Contents

1 Introduction

In today's tech-connected world, information moves at a very fast pace. However, this hasn't been the case throughout history. The dynamics of information transfer have changed over time, primarily as a result of three factors: the information's inherent nature, the number of channels for information transfer and the network structures of people who receive information. We consider these factors in building a model to measure what affects the evolution and and influencing-ability of information in society's networks.

In our model, we quantify three variables to calculate the probability that a person transfers information to someone else after he/she first receives it. These three variables are, as mentioned above, the information's inherent nature (is it, say, scandalous or humorous?), the number of channels for distribution, and the extent to which an individual's personal network is aware of the information. We use discrete values for the first two variables, and dynamically calculate the third, to construct a simple probability function. Using the probability function and a sample network from Facebook data, we analyze the number of iterations required to transmit the information over the complete network.

2 Assumptions and Clarifications

1. We found no algorithm that randomly constructs a social network comparable to real ones today[1]. To avoid this problem, and to make our model more realistic, we use a sample social network data available from Facebook.
2. We assume that the society is divided into two kinds of people, the Informed and the Uninformed. The Informed are those who have received some information, and the Uninformed are those who are yet to receive it.
3. A person is more likely to receive information if a greater proportion of his contacts are aware of the information. In the same way, a person is more likely to reveal this information to an uninformed friend if most of his/her friends are already informed.
4. The probability that the information is transmitted ahead depends only on three factors: the number of channel of transmission, the sensitivity(how critical a piece of information is) of the information and the number of people around a person around who already have it.

3 Model

We create a graph using data available from Facebook that depicts a sample social network. This data is a list of almost $55,000$ edges that comprise a total of 1034 nodes in a network. These edges represent friendships, and each node represents a person.

As part of navigating the graph, we create a function that gives the probability of some message being transferred between adjacent nodes of the graph. This function relates three variables concerning the information and the individuals of our sample network to the number of iterations needed for the information's complete dissemination. These three variables are: message's sensitivity (S), the strength of information channels at a particular time in history (C), and a person's hit-count (H). The first two inputs are varied in discrete multiples of 0.1 from 0.1 to 1. That is, a very sensitive message (for example, an official?s assassination) might carry a value of 1 while an Internet joke might have a sensitivity of 0.1. Similarly, a message sent during the 1850s might have a C value of 0.2 while a message sent today might have a C value of 0.9, as a reflection of the difference in communication speeds and means between both time periods. Ranges of S and C are associated with particular types of messages and time periods. We define those ranges as follows:

	C	S
0.1	1870s	Trivial Remark
0.2	1920s	...
0.3	1970s	Negativity
0.4	...	...
0.5	1990s	Scandalous Rumor
0.6	...	...
0.7	2010s	Good News
0.8	...	...
0.9	2050	Catastrophe
1.0	...	...

Table 1: Values of S and C over the range 0.1 to 1

All possible combinations of these two inputs therefore make up a $[10 \times 10]$ matrix:

$$\begin{bmatrix} (0.1, 0.1) & (0.1, 0.2) & \dots & (0.1, 1) \\ (0.2, 0.1) & \dots & \dots & \dots \\ \vdots & \dots & \dots & \dots \\ (1, 0.1) & \dots & \dots & (1, 1) \end{bmatrix}$$

H is formally defined as:

$$\frac{k'}{k}$$

where k' represents the number of neighbors who are informed of some message and k represents the degree of a node(i.e. the number of friends of that person). k'? changes throughout execution of the program, so it is a dynamic variable. On the other hand the other two variables, C and S, are static and controllable.

Three variables determine the probability, $\mathbb{P}$, that a message will be transmitted between two nodes. Specifically

$$\mathbb{P} = \frac{S + C + H}{3}$$

This calculated value is the probability that a message will be transmitted ahead or not at a particular node. Also, it is important to note that the value of $\mathbb{P}$ varies with each node, as a result of H.

Having developed a probability function, we then develop a directed graph traversal program that simulates a run through the sample facebook network. By 'run', we mean a simulation of the process of a message disseminating throughout the entire network. This graph traversal program accepts a pair value of (S, C) and returns the number of iterations needed so that all nodes are informed about the message.

Pseudocode of the Basic Visiting Mechanism:

```
stack = newStack()
stack add firstNode

while stack not empty do
    secondStack = newStack()
    aNode = popStack( )
    for all Unvisited and Unisolated Neighbors of aNode do
        if Neighbor Transmitted Message then
            secondStack add all neighbors of Neighbor
        else
            secondStack add Neighbor
        end if
    end for
    stack = secondStack
end while
```

For each of the hundred combinations of inputs, we run a simulation 100 times and average the iterations needed for a particular input. The resulting matrix is shown below. Any row or column can be taken for S and the other dimension for C (this matrix is close to symmetric because in the probability function $\mathbb{P}$, S and C can be freely interchanged. The only reason why the matrix is not completely symmetric but is close it because of the random function that is used to determine if the message is sent forward or not.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.1	33.88	27.58	23.24	20.71	18.20	16.60	15.85	14.64	13.72	12.67
0.2	30.81	22.70	20.08	18.41	17.10	15.85	14.01	13.84	12.45	11.94
0.3	24.77	19.90	18.50	17.25	16.04	14.10	13.60	12.74	11.90	11.11
0.4	20.94	18.50	16.24	15.04	14.17	13.65	12.31	12.07	11.48	10.38
0.5	17.60	17.44	16.04	13.91	13.05	12.54	12.42	11.51	10.87	10.30
0.6	17.42	15.94	13.92	13.15	12.27	11.82	11.40	11.14	10.17	10.12
0.7	15.21	14.55	13.58	12.15	12.35	11.25	10.75	09.71	09.87	09.75
0.8	14.37	13.38	12.35	12.11	11.37	11.01	09.97	09.95	09.38	09.10
0.9	13.32	12.38	11.94	11.47	10.62	10.50	09.80	09.64	09.15	09.01
1.0	12.31	12.18	11.40	10.75	10.41	10.08	09.95	09.22	08.94	08.44

Table 2: Average value of number of iterations needed to disseminate information

4 Result

We can use these results to answer most of the following questions:

1. Use your model to predict the communication networks' relationships and capacities around the year 2050.
2. Use the theories and concepts of information influence on networks to model how public interest and opinion can be changed through information networks in today's connected world.
3. Determine how information value, people's initial opinion and bias, form of the message or its source, and the topology or strength of the information network in a region, country, or worldwide could be used to spread information and influence public opinion.

1. The average iterations needed in 2050 for different channel degrees are shown below.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
2050	12.31	12.18	11.40	10.75	10.41	10.08	09.95	09.22	08.94	08.44

Table 3: Number of iterations projected for year 2050

Our results indicate that, in 2050, news of a trivial remark will disseminate through a network (on the order of the 1000 nodes we considered) in ≈ 12.31 iterations while news of a catastrophe will disseminate in ≈ 8.44 iterations. Now consider that in 1870, news of some trivial remark took ≈ 33.88 iterations to disseminate through a network on the scale of ours, but news of a catastrophe spread in 12.67 iterations. This reveals that, as time has progressed from the 1870s until today, news of the utmost importance has taken increasingly less and less time to spread. The theoretical speed of the most important information has grown by ≈ 50%, assuming that social networks in the past were structured in the same way that our sample facebook dataset is. On the other hand, as time has progressed, the time required for news of the least importance to spread has significantly dropped. The theoretical speed of society's least important information has grown by ≈ 200%. As an aside, our model confirms the impression that, because all types of information now travel at similar rates, a wide range of information competes for our attention today in a way unlike before.

2. A useful way to interpret our iteration data is to superimpose graphs of $I = f(S)$ where I represents the number of iterations needed for different sensitivity values, S. These graphs are each held at a constant C, or channel value. The resulting graph is below:

   The graph shows that, for a connected network similar to our sample Facebook one, once either sensitivity S or channel number C reaches their limit (i.e take on values of 1), changing the other input produces negligible change in the number of iterations

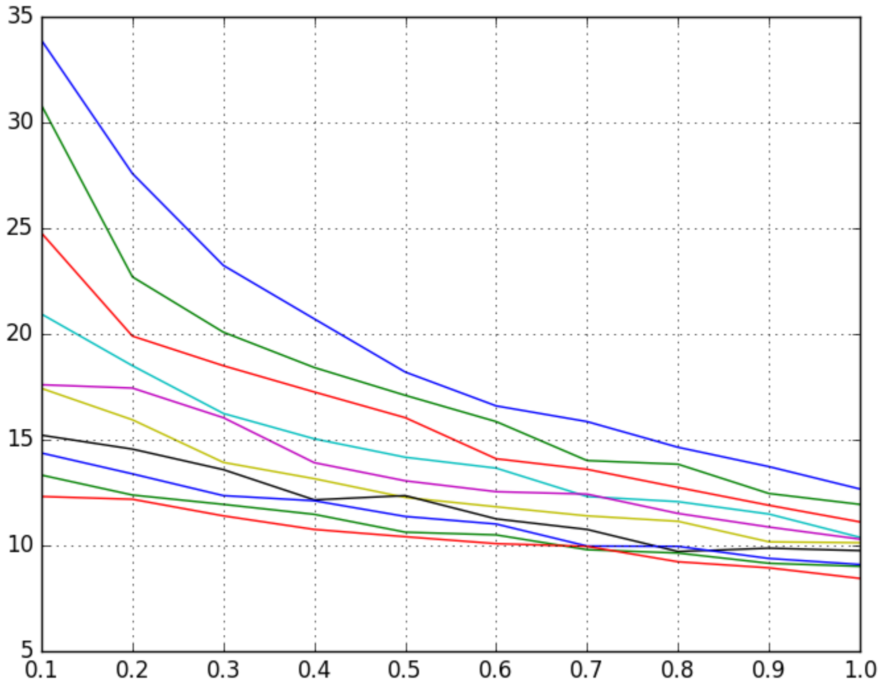

Figure 1: Descents of different sensitivities given constant channel values

needed for a network traversal. Remember that, because C and S were interchangeable in our probability function, the x-axis can be understood to be either C or S. By contrast, when either input is at its minimum, changing the other variable produces significant changes in iteration time. This is an important result for public policy-makers hoping to disseminate a message as quickly as possible. It implies that, instead of focusing efforts to both increase the perceived importance of some message as well as increase the number of communication channels, policy makers could focus their resources on one input. This would produce an intended effect to a similar degree and save resources. A related consequence of our model is that coordinates with a constant sum (i.e $S+C = k$, where k is some constant) have similar iteration numbers. This is a generalization of our first observation: if spending resources on one input is too costly, those resources can be redistributed to the less costly input without any decrease in impact.

3. As explained in #2, above, both information value (its "sensitivity") and channel number affect the number of iterations needed for a message's dissemination. Both also have an interesting relationship in that either input taken to an extreme disseminates information as well as some mixture of both variables taken to partial extents. Moreover it is observed that for a constant sum value of S and C, the average value of iterations needed comes out to be very similar. This means that policy implementers can make changes optimally in S and C values to result in lower iteration values.

Admittedly, our model fails to answer this report's first two questions, which concern how the speed of information transfer relates to its inherent value. We were in the process of forming a model that could discern the particulars of some information from its speed but came short of time to do so. These were our thoughts and work:

Solely examining data taken during the modern era (when channel values were greater than or equal to 0.7), we attempted to form probability functions that would relate the relative frequency of some message to its identity (whether the message was a rumor, good news or a catastrophe). We envisioned claiming, for example, that a message requiring 8 iterations would, on average, be $\frac{1}{10}$ of the time a rumor, $\frac{3}{10}$ a catastrophe and, at all other times, good news. We believe that we could have constructed such a function, because particular sensitivities ($S > 0.6$) exhibited a distribution iterations closely following the normal curve. These sensitivities showed this distribution over 1000 trials, with 250 trials for each value of $C, C > 0.6$. A normal curve is drawn for $S = 0.7$, histograms shown for the remaining sensitivities:

With more time, we would have then compared the theoretical function arising from this data with actual ones from a twitter dataset we were also in the process of using. By aggregating how much time it took for sub-networks of size 1000 to be examined in the macro-twitter network, we could have observed whether tweeting times also followed a normal distribution.

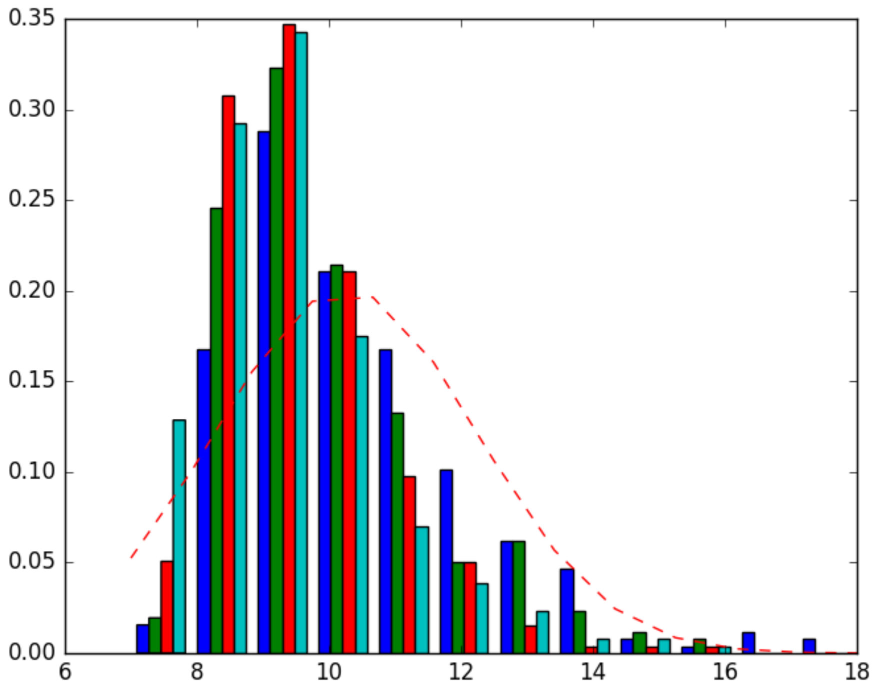

Figure 2: A normal distribution of trials with $S > 0.6$

5 Strengths and Weaknesses

5.1 Strengths

1. Our simulation uses a probability function that tries to model the random nature of information transfer.
2. The function considers how different combinations of a message's sensitivity and its ease of flow (the number of channels) affect the number of iterations needed for message dissemination.
3. It considers how the local structure of a network around any node (its ego network) influences the node's chance of receiving a message.

5.2 Weaknesses

1. We assume that social networks throughout time can be modeled using a network from today.
2. Two of the inputs in our probability function are static; no smaller inputs guide what values they take on in some other function.
3. Our model doesn't allow us to achieve some other objectives of this report:to discern a piece of information's identity from its speed of transfer, to understand the effects of network topology, and the effects of bias.

6 Future Work

1. Our simulation runs trials on one, unvarying network. This is obviously unfavorable as a model for networks throughout history, but it is also unfavorable for modern networks, because we have no concrete reason to believe that our network is representative of other networks. A future model should, consequently, consider how variations in sensitivity and channel count play out in different network structures. We therefore recommend that a future model either develop such a randomization scheme or employ existing, developing randomization programs[5].

2. Literature on Networking Theory borrows heavily from research on the networking of epidemics. For example, some Networking Theory researchers adopt the epidemiology convention to designate the participants of a network as either "Immunized", "Disseminated", or "Susceptible".[4] Their models depart from ours in that they also consider "Immunized" participants, which is the concept that, in a network, some participants behave as roadblocks. Disease or message, once they contract or receive the item, these participants will not transmit anything forward. Using a probability function, our model considers that some individuals may be less likely than others to transmit a message but, ultimately, everyone always transmits forward. The idea that some people choose not to transmit forward, by contrast, is more reasonable. We therefore recommend that future models seed some immunized participants throughout their graph. These participants act as roadblocks.

3. Geographical topology also factors in message dissemination, although a greater physical separation between two nodes only sometimes implies a greater iteration count between two nodes. Future models should negotiate this uncertainty by sometimes giving weight to geographic distance and sometimes ignoring it, at least when models today are concerned.

References

[1] Julian McAuleym, Jure Leskovec: Learning to Discover Social Circles in Ego Networks

[2] Easley David, Kleinberg Jon: Networks, Crowds, Markets: Reasoning About a Highly Connected World

[3] Fagiolo, Giorgio: Clustering in complex directed networks

[4] Hui Zhua, Cheng Huanga, Rongxing Lub, Hui Lia: Modelling information dissemination under privacy concerns in social media

[5] Newman, MEJ: Random Graphs as Models of Networks

[6] Jure Leskovec, Stanford Network Analysis Project, `https://snap.stanford.edu/data/egonets-Facebook.html`,Social Circles:Facebook

7 Appendix

Two programs are included. The first one is the simulator program, which contains three methods, the only relevant one of which is runMessage(). The second one constructs the edge data from Facebook [6] into an array and then runs the data through the simulation of the first program. It is from the main method of the second program, therefore, that the entire simulation is run. This code is also available on github @

https://github.com/yogabbagabb/PythonProjects under the directory MCM.

7.1 Python Code for Simulation

```
'''
Created on Jan 30, 2016

@author: ahanagrawal
'''

import numpy as np
import random

import networkx as nx

'''
[i][vertexNumber] = neighbors
[i][vertexNumber + 1] = hits
[i][vertexNumber + 2] = informed or uninformed (1 or 0)
'''
def createGraph(vertexNumber, timePeriod):
    N = [[0 for x in range(vertexNumber + 3)] for x in range(vertexNumber)]
    for i in range(vertexNumber):
        for j in range(vertexNumber):
            N[i][j] = 1 if (random.random() < timePeriod) else 0
        N[i][vertexNumber] = (np.sum(N[i][0:vertexNumber])   N[i][i])

    return N

'''
N is some square matrix

sensitivity and channels are inputs

neighborList is a list of adjacency matrix entries. It initially contains one node.
'''
def runMessage(N, sensitivity, channels, startingNode, neighborList):

#       if N[currentNode][len(N) + 2] == 0:
    trials = 0

    while (len(neighborList) != 0):
        trials += 1
        newList = []

        while (len(neighborList) != 0):

            neighborIndex = neighborList.pop()


            if (N[neighborIndex][len(N) + 2] == 0
                and N[neighborIndex][len(N)] != 0):

                probability = (sensitivity + channels
                + getHitProportion(N, neighborIndex))/3

                if (random.random() <= probability):
                    appendNeighbors(newList, neighborIndex, N)
                    N[neighborIndex][len(N) + 2] = 1

                else:
                    newList.append(neighborIndex)

        neighborList = newList


    return trials


def appendNeighbors(someList, index, N):
    for i in range(len(N)):
        if (N[index][i] == 1):
            someList.append(i)


def getHitProportion(N, vertex):
    hits = 0
    for i in range(len(N)):
        if (N[vertex][i] == 1 and N[i][len(N) + 2] == 1):
            hits += 1
        N[vertex][len(N) + 1] = hits

    neighbors = N[vertex][len(N)]


    return (hits/neighbors)
```

```
if __name__ == '__main__':
    array = createGraph(1035,0.25)
    someList = [0]

    print(runMessage(array, 0.25, 0.25, 0, someList))
#     a = np.asarray(array)
#     print(a)
#     a,b = np.hsplit(a, [len(a)])
#     g = nx.DiGraph(a)
#     g = nx.to_networkx_graph(a)
#     print(nx.average_clustering(g))
#     print(a)


'''
Created on Jan 30, 2016

@author: ahanagrawal
'''
from MCM.InfoSpread import runMessage
import numpy as np

def resetFacebookMatrix(N):
    for i in range(len(N)):
        N[i][len(N) + 1] = 0
        N[i][len(N) + 2] = 0


def constructMatrix(fileName):

    N = [[0 for x in range(1035)] for x in range(1038)]

    f = open(fileName)
    nodeDict = {}
    line = f.readline()

    while(line != ""):
        lineArray = line.split()
        firstIndex = nodeDict.get(lineArray[0])
        secondIndex = nodeDict.get(lineArray[1])
        if  firstIndex == None:
            firstIndex = index
            nodeDict[lineArray[0]] = firstIndex

        if  secondIndex == None:
            secondIndex = index
            nodeDict[lineArray[1]] = secondIndex

        first = min(firstIndex, secondIndex)
        second = firstIndex + secondIndex   first

        N[firstIndex][secondIndex] = 1
        N[second][first] = 1


        line = f.readline()

    for j in range(len(N)):
        N[j][len(N)] = np.sum(N[j][0:len(N)])

    return N, index

if __name__ == '__main__':

    N,index = constructMatrix("107.edges")
    someList = [0]
    print(runMessage(N, 1, 1, 0, someList))
    print(index)
#      print(len(N))
#      f = open("107.edges")
#
#      a = f.readline()
#      b = f.readline()
#      print(b)
#      print(a)
```