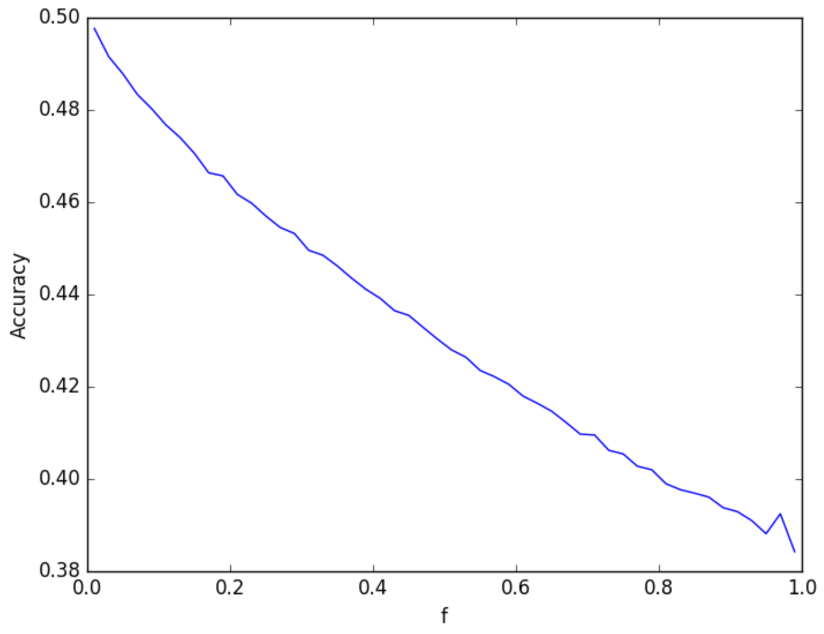
Donovan Guelde
CSCI 5352
PS 5

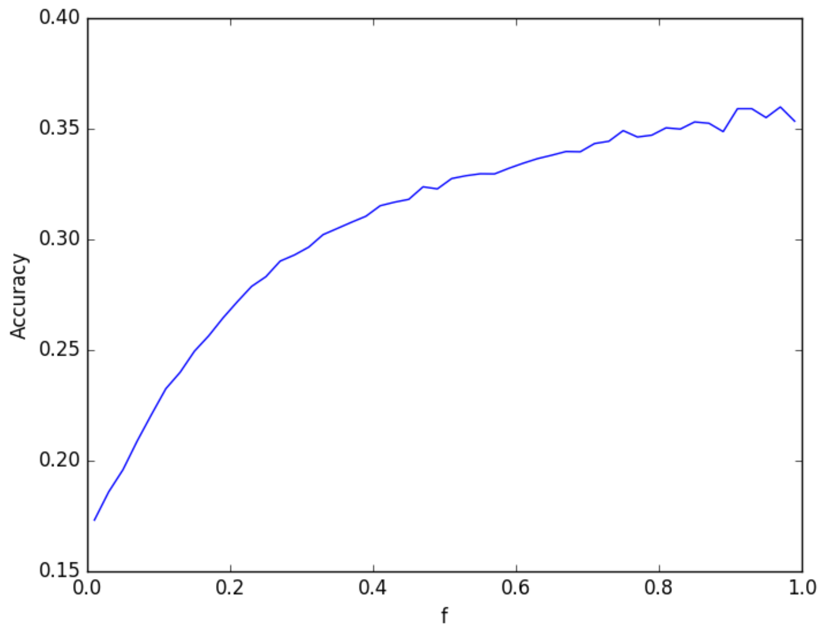
1. a. To investigate the "guilt by association" (GbA) heuristic, the experiment as described in Problem Set 5, question 1.a. was performed on the "Norwegian Board of Directors (2002-2011, projection)", network net1m_2011-08-01, and the "Malaria var DBLa HVR networks", network HVR_5. To achieve consistent results, 1000 iterations were performed and results averaged for each f value tested, f being the fraction of vertex nodes that were visible. The range of f values was [0.01,0.99], and f was increased in increments of 0.02, giving 50 data points. Results for network net1m_2011-08-01 are as follows:

Accuracy as a Function of f for GbA Heuristic on net1m_2011-08-01

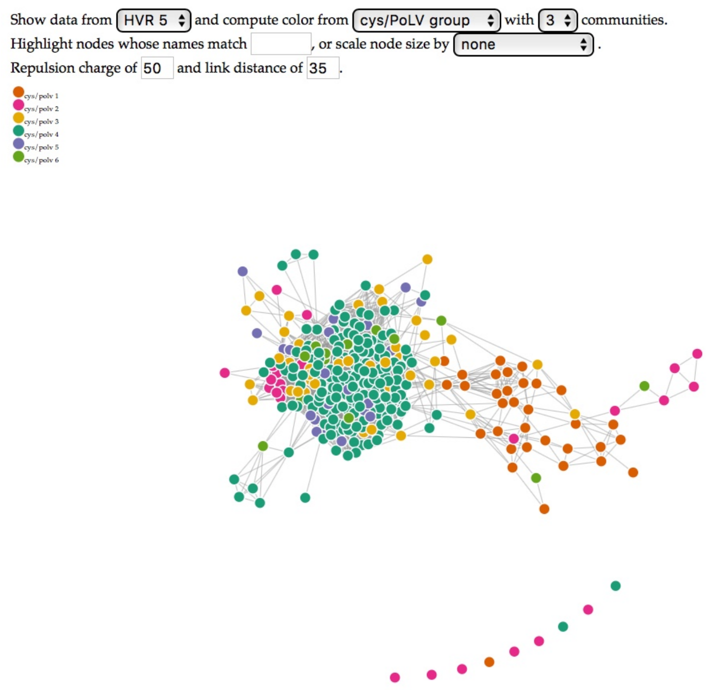


At f=0.01, accuracy is very nearly 0.5, which is expected. This network has vertices labeled by gender, and a simple random choice is expected to produce an accuracy of 0.5 in a binary classification. As f increases, the accuracy of the GbA heuristic decreases, to a minimum of 0.384. This may seem surprising, considering that board membership is male-dominated in the United States, with only 17.9% of board seats filled by women(https:en.wikipedia.org/wiki/Gender_representation_on_corporate_boards_of_directors). However, it is important to view these results while taking recent history into account, particularly the enactment in Norway of a gender representation law requiring at least 40% representation of each gender in corporate boards (https://toreopsahl.com/2010/09/30/article-for-the-few-not-the-many-the-effects-of-affirmative-action-on-presence-prominence-and-social-capital-of-women-directors-in-norway/). Such heavy mixing of genders drastically lowers the accuracy of the GbA heuristic. Corporate boards dominated by males, as in the US, but neither are they dominated by women, either (as per the Norwegian law). The results shown by the above graph show that nodes in the net1m_2011-08-01 network do not exhibit homophily.

Accuracy as a Function of f for GbA Heuristic on network HVR_5



The results of GbA on network HVR_5 show similarly expected results at an f value near 0.0. This network uses six unique node labels, and a random assignment of labels can be expected to be correct $\frac{1}{6}$ (0.166667) of the time. As f increases, accuracy also increases, up to 0.353. While this is still a relatively low accuracy, it is more than double that of random assignment. This indicates that vertices in the HVR_5 network do show some degree of homophily. This conclusion appears to be supported by the following visualization of the HVR_5 network(http://danlarremore.com/var/), linked to by the paper associated with the network file (D. B. Larremore, A. Clauset, and C. O. Buckee, "A network approach to analyzing highly recombinant malaria parasite genes." PLOS Computational Biology 9(10), e1003268 (2013).)

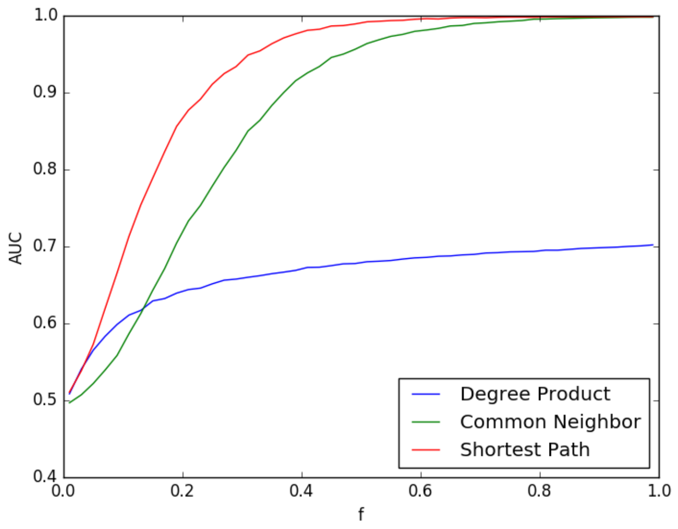

Show data from HVR 5 and compute color from cys/PoLV group with 3 communities.
Highlight nodes whose names match , or scale node size by none.
Repulsion charge of 50 and link distance of 35.

From this visualization, it appears that the homophily in this network may be largely driven by the 'cyc/polv. 4' nodes. Nodes with this label appear to be the most numerous and have a high degree of homophily.

One interesting aspect of the results from both networks is the decrease in smoothness of the plots as f nears 1.0. This can possibly be explained by the drastically reduced size of the test set. As f approaches 1.0, the size of the test set approaches 0 (a fact that I had to account for in coding for this question, as a test set of size 0 would cause my code to crash). With such a small test set size, variance of results increases, even with 1000 repetitions per value of f.

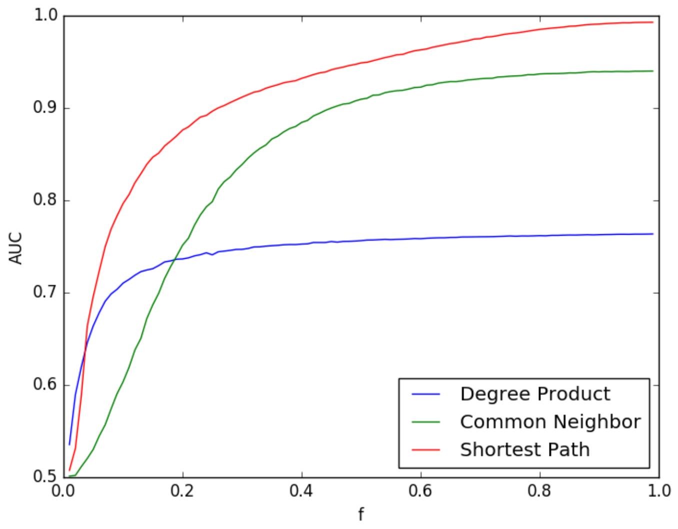
1. b. Utilizing the three scoring functions described in Problem Set 5, question 1.b., accuracy of edge prediction was investigated on the networks net1m_2011-08-01 and HVR_5. Accuracy, as determined by AUC, was measured over 10 iterations per f value, where f=[0.01,0.99], increased in increments of 0.02. The scikit learn roc_auc_score() function was used to calculate AUC value.

Accuracy (AUC) as a Function of f for Edge Prediction
via Three Scoring Methods on net1m_2011-08-01



The accuracy of the common neighbor and shortest path scoring systems, as determined by UAC, increases rapidly for the net1m_2011-08-01 network. This network is a projection of a bipartite graph, and as such, vertices(board members) who sit on the same board (vertices that are linked to the same node in the bipartite network) form a clique in the simple network. Therefore, common neighbor and shortest path scores provide an excellent link predictor. All vertices on the same board (same bipartite network pairing) have a shortest path distance of 1(2 if the linking edge has been hidden), and share all of the same common neighbors (common neighbor score of 1 by our scoring methodology). By similar reasoning, degree product should also be a good predictor, since board members in the same clique likely have a common degree. The predictive power of the degree product score is lessened by the fact that individuals may sit on multiple corporate boards, lessening the correlation of board membership and degree. Different boards may also have the same number of members, giving possibly unconnected cliques the same degree product scores.

Accuracy (AUC) as a Function of f for Edge Prediction
via Three Scoring Methods on HVR_5 Network



The HVR_5 network shows similar edge prediction results as the net1m_2011-08-01 network. As we saw in question 1.a., the HVR_5 network shows strong homophily among like vertices. This homophily is analogous to the cliques formed in net1m_2011-08-01, making common neighbor and shortest path scores good edge predictors, while degree product is a weaker predictor. Due to the apparent absence of strong cliques that are present in net1m_2011-08-01, prediction accuracy of shortest path and common neighbor are not as strong as they are in network net1m_2011-08-01.

2. To answer question 2, the Yeast transcription network (2002) and the US airport networks (2010) (complete US airport network in 2010) were used. To get accurate results on average cascade size, 2,000 iterations were performed for each node as 'Patient Zero'.

US airport Network results:

Node	Airport	Average Cascade Size	Degree
114	Atlanta International	9.5275	314
709	Dulles International	8.968	299
1200	Chicago O'Hare	8.9295	296
766	John F. Kennedy International	8.824	291
877	Los Angeles International	8.754	292
389	Denver International	8.4115	274
500	Newark Liberty International	8.3655	273
1068	Minneapolis Saint Paul International	8.2405	269
711	George Bush Intercontinental (Houston)	8.209	267
1016	Miami International	8.0385	261

Cascade size for the US airport network reflects the relative importance of airports in the US. The airport network is made up of a number of large 'hubs' that connect to other hubs, as well as smaller, regional airports. Therefore, these large hub airports are more likely to be the starting point for the largest cascades. Due to the low probability of transmission of $\frac{1}{c} = \frac{1}{37.061}$, only the largest airports can be expected to spread to even one or two neighboring nodes. The probability of a small regional airport infecting even one neighbor is approaching zero. Only the largest airports, linking to other very large hubs, are able to start a cascade that will reliably spread. A small airport may be able to occasionally spread a cascade to a large hub, but not reliably, over multiple iterations of the simulation. The degree of the airport nodes correlates very well with cascade size and ranking, with the lone exception being node 766, JFK International, that has degree one less than the next node. This can easily be explained by JFK having fewer connections to smaller airports, which do not help to spread the cascade.

Yeast Transcription Network results

Node	Average Cascade Size	degree
556	12.387	71
209	9.528	53
578	8.0215	44
617	7.1135	38
625	7.092	38
332	6.715	36
360	6.663	35
361	6.132	32
221	5.661	29
355	5.22	26

Average cascade size for the Yeast Transcription Network was more varied than the transportation network, with larger maximum values, and lower values appearing in the top ten results. The maximum values are due, at least in part, to the higher probability of neighbor infection of $\frac{1}{6.267}$, compared to the US airport network. The smaller average cascade sizes may be due to the disconnectedness of the yeast transcription network. Using igraph's clusters() function, the yeast transcription network was found to have 11 components. Most vertices were members of the same giant component, so this would not explain the large discrepancy. The varying cascade sizes correlate very well with the degree of the vertices, as shown in the table above. Compared to the airport network, the yeast network has a wider range of degree in the top ten, giving a wider range of cascade sizes.

Code for 1.a.

```
# Donovan Guelde
# CSCI 5352 PS 5

import numpy as np
import random
import matplotlib.pyplot as plt

INPUTSET= 0 #0-toy data set, 1-boardmember set, 2-malaria
numberTrials=10000
fStep=0.02

def readFile(fileName,mData):
        with (open(mData,'r')) as f:#get metadata
                if (INPUTSET!=2):
                        f.next() #skip label row of boardmember,toy set data
                maxNode=0 #number of nodes in network
                for line in f:
                        maxNode+=1
                f.seek(0,0)
                metadata = np.zeros((maxNode))
                if (INPUTSET!=2):
                        f.next()
                counter=0
                for line in f:
                        if (INPUTSET!=2):
                                node,gender=line.split()[0],line.split()[-1:][0]
                                node=int(node)-1 #index from 0, not 1
                                gender=int(gender)
                                metadata[node]=gender
                        else:
                                metadata[counter]=line
                                counter+=1
        f.close()
        metadata = metadata.astype(int)
        # build an n x n simple network.  Uses edge weights to signify class of neighbor node
        # ex.  A(i,j) = 2, A(j,i) = 1--> i and j are linked, j is class 2, i is class 1
        with (open(fileName,'r')) as f:
                lines = f.readlines()
                matrix = np.zeros((maxNode,maxNode))
                for line in lines:
                        if (INPUTSET!=2):
                                node,neighbor = map(int,line.split())
                        else:
                                node,neighbor = map(int,line.split(','))
                        node-=1 #start at [0], not [1]
                        neighbor-=1
                        matrix[node][neighbor]=metadata[neighbor]
                        matrix[neighbor][node]=metadata[node] # undirected
        f.close()
        temp = np.where(np.sum(matrix,axis=1)==0) #delete vertices with no neighbor info (different year, data set, etc.)
        matrix=np.delete(matrix,temp,axis=0)
        matrix=np.delete(matrix,temp,axis=1)
        metadata=np.delete(metadata,temp)
        return matrix,metadata

def main():

        if(INPUTSET==0):
                networkFile='toyNetwork.txt'
                metadataFile='toyMetadata.txt'
        if(INPUTSET==1):
                networkFile="net1m_2011-08-01.txt"
                metadataFile="data_people.txt"
        if(INPUTSET==2):
                networkFile='HVR_5.txt'
                metadataFile='metadata_CysPoLV.txt'
        associationMatrix,metadata=readFile(networkFile,metadataFile)
        length = len(metadata)
        numberCategories=metadata.max()-metadata.min()+1
        possibleChoices=np.arange(1,numberCategories+1)
        f=.01
        fCounter=0
        resultsOverF=np.zeros(((0.99-f)/fStep)+1) #store accuracy results for each f value
        fValues=np.zeros(((0.99-f)/fStep)+1) # store f values used for replot, if necessary
        while (f < 1.):
                iterationResults=np.zeros((numberTrials)) #results on each iteration
```

```
                        iterationCounter=0
                        for iteration in xrange(numberTrials):
                                trainMatrix=np.copy(associationMatrix) #make a copy so we can alter it w/out losing oiginal
                                randomLabels=np.random.randint(1,high=numberCategories+1,size=length)
                                randomValues = np.random.random(length) #matrix of 'coin flips' to compare against f for our test se
                                hiddenNodes=np.where(randomValues>f)
                                while (len(hiddenNodes[0])==0): #test set length 0 makes no sense...try again
                                        randomValues = np.random.random(length)
                                        hiddenNodes=np.where(randomValues>f) #we hide the label on these nodes
                                predictions=np.zeros(len(hiddenNodes[0])) #make predictions for nodes w/ hidden labels
                                trainMatrix[:,hiddenNodes]=0 #set A(i,j) to 0 when j is hidden (can still see A(j,i) to make predict
                                findMajority=np.zeros((len(hiddenNodes[0]),numberCategories)) #store 'votes' for each vertex in sep
                                for index in range(0,numberCategories):
                                        findMajority[:,index]=((trainMatrix==index+1).sum(1))[hiddenNodes] #neighbor vote total for
                                predictions=np.zeros(len(hiddenNodes[0])) #store predictions
                                predictions[np.where(findMajority[:,0]==findMajority[:,1])]=randomLabels[np.where(findMajority[:,0]
                                #print findMajority,'\n',predictions
                                predictions[np.where(predictions==0)]=(np.argmax(findMajority[np.where(predictions==0)],axis=1))+1
                                correct=float(np.sum(predictions==metadata[hiddenNodes]))
                                iterationResults[iterationCounter]=correct/len(hiddenNodes[0])
                                iterationCounter+=1

                        resultsOverF[fCounter]=np.average(iterationResults) #average accuracy of iterations over 1 f value
                        fValues[fCounter]=f
                        f+=fStep
                        fCounter+=1
                plt.plot(fValues,resultsOverF)
                plt.xlabel('f')
                plt.ylabel('Accuracy')
                #plt.savefig('./{}{}Iterations.png'.format(networkFile[:-4],numberTrials))
                plt.show()
                #np.savetxt('./{}{}accuracy.txt'.format(networkFile[:-4],numberTrials),resultsOverF)
                #np.savetxt('./{}{}fValues.txt'.format(networkFile[:-4],numberTrials),fValues)
main()
```

Code for 1.b.

```
# Donovan Guelde
# CSCI 5352 PS 5

import numpy as np
import matplotlib.pyplot as plt
import igraph
from sklearn.metrics import roc_auc_score as auc
import time

INPUTSET=0 #0-toy data set, 1-boardmember set, 2-malaria
numberTrials=10
fStep=0.02

#use igraph to quickly(relatively) calculate common neighbor score
#uses association matrix to make igraph instance,
#precalculates adjacency lists to find common neighbor score
#http://stackoverflow.com/questions/28352211/eficient-common-neighbors-and-preferential-attachment-using-igraph
class GraphCalculations(object):
        def __init__(self,graph):
                self.graph=graph
                self.adjlist=map(set,graph.get_adjlist())
```

```
    def common_neighbors(self,i,j):
        return np.divide(float(len(self.adjlist[i].intersection(self.adjlist[j]))),len(self.adjlist[i].union(self.a

##########################################################################################
# Some helper functions to speed things up using triangular matrices rather than full n x n #
##########################################################################################

#input upper triangle array, returns symmetric array
def makeSymmetricFromTriangle(array):
    array = np.add(array,array.T) - np.diag(array.diagonal())
    return array

#returns symmetric array based on average of A[i,j] and A[j,i]
def makeSymmetric(array):
    array=(array+array.T)/2
    return array

#takes in an array, returns upper triangle as a list
def getTriangleMatrixAsList(array):
    arrayList = array[np.triu_indices_from(array)].tolist()
    return arrayList

def readFile(fileName,mData):
    with (open(mData,'r')) as f:#get metadata
        if (INPUTSET!=2):
            f.next() #skip label row of boardmember,toy set data
        maxNode=0 #number of nodes in network
        for line in f:
            maxNode+=1
        f.seek(0,0)
        metadata = np.zeros((maxNode))
        if (INPUTSET!=2):
            f.next()
        counter=0
        for line in f:
            if (INPUTSET!=2):
                node,gender=line.split()[0],line.split()[-1:][0]
                node=int(node)-1 #index from 0, not 1
                gender=int(gender)
                metadata[node]=gender
            else:
                metadata[counter]=int(line)
                counter+=1
    f.close()

    metadata = metadata.astype(int)
    # build an n x n simple network.
    with (open(fileName,'r')) as f:
        lines = f.readlines()
        matrix = np.zeros((maxNode,maxNode))
        for line in lines:
            if (INPUTSET!=2):
                node,neighbor = map(int,line.split())
            else:
                node,neighbor = map(int,line.split(','))
            node-=1 #start at [0], not [1]
            neighbor-=1
            matrix[node][neighbor]=1
```

```python
                        matrix[neighbor][node]=1 # undirected
        f.close()
        #matrix = matrix.astype(int)
        temp = np.where(np.sum(matrix,axis=1)==0) #delete vertices with no neighbor info (different year, data set, etc.)
        matrix=np.delete(matrix,temp,axis=0)
        matrix=np.delete(matrix,temp,axis=1)
        metadata=np.delete(metadata,temp)

        #matrix=np.ascontiguousarray(matrix)
        metadata=np.ascontiguousarray(metadata)
        return matrix,metadata

def main():
        np.set_printoptions(linewidth=140)
        if(INPUTSET==0):
                networkFile='toyNetwork.txt'
                metadataFile='toyMetadata.txt'
        if(INPUTSET==1):
                networkFile="net1m_2011-08-01.txt"
                metadataFile="data_people.txt"
        if(INPUTSET==2):
                networkFile='HVR_5.txt'
                metadataFile='metadata_CysPoLV.txt'
        matrix,metadata=readFile(networkFile,metadataFile)
        length = len(metadata)
        shape=length,length
        numberCategories=metadata.max()-metadata.min()+1
        f=.01
        fCounter=0
        degreeProductAccuracyOverF=np.zeros(((.99-f)/fStep)+1) #store accuracy results for each f value
        commonNeighborAccuracyOverF=np.zeros(((.99-f)/fStep)+1)
        shortestPathAccuracyOverF=np.zeros(((.99-f)/fStep)+1)
        fValues=np.zeros(((.99-f)/fStep)+1) # store f values used for replot, if necessary
        trueLabels=getTriangleMatrixAsList(matrix) #true edge set in list format

        while (f <= 1.0):
                start = time.time()
                degreeProductIterationResults=np.zeros((numberTrials)) #results on each iteration
                commonNeighborIterationResults=np.zeros((numberTrials))
                shortestPathIterationResults=np.zeros((numberTrials))
                iterationCounter=0
                #start = time.time()
                commonNeighbors=np.empty((length,length))
                for iteration in xrange(numberTrials):
                        #determine holdout, generate tie-breaking noise, hide edges
                        associationMatrix=np.copy(matrix) #copy original network
                        randomValues = np.random.random((length,length)) #matrix of 'coin flips' to compare against f for o
                        hiddenEdges=np.where(randomValues>f) #flip coin
                        hiddenEdgeList=associationMatrix[hiddenEdges]
                        associationMatrix[hiddenEdges]=0 #hide edges
                        associationMatrix=makeSymmetricFromTriangle(associationMatrix) #use upper triange (after coin flips
                        randomNoise=np.divide(np.random.random((length,length)),length)

                        #generate degree product scores
                        degreeList=np.sum(associationMatrix,axis=1)
                        degreeProduct=np.add(np.outer(degreeList,degreeList),randomNoise) #degree product matrix with noise
                        degreeProductScores=getTriangleMatrixAsList(degreeProduct)

                        #generate normalized common neighbor score
                        g = igraph.Graph.Adjacency((associationMatrix.astype(bool)).tolist())
                        neighborStruct=GraphCalculations(g)
                        for index in range(0,length): #make upper triangle matrix via loops
                                for index2 in range(index,length):
                                        commonNeighbors[index][index2]=neighborStruct.common_neighbors(index,index2)
                        commonNeighbors=np.nan_to_num(commonNeighbors)
                        commonNeighbors=np.add(commonNeighbors,randomNoise)
                        commonNeighborScores=getTriangleMatrixAsList(commonNeighbors)

                        #generate shortest path score
                        shortestPath=np.asarray(g.shortest_paths_dijkstra())
                        shortestPath=np.add(shortestPath,randomNoise)
                        shortestPath=np.reciprocal(shortestPath) #nodes with no path will have pathlength = (1/noise) -> ve
                        shortestPathScores=getTriangleMatrixAsList(shortestPath)

                        #get/store AUC scores for iteration
                        degreeProductResults=auc(trueLabels,degreeProductScores)
                        commonNeighborResults=auc(trueLabels,commonNeighborScores)
                        shortestPathResults=auc(trueLabels,shortestPathScores)
                        degreeProductIterationResults[iterationCounter]=degreeProductResults
                        commonNeighborIterationResults[iterationCounter]=commonNeighborResults
                        shortestPathIterationResults[iterationCounter]=shortestPathResults
                        iterationCounter+=1

                #accuracy results for each f value
                degreeProductAccuracyOverF[fCounter]=np.average(degreeProductIterationResults)
                commonNeighborAccuracyOverF[fCounter]=np.average(commonNeighborIterationResults)
                shortestPathAccuracyOverF[fCounter]=np.average(shortestPathIterationResults)
                fValues[fCounter]=f
                print f,time.time()-start
                f+=fStep
                fCounter+=1



        plt.plot(fValues,degreeProductAccuracyOverF)
        plt.plot(fValues,commonNeighborAccuracyOverF)
        plt.plot(fValues,shortestPathAccuracyOverF)
        plt.legend(['Degree Product','Common Neighbor','Shortest Path'],loc=4)
        plt.xlabel('f')
        plt.ylabel('AUC')
        #plt.savefig('./predictEdges{}{}Iterations.png'.format(networkFile[:-4],numberTrials))
        plt.show()

        #np.savetxt('./degreeProduct{}{}accuracy.txt'.format(networkFile[:-4],numberTrials),degreeProductAccuracyOverF)
        #np.savetxt('./commonNeighbors{}{}accuracy.txt'.format(networkFile[:-4],numberTrials),commonNeighborAccuracyOverF)
        #np.savetxt('./shortestPath{}{}accuracy.txt'.format(networkFile[:-4],numberTrials),shortestPathAccuracyOverF)
        #np.savetxt('./predictEdges{}{}fValues.txt'.format(networkFile[:-4],numberTrials),fValues)


main()
```

Code for 2.

```
#Donovan Guelde
#csci 5352 PS5 extra credit
#Yeast transcription network (2002)
```

```
#        http://www.weizmann.ac.il/mcb/UriAlon/download/collection-complex-networks for yeast network,
#US airport networks (2010)
#        http://opsahl.co.uk/tnet/datasets/USairport_2010.txt

import numpy as np
import random
import time
import math
import matplotlib.pyplot as plt
import igraph

def readFile(fileName):
        with (open(fileName,'r'))as f:
                matrix=np.zeros((1858,1858)) #yeast network has n=688 nodes, airport network has 1574 (labels go up to 1858)
                for line in f:
                        line=line.split()
                        node,neighbor=int(line[0])-1,int(line[1])-1 #minus 1 to begin at 0
                        matrix[node][neighbor]=1
                        matrix[neighbor][node]=1#undirected, unweighted
        f.close()
        return matrix

class GraphHelper(object): #some easy, fast igraph help...not really helpful in this program
        def __init__(self,graph):
                self.graph=graph
                self.adjlist=map(set,graph.get_adjlist())

def main():
        #FILENAME="yeastInter_st.txt"
        FILENAME="USairport_2010.txt"
        ITERATIONSPERNODE=2000 #iterations on each node
        matrix = readFile(FILENAME) #numpy matrix
        networkSize=len(matrix)
        g = igraph.Graph.Adjacency((matrix>0).tolist())
        c = igraph.mean(g.degree())
        p = 1./c #transmission probability

        epidemicSize=np.zeros(networkSize) #average cascade size per node
        cascadeSize=np.zeros(ITERATIONSPERNODE) #cascade size per run on patient Zero node
        possibleNewInfections=[] #neighbors of contageous nodes
        newInfections=[] #newly infected nodes at a single time t
        for patientZero in xrange(networkSize): #everybody gets a turn...
                print patientZero
                for iteration in xrange(ITERATIONSPERNODE):
                        start = time.time()
                        immunity=np.random.rand(networkSize) #immunity chance for nodes
                        condition=np.zeros(networkSize) #0=susceptible, 1=contageous, 2=infected but not contageous
                        condition[patientZero]=1
                        newInfection=True
                        while(newInfection):
                                newInfection=False
                                diseaseSpreaders=np.where(condition==1)
                                condition[condition==1]=2 #not contageous any more
                                try: #will throw error if no neighbors (if patient zero has no edges...)
                                        exposed=[neighbors[spreader] for spreader in diseaseSpreaders][0]
                                except TypeError:
                                        continue
                                exposed=np.intersect1d(exposed,np.where(condition==0)) #remove non-susceptible from list
                if(len(exposed)==0): continue #if no susceptible, finished
                newInfections=np.intersect1d(exposed,exposed[np.where(immunity[np.array(exposed)]<p)]) #cas
                condition[newInfections]=1 #contageous
                if newInfections.sum()>0:
                    newInfection=True
            cascadeSize[iteration]=len(np.where(condition!=0)[0]) #if contageous or infected, you count as sick
        epidemicSize[patientZero]=np.average(cascadeSize)
    outputFile=FILENAME[:-4]+"_undirected_{}_iterations_results.txt".format(ITERATIONSPERNODE)
    with (open(outputFile,'w'))as f:
        for index in range(0,networkSize):
            winner=np.argmax(epidemicSize)
            f.write('{} {}\n'.format(winner+1,epidemicSize[winner]))
            epidemicSize[winner]=0
    f.close()
        #np.savetxt('./q1c/E{}length.txt'.format(E),epidemicLength)
        #np.savetxt('./q1c/E{}size.txt'.format(E),epidemicSize)
        #plotResults(epidemicSize,epidemicLength,pValues,N,E,C)
        #E+=ESTEP #end of e-loop

main()
```