

Weight Centrality in Diffusion Networks

Nicole Taheri¹, Aditya Mittal², Kathryn Hymes³

ABSTRACT

Among social networks, a common issue is choosing the initial set from which a behavior is spread. Representing a social network as a graph, $G(V, E, W)$, we let the members of the graph be the vertices, while relationships between the members are the edges. The interactions between the members are the edge weights. The two most widely used methods are *degree centrality*, which chooses as the initial set the nodes of highest degree, and *distance centrality*, which chooses as the initial set the nodes with the maximum number of other nodes within a certain radius (where the radius is the number of nodes between two nodes). We create an algorithm that is based on the weights of the incident edges to each node, equivalently the number of interactions between each member. We compared our *weight centrality* algorithm to a degree centrality algorithm on the same network; our algorithm performed better than the standard degree centrality algorithm when at least 9 nodes were chosen to be in the initial set.

Background

In the literature there are many studies on the effects of diffusion through social networks. The goal is to choose an initial target set of users in the network such that the diffusion through the network is maximized. The problem of finding this initial set exactly is NP-hard, since it would require checking every possible initial target set. However, there are many heuristics that attempt to maximize the diffusion through the network, including degree centrality, distance centrality, clustering and cascading threshold models. In our work, we present a new heuristic based on weight centrality and compare it with the more common heuristics of degree centrality and distance centrality.

Degree centrality is the heuristic that maximizes the diffusion through the network based on choosing the initial target set to be nodes with the highest degrees. Distance centrality, on the other hand, is based on choosing the nodes with the greatest amount of nodes within a certain distance. Our heuristic of weight centrality is based on selecting the initial target set as nodes which have heavy incident edges. In this heuristic, the edge weights represent the interaction between the nodes.

There are a number of concepts that can be taken into consideration when applying these heuristics. Node saturation is our concept of the limit to the interaction that can occur between the nodes. Selecting an initial target set which includes saturated nodes will not increase the diffusion through the network. Depending upon the nature of the network, other factors such as coordination come into play.

Nicole Taheri, Aditya Mittal, and Kathryn Hymes are M.S. students in Institute of Computational and Mathematical Engineering at Stanford University. This project has been done for Amin Saberi's Discrete Math and Algorithms class CME 305.

¹ ntaheri@stanford.edu

² admittal@stanford.edu

³ kehymes@stanford.edu

The idea of coordination is discussed in greater depth in *The Role of Compatibility in the Diffusion of Technologies through Social Networks*. (Immorlica, Kleinberg and Mahdian). Coordination requires that two neighbors both either interact with each other or they both do not interact with each other if they want to maximize their utility. This concept leads to the idea that in order to maximize the diffusion between the nodes we need friends to coordinate with each other, thereby maximizing interaction.

Algorithm

We create an algorithm which takes a directed graph $G(V, E, W)$ as a sparse matrix and determines based on the weight centrality principal the set of nodes that should be activated initially. In order to do this we first compute a node weight matrix, W , which has its first column the identification numbers of the unique elements, and the second column equals the sum of the edge weights of the outgoing edges of the corresponding node. This value in second column will become the weight value that is associated with each node.

Next we start building an initial set I from an empty set by adding one node at a time such that each addition makes the initial set's average weight higher. We randomly choose a node to possibly add to the initial set I ; if this node increases the average node weight, it is added, otherwise another node is randomly chosen. We create a threshold value *threshold* that is set to a specified value to give us a relative measure of the increase in the average each time a node is added to the set. We compute the *currentHold*, where

$$current\ hold = \frac{\sum_{i=1}^n w_i}{n} - \frac{\sum_{i=1}^{n-1} w_i}{n-1}$$

which refers to the difference between the average node weight of the current initial set and that of the previous node weight. If for more than $\alpha = 5$ additions to the initial set, the *currentHold* is less than *threshold*, the algorithm terminates, deciding that the average will not increase substantially enough for motivation to add more nodes to the initial set. We now return the nodes in the active set as the initial set I that should be activated.

Using this output, the degree centrality algorithm is called. The degree centrality algorithm takes in the number of nodes, n , that are output from the weight centrality algorithm, and finds the n nodes that have the highest degree. These nodes then become the node set that the degree centrality algorithm outputs as the initial set, I , to activate.

Moreover, both the weight centrality and degree centrality algorithms call a separate function *how_many.m* which computes the number of nodes that are within a connected component of at least one of the nodes in the output initial set I . In other words, *how_many.m* computes the number of nodes that are reachable through a directed path from the initial set I .

Discussion

The basis of our idea comes from the thought process that the number of interactions between nodes may be more important than the number of edges incident to a node. Below are a few examples that demonstrate this idea.

Suppose we want to police the traffic in a road network, where an edge represents a road, a node represents an intersection and the weight on an edge represents the amount of traffic on the corresponding road. The goal is to put police in the areas so that they will catch the most perpetrators of the law. Degree centrality corresponds to placing the policemen on the nodes where the most roads meet; weight centrality corresponds to placing the policemen on the nodes with the maximum amount of traffic. It should be clear that the weight centrality algorithm will have more success than the degree centrality version in this case.

Another commonly observed example is an online social network. In order to spread an idea or behavior among members of the community, degree centrality chooses the people with the most friends, while weight centrality chooses those that interact most with their friends. In this case a weight centrality algorithm may be more effective, since those who interact with their friends are more likely to have a larger influence over others.

Next, suppose we have a fish network and we want to spread food all through the fish network. One option is to give the food to the fish that have a lot of friends, are popular politicians, but keep all the food to themselves. The other option is to find the fish who interact and generously give food to everyone they know. Then of course, we give the food to the generous fish using our weight centrality algorithm.

Experimental Results and Analysis

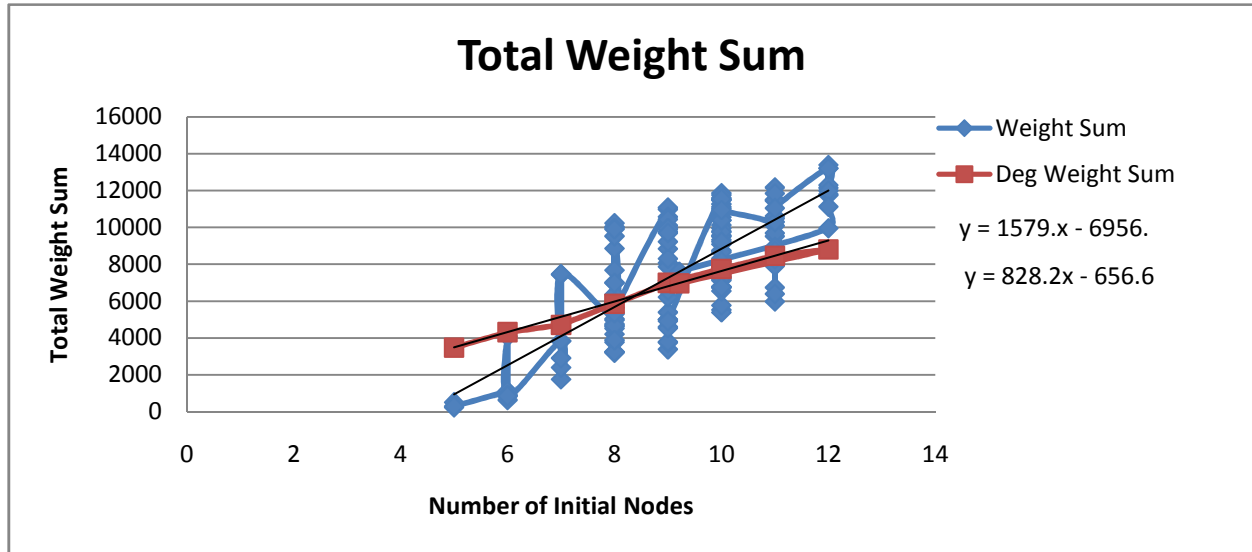
We run our weight centrality algorithm as well as the degree centrality algorithm on our data⁴. Each row of the data that we use consists of the user identification numbers and interactions between the network's users. We use this data to define a graph $G(V, E, W)$ where the users are vertices and the weight on the edges between the vertices is the number of interactions between the two users. Using Matlab code we constructed for each of the two algorithms we compare the performance of the two algorithms for two different tasks. The two performance comparisons include maximizing the interactions between the users and maximizing the number of users we can influence.

Maximizing the Interactions between Users

To maximize the interactions between users we try to maximize the weights on the edges of the initial set I we define using our algorithms. Graph 1 displays the total weights of the initial set I for 141 runs of both algorithms against the number of nodes in the initial set I . For each run the number of nodes, n , in the initial set I is determined by the run of the weight centrality algorithm. The degree centrality

⁴ The data we use for this project is confidential and this work should not be distributed outside of class.

algorithm then computes an initial set as the n nodes with highest degree. Then we use a separate function to compute the weights of the initial set for both algorithms and plot the data to get Graph 1.



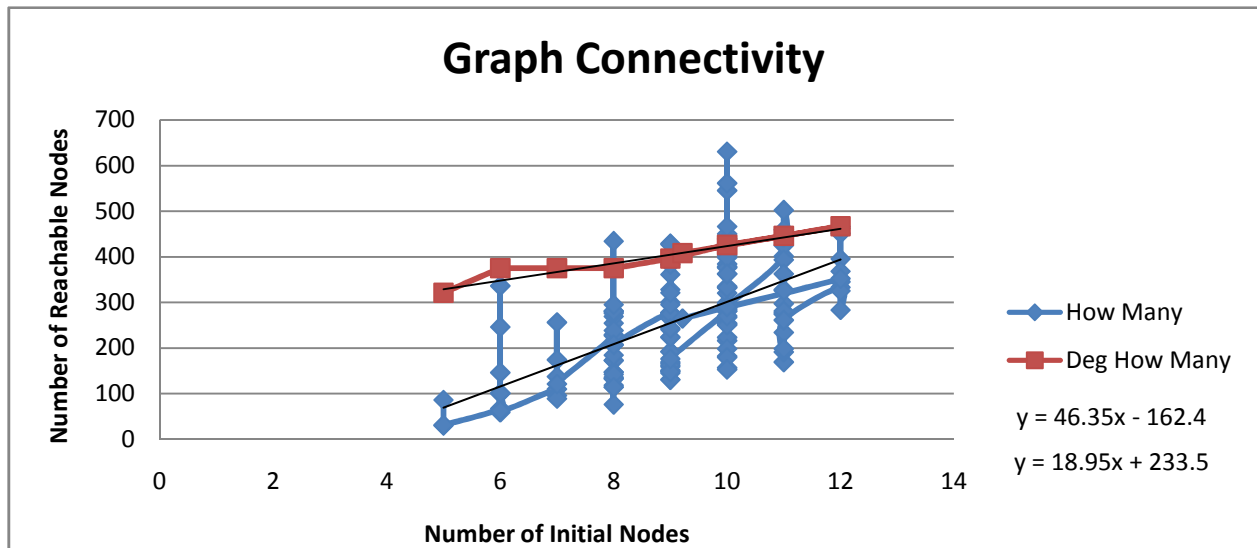
Graph 1

We see from Graph 1 that if the number of initial nodes we pick are 5 or 6, then we should use the deg centrality algorithm's initial set because it maximizes the total interaction. If the number of initial nodes we pick are more than 6, then we should use the initial set from the weight centrality algorithm that gives us the maximum total weight. For larger initial sets we see that the weight centrality will always do better than degree centrality from the graph because its slope is almost twice as much.

This raises the question of how many nodes should we pick to initially influence. Of course the more the better, but we don't want to influence every node in the whole network, so we would like to find the optimal number of nodes to influence. For our data we believe that the optimal number of nodes to pick is 10, and more reasoning for this belief is provided in the section where we also try to maximize the number of users influenced. From the density of points in Graph 1 we see that our algorithm chose to pick out 10 nodes more often than other values, based on our threshold settings used for termination criterion in the weight centrality algorithm. We also compute that on average the algorithm terminated after adding 9.21 nodes to the initial set.

Maximizing the Number of Users Influenced

In order to maximize the number of users influenced, we use our Matlab function *how_many.m* to determine the number of nodes reachable from the initial set in a directed fashion. Graph 2 then displays the number of reachable nodes from the initial set I for 141 runs of both algorithms against the number of nodes, n , in the initial set I , computed as before.



Graph 2

We see from Graph 2 that in order to maximize the number of reachable nodes we should pick the initial set that is from the data point with 10 initial nodes. If we pick 10 initial nodes we can maximize the number of reachable nodes by using the initial set from the data point that has the maximum value in Graph 2.

Using the slopes from the equations of the trend lines in Graph 2 we extrapolate the data to determine that the trend lines will intersect when the number of chosen nodes is 14.45. Almost always our algorithm terminates before choosing 14 nodes and in the optimal case we decide that using 10 nodes is best, therefore, we would ideally use the initial set from the degree centrality algorithm if we were maximizing the number of users influenced.

Conclusion

We conclude that degree centrality should be used when the goal is to maximize the number of people that are approached with the application. However, when the goal is to maximize the usage of the application, our weight centrality algorithm should be used. The reason for this conclusion is that on average degree centrality algorithm approaches more users, whereas the weight centrality algorithm on average maximizes the interaction and application usage between the users.

Matlab Code

Function getloop

```
function nodes = getloop(interactions)
```

```
%determine the average number of interactions
```

```
[m,n]=size(interactions);
```

```

avg_iact=sum(interactions(:,3))/m;
%get a random row
%row = ceil(rand*m);

%compute the node weight matrix
id_vector(1)=interactions(1,1);
weight_matrix(1)=interactions(1,3);
for i=2:m
    flag = 0;
    p=length(id_vector);
    for j=1:p
        if (interactions(i,1)==id_vector(j))
            weight_matrix(j)=interactions(i,3)+weight_matrix(j);
            flag=1;
        end
    end
    if (flag==0)
        weight_matrix(p+1)=interactions(i,3);
        id_vector(p+1)=interactions(i,1);
    end
end

%start an active set
act_w=0;
node_cnt=0;
p=length(weight_matrix);
avg_w=0;
threshold=50;
currentHold = 20;
threshold_test=0;
while(threshold_test<5)
    prev_avg_w=avg_w;
    yes = 1;
    while (yes==1)
        row = ceil(rand*p);
        %look for the row in the node set
        yes = 0;
        r = node_cnt; %size(node_set);
        for k=1:r
            if (node_set(k)==row)
                yes = 1;
            end
        end
    end
    %Assume node is in the active set and compute the average
    node_cnt=node_cnt+1;
    act_w=act_w+weight_matrix(row);
    avg_w=act_w/node_cnt;

    if (prev_avg_w < avg_w) %Our assumption is correct
        node_set(node_cnt) = row;
        currentHold = avg_w-prev_avg_w;
        if (currentHold<threshold) %threshold_test should run multiple times
            threshold_test=threshold_test+1;
        end
    else %Our assumption was wrong

```

```

    act_w=act_w-weight_matrix(row);
    node_cnt=node_cnt-1;
    avg_w=act_w/node_cnt;
end
end

```

%output the nodes in the active set

```

nodes=node_set;
weight = avg_w*node_cnt;

```

```

[deg_nodes, deg_weight, deg_how_many] = deg_cen(length(node_set), interactions, weight_matrix);

```

```

for i=1:length(node_set)
    node_set_id(i) = id_vector(node_set(i));
end
howmany = how_many(node_set_id, interactions, 2);

```

```

fid = fopen('exp1.txt', 'at');
fprintf(fid, '%d\t', length(node_set));
fprintf(fid, '%d\t', weight);
fprintf(fid, '%d\t', howmany);
fprintf(fid, '%d\t', deg_weight);
fprintf(fid, '%d\n', deg_how_many);
fclose(fid);

```

Function callloop

```

function a = callloop(interactions)
for i = 1:500
    getloop(interactions);
end
a = 1

```

Function how_many

```

function howmany = how_many(node_set_id, interactions, k)
[m,n]=size(interactions);
o=1;
p=2;

index = length(node_set_id) + 1;
while (o~=p)
    q=p-o;
    o=length(node_set_id);
    % index=o+1;
    for i = 1:m
        for j = q+1:o
            if (node_set_id(j)==interactions(i,1) && interactions(i, 2)~=0)
                node_set_id(index)=interactions(i,2);
                interactions(i, 2) = 0;
                index=index+1;
            end
        end
    end
end
end

```

```

    p=length(node_set_id);
end

p = length(node_set_id);
index = 0;
yes =0;
used_id(1)=node_set_id(1);
count=1;
for i = 2:p
    index=index+1;
    for j = 1:length(used_id)
        if(node_set_id(i)==used_id(j))
            yes=1;
        end
    end
    used_id(index)=node_set_id(i);
    if (yes==0)
        count=count+1;
    end
    yes=0;
end
    howmany = count

```

Function deg_cen

```
function [nodes, deg_cen_weight_sum, deg_how_many] = deg_cen(n, interactions, weight_matrix)
```

```
%determine the average number of interactions
[y,z]=size(interactions);
```

```
%compute the node weight matrix
```

```
id_vector(1)=interactions(1,1);
deg_vector(1)=1;
for i=2:y
    flag = 0;
    p=length(id_vector);
    for j=1:p
        if (interactions(i,1)==id_vector(j))
            deg_vector(j)=deg_vector(j)+1;
            flag=1;
        end
    end
    if (flag==0)
        deg_vector(p+1)=1;
        id_vector(p+1)=interactions(i,1);
    end
end
end
```

```
dw_matrix = [deg_vector; weight_matrix; id_vector]';
dw_matrix = sortrows(dw_matrix, 1);
```

```
p = length(deg_vector);
deg_cen_weight_sum = sum(dw_matrix(p-n+1:p, 2))
nodes = dw_matrix(p-n+1:p,3);
```

```
for i=1:length(nodes)
```

```
node_set_id(i) = nodes(i);  
end  
  
deg_how_many = how_many(node_set_id, interactions, 1);
```

Acknowledgments

We would like to thank Professor Amin Saberi and Adam Guetz and Chen Gu for a great course in Discrete Math and Algorithms for which we did this project.

References

- [1] Immorlica, Nicole, et al. "The Role of Compatibility in the Diffusion of Technologies through Social Networks."
- [2] Kempe, David, Jon Kleinberg and Eva Tardos. "Influential Nodes in a Diffusion Model for Social Networks."
- [3] Kempe, David, Jon Kleinberg and Eva Tardos. "Maximizing the Spread of Influence through a Social Network."
- [4] Kleinberg, Jon. "Cascading Behavior in Networks: Algorithmic and Economic Issues."