Recovering Information Starting Points

Rachael Maltiel and Jacqueline Corbitt

August 15, 2008

Abstract

In this paper, we describe several algorithms to estimate the best nodes to begin spread of information throughout the graph. The graph models a social network with the nodes representing people and edges representing communication channels. We deduce the best place to start the information by experimenting with random start points. We also analyze our methods for how much improvement in time it takes the information to spread throughout the graph, both compared to each other and to choosing a random node.

Contents

1 Introduction of the Problem

1.1 Basic Graphs

Our problem utilizes graphs to model the spread of information. We can think of nodes as people, and edges as relationships involving communication between these nodes. (Because of this easy and comprehensible analogy, this problem has been termed "the rumormonger problem.") We work on the inverse problem of determining the edges in the graph given certain output data.[1]

If information is given to node 0, we presume that after one timestep (referred to as a "day"), node 0 has passed the information to all of its connections. Any node that shares an edge with node 0 (i.e., is a neighbor of node 0) will receive the information on day 1. By day 2, all the nodes which received the information on day 1 have spread the information to all their connections, and so on until all connected nodes have received the information. In a graph which is connected, all the nodes will eventually receive the information, but the order of nodes will depend on which is the first node to receive information. In each run, a different starting node will give a different order. An example is shown in Figure 2 with the first run beginning on node 0 and the second run beginning on node 6. As can be seen, the information passes much more quickly when it is started with node 6.

1.2 Basic Definitions

Adjacency Matrix: Symmetric matrix M such that

$$M_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are incident} \\ 0 & \text{otherwise} \end{cases}$$

Distance: The smallest number of edges in a path between vertices i and j, denoted as $d(i, j)$.

[1]These graphs are generated randomly. A given probability determines the independent likelihood of each possible edge existing. For graphs of 10 nodes we generally use a probability of 0.35 and for graphs of 100 nodes we use a probability of 0.05.

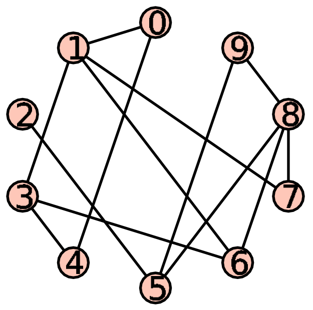


Figure 1: A typical 10-node graph

	Run One	Run Two
Day 0	0	6
Day 1	1, 4	1, 3, 8
Day 2	3, 6, 7	0, 4, 5, 7, 9
Day 3	8	2
Day 4	5, 9	-
Day 5	2	-

Figure 2: Sample Run Data

Distance Matrix: Symmetric matrix M such that

$$M_{ij} = \begin{cases} 0 & i = j \\ d(i,j) & \text{otherwise} \end{cases}$$

Degree: The number of edges incident to a given node.

Center: $u : max\{d(u,v)|$ for $v \in G\} \leq max\{d(u',v)|$ for $v \in G\}$ for all $u' \in G$
A node that has the shortest path length to the farthest node. In our problem, this is a node that takes the fewest number of days to spread the information to all other nodes. The solution to the forward problem of finding the center is straightforward.

Median: $u : min\{\sum_{v \epsilon G} d(u,v)\}$ for all $u \epsilon G$
A node which yields the smallest sum when the shortest paths to all other nodes are summed. In our problem, this is a node that takes the smallest average number of days to contact all nodes. The solution to the forward problem is to sum up the shortest connections from each node to all other nodes. A node with the smallest sum is a median.

1.3 Goals

Our goal is *to predict the best nodes with which to begin a run to have the information spread the fastest without viewing the actual graph.* "Best" can be defined by either the center(s), the median(s), or both.[2] In this paper, we discuss several algorithms and their effectiveness at achieving this goal. We will also discuss further directions this research can be taken.

2 Fractional Method

2.1 The Basic Method

One can deduce the presence or absence of some connections from just one run. For example, from Figure 2 it is easily deduced that node 0 is connected to nodes 1 and 4, since is the only way nodes 1 and 4 can have the information by day 1. (Henceforth, nodes shall be referred to only by their number, unless it causes confusion in context.) Node 8 is connected to 5 and 9. Node 2 is definitely not connected to 3, 6, or 7 since it takes more than a day for the information to pass between these nodes. However, some connections are still in question. Node 2 may be connected to only 5, only 9, or both. Our method represents these possible connections by modifying the adjacency matrix. Instead of recording a 1 or a 0 in the (2,5), (5,2), (2,9), or (9,2) positions, a fraction is calculated and recorded. In this case, the fraction recorded in all four of those positions would be 1/2. This does not correctly represent the accurate probability of the connection; instead it is meant to symbolize one edge spread between this many nodes (possibilities for more accurate probabilities will be discussed in Section 5). We calculate these fractions by taking the reciprocal of the number of possible nodes from which it could have received the information. That is to say, for a node a which received information on day x, it could have heard the information from any of n nodes which received the information on day $x-1$. Therefore, the probability for node a to be connected to any of the n nodes would be represented by $1/n$. In the case above, the 2 in the denominator of 1/2 indicates the number of nodes (node 5 and node 9) from which it is possible that node 2 got the information. Also, since no information can be deduced about the connection between 5 and 9 (or any nodes that get the information on the same day), the entries in (9,5) and (5,9) will be -1 as a placeholder. In other words:

1. Create a matrix `M` which is `m x m`, where `m` is the number of total nodes in the graph. Each entry is -1 (placeholder).
2. Put 0's on all diagonals (no node is connected directly to itself).
3. For all nodes a that received the information on Day x and for any node b that received the information on Day $x-1$ of which there are n, enter $1/n$ in position `M[a, b]`.

[2]Often, several nodes are both centers and medians.

	0	1	2	3	4	5	6	7	8	9	Row Sum
0	0	1	0	0	1	0	0	0	0	0	2
1	1	0	0	1/2	-1	0	1/2	1/2	0	0	2.5
2	0	0	0	0	0	1/2	0	0	0	1/2	1
3	0	1/2	0	0	1/2	0	-1	-1	1/3	0	1.33
4	1	-1	0	1/2	0	0	1/2	1/2	0	0	2.5
5	0	0	1/2	0	0	0	0	0	1	-1	1.5
6	0	1/2	0	-1	1/2	0	0	-1	1/3	0	1.33
7	0	1/2	0	-1	1/2	0	-1	0	1/3	0	1.33
8	0	0	0	1/3	0	1	1/3	1/3	0	1	3
9	0	0	1/2	0	0	-1	0	0	1	0	1.5

(a) Data from only Run One

	0	1	2	3	4	5	6	7	8	9	Row Sum
0	0	1	0	0	1	0	0	0	0	0	2
1	1	0	0	1/2	1/3	0	1	1/2	0	0	3.33
2	0	0	0	0	0	1/2	0	0	0	1/2	1
3	0	1/2	0	0	1/2	0	1	1/3	1/3	0	2.67
4	1	1/3	0	1/2	0	0	0	1/2	0	0	1.33
5	0	0	1/2	0	0	0	0	0	1	-1	1.5
6	0	1	0	1	0	0	0	0	1	0	3
7	0	1/2	0	1/3	1/2	0	0	0	1/3	0	1.67
8	0	0	0	1/3	0	1	1	1/3	0	1	3.67
9	0	0	1/2	0	0	-1	0	0	1	0	1.5

(b) Data from Run One and Run Two

Figure 3: Fractional Matrices using data from Figure 2

4. For all other matrix entries (all other connections) which are still -1, check to see if connection is impossible. The entry is impossible if the days on which the information is received have a difference of greater than 1. If the nodes of the entry receive information on the same day, leave connection as -1. Else, change entry to 0.

Given only the data from run one, our matrix appears in Figure 3(a)[3].

Because this graph does not have many connections, we already know much of the information about the graph's actual connections. However, with another run, even more data can be obtained. We can use the information from the second run to modify our matrix.

1. Iterate through our original matrix and look for entries which are not either 1 or 0, as these represent connections or lack of connections that we know with certainty already.

[3]The last column represents the row sum, treating -1's as 0's because nothing is known about them. The reason for row sums will be discussed in Section 2.3.

	Run One	Run Two
Day 0	0	4
Day 1	2, 4	0, 2
Day 2	1	1
Day 3	3	3

(a) Five Node Sample Data

	0	1	2	3	4
0	0	0	1	0	1
1	0	0	1/2	1	0
2	1	1/2	0	0	1
3	0	1	0	0	0
4	1	0	1	0	0

(b) Fractional Matrix

Figure 4: Sample Data with more information than Fractional Method can catch

2. If a fraction or a place-holding -1 is found, look at the data for that connection for run two.

3. If there is no information about that connection in run two (i.e. both nodes received the information on the same day), make no changes.

4. If the connection is shown impossible, replace that connection in the matrix with a 0.

5. If the connection is shown certain (i.e. there is only one node the information could have come from), replace that connection in the matrix with a 1.

6. If the connection is neither impossible nor certain, calculate what the fractional probability would be. If it's a larger fraction than the original entry, or the original entry was -1, replace the original entry with the new fraction. Else, leave the original entry.

7. Find the next entry neither 1 nor 0 and continue.

Using this idea, the modified matrix becomes the matrix in Figure 3(b). As you can see, this second run made a significant difference to the matrix. More connections were discovered or proved impossible, and there is now only one connection for which no information is given.

The information obtained through this algorithm lacks some information that we can gather from the run data, however. Let's take a look at another example, this time with only five nodes in Figure 4(a). We'll just look at the runs, not the graph.

According to the first run, 2 and 4 are connected to 0, but 1 is not. According to the second run, 0 and 2 are connected to 4, but 1 is not. It is simple to deduce that 1 must be connected to 2, but our previous algorithm would not show this. After having been given these two runs, that algorithm would give us the matrix in Figure 4(b).

Thus we can see that in some cases our algorithm cannot see as much as the human eye. In this case, it is possible to actually discover the whole graph with these two runs, but we must be clever with our algorithm.

	0	1	2	3	4
0	0	0	1	0	1
1	0	0	1/2	1	1/2
2	1	1/2	0	0	-1
3	0	1	0	0	0
4	1	1/2	-1	0	0

Figure 5: `M1`

Node	0	1	2	3	4
Day	0	2	1	3	1

Figure 6: `dayinfo1`

2.2 An Improvement

In order to recover more information and solve the above problem, we describe an improved method.

1. Write a matrix as seen in Figure 5, using previous method and entering only the data from the first run (`M1`).
2. Create an array, Figure 6, with a length the same as the number of nodes n. The nth index's entry is the day that node received the information (i.e. for run one of the previous example, `dayinfo1`[0] = 0, `dayinfo1`[1] = 2, `dayinfo1`[2]=1, etc).
3. Write a second matrix as seen in Figure 7 using the previous method and entering only the data from the first run (`M2`).
4. Create an array (`dayinfo2`) using the same strategy as `dayinfo1`, but using the data from run two. See Figure 8.
5. Start to put this information into a final matrix (`Mfinal`). Iterate through `M1` and `M2` and record all 1's and 0's (all certain and impossible connections). Leave other entries as -1. See Figure 9.

	0	1	2	3	4
0	0	1/2	-1	0	1
1	1/2	0	1/2	1	0
2	-1	1/2	0	0	1
3	0	1	0	0	0
4	1	0	1	0	0

Figure 7: `M2`

Node	0	1	2	3	4
Day	1	2	1	3	0

Figure 8: `dayinfo2`

	0	1	2	3	4
0	0	0	1	0	1
1	0	0	-1	1	0
2	1	-1	0	0	1
3	0	1	0	0	0
4	1	0	1	0	0

Figure 9: The first draft of `Mfinal`

6. Write a new matrix (`M1iffy`) according to this procedure: Begin with all entries in `M1iffy` as [0]. For each entry in `M1`, record which entries are fractional. Then check the same connection in `Mfinal`. If the same connection in `Mfinal` is a 1, then the entry at that position in `M1iffy` becomes a 0. If the same connection in `Mfinal` is a 0, then mark the connection as a 2 in `M1iffy`. If the same connection in `Mfinal` is -1, put a 1 in `M1iffy`. (All these numbers are meant simply as symbolic placeholders.) See Figure 10.

7. Write a new matrix (`M2iffy`) according to the above procedure, but iterating through `M2`. There are now two new matrices recording all the unsure connections and all possible connections which have been shown to not be connections by the other run of data. See Figure 11.

8. In each `iffy` matrix, look for each 2. This represents a connection that the first run showed was possible but that the second run proved nonexistent. For the two nodes for which a connection has been disproven, the dayinfo of one node is x and the dayinfo of the other node is $x-1$. Let's call the first node a and the second node b. Thus, we can update the probabilities of a connection between node a with the other nodes with dayinfo $x-1$.

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	1	0	2
2	0	1	0	0	0
3	0	0	0	0	0
4	0	2	0	0	0

Figure 10: `M1iffy`

	0	1	2	3	4
0	0	2	0	0	0
1	2	0	1	0	0
2	0	1	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

Figure 11: `M2iffy`

	0	1	2	3	4
0	0	0	1	0	1
1	0	0	1	1	0
2	1	1	0	0	1
3	0	1	0	0	0
4	1	0	1	0	0

Figure 12: The final version of `Mfinal`

n is the number of nodes with dayinfo $x - 1$. The original entry for all said connections between node a and all nodes with dayinfo $x - 1$ is $1/n$. We can now update this fraction to be $1/(n - 1)$. This, for example, can change an entry from 1/3 to 1/2, or from 1/2 to 1. If the number of nodes with dayinfo $x - 1$ except for node b is 1, that means that we have proven a definite connection between node a and the remaining node with dayinfo $x - 1$. We now set this connection to 0 in the iffy matrix and record a 1 in the entries in `Mfinal`. [4] Then we continue looking through the iffy matrix until the next 2 is found.

9. At this point, all the certain and disproven connections - all 1's and 0's - are recorded in `Mfinal`. Finally, we iterate through both `M1` and `M2` one last time, comparing all connections which are still listed as `-1` in `Mfinal`. For each such connection in `M1` and `M2`, we take the larger fraction and record that in Mfinal. If there is no information about a connection in either `M1` or `M2`, the connection is left as `-1` in `Mfinal`.

This example may be trivial but it demonstrates the algorithm clearly.

In effect, this algorithm can either increase the fraction to accurately take into account how many nodes a given node may be connected to or change a fraction to 1 when a connection can be determined given both sets of data. The effectiveness of this improvement in the algorithm is debatable. In graphs with 10 or fewer nodes, this could make a significant difference in our knowledge about connections. However, for a 100-node graph, several extra matrices are

[4]Remember that since our graph is symmetric, connections will be updated in both the (i, j) and the (j, i) entries.

created which may not make important modifications. This could slow down the run time significantly while only making small changes. The importance of this discussion is made more evident in the next section.

2.3 Analyzing the Method: Row Sums

In a real adjacency matrix, each row sum would indicate the degree of the node (the index of the row being the node's number). Although the matrix created through this algorithm is not a real adjacency matrix, it is an approximation in that row sums can still give valuable information. In a real adjacency matrix, the row with the highest sum would indicate the node with most connections, and in our pseudo-adjacency matrix, the row sum often is very close to the actual row sum of the real adjacency matrix (if the graph was given). During the calculation of row sums, each -1 is treated as a 0, since an edge's nonexistence is more likely than its existence.

In this algorithm, we are looking for the highest row sum. This indicates the nodes which are more likely to have a high degree, and thus the nodes which are more likely to be better start points. In the matrix in Figure 3(b), 8 is indicated as the best start point, with nodes 6 and 1 following close behind. If we look at the actual graph and calculate the centers and medians, we discover that nodes 6, 7, and 8 are centers, while nodes 6 and 8 are medians. Thus our algorithm chose the medians as the best start points, which also happen to be centers. Since the medians are also centers, this test does not demonstrate the accuracy of our method in determining centers; judging from this one test, it seems that our algorithm has chosen the medians. A statistical analysis to determine how accurate our methods are in both determining centers and medians will be given in Section 4.

At first glance, it would seem that the previous modification to our algorithm would have a significant effect on the row sum, as it has the potential to increase fractional entries to 1. However, often our algorithm merely changes a 1/3 to a 1/2 (not a significant difference), or, if no changes are necessary, leaves the row sum unchanged. For the time being, since the random graphs we have experimented with rarely have more than 100 nodes, making a few extra matrices does not unreasonably affect the run time of the operation. This addition to the algorithm will only become an issue in graphs with hundreds of nodes.

3 Cost Method

3.1 The Basic Method

In Fractional Method, we use an estimate of which nodes are connected to each other and each node's estimated degree to determine the best starting nodes. In Cost Method, we estimate how far apart every node is from every other node — or at least find an upper bound on the minimum distance between nodes. Both of these are important in finding the median of a graph. From Figure 2, for example, we can conclude that 0 has a distance of one from 1 and 4, 0 has a distance of two from 3, 6, and 7, etc.[5] However, just looking at the first set of data, we don't know if 1 and 4 are connected. We do know that (0,1) and (0,4) are edges so we know that the upper bound on the minimum distance between 1 and 4 is two (a path can go from 1 to 0 to 4 to get between nodes 1 and 4). By the same logic, 1 and 4 have a distance of at most three from nodes 3, 6, and 7, etc. An important deduction is that we can add the respective days which the nodes receive information on to find the maximum distance between the two nodes.

1. Put 0's on the diagonals as there are no connections between a node and itself.

2. For each connection (0, a), where $0 < a < n$, the number of total nodes, add the number of day node 0 learned the information with the number of day node a learned the information on (i.e. for this data, node 0 received information on day 0 and node 1 received information on day 1. $0 + 1 = 1$, so the entry (0,1) is set to 1). Only look at run one.

3. Repeat for (x, a), where $0 < x < n$ and $x < a < n$.

4. To keep the graph symmetric, for every calculated (i, j) connection, the (j, i) connection is set to the same.

5. Repeat the process, but this time examining run two. If a calculated connection is less than the entry already in the matrix, then the new, smaller entry replaces the old entry. If not, the old entry remains. Remember to keep the graph symmetric; if an (i, j) entry is changed, the corresponding (j, i) entry should be changed as well.

We can put these bounds of the shortest path lengths between two nodes in a modified Distance Matrix. The matrix generated using this algorithm appears in Figure 13(a) (the last column is the row sum for that row) using the data from run one in Figure 2.

Now, we calculate distances between nodes the same way we did before, but this time using data from run two. Then, for each connection, we compare the lengths and keep the smallest in our matrix. The updated and modified matrix, using data from both runs, appears in Figure 13(b).

The second run made a huge improvement on the first matrix. Note that the only row sum it did not affect was the row sum of node 0. This is because run one began on node 0, allowing this run to provide the information for the actual minimum distances to each node. It will always be the case that the start nodes for the two runs will have their actual minimum distances to each node. All the distances between all other nodes are only estimated. We call this "starting point bias." Because of this effect, our method sometimes recognizes

[5]Note that the distance from each node to the beginning node is the same as the number of days from each node to the beginning node; or in other words, it takes one day to travel a distance of one.

	0	1	2	3	4	5	6	7	8	9	Row Sum
0	0	1	5	2	1	4	2	2	3	4	24
1	1	0	6	3	2	5	3	3	4	5	32
2	5	6	0	7	6	9	7	7	8	9	64
3	2	3	7	0	3	6	4	4	5	6	40
4	1	2	6	3	0	5	3	3	4	5	32
5	4	5	9	6	5	0	6	6	7	8	56
6	2	3	7	4	3	6	0	4	5	6	40
7	2	3	7	4	3	6	4	0	5	6	40
8	3	4	8	5	4	7	5	5	0	7	48
9	4	5	9	6	5	8	6	6	7	0	56

(a) Data from only Run One

	0	1	2	3	4	5	6	7	8	9	Row Sum
0	0	1	5	2	1	4	2	2	3	4	24
1	1	0	4	2	2	3	1	3	2	3	21
2	5	4	0	4	5	5	3	5	4	5	40
3	2	2	4	0	3	3	1	3	2	3	23
4	1	2	5	3	0	4	2	3	3	4	27
5	4	3	5	3	4	0	2	4	3	4	32
6	2	1	3	1	2	2	0	2	1	2	16
7	2	3	5	3	3	4	2	0	3	4	29
8	3	2	4	2	3	3	1	3	0	3	24
9	4	3	5	3	4	4	2	4	3	0	32

(b) Data from Run One and Run Two

Figure 13: Cost Matrices using data from Figure 2

	0	1	2	3	4	5	6	7	8	9	Row Sum
0	0	1	5	2	1	4	2	2	3	4	24
1	1	0	4	2	2	3	1	3	2	3	21
2	5	4	0	4	5	3	3	5	2	3	34
3	2	2	4	0	3	3	1	3	2	3	23
4	1	2	5	3	0	4	2	3	3	4	27
5	4	3	3	3	4	0	2	4	1	2	26
6	2	1	3	1	2	2	0	2	1	2	16
7	2	3	5	3	3	4	2	0	3	4	29
8	3	2	2	2	3	1	1	3	0	1	18
9	4	3	3	3	4	2	2	4	1	0	26

Figure 14: Improved Cost Matrix

the start nodes as a good starting point even when this is not the case. With a large enough graph, we can throw out the start nodes in our search for centers and medians. However, in a small graph like this one, the chance of one or both of the start nodes being a center or median is high, so throwing these nodes out would harm our results.

Since we want the node(s) with the smallest sum of shortest path lengths between itself and the other nodes, we want the node that gives the smallest row sum (since each column along the row represents a bound of a shortest path length). According to our second matrix, node 6 is the best choice, with nodes 1, 3, 8, and 0 also recommended. As previously noted, the centers of this graph are nodes 6, 7, and 8, while the medians are 6 and 8. It's useful to note that node 6 did well; it is also interesting to note that nodes 0 and 8 were tied in their row sums. While it is apparent even just from looking at the data that 0 is a poor choice for a start node, 0 was a start point for run one and so its row sum is its actual sum of shortest path lengths. Node 8, however, must loop back through the starting node, increasing its estimated shortest path lengths. Also, node 6 was another starting point. While by the definition of median 6 should have the same row sum as node 8, it actually did far better because 6 was a starting point.

Another problem with this method is that in run one, we assumed we had to travel all the way back to the start node every time when we were counting distances. However, the interconnections between nodes 2, 5, 8, and 9 could be calculated more exactly, since the only node on day 3 is 8, as seen in Figure 2. Thus, instead of counting all the steps from 5 to 0 and from 0 to 8 (7 steps), 5 can just be connected to 8 (1 step). This is a vast improvement in the bound of the shortest path length. At the time of this writing, this improvement has not been written into the code yet. This can be done easily by hand, however, and combined with data from run two as in Figure 14.

Glancing over the row sums, we can see that this matrix gives us a far more accurate idea of the best nodes to choose; node 6 again comes in first, but it is closely followed by node 8. In sparse graphs such as this one, we can see that this modification to the algorithm would make a significant difference in the accuracy of our method. In more dense graphs, where the chance of only one node being contacted on one day is very small, this change in code would not help at all.

3.2 A Different Interpretation

Instead of comparing all row sums in the final matrix, we can take a shortcut by only looking at row sums of rows which have the smallest numbers. In this example, for instance, we wouldn't look at any row sum with a 5. We can do this because it is unlikely that any good start nodes would have such a high upper bound on their minimum distance to any node. If we look at the modified matrix in Figure 14, we realize some rows have neither 4s nor 5s, so we can throw out those rows. This leaves us with rows 6 and 8, the best nodes in the matrix! Even looking at Figure 13(b), which isn't as accurate as Figure 14, we can ignore rows with entries of 5, leaving us to compare the nodes 1, 3, 6, and 8. When comparing large graphs (particularly when doing row sums by hand), this method makes comparisons easier.[6]

4 Statistical Comparison of Fractional Method and Cost Method

4.1 Methodology

To determine how accurate our algorithms are in determining the best starting points in a network, we constructed confidence intervals comparing the effectiveness on average of choosing the best nodes from Fractional Method or Cost Method and choosing a random node in the graph. We wanted to evaluate how much more effective both our methods were than simply choosing a random node and we also wanted to compare the methods to each other. We evaluated all methods on their ability to predict the center or median. To compare how close an algorithm is to the center, we calculated the number of days it took the actual center to reach all nodes and compared this to the number of days it took the nodes our algorithm predicted to reach all nodes:

1. Find the actual center of the graph. Calculate the number of days it takes to reach all nodes.
2. For each node our algorithm chooses, calculate the number of days it takes to reach all nodes.
3. Find the difference between these.

[6]It's not always accurate, but is often a good indicator of which nodes are ideal.

4. Repeat for i iterations and then calculate the mean of all of these differences.

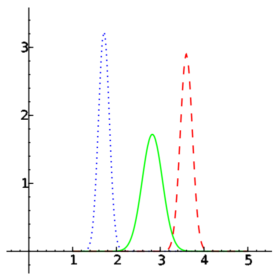
We use a similar method to compare how close each algorithm is to the median:

1. Find the actual median of the graph. Find the sum of the shortest paths to each node.
2. For each node our algorithm chooses, find the sum of the shortest paths to each node.
3. Calculate the difference between these two sums.
4. Repeat for i iterations and then calculate the mean of all of these differences.

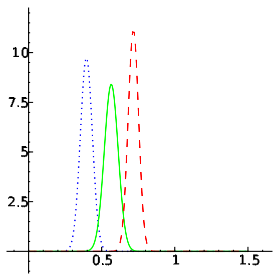
From these differences, we computed the mean and standard deviation and compared the normal plots.

4.2 Results

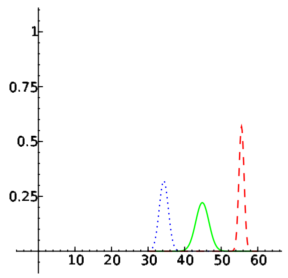
The graphs in Figure 15 show plots comparing the difference between centers and medians. We did this for both 10 and 100 node graphs for both algorithms and a random node, running 100 simulations in each case. Fractional Method is represented by the blue dotted line, Cost Method is represented by the green solid line, and choosing a node randomly is represented by the red dashed line. For centers for 10 node graphs, the mean of the difference from the center (calculated as described previously) is .395 days for Fractional Method with a standard deviation of .4101, .565 days for Cost Method with a standard deviation of .4749, and .7164 days with a standard deviation of .3583 for choosing a node randomly. For medians for 10 node graphs, the mean of the difference from the median (calculated as described previously) is 1.715 for Fractional Method with a standard deviation of 1.2355, 2.82 for Cost Method with a standard deviation of 2.3177, and 3.5967 with a standard deviation of 1.3751 for choosing a node randomly. For centers for 100 node graphs, the mean of the difference from the center is .9667 for Fractional Method with a standard deviation of .5061, 1.1667 days for Cost Method with a standard deviation of .3824, and 1.4156 with a standard deviation of .3749 for choosing a node randomly. For medians for 100 node graphs, the mean of the difference from the median is 34.2967 for Fractional Method with a standard deviation of 12.436, 44.84 for Cost Method with a standard deviation of 18.0246, and 55.5538 with a standard deviation of 7.0242 for choosing a node randomly. We can say with 99% confidence that the mean of Fractional Method is less than the mean of Cost Method, while we can say with 95% confidence that the mean of Cost Method is less than the mean from choosing a node randomly. The mean of Fractional Method is also a significant improvement on the mean when using a random node. We can summarize these results by saying that while Cost Method generally provides



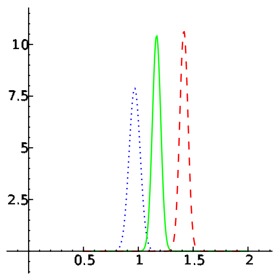
(a) Comparison of Difference of Average Distance (Difference of Medians) for 10 Node Graphs



(b) Comparison of Difference of Average Number of Days (Difference of Centers) for 10 Node Graphs



(c) Comparison of Difference of Average Distance (Difference of Medians) for 100 Node Graphs



(d) Comparison of Difference of Average Number of Days (Difference of Centers) for 100 Node Graphs

Figure 15: True Mean Comparisons

a significant improvement over choosing a random node, Fractional Method's improvement is larger.

5 Probability Method

5.1 Introduction

While our first methods provide an estimate about the center and median nodes of the graph based on the known data, they do not account for the probability of our estimate being correct. In a graph, there is a certain probability that a given edge is in the graph dependent upon how many edges the graph already contains. Our project utilizes Erdős-Rényi Random Graphs. The graphs take the input of a number of nodes and a probability of an edge existing between any two nodes, and the graph is generated by using the probabilities to determine which edges exist. When we look at the inverse problem, there is a conditional probability to find if a certain additional edge exists in the graph when we already know that certain edges definitely exist in the graph since the edges are not independent. The Probability Method takes this conditional probability into account.

In the Probability Method, we find all the possible graphs given our run data. From this data, there are certain connections known either to exist or not exist while other connections remain as possibilities. By grouping the edges into Yes, No, and Maybe categories, the possible graphs can be formed by inserting edges from all the possible combinations of Maybe edges. We can find the median of each possible graph. Then by summing the probability of each graph with a given median, we can find the probability of that node being the median.

5.2 Building the Algorithm

5.2.1 Sorting the `Yes`, `No`, and `Maybe` Edges

The first step of sorting the `Yes`, `No`, and `Maybe` edges (which will be referred to as the `Yes`'s, `No`'s, and `Maybe`'s) proved to be more difficult than originally anticipated. As discussed in the Fractional Method, while it is easy to find obvious connections or lack of connections, some connections can be determined through two runs of data in a manner that is easy for us to determine but hard for the computer.

A clear way to display the information from two runs is in a grid, as seen in Figure 16, using data from Figure 2. Each node is placed in the grid according to when each learned the information. Any node that learned the information on the Day 0 in run one is placed on the top row, the Day 1 on the second row, etc. The day it learned the information in run two determines the node's column placement with nodes starting with the information being placed in the farthest left column, nodes receiving the information on the second day being placed in the next column, etc. For example, since node 4 learned the information on Day 1 in run one and Day 2 in run two, 4 is placed in the second row from the top and the third column from the left.

Based off this placement, there are allowable manners in which the information can flow. Since each node must learn the information from a node that occurs on the day before in both runs, lines can only cross at most one horizontal line and at most one vertical line. If there is only one possible line in either an upward or leftward direction from a given node, then this line can be drawn in as a `Yes` (represented with a solid line). This is because that node must get the information both from above and from the left, representing the two separate runs. If there is more than one node either above or left from which the data could have come, then a dotted line is drawn in representing a `Maybe`. If two nodes occur in the same box, as with 5 and 9, not enough information is known. This edge is a possible edge so a dotted line representing a `Maybe` can be drawn in. Any nodes that are too far away from either run (so not one cell apart in either a upward or sideways direction) can be classified as `No`. If the edge is too far in either direction, then the information could not have passed between these two nodes in at least one run, indicating that the edge does not exist.

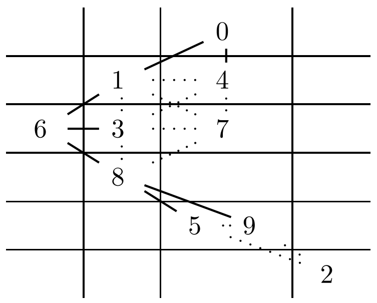


Figure 16: Visualization for sorting Yes's and Maybe's

Creating an algorithm for the computer proved difficult.

1. The method iterates through all the possible node connections and looks at the `dayinfo` of each node compared to each other node (the `dayinfo` is an array with the day that the node first appears in a given run, as in Fractional Method). If x and y are nodes, then if `dayinfo[`x`]` and `dayinfo[`y`]` have a difference of more than 1 on either run, then the edge must be a `No`. This means that on one of the runs, the nodes were more then two days apart in learning the information meaning that there cannot be a connection. If there was a connection, the first node to learn the information would pass it along to the other node on the next day, giving the nodes a difference in their `dayinfo` of 1. So if the `dayinfo` difference is more than 1, the two nodes (indicating an edge) are put in the `No` category but if it is less than or equal to 1 then the two nodes are put in the `Maybe` category.

2. After this, the `Maybe`'s must be sorted to remove the `Yes` edges. An edge has a definite connection if there is only one node from which a given node could have received the information. To determine when this occurs, we created a list that stored each node's location in the grid. Using this, the program jumps between two rows or columns. Let's say we are looking for possible sources of information for node x. First we jump between rows and the program counts the number of nodes that occur in the previous row from x's row that are within one column to the left or right of x's column. If there is only one node located in this position, then the connection must exist as this the only source of information for x. This edge is then moved to the `Yes`'s and removed from the `Maybe`'s. Then, we look at the column before x's column, again only looking for nodes one above or one below x's row. After this process, the possible edges have been sorted into `Yes`, `No`, and `Maybe` edges as can be determined from the run data.

3. Within the category of `Maybe`'s, there are two different types of possible edges. Some edges are unnecessary to get the specific run data. Some

`Maybe` edges, however, must be included in the graph to be able to get the run data that we have. For example, in the visualization in Figure 16, currently node 4 has no way of learning the information in run two. At least one of the possible edges that 4 has from the left, (1,4) and (3,4), must exist in the graph. These edges will all be `Maybe` edges but we would need at least one of them for the run data to occur. We created another category, known as `Maybe_Necessary` edges. By checking if a node has a connection from the prior day, this list groups the potential edges from where the information could have come. When building a possible graph, at least one edge from each of these groups must be part of the graph. The edges that are not necessary for the data to occur as it does, but still could be part of the graph, remain in the `Maybe` category (for example, edge (3,8)).

At this point, we have deduced all the information that is possible regarding the sorting of the edges from our data. Any edges that we know cannot exist are in the `No` category. Any edge that we know is a connection is in the `Yes` category. All that remains are possible connections. These are in the `Maybe` and `Maybe_Necessary` category, depending on if they are needed for the information to pass as it passes in the data. Any combinations of these edges can exist as long as at least one edge from each `Maybe_Necessary` group is taken.

5.2.2 Finding the Medians' Probabilities

Once all edges are sorted, we can create all possible graphs and determine their medians. We must be sure to include at least one edge from every vertex that does not yet have a connection explaining how it learned the information when it did. These edges come from the `Maybe_Necessary` category. Extra edges from the `Maybe`'s make different possible graphs as well. We can also determine the probability of each possible graph being the actual graph given the run data.

We use Bayes' law to compute the conditional probabilities of each graph. The weight of a possible graph is

$$w(g) = p^n \cdot (1-p)^{M-n}$$

where p is the probability[7] of an edge existing in the graph, n is the number of `Maybe` edges included in this possible graph, and M is the total number of `Maybe` edges (so $M - n$ is the number or `Maybe` edges not included in the possible graph g). As these weights do not sum to 1[8], they are weighted by dividing by the sum of all the weights. So,

$$W = \sum_g w(g)$$

[7]Although in a true inverse problem, this p would not be known, it can be estimated. The algorithm is effective even with an estimated p. A method for estimating this p is outside the realm of this paper and could be done through evaluating the approximate density of the network.

[8]The weights do not sum to 1 because some combinations of `Maybe` edges are not possible graphs, namely graphs that do not include all the necessary `Maybe_Necessary` edges

Node	Actual Probability	Estimated Probability
0	0	0
1	0	0
2	0	0
3	0.1856	0
4	0	0
5	0	0
6	0.5649	0.8644
7	0.0583	0
8	0.5245	0.2832
9	0	0

Figure 17: Probabilities of node being the median

The probability of a given graph g then becomes

$$P(G = g) = \frac{w(g)}{W}$$

Now, by summing the probabilities of each possible graph where a given median occurs, the probability of a node being the median is determined. The probability of of node x being the median is the sum of the probabilities of all graphs g where x is the median of g. Figure 17 depicts the actual and estimated probabilities[9] using the data from Figure 2. Estimated probabilities only look at some of the possible graphs, as explained in the next section. As can be seen, nodes 6 and 8 have the highest probability of being the median, which indeed they are.

5.3 Advantages and Disadvantages

The method has the obvious advantage of indicating how "correct" the median prediction is. While our other methods provide us with several good answers that are likely to be the best starting nodes, they do not indicate *how likely* this prediction is. Probability Method, on the other hand, determines the exact probability of each node being the median. The reliability of the answer is the answer. There is no guesswork in this method. This can however be misleading. Sometimes even if a node has only a 0.05 chance of being the median, the actual graph may be one of the few graphs where this node is the median.

This method also has the disadvantage of a long run time. When graphs have more than even ten nodes, the number of possible graphs is extremely large. Finding even just this number of graphs takes a long time to run, let alone having the program make the graphs and find their medians. A slight modification on the algorithm is to have it make only all of the "necessary" graphs. This is

[9]The probabilities do not sum to 1 because in some graphs more than one node is the median.

done by *only looking at the possible combinations of the `Maybe_Necessary` edges.* There are significantly fewer possible combinations when the extra `Maybe`'s are ignored. Since in all of our graphs, we are working with p's that are less than 0.5, there is more chance of an edge not being there than of an edge being there. Therefore, adding extra `Maybe` edges when we have already added several `Maybe_Necessary` edges creates graphs that have a very low probability of being the actual graph. Ignoring these extra `Maybe` edges, particularly as p becomes smaller making extra edges even less likely, does not largely impact the probabilities. However, this decrease in the number of possible combinations only allows a quick run time for slightly larger graphs. By about thirteen nodes[10], even this method cannot run quickly. This method is best suited to problems with very few nodes.

6 Further Directions

6.1 New Methods

We have developed methods that look both at the degree of nodes (Fractional Method) and the shortest path lenghts to other nodes (Cost Method). We also looked at the probability of certain graphs existing given our run data (Probability Method). However, in determining the center of the graph, there are several facts about adjacency matrices that may also play a role. For example, squaring adjacency matrices gives information about the number of paths of length two between various vertices in the graph. However, although our Fractional Method makes a modified adjacency matrix, we do not have an actual adjacency matrix. We have not evaluated how our pseudo-adjacency matrix from Fractional Method reacts to this manipulations. Much work could be done in this area. Another way in which adjacency matrices are sometimes used to find information about a graph is by looking at their determinants. We also did not analyze the determinants of our psuedo-adjacency matrices but this may provide more accurate methods for uncovering the best starting nodes.

6.2 Graph Model Modifications

For our problem, we worked with very basic graphs. Our model was based on the fact that any node would immediately pass the information along to any other node to which it was connected. However, using more realistic and complex models of graphs can enhance the problem.

One example is a directed graph. While it may be the case that person A would tell a rumor to person B, person B might not necessarily tell a rumor to person A. So while an edge could point from node A to B, there would not be an edge from A to B. This models realistic human behavior and creates a more

[10]The run time depends on the number of `Maybe_Necessary` edges, which can vary even within graphs with the same number of nodes. Because of this, there is not an exact number of nodes that serve as the cut off point for a decent run time of this algorithm.

complex graph. It is likely that information regarding the best starting points could be recovered from this sort of graph.

Additionally, a weighted graph could be used to reflect information not being immediately passed along. A large weight could indicate that while node A will eventually tell node B, A will not pass along the information for several days. This would make it much harder to determine who passed information along to whom. It would also have similar properties to an electrical network if these weights are thought of as conductivities or resistors.

A regular graph with each node having the same degree (or the limited degree in a modified regular graph) could also illustrate another way information could be passed along. In real life, a person with a lot of connections may not tell the information along to everyone he or she knows but instead only to the first few people they see while the information is still new. A graph could model this behavior by having each node pass the information along to a certain number (or certain maximum number) of nodes. When a node has a degree higher than this number of nodes, there could be a probabilistic analysis as to which nodes the original node is most likely to pass the information.

One final model could take into account a person's willingness to act upon learning information. For example, an advertiser wants the information to get to people who will actually buy the product. If we have two different types of nodes, one to represent people who will buy the product and people who will not, work could be done to analyze how to get the information to the nodes who will buy the product. Reaching the nodes who will not buy the product is of less importance in passing along the information. While everyone still passes information along, only certain nodes act on this information and it is most important that the information gets to these nodes.

In our methods, we first planted the information with one starting point in our sample runs[11]. However, analyzing data with runs that use multiple starting points could provide information about finding the best group of starting points. In many applications, this would provide a more realistic model. For example, in large scale advertising, a company does not advertise to one person and expect them to tell all his or her friends. Instead, an advertising campaign would try to target the best group of people, hoping that these people would then pass along the information. More analysis could be done regarding when data comes from runs with multiple starting points as well as choosing the best group of starting points. We know that we would not want two (or more) starting points that are connected to the same nodes as this would not cause the information to move along faster. Instead, we would want two (or more) starting points that pass the information along to different nodes, causing the information to pass faster then if it were started with one node alone. However, analysis beyond that has not yet been done.

Lastly, our graphs used Erdős-Rényi Random Graphs. Generally, social net-

[11]Fractional Method can run with more than one starting node whereas Cost Method can only have one starting node in each run. This is because if Cost Method starts with multiple starting nodes, we never have a single that we know is certainly connected to multiple points through which we can connect other nodes.

works are modeled using Power Law (Zipf) Distributions[3]. These distributions acknowledge the fact that in a social network, few nodes are connected to a lot of nodes while many nodes are connected to only a few nodes. This more accepted model of social networks would provide more accurate analysis about how information travels in a social networks.

6.3 Runtime Analysis

While we have analyzed how accurate our algorithms are at finding the actual center or median, we have not done any analysis about the speed of our algorithms. An algorithm with a fast runtime, even if it is slightly less accurate but still an improvement, can be useful in certain situations. Runtime analyses could be done for the algorithms to compare them in this manner.

7 Summary

While Fractional Method utilizes estimated degree of nodes in choosing best starting points, Cost Method takes into account a maximum shortest path length between nodes. Fractional Method does a better job of picking a starting point that has the information spread throughout the graph in a similar manner to a center or median. Both, however, provide a statistical improvement upon picking a random node from the graph. The analysis also indicates that the methods work better for larger graphs.

Probability Method takes into account the exact probability of a given node being the median. While this method can pick the best starting nodes, it is slow. It takes too long to run on large graphs and therefore cannot be quickly used for as many situations. An estimated version speeds up the process somewhat but still cannot be used on large graphs.

8 References

References

[1] Evans, James Robert and Edward Minieka. Optimization algorithms for networks and graphs. New York: Marcel Dekker, Inc., 1992.

[2] Foulds, L.R.. Graph Theory Applications. New York: Springer Verlag, 1995.

[3] Chung, Fan and Linyuan Lu and Van Vu. "Eigenvalues of Random Power law Graphs." Annals of Combinatorics 7(2003): 21-33.