CSCI 5352

Cassandra Spath **Problem Set 5** November 6, 2019

1. (100 pts total) Minimum Violations Rankings. Here, you will implement the minimum violations algorithm and explore rankings in directed random networks

   (a) (60 pts) Coding up and testing the MVR algorithm. Recall the MVR algorithm that we discussed in class: let π be an ordering of the network's n vertices, such that π_i is the ranking of vertex i, and using the convention that n is the best ranking and 1 is the worst ranking. Let $V(\pi)$ be the number of directed edges that violate the ordering by pointing from a lower ranked vertex to a higher ranked vertex. In other words, in a network where A_{ij} denotes an edge from i to j,

   $$V(\pi) = \sum_{ij} A_{ij} I_{\pi_i \leq \pi_j}$$

   where $I_{\pi_i \leq \pi_j}$ is an indicator function taking on the value 1 when $\pi_i \leq \pi_j$, and 0 otherwise.

   Test cases:

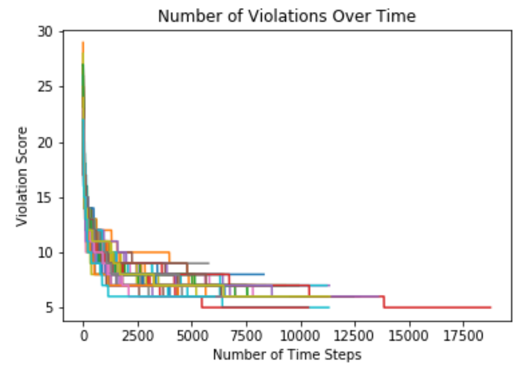
   i. a directed chain of $n = 50$ vertices such that $A_{i,i+1} = 1$ for $i = 1, 2, \ldots, 49$.

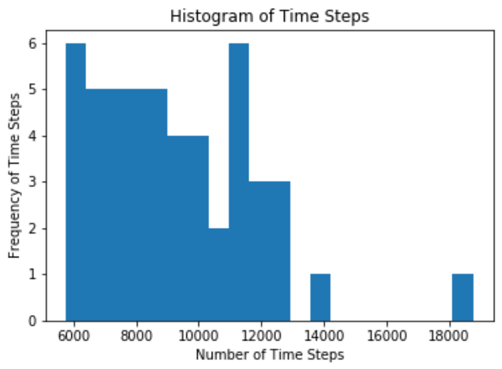
   ii. a $G(n = 50, p = 0.15)$ random network in which edges are made directed by forcing each undirected edge between i and j to point from i to j when $i < j$ and from j to i when $j < i$.

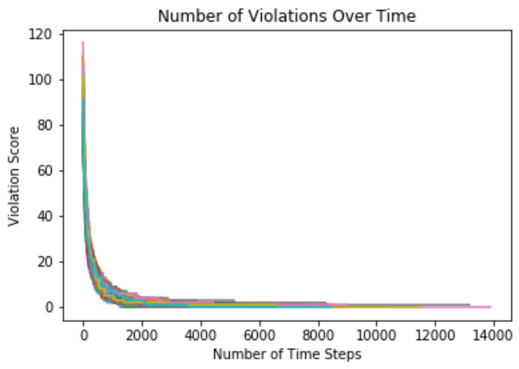
   - Expected final ranking of the algorithm to be for both cases.
   - Plot line-plot of number of violations V vs the number of timesteps t for each repetition of the algorithm and a histogram of the total number of timesteps elapsed when the algorithm stop.

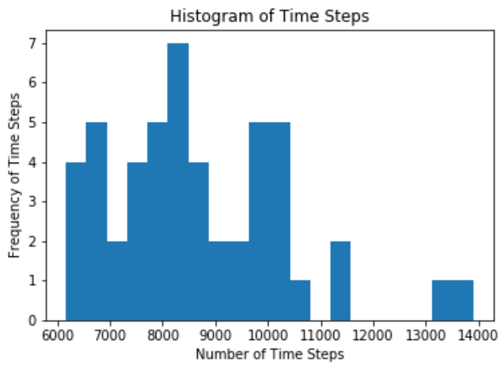
   I would expect that the ranking of the chain of nodes to get close to the correct ranking where each node has the ranking of the node number. I expect that this ranking will almost be achieved with a couple of nodes having violations since they weren't chosen for a switch.

   I expect that the random graph will have similar results since the nodes have a clear ranking of node number. There will be more nodes in the random graph since there 184 on average compared to 49 for the chain. Since there are more edges in the random graph, I would expect that on average the number of violations will be smaller for the random graph since there are more edges that could found that result in a swap that improves the number of violations.


Number of Violations Over Time
Violation Score
Number of Time Steps


Histogram of Time Steps
Frequency of Time Steps
Number of Time Steps


Number of Violations Over Time
Violation Score
Number of Time Steps


Histogram of Time Steps
Frequency of Time Steps
Number of Time Steps

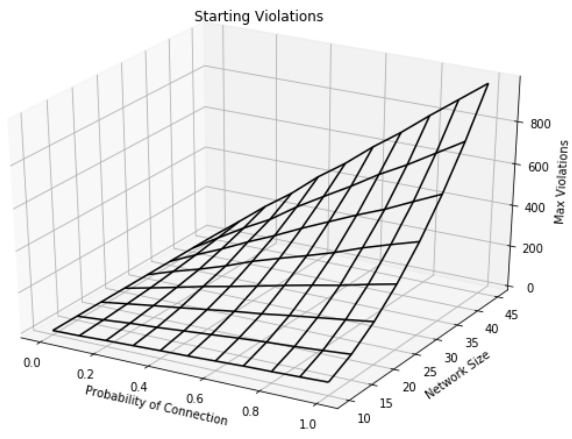
(b) (40 pts) Misled! Rankings in randomness! Here, you will generate random directed networks, in which we expect no meaningful linear hierarchy to exist. However, just like modularity maximization, we can always minimize violations, even if the resulting communities or hierarchy are simply fluctuations in randomness. You will explore how the parameters of a directed G(n, p) network affect the apparent strength of hierarchies that the MVR algorithm finds.

Conduct an exploration of the effects of n and p on the expected number of violations found by the MVR algorithm when applied to directed $G(n, p)$ networks. Write a brief one-page summary of this exploration by telling me (1) how you attacked the problem, (2) what you found, (3) and what you expect to happen if we dramatically increase the size of the network or the probability of connection p. You may include any figures or mathematics that you wish!

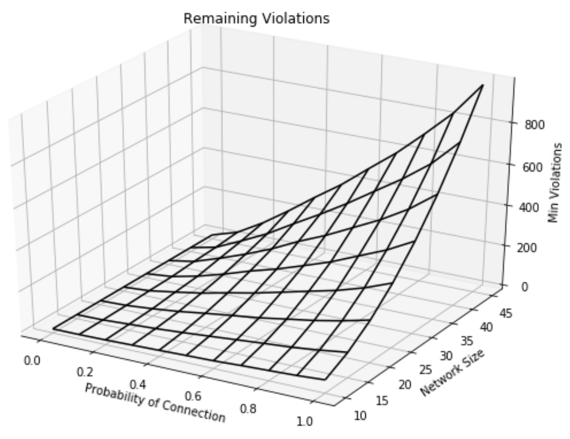
I approached the problem by examining different combinations of n and p. This allowed me to get an idea of the effects of these values on the number of violations. For each combination, I ran 50 instances to eliminate noisy results.

When $p = 0$, there are no edges in the graph which means that there no violations. For any smaller values of p, there are few edges which makes it easy to eliminate any violations since there are many valid rankings.

On the other end of these spectrum, when $p = 1$, all possible edges in the graph making it impossible to eliminate violations since there are so many violations. For any near 1 value of p, there are many violations due to the number of edges that exist in the graph.



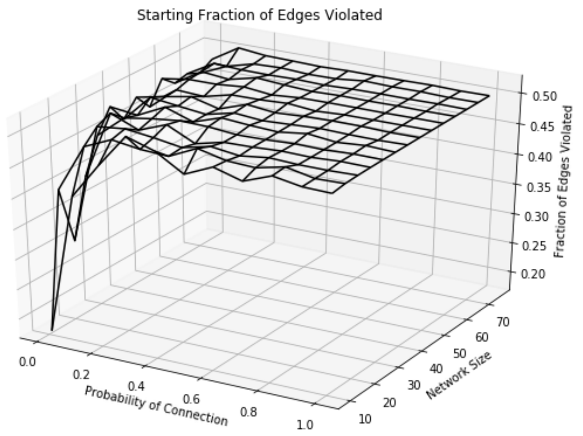
Starting Number of Violations



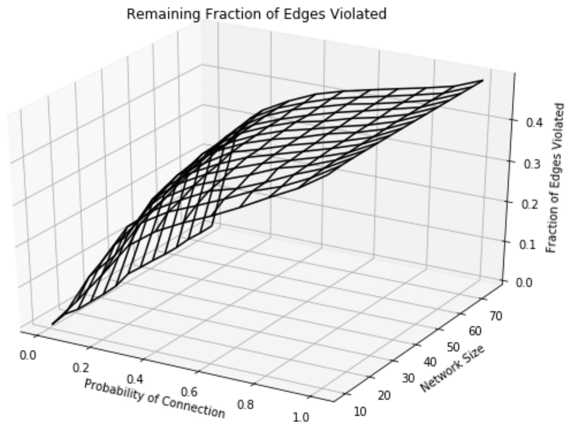
Ending Number of Violations

When evaluating the absolute number of violations in the graph, the higher number of nodes results in more edges in the random graph and more violations. For the starting number of violations, the graph is close to flat plane. This is because the number initial violations is approximately number of edges in the graph. Thus, it is directly proportional to the probability of an edge and the number of edges.

When examining the number of violations after running MVR, there is a curve in the graph. For larger values of p, there are a larger number of violations. However, there is a clear decrease in the number of violations from the starting point. To better show the decrease in the number of violations, we will look at graphs for the fraction of edges being violated.



Starting Fraction of Violations



Ending Fraction of Violations

For the fraction of edges violated, for any nonzero p value, approximately half the edges are initially violated. This is true regardless of the n value. This logically makes sense because a random ranking of a random graph will result in about half the edges being violated.

For the fraction of edges violated at the end, values of p greater than 0.8 still have approximately 40% to 50% of the edges violated. For values of p less than 0.1 there are less than 15% of the edges violated. The remaining p values from 0.1 to 0.8 have a increasing fraction of violations from 15% to 40%. For smaller values of n, the fraction of edges violated at the end of running MVR will decrease slightly since there are fewer edges.

Algorithm 1: Python Code for Problems

```
import numpy as np
import matplotlib.pyplot as plt
import os
import networkx as nx
import random
from mpl_toolkits import mplot3d
import math
import copy
from statistics import mean

#PROBLEM 1a
#create directed chain of nodes with given n
def createDirectedChain(n=50):
    G = nx.DiGraph()
    for i in range(n-1):
        #create edges
        G.add_edge(i, i+1)
    return G

#create random directed network with given n and probability
def createRandomGraph(n=50, p=0.15):
    G = nx.DiGraph()
    G.add_nodes_from(list(range(n)))
    for i in range(n):
        for j in range(i+1, n):
            if random.random() < p:
                G.add_edge(i,j)
    return G

def violationScore(G, ordering):
    v = 0
    for (i,j) in G.edges:
        v += (ordering[i] < ordering[j])
    return v

def MVR(G):
    #create random ordering
    ordering = list(range(len(G)))
    random.shuffle(ordering)
    #get initial violations score
    violation = violationScore(G, ordering)
    vList = [violation]
    #run algorithm
    t = 1
    timeSinceSwap = 0
    stoppingPoint = math.factorial(len(G))*2/math.factorial(len(G)-2)
    while(timeSinceSwap < stoppingPoint):
        #pick 2 random nodes
        nodeI = random.randint(0,len(G)-1)
        nodeJ = random.randint(0,len(G)-1)
        #create new proposed ordering with swap
        proposedO = copy.deepcopy(ordering)
        temp = proposedO[nodeI]
        proposedO[nodeI] = proposedO[nodeJ]
        proposedO[nodeJ] = temp
        #get violations score for proposed ordering
        proposedV = violationScore(G, proposedO)
        #decide which ordering to keep
        if proposedV < violation:
            violation = proposedV
            ordering = proposedO
            timeSinceSwap = 0
        elif proposedV == violation:
            ordering = proposedO
            timeSinceSwap += 1
        else:
            timeSinceSwap += 1
        #increment time steps
        vList.append(violation)
        t += 1
    return (t, vList)

#test on directed chain
numTests = 50
G = createDirectedChain(50)
timeStepsList = []
#create line plot
plt.figure()
plt.xlabel("Number␣of␣Time␣Steps")
plt.ylabel("Violation␣Score")
plt.title("Number␣of␣Violations␣Over␣Time")
for t in range(numTests):
    #run MVR on graph
    (timeSteps, violations) = MVR(G)
    #plot violations
    plt.plot(list(range(timeSteps)),violations)
    #add timesteps to list for histo
    timeStepsList.append(timeSteps)
plt.savefig("ChainList.png")
#create histogram of times
plt.figure()
plt.xlabel("Number␣of␣Time␣Steps")
plt.ylabel("Frequency␣of␣Time␣Steps")
plt.title("Histogram␣of␣Time␣Steps")
plt.hist(timeStepsList, bins=20)
plt.savefig("ChainHist.png")
```

```
#test on random graphs
numTests = 50
timeStepsList = []
#create line plot
plt.figure()
plt.xlabel("Number_of_Time_Steps")
plt.ylabel("Violation_Score")
plt.title("Number_of_Violations_Over_Time")
for t in range(numTests):
    #create graph
    G = createRandomGraph()
    #run MVR on graph
    (timeSteps, violations) = MVR(G)
    #plot violations
    plt.plot(list(range(timeSteps)),violations)
    #add timesteps to list for histo
    timeStepsList.append(timeSteps)
plt.savefig("RandomLine.png")
#create histogram of times
plt.figure()
plt.xlabel("Number_of_Time_Steps")
plt.ylabel("Frequency_of_Time_Steps")
plt.title("Histogram_of_Time_Steps")
plt.hist(timeStepsList, bins = 20)
plt.savefig("RandomHist.png")

#PROBLEM 1b
#create random directed network with given n and probability
def createRandomDiGraph(n=50, p=0.15):
    G = nx.DiGraph()
    G.add_nodes_from(list(range(n)))
    for i in range(n):
        for j in range(n):
            if not i == j and random.random() < p:
                G.add_edge(i,j)
    return G

#ABSOLUTE VIOLATIONS
#test on random graphs
numTests = 50
minVio = []
maxVio = []
aveVio = []
probs = np.arange(0,1.01,0.1)
n = np.arange(10,80,5)
for nval in n:
    minS = []
    maxS = []
    aveS = []
    for p in probs:
        maxV = []
        minV = []
        aveV = []
        print(str(nval)+", "+str(p))
        for t in range(numTests):
            #create graph
            G = createRandomDiGraph(nval,p)
            #run MVR on graph
            (timeSteps, violations) = MVR(G)
            #add timesteps to list for histo
            aveV.extend(violations)
            maxV.append(violations[0])
            minV.append(violations[timeSteps-1])
        minS.append(mean(minV))
        maxS.append(mean(maxV))
        aveS.append(mean(aveV))
    minVio.append(minS)
    maxVio.append(maxS)
    aveVio.append(aveS)

#plot minimum
X, Y = np.meshgrid(probs,n)
fig = plt.figure(figsize=(10,7))
ax = plt.axes(projection='3d')
ax.plot_wireframe(X, Y, np.asarray(minSteps), color='black')
ax.set_title("Remaining Violations")
ax.set_xlabel("Probability of Connection")
ax.set_ylabel("Network Size")
ax.set_zlabel("Min Violations")
plt.savefig("min.png")

#plot maximum
fig = plt.figure(figsize=(10,7))
ax = plt.axes(projection='3d')
ax.plot_wireframe(X, Y, np.asarray(maxSteps), color='black')
ax.set_title("Starting Violations")
ax.set_xlabel("Probability of Connection")
ax.set_ylabel("Network Size")
ax.set_zlabel("Max Violations")
plt.savefig("max.png")

#FRACTION VIOLATIONS
#test on random graphs
numTests = 50
minVio = []
maxVio = []
aveVio = []
probs = np.arange(0,1.01,0.1)
n = np.arange(10,80,5)
for nval in n:
    minS = []
    maxS = []
    aveS = []
    for p in probs:
        maxV = []
        minV = []
        aveV = []
        print(str(nval)+", "+str(p))
        for t in range(numTests):
            #create graph
            G = createRandomDiGraph(nval,p)
            #run MVR on graph
            (timeSteps, violations) = MVR(G)
            violations[:] = [x / (len(G.edges)+1) for x in violations]
            #add timesteps to list for histo
            aveV.extend(violations)
            maxV.append(violations[0])
            minV.append(violations[timeSteps-1])
        minS.append(mean(minV))
        maxS.append(mean(maxV))
        aveS.append(mean(aveV))
    minVio.append(minS)
    maxVio.append(maxS)
    aveVio.append(aveS)

#plot minimum
X, Y = np.meshgrid(probs,n)
fig = plt.figure(figsize=(10,7))
ax = plt.axes(projection='3d')
ax.plot_wireframe(X, Y, np.asarray(minSteps), color='black')
ax.set_title("Remaining Fraction of Edges Violated")
ax.set_xlabel("Probability of Connection")
ax.set_ylabel("Network Size")
ax.set_zlabel("Fraction of Edges Violated")
plt.savefig("minNorm.png")

#plot maximum
fig = plt.figure(figsize=(10,7))
ax = plt.axes(projection='3d')
ax.plot_wireframe(X, Y, np.asarray(maxSteps), color='black')
ax.set_title("Starting Fraction of Edges Violated")
ax.set_xlabel("Probability of Connection")
ax.set_ylabel("Network Size")
ax.set_zlabel("Fraction of Edges Violated")
plt.savefig("maxNorm.png")
```