Pomona College

Senior Thesis in Mathematics

Modeling Diffusion in Dynamic Networks

Author:
Helen Guo

Advisors:
Dr. Ami Radunskaya &
Dr. Maryann Hohn

Submitted to Pomona College in Partial Fulfillment
of the Degree of Bachelor of Arts

May 18, 2020

Abstract

This paper examines how the evolution of networks based on different affiliation strategies affects the rate at which spreading (diffusion) occurs. Using MATLAB simulations, we analyze how the rate of spreading differs in networks that evolve based on indiscriminate (randomly-evolving), degree, closeness, and betweenness affiliation strategies. We first introduce spreading dynamics at the beginning of network evolution, then examine how diffusion rates change when spreading is introduced later in the evolution process, after long-term network characteristics have been established.

Our results show that diffusion occurs more slowly in networks governed by degree, closeness, and betweenness affiliation strategies than in an indiscriminately-evolving network. This slowing becomes more dramatic when spreading is introduced later in the evolution process. Networks driven by all three discriminate affiliation strategies exhibit similar spreading rates, despite different long-term characteristics of the network. Future work may determine factors that differentiate the diffusion rates of the discriminate dynamic networks.

Contents

Chapter 1

How Preference-Based Social Networks Evolve

Social network analysis provides a powerful basis for understanding how complex organizations can evolve from social affiliation choices. Networks are represented mathematically as graphs and can simulate many types of interaction, including but not limited to interactions between people in a society, organisms in a species, cells in a body, and avatars on a social media platform. On a basic level, these graphs consist of **nodes** representing actors and **edges** representing ties that connect the nodes. A node's **indegree** is the number of edges that other nodes extend to it. A node's **outdegree** is the number of edges it extends to other nodes. To quantify the influence of a node on the network, three main network affiliation measures are defined:[1]

Degree affiliation, also known as the "popularity measure," is the most intuitive affiliation measure, which quantifies the proportion of connections in the graph attached to a node v_i in a network of n nodes. In other words, it is calculated by dividing the number of indegrees of node v_i by $n - 1$, the number of nodes in the network excluding v_i:

$$P(v_i) = \frac{d_{in}(v_i)}{n-1}$$

Betweenness quantifies to what degree a node serves as an intermediary between nodes. If node v_j extends an outdegree to node v_i, which extends an outdegree to node v_k, then v_i is on the shortest path between v_j and v_k. To solve for the betweenness of node v_i, we first count the number of shortest paths that passes through it. Let $count(v_i)$ represent the number of shortest paths that passes through the node v_i. Since v_i lies on the shortest path between two nodes for every shortest path counted, we double $count(v_i)$ and then divide by $(n-1)(n-2)$, the total number of shortest paths:

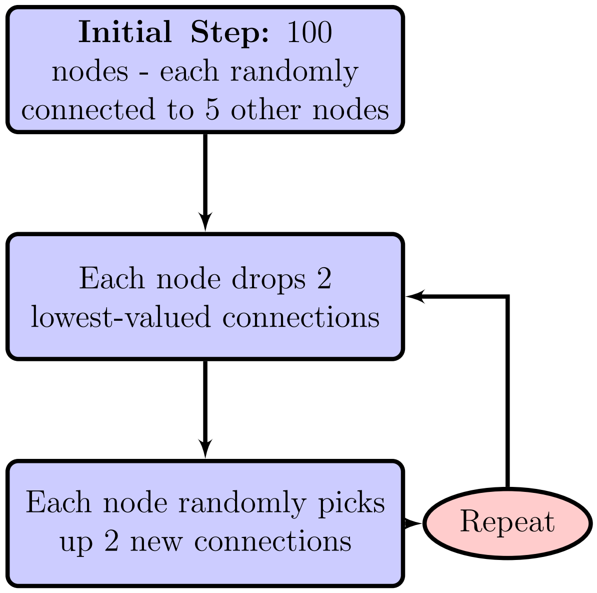
$$B(v_i) = \frac{2count(v_i)}{(n-1)(n-2)}$$

Closeness is related to the average distance of a node to all other nodes. Distance between two nodes is quantified by one greater than the number of nodes that lie in between them. So if node v_i extends an outdegree to node v_k, then the distance between them is 1. Closeness is calculated by taking the reciprocal of the average distance for a node v_i to any other node v_j, so that a higher value represents greater closeness. To solve for the value, we divide $n-1$ (the number of nodes in the network excluding v_i) by the sum of the distances between node v_i and all other nodes v_j:

$$C(v_i) = \frac{n-1}{\sum d(v_i, v_j)}$$

Fefferman and Ng (2007) models how social networks driven by a specific network affiliation measure evolve.[1] Their model network has 100 nodes. Each node begins with five randomly-chosen outdegrees. At every time step, each node keeps its connections to three nodes with the highest affiliation measures and drops the other two nodes. The node then connects to two new nodes randomly to replace the ones it dropped. Figure 1.1 outlines this procedure.

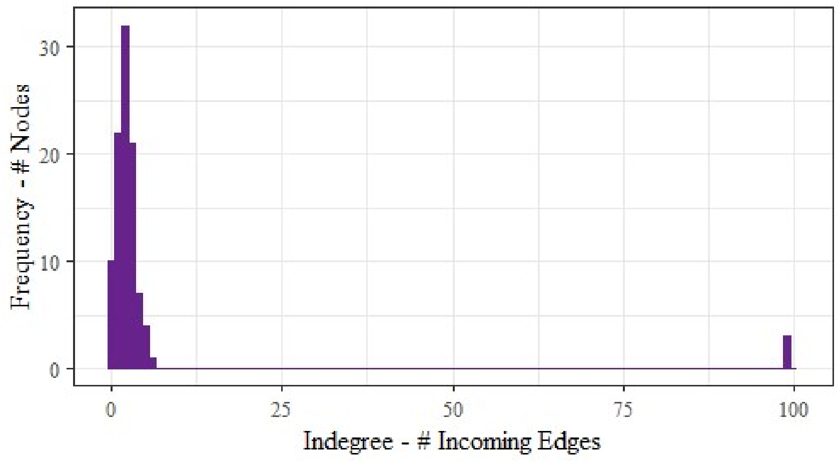
Figure 1.1: Model of Dynamic Network Evolution


Initial Step: 100 nodes - each randomly connected to 5 other nodes
Each node drops 2 lowest-valued connections
Each node randomly picks up 2 new connections
Repeat

In the 100-node model of dynamic network evolution described in Fefferman and Ng (2007), each node initially randomly connects to five other nodes, then drops two of their lowest-valued connections and randomly picks up two new connections at each time step.[1]

Dynamic networks driven by the affiliation strategies defined above exhibit unique long-term behavior, as mentioned in Wilson et al. (2020).[3] In networks driven by degree affiliation, we can see long-term behavior where there is a set of **core nodes** that have attained the maximum number of indegrees. These nodes remain maximally popular in infinite time since all nodes prefer to connect to nodes with the most indegree edges. In Figure 1.2 below, we generated a histogram of indegree values for a network of 100 nodes driven by degree affiliation at time step $t = 300$. Clearly, there is a network core, since some nodes have acquired 99 indegrees.

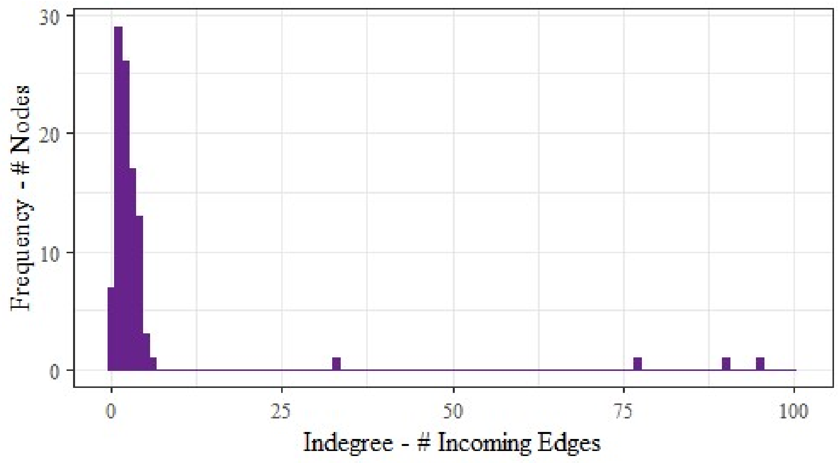
Figure 1.2: Node Indegrees in a Degree-Driven Network



A dynamic network driven by degree affiliation acquires a set of core nodes. The histogram above reflects a degree-driven network of 100 nodes evolved for 300 time steps by the process charted in Figure 1.1 through MATLAB simulation. The network reaches an equilibrium where three nodes have attained the maximum number of indegrees and remain maximally popular in infinite time.

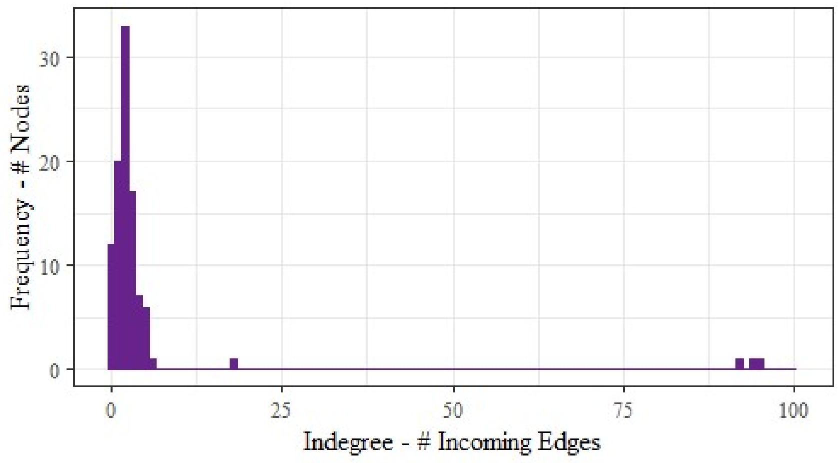
In contrast, nodes that become highly popular in networks driven by betweenness and closeness can lose their indegrees more easily than in degree-driven networks. The nodes that make up the core, identified by having acquired a higher indegree value than the majority of other nodes, can change. Core nodes do not necessarily attain the maximum number of indegrees as seen in Figure 1.3 and Figure 1.4.

Figure 1.3: Node Indegrees in a Betweenness-Driven Network



A dynamic network driven by betweenness affiliation results in a small number of nodes that have higher indegree values than the majority of nodes. The histogram above reflects a betweenness-driven network of 100 nodes evolved for 300 time steps by the process charted in Figure 1.1 through MATLAB simulation.

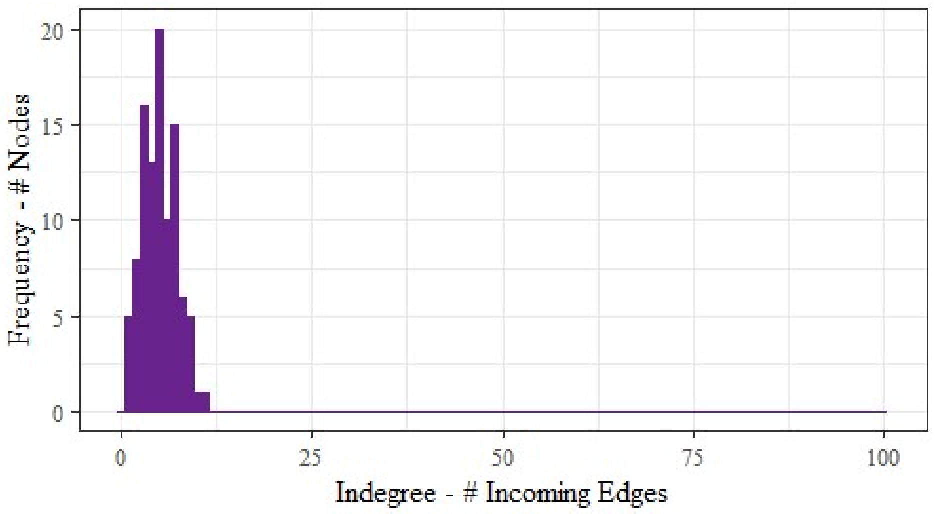
Figure 1.4: Node Indegrees in a Closeness-Driven Network



A dynamic network driven by closeness affiliation results in a small number of nodes that have higher indegree values than the majority of nodes. The histogram above reflects a closeness-driven network of 100 nodes evolved for 300 time steps by the process charted in Figure 1.1 through MATLAB simulation.

In the extreme case of a network in which nodes are affiliating indiscriminately without preference, a core is not likely to emerge as seen in Figure 1.5.

Figure 1.5: Node Indegrees in an Indiscriminate Network



A randomly-evolving network does not result in any node having a significantly higher indegree value than other nodes. The histogram above reflects a network of 100 nodes evolved without affiliation strategy for 300 time steps by the process charted in Figure 1.1 through MATLAB simulation.

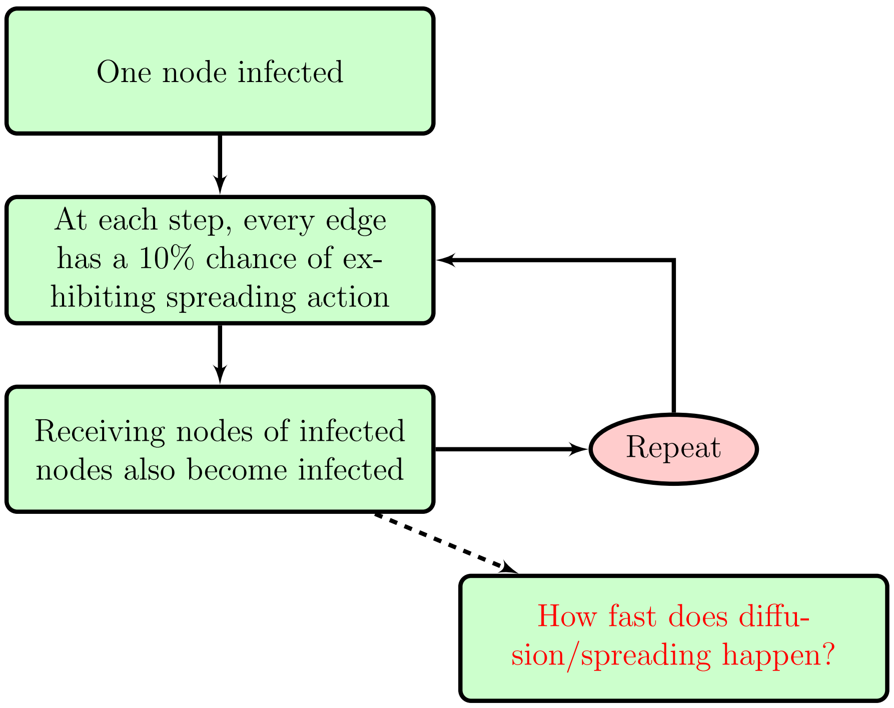
Chapter 2

Exploring Diffusion Rates in Dynamic Networks

In previous research discussed in Chapter 1, we observed that dynamic networks with a set affiliation strategy evolve to have a set of core nodes with significantly higher indegrees than others.[3] This paper sets out to explore how the evolution of discriminate dynamic networks affects diffusion rates using MATLAB simulations. We define **diffusion** as the spread of a substance, such as an infection or information, throughout a network of nodes. We introduce a theoretical infection at the beginning of network evolution, then examine how quickly the infection spreads to the entire network. We then introduce the infection later in network evolution to discern if diffusion rates change after long-term network characteristics have been established.

We take an intuitive approach to modeling diffusion in dynamic networks. Once a node in the network is infected, we want the number of active edges to resemble a Poisson point process. To do this, we code for each edge to have a 10% chance of becoming “active” at each time step. If an outward edge of an infected node becomes “active”, the receiving node then becomes infected as well. Since the probability of an edge becoming “active” is constant at each time step, the number of active edges in the network over time is a Poisson point process. Figure 2.1 outlines this procedure.

Figure 2.1: Model of Network Diffusion

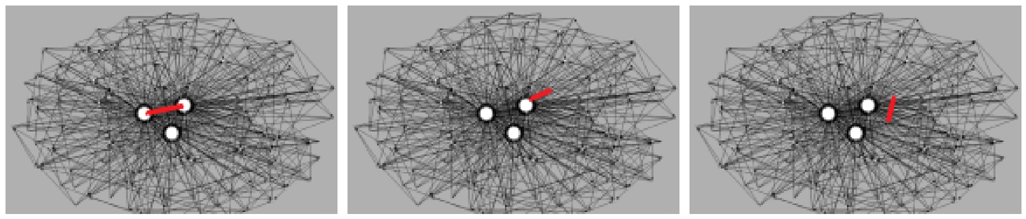

One node infected
At each step, every edge has a 10% chance of exhibiting spreading action
Receiving nodes of infected nodes also become infected
Repeat
How fast does diffusion/spreading happen?

In our model of network diffusion, each edge has a 10% chance of becoming "active" at each time step. If an outward edge of an infected node becomes "active", the receiving node then becomes infected as well.

It is important to note that "active" edges of nodes not yet infected cannot infect other nodes. For example, we can think of "active" edges as emails sent between coworkers and the "infection" as a secret sent by email; only employees who already have been "infected" with the secret can pass on the secret to others with an email (an "active" outdegree).

We hypothesize that in networks that develop a core, the nodes with a higher indegree value are more likely to receive a particular message or become infected. Hence, it may take longer for a message or infection to diffuse to peripheral nodes. Figure 2.2 displays a possible network with three core nodes and highlights conjectured "active" edges in red.

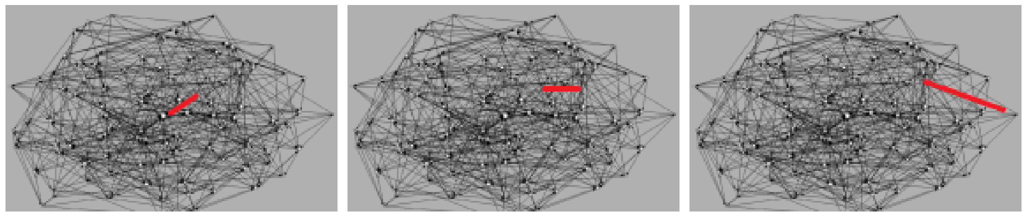
Figure 2.2: Diffusion in a Discriminate Network (Conjectured)



We hypothesize diffusion will occur more slowly in affiliation-driven networks which develop a core than in indiscriminate (randomly-evolving) networks. Base images are taken from Fefferman and Ng (2007).[1] The red lines depicting active edges are overlain manually.

In an indiscriminate (randomly-evolving) network, we hypothesize diffusion will occur more quickly. Since a set of core nodes does not manifest, edges between any two nodes are likely to become active. Figure 2.3 displays an indiscriminately-evolving network and highlights conjectured "active" edges in red.

Figure 2.3: Diffusion in an Indiscriminate Network (Conjectured)



We hypothesize diffusion will occur more quickly in indiscriminate (randomly-evolving) networks than in affiliation-driven discriminate networks. Base images are taken from Fefferman and Ng (2007).[1] The red lines depicting spreading interactions are overlain manually.

Chapter 3

Model Development

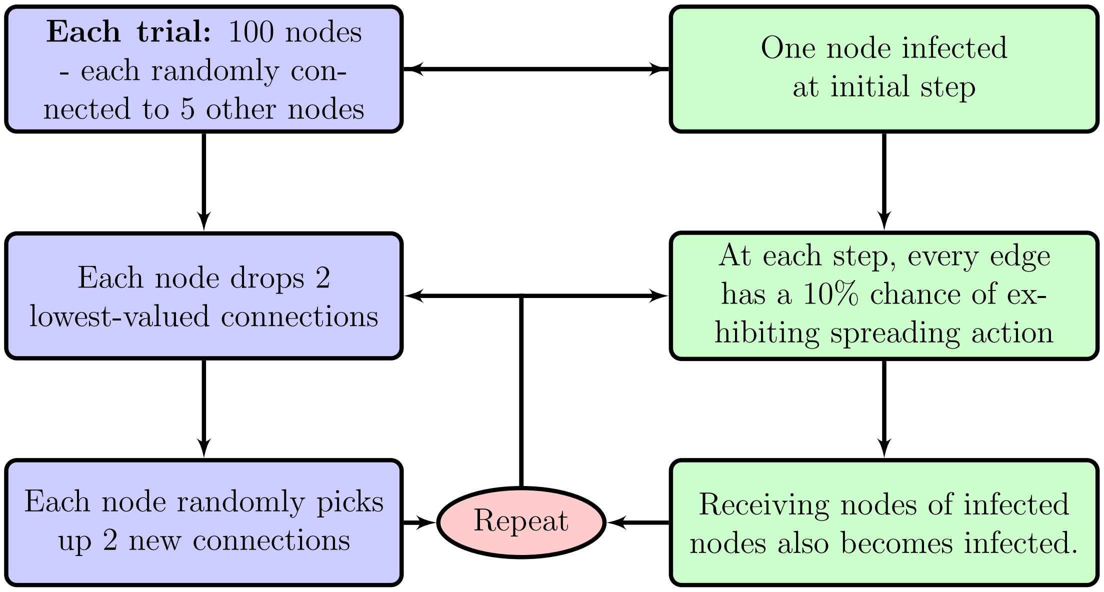
In MATLAB, we simulate diffusion as defined in Section 2 over a replica of Fefferman and Ng's model of network evolution. We adopt and modify code [included in the appendices] used in Brooks et al. (2018), which conducted mathematical analysis on the impact of social structure on ectoparasite load in allogrooming populations.[2] Degree, betweenness, and closeness affiliation strategies are defined as in Chapter 1. Here in Chapter 3, we evolve a network of 100 nodes indiscriminately (as a control), as well as based on the three affiliation strategies.

At time step zero, t_0, each node is randomly connected to five other nodes. Then at each time step, each node keeps three connections (with the highest affiliation measures according to the network's preference) and drops two (with the lowest affiliation measures). Each node picks up two new connections randomly. An indiscriminate network (without an affiliation strategy) drops two and picks up two new connections at each time step randomly.

With these rules in place, we add our model of diffusion. Experiments (a)-(d) randomly designate an "infected" node in the network to be the source of diffusion at the initial time step. Experiment (a) runs 50 independent simulations of a randomly-evolving network until every node in the network becomes "infected". Experiments (b) runs 50 independent simulations of a degree-driven network until every node in the network becomes "infected". Experiments (c) runs 50 independent simulations of a closeness-driven network until every node in the network becomes "infected". Experiments (d) runs 50 independent simulations of a betweenness-driven network until every node in the network becomes "infected".

To summarize, for each affiliation measure (including indiscriminate affiliation), we simulate 50 trials of dynamic network evolution and diffusion by the process illustrated in Figure 3.1.

Figure 3.1: Model of Diffusion in Dynamic Network Experiments (a)-(d)

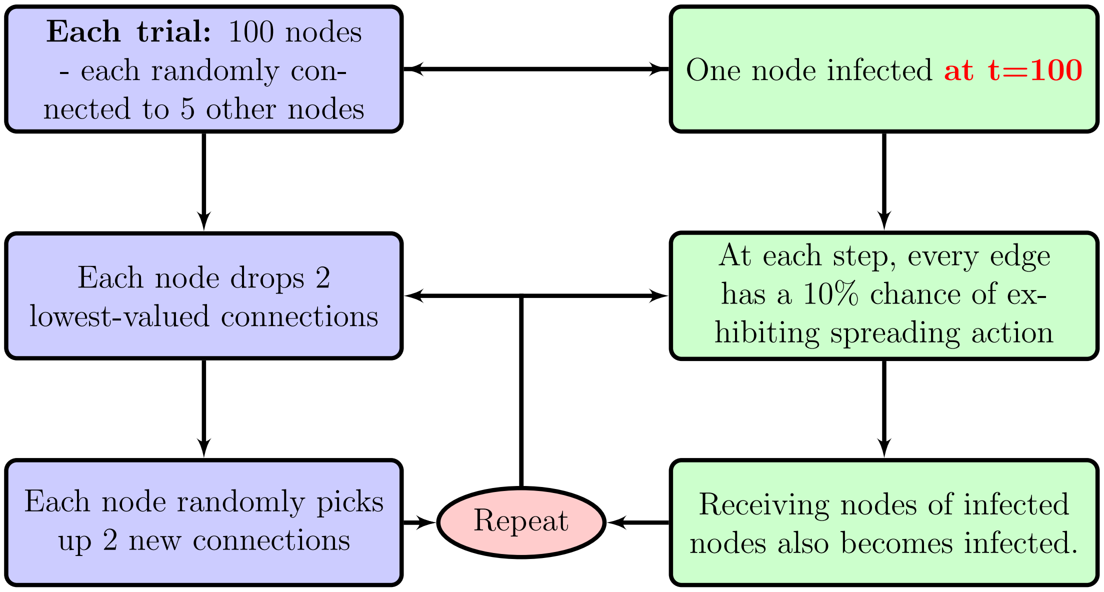


Each trial: 100 nodes - each randomly connected to 5 other nodes
One node infected at initial step
Each node drops 2 lowest-valued connections
At each step, every edge has a 10% chance of exhibiting spreading action
Each node randomly picks up 2 new connections
Repeat
Receiving nodes of infected nodes also becomes infected.

We use the dynamic network model on the left from Fefferman and Ng (described in Chapter 1).[1] We then introduce the spreading, or diffusion, process pictured on the right. Early experiments (a) - (d) initiate diffusion at the first time step.

Experiments (e)-(h) repeat experiments (a)-(d) but do not designate an initial source of diffusion (randomly infected node) until time step 100, after long-run network characteristics such as a core have been established. Figure 3.2 illustrates our process.

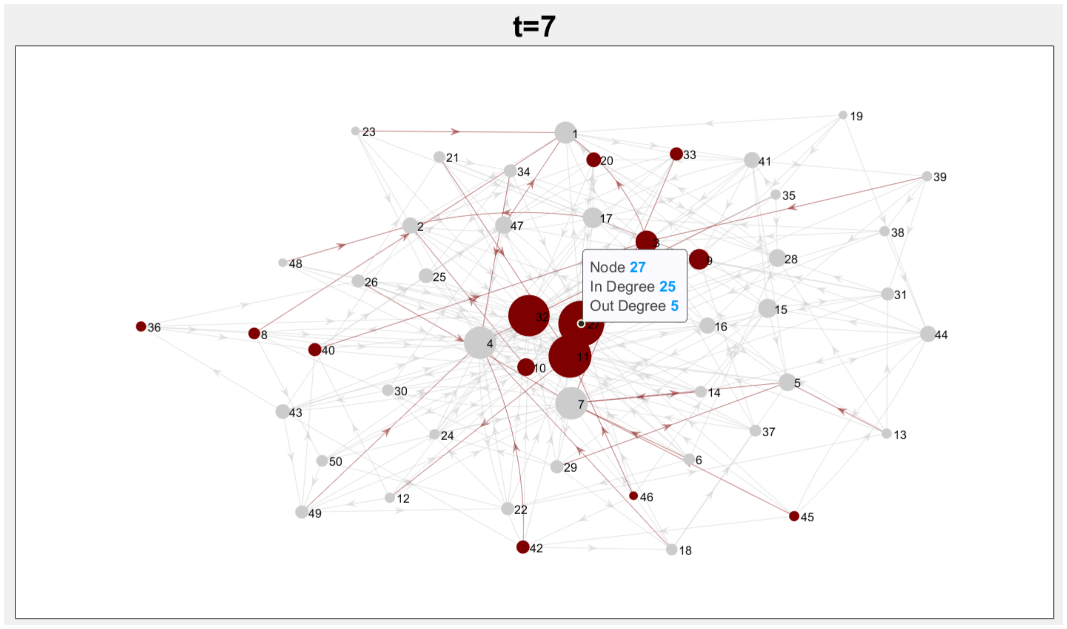
Figure 3.2: Model of Diffusion in Dynamic Network Experiments (e)-(h)


Each trial: 100 nodes - each randomly connected to 5 other nodes
One node infected **at t=100**
Each node drops 2 lowest-valued connections
At each step, every edge has a 10% chance of exhibiting spreading action
Each node randomly picks up 2 new connections
Repeat
Receiving nodes of infected nodes also becomes infected.

We use the dynamic network model on the left from Fefferman and Ng (described in Chapter 1).[1] We then introduce the spreading, or diffusion, process pictured on the right. Later experiments (e)-(h) initiate diffusion at time step 100 to capture results for more developed dynamic networks.

Our model allows for visual representation of network evolution at each time step as seen in Figure 3.3. Nodes with a higher number of indegrees are larger and closer to the center of the diagram. Infected nodes and active edges are colored in red while uninfected nodes are colored in gray.

Figure 3.3: Visual Representation of Network Evolution



A 100-node degree-driven network infected at the initial time step is pictured above at time step seven. Infected nodes and spreading interactions at the time step are pictured in red. Node 27, the most popular node at time step seven, has 25 indegrees and five outdegrees.

For each trial of an experiment, we store the number of infected nodes at each time step in a matrix with 50 columns. The columns differ by trial, and the rows differ by time step. To compare the rate of spreading for our experiments, we look at the mean spreading rate for all 50 trials.

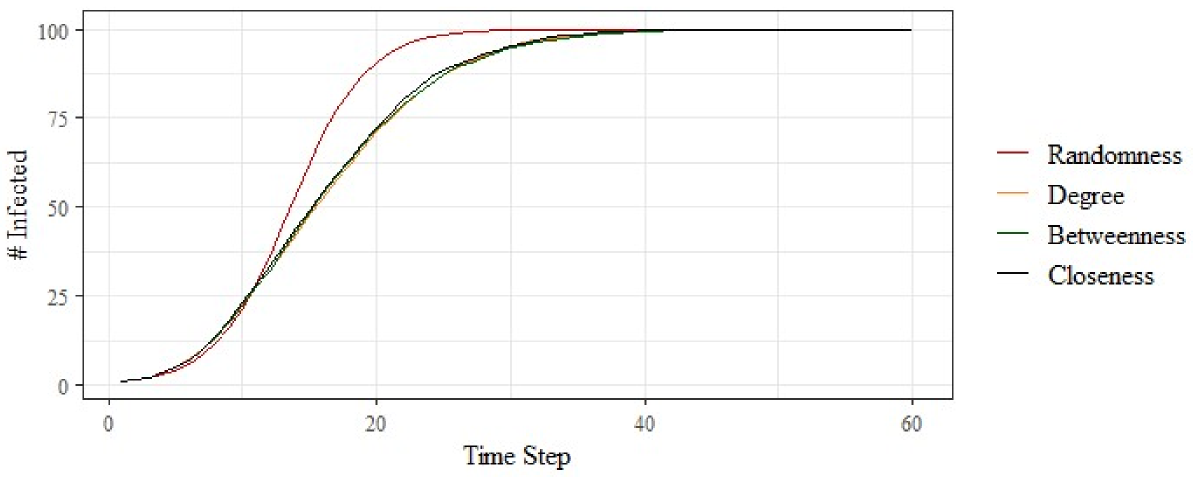
Chapter 4

Simulation Results

We simulated 50 trials of network evolution and diffusion for experiments (a)-(d), which model networks driven by indiscriminate, degree, closeness, and betweenness affiliation criteria respectively. Then we plotted and compared the mean diffusion rates of the 50 trials for each experiment as seen in Figure 4.1. Dynamic networks driven by degree, betweenness, and closeness affiliation metrics have discernibly slower diffusion rates than randomly-evolving networks. All three discriminate dynamic networks exhibit similar spreading rates.

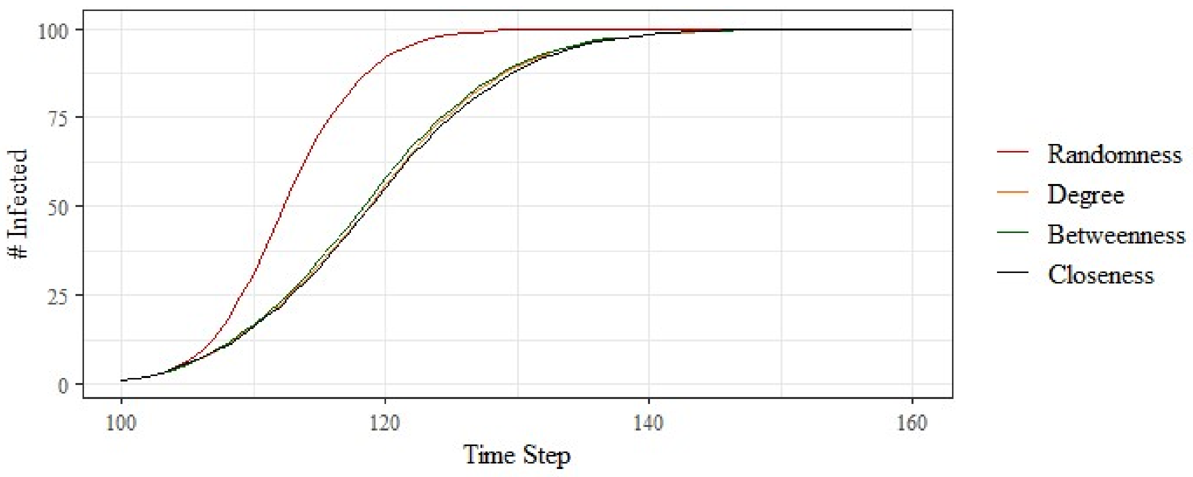
In comparing these results with the mean diffusion rates of (e)-(h) in Figure 4.2, we see that rates of diffusion slow even further in discriminate networks when infection is introduced later at time step 100. The mean diffusion rate of the randomly-evolving networks infected later (e) does not vary significantly from that of the randomly-evolving networks infected earlier (a).

Figure 4.1: Mean Diffusion Rates of Initial Infection Experiments (a)-(d)



Spreading occurs slower in networks driven by an affiliation strategy than in randomly-evolving networks. This distinction becomes apparent even when diffusion is initiated at the first time step, before long-term network characteristics have been established.

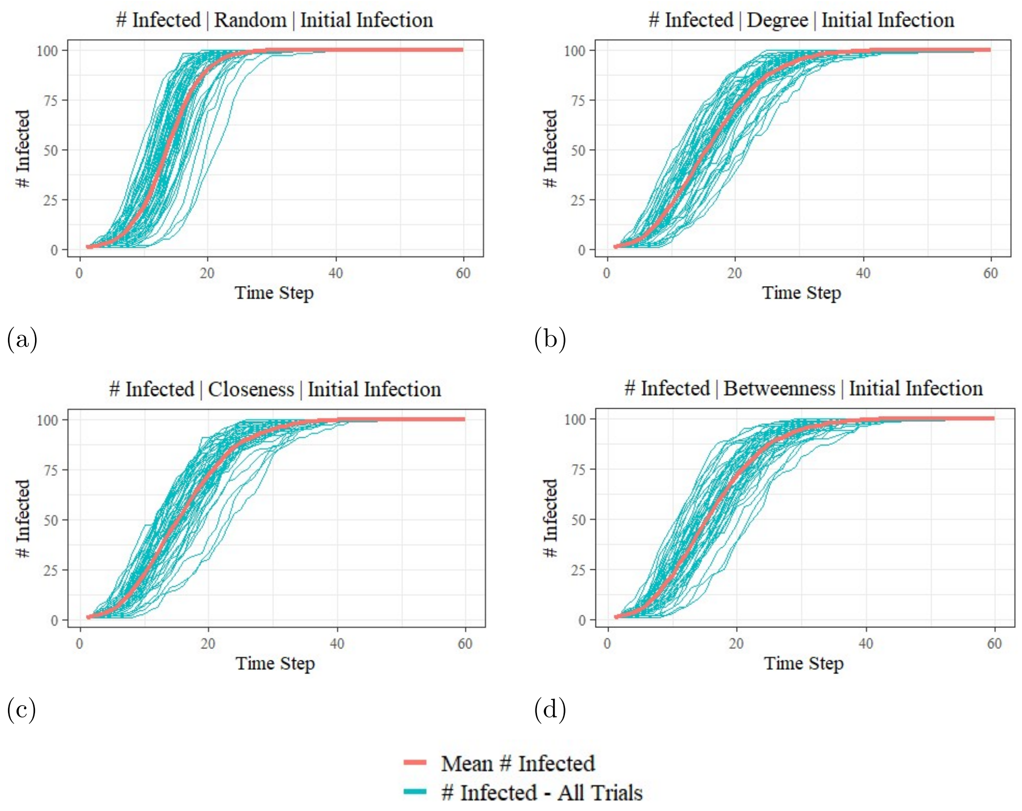
Figure 4.2: Mean Diffusion Rates of Late Infection Experiments (e)-(h)



When diffusion is initiated at time step 100, the slowing of diffusion in networks driven by an affiliation strategy becomes even more drastic.

Additionally, we plotted each of the 50 trials in blue and the mean diffusion rate of all trials in red. The holistic data supports our earlier observation that diffusion in networks driven by affiliation strategies proves slower than in randomly-evolving networks. The results of experiments (a)-(d) are shown in Figure 4.3.

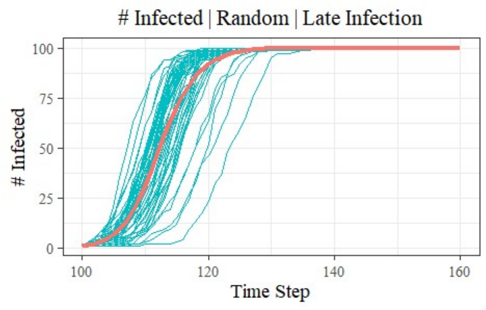
Figure 4.3: Diffusion Rates of Initial Infection Experiments



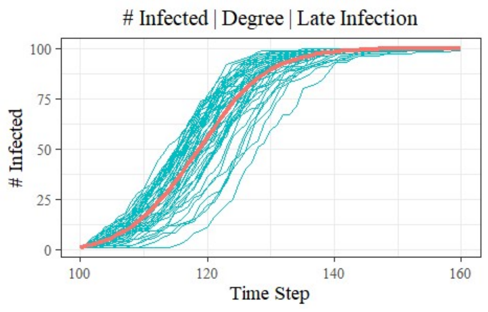
Diffusion occurs slower in networks driven by an affiliation strategy pictured in (b),(c), and (d) than in randomly-evolving networks pictured in (a). The blue curves represent all 50 trials of each network type when diffusion is initiated at the first time step. The red curve represents the mean diffusion rate of all 50 trials of each network type.

In experiments (e)-(h), spreading is implemented at time step $t = 100$. Diffusion rates slow down even further in networks driven by degree, closeness, and betweenness as seen in Figure 4.4. The mean diffusion rate of the randomly-evolving networks (e) does not vary significantly from that of the randomly-evolving networks infected earlier (a).

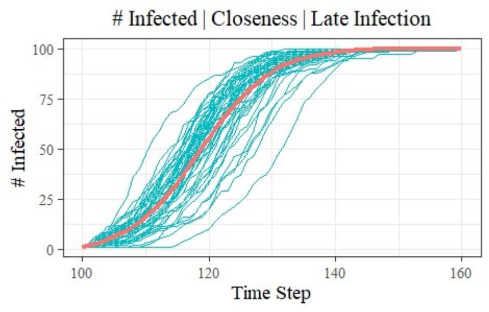
Figure 4.4: Diffusion Rates of Late Infection Experiments



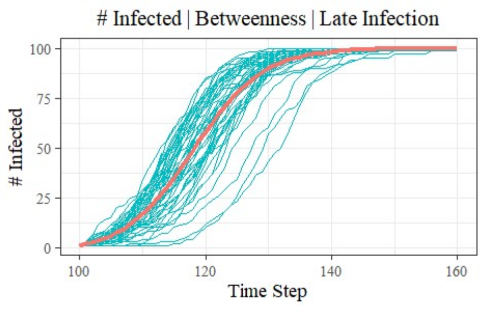
(e)



(f)



(g)



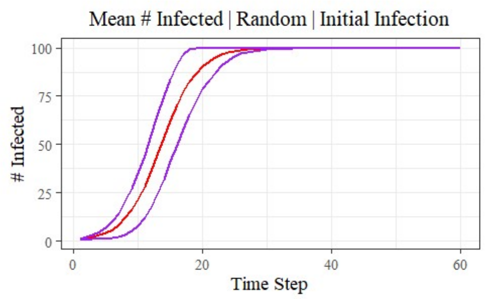
(h)

🟥 Mean # Infected
🟦 # Infected - All Trials

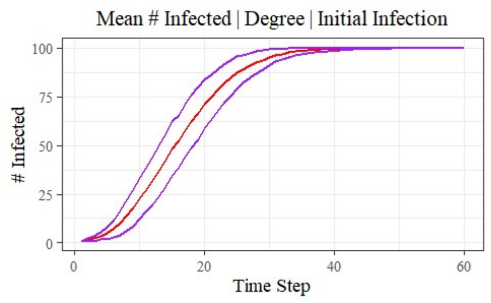
When diffusion is initiated at time step 100, spreading occurs even slower in networks driven by an affiliation strategy pictured in (f),(g), and (h) than in randomly-evolving networks pictured in (e). The blue curves represent all 50 trials of each network type. The red curve represents the mean diffusion rate of all 50 trials of each network type.

To depict our findings more clearly, we plotted the mean diffusion rates of the trials bounded one standard deviation above and one standard deviation below. The results of experiments (a)-(d), and (e)-(h) are included in Figure 4.5 and Figure 4.6.

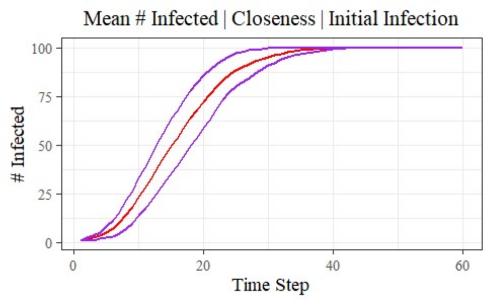
Figure 4.5: Bounded Mean Diffusion Rate of Initial Infection Experiments



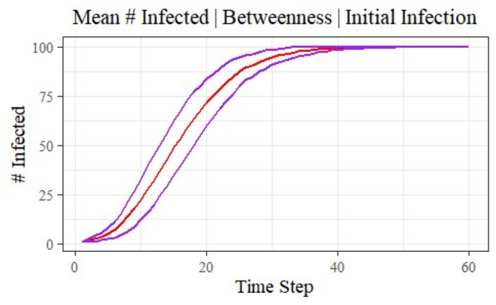
(a)



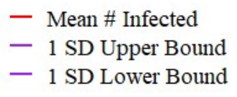
(b)



(c)

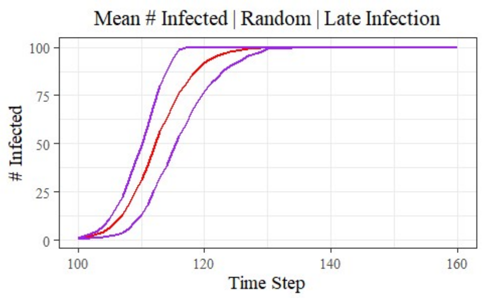


(d)

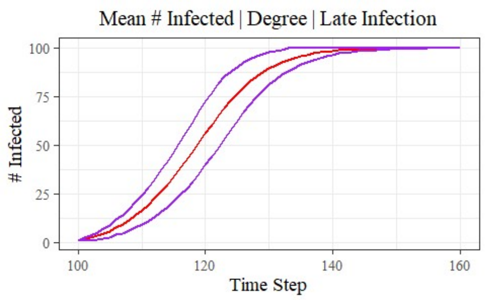


Diffusion occurs slower in networks driven by an affiliation strategy [see (b),(c), and (d)] than in randomly-evolving networks [see (a)]. The red curve represents the mean diffusion rate of all 50 trials of each network type when diffusion is initiated at the first time step. The purple curves represent a 1-standard-deviation bound on the mean diffusion rate.

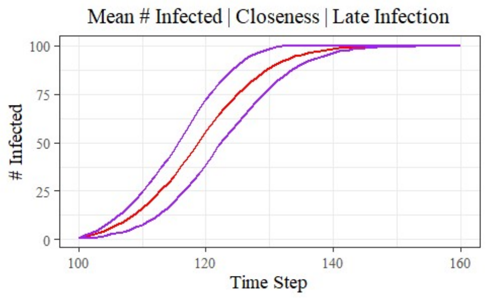
Figure 4.6: Bounded Mean Diffusion Rates of Late Infection Experiments



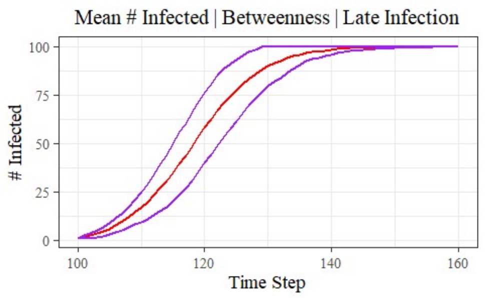
(e)



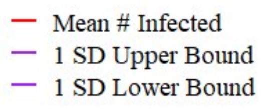
(f)



(g)



(h)



When diffusion is initiated at time step 100, spreading occurs even slower in networks driven by an affiliation strategy [see (f),(g), and (h)] than in randomly-evolving networks [see (e)]. The red curve represents the mean diffusion rate of all 50 trials of each network type. The purple curves represent a 1-standard-deviation bound on the mean diffusion rate.

Chapter 5

Conclusion & Future Directions

Simulations show that diffusion in dynamic networks that develop a set of core nodes occurs more slowly than in indiscriminate (randomly-evolving) networks. The more evolved the core is, the more slowly diffusion occurs. These results support our conjecture that spreading interactions are more likely to occur between nodes that have high indegree values. Nodes with more indegree edges are more likely to become infected, so it takes longer for the infection to diffuse to peripheral nodes. In the randomly-evolving network, edges are likely to become active between any two nodes, resulting in faster diffusion.

In future research, we can count how frequently nodes in the core are receiving spreading interactions in comparison to periphery nodes. These results might lend further support to our explanation.

Perhaps curiously, all three non-random dynamic networks exhibit similar spreading rates despite the variation in characteristics of the core discussed in Chapter 1. Our model may be too limited in scale to capture the effects of the variation in network affiliation on diffusion rates. In preliminary explorations, we have found that varying the number of connections each node keeps up from 1 to 4 does not create discernible variation in the diffusion rates of discriminate dynamic networks. Perhaps by revising our model to include a higher number of total network edges and number of edges each node can keep, we might capture these effects.

Bibliography

[1] Fefferman, N. H., and Ng, K. L. The role of individual choice in the evolution of social complexity. *Annales Zoologici Fennici 44* , 1 (2007), 58–69

[2] Brooks, H. Z., Hohn, M. E., Price, C. R., Radunskaya, A. E., Sindi, S. S., Williams, N. D., et al. (2018). "Mathematical analysis of the impact of social structure on ectoparasite load in allogrooming populations," in Understanding Complex Biological Systems with Mathematics. Association for Women in Mathematics Series, Vol. 14, eds A. Radunskaya, R. Segal, and B. Shtylla (Cham: Springer), 47–61

[3] Wilson, S. N., Sindi, S. S., Brooks, H. Z., Hohn, M. E., Price, C. R., Radunskaya, A. E., Williams, N. D., and Fefferman, N. H. How emergent social patterns in allogrooming combat parasitic infections. Frontiers in Ecology and Evolution 8 (2020), 54

Appendix A

MATLAB Code: Setting Simulation Parameters

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Part 1: Parameters which a for loop will cycle
   through %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% I only recommend changing NUM_TRIALS: the total
   number of simulations to run


NUM_TRIALS               =1;          %# of trials per
   parameter combination
activeedgeProb           = 0.1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Part 2: Parameters for Network Dynamics and Infection
    Initiation %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

runPrefix    = 'Run';


numHosts        = 100;      %# hosts/nodes in graph
```

```matlab
numSteps        = 300;      %# of total steps in simulation
infectionStep = 1;      %step the infection will be introduced
num_neighbors = 5;
num_neighbors_keep = 3;

coreCriteria   = 2;         %Where do you want the infection introduced?
                            % 0 = Periphery Node
                            % 1 = Core Node
                            % 2 = Random Node

freezeStep    = 300;        %When to "freeze" dynamic network:
                            % - Dynamic Network: freezeStep > numSteps
                            % - Static Network: freezeStep = infectionStep

criteria      = 1;          %Centrality criteria:
                            % 0 = Random
                            % 1 = Degree
                            % 2 = Closeness
                            % 3 = Betweenness

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Part 3: Calling the Matlab Code %
%  Do not recommend changing!     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Determine the Network Type
if freezeStep >= numSteps
    networkType = 'Dynamic';
else
    networkType = 'Static';
end

```

```
% Network Centrality Choice
switch criteria
    case 0
        simulationType = 'Random';
    case 1
        simulationType = 'Degree';
    case 2
        simulationType = 'Closeness';
    case 3
        simulationType = 'Betweenness';
end

%Injection Location Choice
switch coreCriteria
    case 0
        injectionLocation = 'Periphery';
    case 1
        injectionLocation = 'Core';
    case 2
        injectionLocation = 'Random';
    end

% Open file that contains the parameter values for each
    run
filePrefix = ['SpreadResults/Network_',networkType,'
    _InjectionSite_',injectionLocation,'_Centrality_',
    simulationType];
trialInfo = [filePrefix,'__runInfo.txt'];
FID = fopen(trialInfo,'a+');
Matrix_Infected_File = sprintf('%s_MatrixInfected.txt',
    filePrefix);
Matrix_Infected = fopen(Matrix_Infected_File,'w');

                MATRIX_NUMBER_INFECTED = [];
                for n = 1:NUM_TRIALS

                    RS1 = randi(2^32);
                    RS2 = randi(2^32);

                    outputPrefix = sprintf('%s_%s_%04d',
                        filePrefix,runPrefix,n);
                    MATRIX_NUMBER_INFECTED(n,:) =
                        simplifiedSpreadingModel(numHosts,
                        numSteps,...
                    infectionStep,freezeStep,criteria,...
                    coreCriteria,RS1,RS2,outputPrefix, ...
                     num_neighbors, num_neighbors_keep,
                         activeedgeProb);

                end
                TimeToFifty=zeros(1,NUM_TRIALS);
                for i = 1:NUM_TRIALS
                    I=find(MATRIX_NUMBER_INFECTED(i,:)==50)
                        ;
                    if isempty(I)
                        TimeToFifty(i)=-1;
                    else
                        TimeToFifty(i)=I(1);
                    end
                end
fprintf(Matrix_Infected, ' %i', MATRIX_NUMBER_INFECTED
    ');

fclose(FID);
fclose(Matrix_Infected);
```

Appendix B

MATLAB Code: Diffusion in Dynamic Networks

```
function MATRIX_NUMBER_INFECTED =
    simplifiedSpreadingModel(numHosts,numSteps,...
      infectionStep,freezeStep,criteria,...
     coreCriteria,randomSeed1,randomSeed2,outputPrefix,
        ...
      num_neighbors, num_neighbors_keep, activeedgeProb)

%close all;
%clear;
warning('off','all')
addpath('./MIT_Code')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BEGIN PARAMETERS FOR THE MODEL TO MODIFY %
%     Change/Modify Values Here to Test      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Number of Hosts
NUM_HOSTS                  = numHosts;

%Number of Neighbors (Outgoing Edges)
NUM_NEIGHBORS              = num_neighbors;
```

```

%Number of Neighbors to Keep/Drop
NUM_NEIGHBORS_KEEP      = num_neighbors_keep;
NUM_NEIGHBORS_DROP      = NUM_NEIGHBORS-
    NUM_NEIGHBORS_KEEP;
MATRIX_NUMBER_INFECTED = [];
%Criteria to Maximize:
% 0 == Random;
% 1 == Degree
% 2 == Closeness;
% 3 == Betweenness;
CRITERIA = criteria;

 if(~(CRITERIA==0 || CRITERIA==1 || CRITERIA == 2 ||
    CRITERIA == 3))
      error('Criteria must be set to a valid choice:
          0,1,2 or 3');
 end

CORECRITERIA = coreCriteria;
 if(~(CORECRITERIA==0 || CORECRITERIA==1 || CORECRITERIA
    ==2))
      error('Core Criteria must be set to a valid choice:
           0,1 or 2');
 end

%Number of iterations (Steps) in the model
NUM_STEPS             = numSteps;

%Where Infection Begins in the model;
INFECTION_STEP        = infectionStep;

%The Step at which we will FREEZE the network dynamics
FREEZE_STEP           = freezeStep;

%If set to 1 will initialize with a cycle
DEBUG = 0;

```

```
%Seed/Set-up the randomStreams
%We want separate randomStreams for paraistes and nodes
    to decouple
%completey the dynamics/repeat if needed;
seedNetwork   = RandStream('mt19937ar','Seed',
   randomSeed1);
seedINFECTION = RandStream('mt19937ar','Seed',
   randomSeed2);

%Create Output File for Debuggind
debugFile = sprintf('%s_debug.txt',outputPrefix);
debugOut = fopen(debugFile,'w');

%Create Output Files:
docFile = sprintf('%s_command.txt',outputPrefix);
docOut  = fopen(docFile,'w');

adjFile = sprintf('%s_adjacency.txt',outputPrefix);
adjOut  = fopen(adjFile,'w');

activeFile = sprintf('%s_active.txt',outputPrefix);
activeOut  = fopen(activeFile,'w');

nodeDegreeFile = sprintf('%s_nodeDegree.txt',
   outputPrefix);
nodeDegreeOut  = fopen(nodeDegreeFile,'w');

nodeClosenessFile = sprintf('%s_nodeCloseness.txt',
   outputPrefix);
nodeClosenessOut  = fopen(nodeClosenessFile,'w');

nodeBetweennessFile = sprintf('%s_nodeBetweenness.txt',
   outputPrefix);
nodeBetweennessOut  = fopen(nodeBetweennessFile,'w');

nodeInfectionFile = sprintf('%s_nodeInfection.txt',
   outputPrefix);
nodeInfectionOut  = fopen(nodeInfectionFile,'w');
```

```

numInfectedFile = sprintf('%s_numInfected.txt',outputPrefix);
numInfectedOut  = fopen(numInfectedFile,'w');

graphFile = sprintf('%s_graph.txt',outputPrefix);
graphOut  = fopen(graphFile,'w');

%Print the Command to the OutputFile:
fprintf(docOut,'%s\n',date);
fprintf(docOut, 'function simplifiedSpreadingModel(numHosts,numSteps,infectionStep,freezestep,criteria,coreCriteria,randomSeed1,randomSeed2,outputPrefix)\n');
fprintf(docOut, 'function simplifiedSpreadingModel(%i,%i,%i,%i,%i,%i,%i,%i,%s)\n',numHosts,numSteps,infectionStep,freezeStep,criteria,coreCriteria,randomSeed1,randomSeed2,outputPrefix);
fprintf(docOut, '\n');

fprintf(docOut, 'Graph File: Iterate, Betweenness, Closeness, Degree.\n');
fprintf(docOut, 'Node File : One file for each metric; Iterate, Node1(Metric), Node2(Metric). etc.\n');
fclose(docOut);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END PARAMETERS FOR THE MODEL TO MODIFY %
%     Do not change values below here!    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step -1: Infection Parameters %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

INITIAL_INFECTION              = zeros(NUM_HOSTS,1); % individuals/node
INITIAL_NUMBER_INFECTED           = 0;
```

```

%Binomial Distribution Probabilities
p                                = activeedgeProb;

%Error Check:
if(p<0)
    error('Must have activeedgeProb >= 0');
end


%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 0: Initial Set-Up %
%%%%%%%%%%%%%%%%%%%%%%%%%%

connected = 0;
numTrials = 1;
MAX_TRIALS = 10;

INITIAL_EDGES              = zeros(NUM_HOSTS,NUM_HOSTS);
INITIAL_UNDIRECTED_EDGES = zeros(NUM_HOSTS,NUM_HOSTS);

while ( connected == 0 && numTrials < MAX_TRIALS)

    %(a)Set up the directed edges;
    %Edges are from node i to node j
    if(DEBUG == 1)
        %Complete Graph
        %INITIAL_EDGES = ones(NUM_HOSTS,NUM_HOSTS);
        %INITIAL_EDGES = INITIAL_EDGES - eye(NUM_HOSTS)
            ;

        %Cycle:
        for i=1:(NUM_HOSTS-1)
            INITIAL_EDGES(i,i+1) = 1;
            INITIAL_UNDIRECTED_EDGES(i,i+1) = 1;
            INITIAL_UNDIRECTED_EDGES(i+1,i) = 1;
        end
        INITIAL_EDGES(NUM_HOSTS,1) = 1;
        INITIAL_UNDIRECTED_EDGES(NUM_HOSTS,1) = 1;
        INITIAL_UNDIRECTED_EDGES(1,NUM_HOSTS) = 1;

        %IMPORTANT TEST CASE: Parallel vertex did not
           have the same
        %                         betweenness.
        %INITIAL_EDGES(1,2) = 1; INITIAL_EDGES(2,3) =
           1;
        %INITIAL_EDGES(3,4) = 1;
        %INITIAL_EDGES(4,5) = 1;
        %INITIAL_EDGES(5,1) = 1;
        %INITIAL_EDGES(1,6) = 1; INITIAL_EDGES(6,3) =
           1;
    else
        for i = 1:NUM_HOSTS
            NEIGHBORS = randperm(seedNetwork,NUM_HOSTS
               -1,NUM_NEIGHBORS);
            for j = 1:length(NEIGHBORS)
                if(NEIGHBORS(j)<i)
                    INITIAL_EDGES(i,NEIGHBORS(j)) = 1;
                    INITIAL_UNDIRECTED_EDGES(i,
                       NEIGHBORS(j)) = 1;
                    INITIAL_UNDIRECTED_EDGES(NEIGHBORS(
                       j),i) = 1;
                else
                    INITIAL_EDGES(i,NEIGHBORS(j)+1) =
                       1;
                    INITIAL_UNDIRECTED_EDGES(i,
                       NEIGHBORS(j)+1) = 1;
                    INITIAL_UNDIRECTED_EDGES(NEIGHBORS(
                       j)+1,i) = 1;
                end
            end
        end
    end

    %(b) Check to make sure connected
    connected = mbiIsConnected(INITIAL_UNDIRECTED_EDGES
        );
    numTrials = numTrials+1;
end

if(numTrials>=MAX_TRIALS)
    error('Failed to Find a Connected Graph');
end

%Step (c): Compute Centrality Metrics

%Compute Node/Graph Degree (in/out)
[INITIAL_DEG, INITIAL_NODE_DEGREE,INITIAL_OUT_DEGREE] =
    degrees(INITIAL_EDGES);
INITIAL_GRAPH_DEGREE                                  =
    mbiGraphDegree(INITIAL_NODE_DEGREE);

%Compte Node/Graph Closeness
INITIAL_NODE_CLOSENESS  = mbiCloseness(
    INITIAL_UNDIRECTED_EDGES);
INITIAL_GRAPH_CLOSENESS = mbiGraphCloseness(
    INITIAL_NODE_CLOSENESS);

%Compute Node/Graph Betweenness
INITIAL_NODE_BETWEENNESS  = node_betweenness_faster(
    INITIAL_UNDIRECTED_EDGES);
INITIAL_GRAPH_BETWEENNESS = mbiGraphBetweenness(
    INITIAL_NODE_BETWEENNESS);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 1: Run the Host Model Iterations with All
    Metrics %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Store the Initial Computations on the Graph
CURR_NODE_DEGREE          = INITIAL_NODE_DEGREE;
CURR_GRAPH_DEGREE         = INITIAL_GRAPH_DEGREE;

CURR_NODE_CLOSENESS       = INITIAL_NODE_CLOSENESS;
```

```
CURR_GRAPH_CLOSENESS           = INITIAL_GRAPH_CLOSENESS;

CURR_NODE_BETWEENNESS          = INITIAL_NODE_BETWEENNESS;
CURR_GRAPH_BETWEENNESS         = INITIAL_GRAPH_BETWEENNESS;

CURRENT_EDGES                  = INITIAL_EDGES;
CURRENT_UNDIRECTED_EDGES       = INITIAL_UNDIRECTED_EDGES;

%Store the Initial Computations on the Graph
CURRENT_INFECTION         = INITIAL_INFECTION;
NEXT_INFECTION           = INITIAL_INFECTION;
PAST_INFECTION            = INITIAL_INFECTION;


%Begin the Iterations/Sampling;
%There are TWO phases to the iterations:
%        Phase 1: Spreading; Phase 2: Network Resample
for iterate = 1:NUM_STEPS
    if mod(iterate,10) == 0
       iterate
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Phase 0: Store a Copy of the Current
       Configuration %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    NEXT_INFECTION       = CURRENT_INFECTION;
     ACTIVE_EDGES = zeros(NUM_HOSTS,NUM_NEIGHBORS);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Phase 1: SPREADING %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (iterate>= INFECTION_STEP)
       infection_step = iterate;

       if(iterate == INFECTION_STEP)

           %RANDOM
           if(CRITERIA == 0)
```

```
            I            = randperm(seedINFECTION,
                NUM_HOSTS);
        %DEGREE
        elseif(CRITERIA == 1)
            [S,I]        = sort(CURR_NODE_DEGREE, '
                descend');
        %CLOSENESS
        elseif(CRITERIA == 2)
            [S,I]        = sort(CURR_NODE_CLOSENESS, '
                descend');
        %BETWEENNESS
        elseif(CRITERIA == 3)
            [S,I]        = sort(CURR_NODE_BETWEENNESS, '
                descend');
        end

        %(0) Periphery: Pick infection in the
            periphery
        if(CORECRITERIA == 0)
            patient0   = I(randi(seedINFECTION,[
                NUM_NEIGHBORS_KEEP+1,NUM_HOSTS]));
        %(1) Core: Pick infection in the core
        elseif(CORECRITERIA == 1)
            patient0   = I(randi(seedINFECTION,
                NUM_NEIGHBORS_KEEP));
        %(2) Random: Pick a Random host.
        elseif(CORECRITERIA == 2)
            patient0   = randi(seedINFECTION,
                NUM_HOSTS);
        end
        NEXT_INFECTION(patient0) = 1;
    else
        PAST_INFECTION = CURRENT_INFECTION;

        %Spreading the infection

        for i = 1:NUM_HOSTS
        for j = 1:NUM_HOSTS
```

```
                    counter = 0;
                    if(CURRENT_EDGES(i,j)>0)
                       counter = counter+1;
                     if(rand(seedINFECTION) < p)
                         ACTIVE_EDGES(i,counter)=1;
                         if (CURRENT_INFECTION(i)==1)
                             NEXT_INFECTION(j)=1;
                         end
                     end
                    end
                end
            end
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Phase 2: Network Resampling %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    nextConnected = 0;
    numTrials = 1;
    MAX_TRIALS = 10;

    while ( nextConnected == 0 && numTrials <
       MAX_TRIALS)
        NEXT_EDGES             = CURRENT_EDGES;
        NEXT_UNDIRECTED_EDGES =
            CURRENT_UNDIRECTED_EDGES;

        if(iterate>=FREEZE_STEP)
            %Do nothing we have frozen the network
               resampling.
            freeze_iterate = iterate;
            freeze_iterate;
        else

            for i = 1:NUM_HOSTS
                %RANDOM
                if(CRITERIA == 0)
            [S,I] = sort(CURRENT_EDGES(i,:).*
               rand(seedNetwork,1,NUM_HOSTS),'
               descend');
        %DEGREE
        elseif(CRITERIA == 1)
            [S,I] = sort(CURRENT_EDGES(i,:).*
               CURR_NODE_DEGREE,'descend');
        %CLOSENESS
        elseif(CRITERIA == 2)
            eps = .1;
            [S,I] = sort(CURRENT_EDGES(i,:).*(
               CURR_NODE_CLOSENESS+eps*ones(
               size(CURR_NODE_CLOSENESS))  )','
               descend');
            %[S,I] = sort(CURRENT_EDGES(i,:).*(
               CURR_NODE_CLOSENESS)','descend')
               ;

        %BETWEENNESS
        elseif(CRITERIA == 3)
            eps = .1;
            [S,I] = sort(CURRENT_EDGES(i,:).*(
               CURR_NODE_BETWEENNESS+eps*ones(
               size(CURR_NODE_BETWEENNESS))  ),'
               descend');
            %[S,I] = sort(CURRENT_EDGES(i,:).*(
               CURR_NODE_BETWEENNESS),'descend
               ');

        else
            CRITERIA
            exit('Should not be here. Invalid
               criteria value.');
        end

        %SortNodes by Degree: 3 Categories
        %NUM_NEIGHBORS_KEEP = 3;
        %NUM_NEIGHBORS = 5;
```

```
            keepNodes    = I(1:NUM_NEIGHBORS_KEEP);
            deleteNodes  = I(NUM_NEIGHBORS_KEEP+1:NUM_NEIGHBORS);
            sampleNodes  = I(NUM_NEIGHBORS+1:NUM_HOSTS);

            %Delete the host itself so we do not get self edges.
            sampleNodes(sampleNodes==i)=[];

            %Determine New Neighbors:
            newIndex      = randperm(seedNetwork, length(sampleNodes), NUM_NEIGHBORS_DROP);
            newNeighbors = zeros(1, NUM_NEIGHBORS_DROP);
            for j = 1:NUM_NEIGHBORS_DROP
                newNeighbors(j) = sampleNodes(newIndex(j));
            end

            %Drop Previous Neighbors and Add New Neighbors
            for j = 1:NUM_NEIGHBORS_DROP
                if( NEXT_EDGES(i,deleteNodes(j)) == 1)
                    NEXT_EDGES(i,deleteNodes(j)) = 0;
                    NEXT_UNDIRECTED_EDGES(i,deleteNodes(j))  = 0;
                    NEXT_UNDIRECTED_EDGES(deleteNodes(j),i) = 0;
                else
                    deleteNodes(j)
                    exit('Error: Trying to Delete an Edge that is not there!');
                end
```

```

                        if( NEXT_EDGES(i,newNeighbors(j))
                          == 0)
                            NEXT_EDGES(i,newNeighbors(j)) =
                                1;
                            NEXT_UNDIRECTED_EDGES(i,
                              newNeighbors(j)) = 1;
                            NEXT_UNDIRECTED_EDGES(
                              newNeighbors(j),i) = 1;
                        else
                            exit('Error: Trying to Add an
                              Edge that already exists!');
                        end
                    end
                end
            end

            nextConnected = mbiIsConnected(
              NEXT_UNDIRECTED_EDGES);
%             if(nextConnected==0)
%                 error('We disconnected the graph');
%             end
            numTrials = numTrials+1;

        end

        if(numTrials>3)
            numTrials
        end

        if(numTrials > MAX_TRIALS)
            exit('Error in Graph Iteration! Could not
              create connected graph w/in the maximum
              number of iterations!');
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Phase 3: Store Configuration and Recompute and
```

```
    Store Metrics %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                CURRENT_EDGES = NEXT_EDGES;
CURRENT_UNDIRECTED_EDGES = NEXT_UNDIRECTED_EDGES;
CURRENT_INFECTION        = NEXT_INFECTION;

%Compute Node/Graph Degree (in/out)
%CURRENT_EDGES -> Directed Graphs
 [DEG, CURR_NODE_DEGREE,OUT_DEGREE] = degrees(
    CURRENT_EDGES);
CURR_GRAPH_DEGREE                         = mbiGraphDegree
    (CURR_NODE_DEGREE);

%for p=1:NUM_HOSTS
%       if(CURRENT_EDGES(p,p) == 1)
%             CURRENT_EDGES(p,p)
%             iterate
%       end
%end

 if(max(CURR_NODE_DEGREE)>=50)
      CURR_NODE_DEGREE

      CURRENT_EDGES

end
%for p=1:length(OUT_DEGREE)
%      if(OUT_DEGREE(p)!=5)
%            OUT_DEGREE(p)
%      end
%end

%Compte Node/Graph Closeness
CURR_NODE_CLOSENESS   = mbiCloseness(
    CURRENT_UNDIRECTED_EDGES);
CURR_GRAPH_CLOSENESS = mbiGraphCloseness(
    CURR_NODE_CLOSENESS);

%Compute Node/Graph Betweenness
%Commenting out Betweenness because it was causing
    problems!
CURR_NODE_BETWEENNESS  = node_betweenness_faster(
    CURRENT_UNDIRECTED_EDGES);
CURR_GRAPH_BETWEENNESS = mbiGraphBetweenness(
    CURR_NODE_BETWEENNESS);

CURRENT_NUMBER_INFECTED = sum(CURRENT_INFECTION);
MATRIX_NUMBER_INFECTED = [MATRIX_NUMBER_INFECTED,
    CURRENT_NUMBER_INFECTED];
%%%%%%%%%%%%%%%%%
% Print Output %
%%%%%%%%%%%%%%%%%

%Print Graph Information:
 fprintf(graphOut,'%i',iterate);
 fprintf(graphOut,' %.10f %.10f %.10f',
    CURR_GRAPH_BETWEENNESS,CURR_GRAPH_CLOSENESS,
    CURR_GRAPH_DEGREE);
 fprintf(graphOut,'\n');
 fprintf(numInfectedOut, ' %i',
    CURRENT_NUMBER_INFECTED);
%Print Node Information:
 fprintf(nodeDegreeOut, '%i',iterate);
 fprintf(nodeClosenessOut, '%i',iterate);
 fprintf(nodeBetweennessOut,'%i',iterate);
 fprintf(adjOut, '%i', iterate);
 fprintf(activeOut, '%i', iterate);
 for i = 1:NUM_HOSTS
     fprintf(nodeDegreeOut, ' %i',CURR_NODE_DEGREE(i
        ));
     fprintf(nodeClosenessOut, ' %.10f',
        CURR_NODE_CLOSENESS(i));
     fprintf(nodeBetweennessOut, ' %.10f',
        CURR_NODE_BETWEENNESS(i));
     fprintf(nodeInfectionOut,' %.10f',
            CURRENT_INFECTION(i));
        fprintf(adjOut,' %s',vec2str(find(CURRENT_EDGES
            (i,:)),[],[],0));
        fprintf(activeOut,' %s',vec2str(ACTIVE_EDGES(i
            ,:) ,[],[],0));
    end
    fprintf(nodeDegreeOut,'\n');
    fprintf(nodeClosenessOut, '\n');
    fprintf(nodeBetweennessOut, '\n');
    fprintf(nodeInfectionOut,'\n');
    fprintf(adjOut,'\n');
     fprintf(activeOut,'\n');
end
%%%%%%%%%%%%%%%%%%%%%%
% Close Output Files %
%%%%%%%%%%%%%%%%%%%%%%

fclose(nodeDegreeOut);
fclose(nodeClosenessOut);
fclose(nodeBetweennessOut);
fclose(nodeInfectionOut);
fclose(graphOut);
fclose(adjOut);
fclose(activeOut);
fclose(numInfectedOut);
end
```

Appendix C

MATLAB Code: Visualizing Networks

```
clear; close all;
% Networks are plotted with node colors to represent
%infections at each time step. The node sizes represent
%the degree (in-degree + out-degree) of each node.

%% Load in adjacency and infection data txt file
adjData=load('SpreadResults/
    Network_Dynamic_InjectionSite_Random_Centrality_Degree_Run_0001_adjac
    .txt');
infectionData=load('SpreadResults/
    Network_Dynamic_InjectionSite_Random_Centrality_Degree_Run_0001_node
    .txt');
activeData=load('SpreadResults/
    Network_Dynamic_InjectionSite_Random_Centrality_Degree_Run_0001_activ
    .txt');
%% Setup
%remove the first column which contains the time
%(since the row encodes the time data anyway)
adjData = adjData(:,2:end);
infectionData = infectionData(:,1:end);
activeData = activeData(:,2:end);

```

```
firstTime=1; %where to start showing video
maxTime=length(adjData(:,1)); %number of time steps
N=100; %number of nodes
k=5; %number grooming connections
stepSize=10; %so we don't need to see all of them..

%put adjacency data into cells; edges{i,j} contains the
%edges for node j at time i
edges=mat2cell(adjData,1*ones(maxTime,1),k*ones(1,N));


h = figure;
filename = 'infectionAnimated2.gif'; %uncomment to save
    gif with this filename
%% Create and plot network
vid1 = VideoWriter('network.mp4', 'MPEG-4');
open(vid1);
for i=firstTime:stepSize:102
    clear G
    G=digraph;
    G=addnode(G,N);

    %add edges to build graph
    for j=1:N
        G=addedge(G,j*ones(1,k),edges{i,j});
    end

    degree=indegree(G)+outdegree(G); %calculate total
       degree of each node

    p=plot(G); %make network plot
    p.NodeCData = infectionData(i,:); %color nodes to
    p.EdgeCData = activeData(i,:);
    p.MarkerSize = degree; %make size of nodes
       proportional to degree
    set(gcf, 'Units', 'Normalized', 'OuterPosition', [0
        0 1 1]); %make figure very big
    title(['\fontsize{20}t=',num2str(i)]);
    myColorMap = jet(256);
    myColorMap(1,:) = .8; %nodes that are uninfected
       are gray
    colormap(myColorMap);
    colorbar
    caxis([0 max(max(infectionData))])
    drawnow


%     % Capture the plot as an image (if want to save)
%       frame = getframe(h);
%       im = frame2im(frame);
%       [imind,cm] = rgb2ind(im,256);
%       % Write to the GIF File
%       if i == firstTime
%           imwrite(imind,cm,filename,'gif', 'Loopcount
    ',inf);
%       else
%           imwrite(imind,cm,filename,'gif','WriteMode
    ','append');
%       end
currFrame = getframe(h);
writeVideo(vid1,currFrame);
end
close(vid1)
```