_path[i])

    r2 = pweight(G_risk_poisson,k_path[i])

    r3 = pweight(G_risk_xgboost,k_path[i])

    r4 = pweight(G_risk_ANN,k_path[i])

    b = (k_path[i],path_costs[i],r1,r2,r3,r4)
    result.append(b)
print(result)
```

```
      ␣
↪---------------------------------------------------------------------------

       KeyError                                  Traceback (most recent call␣
↪last)

       D:
↪\ProgramData\Anaconda3\envs\py37\lib\site-packages\networkx\algorithms\shortest_paths\weight
↪py in multi_source_dijkstra(G, sources, target, cutoff, weight)
       743     try:
   --> 744         return (dist[target], paths[target])
       745     except KeyError:


       KeyError: 'node 14'


   During handling of the above exception, another exception occurred:


       NetworkXNoPath                            Traceback (most recent call␣
↪last)

       <ipython-input-5-6810c969a325> in <module>
         3 k=4;
         4
   ----> 5 k_path, path_costs = yen(G_network,src,tgt,k);
         6 pos = nx.spring_layout(G_network)
         7 result = []
```

```
    <ipython-input-4-66615782333d> in yen(G, source, target, K)
     16 def yen(G,source,target,K):
     17     # Determine the shortest path from the source to the sink.
---> 18     (c,p) = nx.single_source_dijkstra(G,source,target);
     19     A = [p];  A_cost = [c];
     20     # Initialize the set to store the potential kth shortest path.


    D:
↪\ProgramData\Anaconda3\envs\py37\lib\site-packages\networkx\algorithms\shortest_paths\weight
↪py in single_source_dijkstra(G, source, target, cutoff, weight)
    478     """
    479     return multi_source_dijkstra(G, {source}, cutoff=cutoff,␣
↪target=target,
--> 480                                  weight=weight)
    481
    482


    D:
↪\ProgramData\Anaconda3\envs\py37\lib\site-packages\networkx\algorithms\shortest_paths\weight
↪py in multi_source_dijkstra(G, sources, target, cutoff, weight)
    744         return (dist[target], paths[target])
    745     except KeyError:
--> 746         raise nx.NetworkXNoPath("No path to {}.".format(target))
    747
    748


    NetworkXNoPath: No path to node 14.
```

```
[ ]: df = pd.DataFrame(result,columns␣
      ↪=['path','distance','risk_logit','risk_poi','risk_Xgboost','ANN'])

     df.to_csv('result_k=4.csv')
```