A HIDDEN MARKOV MODEL FOR DESCRIBING THE STATISTICAL EVOLUTION OF SOCIAL GROUPS OVER COMMUNICATION NETWORKS.

By

David Siebecker

A Thesis Submitted to the Graduate Faculty of Rensselaer Polytechnic Institute in Partial Fulfillment of the Requirements for the Degree of MASTER OF COMPUTER SCIENCE

Approved:

__________________________
Malik Magdon-Ismail
Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

July 2003
(For Graduation August 2003)
CONTENTS

LIST OF TABLES ........................................ iv
LIST OF FIGURES ........................................ v
ABSTRACT ............................................. vi
1. Overview ........................................... 1
2. The Model .......................................... 3
   2.1 State Transitions .................................. 3
   2.2 Evolution of Energy ................................ 4
   2.3 Evolution of Group Membership .................. 7
      2.3.1 Classification Discussion .................... 7
      2.3.2 Joining a Group ............................... 8
         2.3.2.1 Determining Domain of Groups to Consider 8
         2.3.2.2 Calculating Relative Probability of Joining Groups 9
         2.3.2.3 Attempting to Join a Specific group ......... 10
      2.3.3 Leaving A Group ............................. 11
      2.3.4 Changing Groups ............................. 11
3. Communications ................................... 13
   3.1 Category One Methods of Communicating .......... 13
   3.2 Category Two Methods of Communicating .......... 14
   3.3 Uses for Communications .......................... 14
   3.4 Building Applications ............................ 15
4. Finding Groups Given Communications .................. 16
   4.1 Kernighan-Lin .................................... 16
   4.2 Maximizing the Complement ...................... 17
   4.3 Finishing ........................................ 17
5. Learning ............................................. 18
   5.1 Learning Initial Energy, Energy per Flake and Threshold Values . . 19
   5.2 Learning Node Classifications .................... 19
      5.2.1 Collecting Data for Experimentation ........ 20
LIST OF TABLES

7.1 Results of Learning on a 1600 Node Contagion Network. . . . . . . . . . 39
7.2 Results of Learning on a 10,000 Node Contagion Network. . . . . . . . 41
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Convergence Error under Various Classifications in the alt.movies Newsgroup</td>
<td>26</td>
</tr>
<tr>
<td>6.2</td>
<td>Convergence Error under Various Classifications in the alt.revisionism Newsgroup</td>
<td>27</td>
</tr>
<tr>
<td>6.3</td>
<td>Comparison of Performance with Various Classifications on the alt.movies Newsgroup</td>
<td>28</td>
</tr>
<tr>
<td>6.4</td>
<td>Comparison of Performance with Various Classifications on the alt.revisionism Newsgroup</td>
<td>29</td>
</tr>
<tr>
<td>6.5</td>
<td>Comparison of Classification Schemes used with Learned Classifications on alt.movies</td>
<td>30</td>
</tr>
<tr>
<td>6.6</td>
<td>Comparison of Classification Schemes used with Learned Classifications on alt.revisionism</td>
<td>31</td>
</tr>
<tr>
<td>6.7</td>
<td>Distribution of Learned Classifications on alt.movies.</td>
<td>32</td>
</tr>
<tr>
<td>6.8</td>
<td>Distribution of Learned Classifications on alt.revisionism.</td>
<td>33</td>
</tr>
<tr>
<td>7.1</td>
<td>Finite state machine describing the contagion model.</td>
<td>36</td>
</tr>
<tr>
<td>7.2</td>
<td>An example of the network over which the contagion was modeled.</td>
<td>38</td>
</tr>
<tr>
<td>7.3</td>
<td>State Map at time 0 and time 50</td>
<td>40</td>
</tr>
<tr>
<td>7.4</td>
<td>State Map at time 100, Actual and Learned Probabilities</td>
<td>40</td>
</tr>
<tr>
<td>7.5</td>
<td>States and Observations Illustrating Impossible State Transition</td>
<td>43</td>
</tr>
</tbody>
</table>
ABSTRACT

The goal of this thesis is the development of an abstract statistical model of the evolution for social networks (communities). Our approach is to specify a set of general models that govern the behavior on a micro-scale level (the micro-laws of the model) and observe the resulting emergent macro-scale behavior. The models are general enough to accommodate established theories of social networks, but at the same time flexible enough to accommodate many more possibilities, so as to be applicable to the analysis of email networks, corporate networks, communications networks, contagion networks and social networks, including human networks (for example terrorist networks). We have developed techniques to extract the structure of such networks from real data, infer the micro-scale behavior that would govern the observed data and use these derived micro-laws for analysis and prediction. We model social groups from the probabilistic actions of a node rather than from the probabilistic network point of view. Thus our group is a based upon the more general concept of a set of nodes rather than a specific structure of nodes and can be used to model structure as well as usage. For example we may have a particular “security” infrastructure distributable across a road network, for example, customs agents. By also modeling the “usage” of this road network, we can then efficiently distribute our security infrastructure (agents) appropriately. Our models allow for both modeling the physical network structure and the usage network structure through the general set/group concept.
CHAPTER 1
Overview

We would like to explain and predict the macro behaviors, i.e. the observable quantities, of social groups. Examples of such quantities are the average size of a social community, the distribution of their sizes, the average number of communities, the stability of the communities (for example length of membership), and the evolution of those parameters in time. Our main goal is to find explanations of such macro behaviors in terms of properties attributable to the individual members themselves, which we call micro-laws. Although the micro-laws may not be possible to observe directly, should they explain the macro behaviors, they can be interpreted as the confirmation of the micro-laws. In this case, the micro-laws can be further used to predict the behavior of communities. Examples of such predictions could be that the society will eventually become a single community, several large communities or stabilize into large communities and smaller ones with continual movement between them.

We begin by making (reasonable) assumptions for the micro-laws, or rather for the behavior of the individuals. These laws determine (probabilistically) whether at any given time a particular individual is in a community (or set of communities) and how the individual determines which communities to join. Knowing these laws, we are then able to simulate the behavior of the entire population and the macro behaviors should then emerge naturally as a consequence of stabilization to statistical equilibrium.

Since we do not claim to know how individuals in a society behave, we instead specify functional forms for the nature of this behavior. These functional forms were picked because they appeared intuitive, satisfying certain intuitive properties such as “if you are a member of a community today, it is more likely that you will be a member of that community tomorrow.” These functional forms thus contain some unspecified parameters governing the detailed dynamics. A different choice for these parameters leads to different macro behavior, and in fact, we have observed phase
transitions where the qualitative behavior abruptly changes when a single parameter is changed very slightly. The development of a large portion of our model has its basis in the work of Monge and Contractor’s recent book [10]. Here the most widely accepted communication theories are presented as well as how these theories are used to model social networks.

The ultimate goal is that most societies can be described adequately by some assignment to the parameters. Thus we can observe the behavior of a society and try to determine what values of these parameters are consistent with the observations. Having determined the parameters, we have a model of the society, which can give insight into the behavior and perhaps even lead to predictions on the future evolution of the society. In addition to being able to model most societies the model must also be able to validate current theories in social science [5, 13, 10].
CHAPTER 2

The Model

A basis for modeling the evolution of groups was needed and we chose a Markovian model as the foundation. At each time-step the model is in a certain state, $S_t$, and at the next time-step the state transitions to a new state, $S_{t+1}$. $S_{t+1}$ is solely dependent upon $S_t$, making it a Markov model. When taking the model which produces an evolution of groups and adding to it the ability to produce communications from those groups, the model becomes a hidden Markov model (HMM). The reason for doing this is because in reality all that can be observed is the communications that the underlying groups, which are not known, are producing, thus the groups are hidden and a HMM is appropriate. For reference, when model is referred to without qualifiers in this document it is taken to mean the model describing the state transitions and not the entire HMM.

2.1 State Transitions

For the state transition between time-steps we shall define the following: $e^t_i$ is the amount of energy that node $i$ has at time $t$. $c_i$ is a real number that specifies a group size preference also known as type or classification. Negative numbers signify a liking for small groups, while positive numbers indicate that $i$ prefers large groups. A $c_i$ close to zero means that the node likes to stay in medium size groups. Also defined is $\tau^j_i$ to be a boolean value representing node $i$’s membership in group $j$ at time $t$. Thus the entire state of the world at time $t$ can be defined as

$$S_t = \{e^t_1, c_1, \{\tau^1_i, \ldots, \tau^j_i\}\}_{i=1}^N.$$  

Formally this is the energy of node $i$, node $i$’s type and node $i$’s membership in groups $1, \ldots, k$ for all nodes $i=1, \ldots, N$ all at time $t$.

Thus to define the state transition we need only describe how $e$ and $\tau$ evolve over the course of a time-step. The evolution of the two is in fact related as will
be shown later in sections 2.2, 2.3. First we shall describe the evolution of $e_i$, the energy level of a node.

2.2 Evolution of Energy

What exactly is the energy of a node? Simply put, energy is a measure of how many groups a node can be a member of and to what extent that node can participate in those groups. Each group that a node joins takes up some of the node’s energy. The more energy spent on current groups, the less likely a node is to want to join a new group. In this respect $c_i$ helps govern a node’s actions. The specific formula allow that a node can spend more energy than it has, and in this case it is likely that the node will want to leave one of the groups of which it is a member. As the simulation runs, the longer a node is in a group the more energy the node spends to be part of that group, the idea being that the node is becoming more involved and might be taking on more responsibility. In terms of the simulation, the energy level is a real number, such that $e_i \in [0,1]$.

With each time-step a node’s energy pushes the node to perform one of three things with respect to $|F_i^t| = \tau_1^t, \ldots, \tau_k^t|$: stay the same, increase or decrease. In response to the node’s action and its predicted action the node’s energy will change. How much the energy level changes is not as important as determining in which direction it changes. The change in energy is based on what action a node takes as well as what action the node was most likely to take. Node $i$’s action at time $t$ ($a_i^t$) is one of three choices, that of keeping the number of groups that $i$ is in, $|F_i^t|$, constant, increasing $|F_i^t|$ by one or decreasing $|F_i^t|$ by one. $F_i^t$ is defined as being the set of groups that node $i$ is in prior to $a_i^t$. This does not mean that $F_i$ is the same set from $t$ to $t+1$, merely that the size of it stays the same, goes up or decreases.

There is a likelihood of each action being chosen that is determined by the node $i$’s surplus energy. The surplus energy ($\epsilon_i^t$) is calculated as,

$$\epsilon_i^t = e_i^t - \gamma * |F_i^t| - \sigma t_k * \beta$$

where $\gamma$ and $\beta$ are constants. Values used in our experiments are 0.05 and 0.1
respectively.

$\epsilon_i^t$ is then used to calculate three values $p_+, p_0, p_-$ (relative likelihood of joining, staying or leaving). $p_+, p_0, p_-$ are normalized so their sum is 1. A random number [0,1] is chosen to represent $a_i^t$. The most likely action, ($P_i^t$), is the greatest of $p_+, p_0, p_-$. The specific formula for $p_+, p_0, p_-$ before normalization are based on a threshold value $\psi$. This value is used to control how much surplus energy is needed before the node becomes highly likely to join a new group, or conversely, if there is a surfeit of energy, to make it highly likely that the node leaves a group.

$$
p_+ = \frac{A_+ \epsilon_i^t}{(1+e^{-\rho_+ \frac{\epsilon_i^t}{\psi}})};
$$

$$
p_0 = A_0 \epsilon_i^t e^{-\rho_0 \frac{\epsilon_i^t}{\psi}};
$$

$$
p_- = \frac{A_- \epsilon_i^t}{1+e^{\rho_- \frac{\epsilon_i^t}{\psi}}};
$$

Now the direction of the energy change can be calculated. Based on $P_i^t$ and $a_i^t$ we can define $F(\epsilon_i^t, P_i^t, a_i^t)$ where $F$ returns the amount of change in energy. In general $F$ has the following characteristics where a value of 1 (0,-1) signifies a join (stay, leave resp.) action:

1. $F(x, 1, 1) > 0$;
2. $F(x, 1, 0) < 0$;
3. $F(x, 1, -1) << 0$;
4. $F(x, 0, 1) > 0$;
5. $F(x, 0, 0) = 0$;
6. $F(x, 0, 1) < 0$;
7. $F(x, -1, 1) >> 0$;
8. $F(x, -1, 0) > 0$;
9. $F(x, -1, -1) < 0$;
The intuition behind $F$ acting in this manner is that when a node adds another group to $F^t_i$ that it should get a boost of energy because it is excited that it successfully joined. If a node removes a group from $F^t_i$ it will lose some energy for the reason that though the node is freeing up a great deal of energy it also does not need as much as it did before and some of the surplus energy disappears. In general if a node does nothing its energy should remain about the same.

The only thing left to determine is specifically how much the energy should change. If a node was expected to join a new group, and it does, it only gets little additional energy, due to the fact that the node did as was expected. On the other hand, if the node leaves when it was expected to join, it signifies that the node does not have a need for all of its access energy and that it should lose a significant amount of it. Additionally, if the node was expected to join and it does nothing, it loses a little energy do to the fact that even though it perform the expected join action, not leaving a group is not as bad as leaving one. The inverse of the above applies for when a node is expected to leave a group.

The exact values returned by $F$ depend on four functions $\lambda(x)$, $\lambda_0(x)$, $\mu(x)$ and $\mu_0(x)$. $\lambda$ is the reward function and $\mu$ is the penalty function. If the energy is going to increase $\lambda$ is used and if energy is going to decrease $\mu$ is used. $\lambda_0$ and $\mu_0$ are used for the lesser rewards and penalties, and are generally defined to be one half the value of $\lambda$ and $\mu$. $F$ also works with a system of diminishing rewards, in that if a node has a lot of energy to begin with and its energy increases, it will only increase by a little amount. The converse, that if $e^t_i$ is small and is going to be reduced, that it is only reduced a small amount holds true as well. Another reason for the diminishing rewards is to enforce that $e^t_i \in [0, 1]$.

1. $F(x, 1, 1) = x + \lambda(x) \ast (1 - x)$;
2. $F(x, 1, 0) = x + \mu_0(x) \ast x$;
3. $F(x, 1, -1) = x + \mu(x) \ast x$;
4. $F(x, 0, 1) >= x + \lambda_0(x) \ast (1 - x)$;
5. $F(x, 0, 0) = 0$;
6. \( F(x, 0, 1) = x + \mu_0(x) \times x \);  
7. \( F(x, -1, 1) = x + \lambda(x) \times (1 - x) \);  
8. \( F(x, -1, 0) = x + \lambda_0(x) \times (1 - x) \);  
9. \( F(x, -1, -1) = x + \mu(x) \times x \);

The choice of \( \lambda \) and \( \mu \) are user configurable, though we have a few options built into the system that provide interesting behavior. Typically used are

\[
\begin{align*}
\mu(x) &= 0.35(0.01 \times x + (1.99 \times x)^{1.05}) \\
\mu(x) &= 0.35(x^2 + x^{1.00}) \\
\mu(x) &= 0.5(x + e^{-x})
\end{align*}
\]

with \( \lambda(x) \) being either valued at either \( \lambda(x) = 1 - \mu(x) \) or \( \lambda(x) = (1 - \mu(x))/2 \).

### 2.3 Evolution of Group Membership

The set of groups that node \( i \) is a member of at time \( t \), \( F_i^t \), changes in two fundamental ways with each time-step. The first manner in which it changes is according to \( a_i^t \), node \( i \)'s action at time \( t \). If \( i \) is going to be joining a new group, it needs to select the group which it is going to attempt to join. Note that it is an attempt to join, as a group can refuse membership to \( i \) if \( i \) has bad qualifications. The qualification of a node, \( q_i^t \), shall be defined later.

The other manner in which \( F_i^t \) changes is that for each time step \( i \) is allowed to attempt to change groups. For each group \( j \in F_i^t \) \( i \) can decide that it would like to attempt to leave group \( j \) and join a new group \( k \). If \( k \) refuses \( i \)'s membership then \( i \) will retain its membership in \( j \). The process of attempting to join a group is the same for both methods of changing \( F_i^t \).

#### 2.3.1 Classification Discussion

Before going more in-depth into the evolution of group membership the classification attribute of the nodes needs to be clarified. Initially the classification
attribute was used to determine if a node was a leader \( (c'_l = 1) \), a socialite \( (c'_s = 3) \) or follower \( (c'_f = 3) \), where leaders prefer small groups, socialites medium groups and followers large groups. Over time the simulation progressed to the point where size preference was a real so that static size preferences were not necessary as they were a limitation to the model. However, some of the formula are based on these static size preferences, though they have been modified to that in general they are not limited by them.

In most cases the formula have been modified to be based on some linear combination of the original static preferences that is then a function of the real classification. Additionally a method was needed to map \( c_i \) to the static classifications \( (c'_i) \). An intermediate variable \( \delta_i \) is introduced to scale \( c_i \) to the range of \([0,1]\) and then \( \delta_i \) can be mapped to the static classifications. If \( \delta_i > 0.5 \) the node is considered to be follower. If \( \delta_i < -0.5 \) the node is considered to be leader. In all other cases the node is a socialite. \( \delta = \frac{(e^{ci} - e^{-ci})}{(e^{ci} + e^{-ci})} \)

2.3.2 Joining a Group

The procedure for joining a group is as follows:

1. Determine Domain of Groups to Consider

2. Calculate Relative Probability of Joining Each Group in Domain

3. Normalize Probabilities and Select

4. Attempt to Join a Specific Group

2.3.2.1 Determining Domain of Groups to Consider

There are two ways of selecting the domain of groups to join, \( D_F \). The first is to consider all the groups of which node \( i \) has knowledge, \( K'_i \). The other is to pick randomly from the entire set of existing groups \( FZ' \).

\( K'_i \) is defined as being all groups of which nodes that \( i \) is in contact with are members. Nodes \( i, l \) are in contact if and only if \( i \in j \) and \( l \in j \), written \( C_{il} \). Thus \( K'_i \) can be formally defined as the set of groups such that \( K'_i = \{ j : l \in j, i \notin j, C_{il} \} \), where \( j \) is a group and \( l \) is a node that \( i \) is in contact with.
The matter of choosing which domain to use is determined by $|K^t_i|$. A random number, $R \in [0, 1]$, is chosen and if $R < 1/|K^t_i|$ then $FZ^t$ is chosen. Otherwise the $K^t_i$ is chosen to be the set under consideration. Because $FZ^t$ can be quite large, it is pared down to be a much more manageable size before a specific group is chosen. We define a constant $P \in [0, 1]$ that is the percentage of $FZ^t$ that we want to consider. From $FZ^t$ we choose $|FZ^t| \cdot P$ groups to form $FZ^t_m$ a modified set of groups from which to consider joining. The reasoning behind this is that it is unrealistic for $i$ to know about every group in $FZ^t$ and thus let $i$ know about a subset of $FZ^t$. In the experiments $P = .15$.

The logic behind using $K^t_i$ is that $i$ is more likely to want to join a group that someone he knows is in. The formula for determining which set to choose is based on the fact that if $K^t_i$ is small, $i$ is likely to want a broader variety of groups to consider than $K^t_i$ allows for. Also note, that when $D_F = FZ^t_m$ that we add to $D_F$ the chance that $i$ will want to create a new group.

2.3.2.2 Calculating Relative Probability of Joining Groups

The next step in figuring out which group to attempt to join is to figure out the relative probability of $i$ wanting to join $j : j \in D_F$. Each $j : j \in D_F$ is assigned a probability $p^i_r(j)$ which is the relative likelihood that $i$ will want to join $j$. It is calculated as being the product of $i$’s affinity for the size of $j$ $^1 (gg(|j|))$, how likely $i$ is going to want to join $j$ based on $i$’s qualifications and the average qualification of $j$ $^2$, $Aff_i(j)$, the affinity of $i$ for $j$.

\[
\begin{align*}
gg_i(x) &= gg'(c's, x) + (gg'(c'f, x) - gg'(c'l, x))/2 * m_{calla} + (gg'(c'l, x) + gg'(c'f, x) - 2 * gg'(c's, x))/2 * m_{calla} \\

& \quad (gg'(c'l, x) + gg'(c'f, x) - 2 * gg'(c's, x))/2 * m_{calla} \\

gg'(w, x) &= (1/(\Gamma((w + 1) * \theta^w))) * (x + 2)^{w-1} * e^{-(x+2)/\theta} \\

Aff_i(j) &= desD + (1 - desD) * \\
\end{align*}
\]

$^1$This is based on the classification of $i$, $c_i$. Leaders like small groups, socialites like medium sized groups and followers like large groups.

$^2$Nodes want to join groups with high qualifications in general, though nodes with very high qualifications are very unlikely to want to join a group with low qualifications.
(1 + \tanh(desA * (q_j - q_i)/q_i)/2)

where \( \theta \), \( desD \) and \( desA \) are constants, currently valued at 4, .9, .75 resp. \( c_i \) is the classification of node \( i \) and takes on the value of 1 (3 5) for leaders (socialites, followers resp.). Note that for the purposes of creating a new group, a new group is considered to be one of size 0 and to have 0 qualification. Also note that \( gg_i(j) \) depends on a \( \Gamma \) function.

So the relative probability of \( i \) wanting to join \( j \) can be written as \( p_i^r(j) = gg_i(|j|) \cdot Aff_i(j) \). Once \( p_i^r(j) \) has been calculated for all \( j \in D_F \) we sum the relative probabilities and normalize them to obtain \( p_n^i(j) \). Using these normalized probabilities we now choose a group by obtaining a random number \( R \in [0,1] \). The resulting group \( j_s \) that is selected is the group that \( i \) will attempt to join.

2.3.2.3 Attempting to Join a Specific group

Now that we have \( j_s \), node \( i \) is going to attempt to join it. Remember that a group can refuse membership to a node. A group is likely to refuse membership to a node with a low qualification compared to the group’s qualification. At this point it is useful to define how both \( q^t_i \) and \( q^t_j \) are calculated.

\[
q^t_i = 1/|F^t_i| \cdot \Sigma r^i_j : j \in F^t_i
\]

\[
q^t_j = 1/|j| \cdot \Sigma q_i : i \in j
\]

Thus \( q^t_i \) is the average rank of \( i \) across all of the groups of which it is a member or ever was a member. \( q^t_j \) is the average qualification of all of the members of \( j \). \( r^j_i \) is \( i \)'s rank in \( j \) and is calculated as being \( 2 - \delta_i \) multiplied by the amount of time that \( i \) has been a member of \( j \).

Using \( q_j \) and \( q_i \) we can calculate \( Aff_j(i) \) the affinity of group \( j \) for node \( i \) to be

\[
Aff_j(i) = desD + (1 - desD) \cdot (1 + \tanh(desA * (q_j - q_i)/q_i)/2)
\]

To determine if \( j \) will refuse membership to \( i \) we pick a random number \( R \in [0,1] \)

\(^3\)For groups that \( i \) has left \( r^j_i \) is calculated based on the rank of \( i \) when it left \( j \)
and if \( R < \text{Aff}_j(i) \) node \( i \) has been allowed to join group \( j \). Note that if membership is refused and \( a^t_i \) was to join, \( a^t_i \) is changed to the stay action, assuming this was the group that \( i \) was trying to join in response to \( a^t_i \).

### 2.3.3 Leaving A Group

When \( a^t_i \) is set to leave a group, the only thing to do is choose the group that node \( i \) is going to leave. The steps involved are as follows:

1. Calculate Relative Probability of \( i \) wanting to leave each \( j \in F^t_i \)
2. Normalize Relative Probabilities
3. Choose Group to Leave

First we calculate the relative probability of \( i \) wanting to leave \( j \mid j \in F^t_i \). This is calculated as

\[
pl_i(j) = r^i_j \cdot gg_i(\vert j \vert).
\]

The next step is to normalize the relative probabilities as \( pln_i(j) = \frac{pl_i(j)}{\sum_{j \in F^t_i} pl_i(j)} \), \( j \in F^T_i \). Finally we choose a group to leave based on \( pln_i \) and a random number \( R \in [0, 1] \).

### 2.3.4 Changing Groups

Earlier it was mentioned that at each time \( t \) node \( i \) is able to change the groups of which it is a member, while still keeping \( |F^t_i| \) constant. The process for doing so is as follows.

1. for each group \( j \in F^t_i \)
2. determine the probability of staying in \( j \ (ps^t_i) \)
3. choose \( R \in [0, 1] \), if \( R < ps^t_i \) then \( i \) will stay in \( j \)
4. if \( R \geq ps^t_i \) then \( i \) will attempt to join a group as outlined above
5. if the join is successful \( i \) will leave \( j \) otherwise it will stay in \( j \)
Determining the probability of $i$ staying in a group $j$ is done as

$$
ps_i^j = (1 - \text{epsStay}) \times ((1 - \text{epsRankMod}) \times r_i^j + \text{epsRankMod})
$$

$$
gg_{i}(j) / \text{affmax}[c_i'] \times \text{Aff}_j(i) + \text{epsStay};
$$

where $\text{epsStay}$ and $\text{epsRankMod}$ are constants and $\text{affmax}[c_i']$ is the maximum affinity for a specific group size over all group sizes. Joining a group proceeds as in section 2.3.2.
Now that the simulation is described, the manner in which communications are derived from the group information should be explained. Just because two nodes are in the same group at a time-step does not mean that they are necessarily communicating at that time-step. Depending on how one wants to use the group data from the simulation, two nodes being in contact could mean that they are communication, or it could mean that they are more likely to communicate. The more groups they have in common, the more likely they are to communicate.

We have developed a few methods of producing communications data from groups generated by the simulation, though they fall into two main categories. Category one methods say that a node is communicating with some structured subset of the members of the groups that is in at every time-step. Category two methods assign a probability to the communication between any two nodes based on the number of groups they have in common.

### 3.1 Category One Methods of Communicating

The general idea with these methods is that we impose a structure onto a group and form communications within that general structure. For example, say that we believe the group should have a ring structure. The highest ranking nodes would form the center ring, the middle ranking nodes form a middle ring and the low ranking nodes an outer ring. If the group is particularly small, the middle and outer rings can be omitted.

For filling in the communications within a particular ring the following is then done: All nodes in the center ring would form a clique. The nodes in the middle and outer rings would form a ring around the center ring, i.e. a node would communicate with some number nodes from the same ring.

In addition to intra-ring communications there are also inter-ring communications. Nodes from the middle ring would communicate with one or two nodes
from the inner ring. Nodes from the outer ring would communicate with one or two
nodes from the middle ring. Additionally there is a small chance that a node from
the outer ring could communicate with a node from the inner ring. Note that nodes
chosen from a higher ranked ring to communicate with are chosen at random.

As mentioned above, this description is for a one, two or three ring structure,
but this methodology can easily be extended to \( n \) rings as well as other structures
such as trees.

3.2 Category Two Methods of Communicating

With category two methods we assign a probability of two nodes (any two
nodes, not just nodes in contact with each other) of communicating at a certain
strength. The more groups the two have in common, the more likely they are to
communicate and the stronger their communication is likely to be. The likelihood
of two nodes communicating is determined by a poisson random variable with dis-
tribution \( (e^{-\lambda} * \lambda^k)/k! \), where \( k \) is the communication strength and \( \lambda = s/N \) where
\( s \) is the number of groups that the two nodes have in common. In the case where
\( s = 0 \) \( \lambda = \frac{1}{N^2} \). Thus it can be determined by using a random number generator
and starting at some predetermined highest communication strength and then re-
ducing the strength until a communication at that strength occurs how strong a
communication between two nodes is.

3.3 Uses for Communications

With methods for generating communications in place, there are two ways
in which they can be used. The first is to use these methods to generate a set
of communications for the groups that the simulation outputs. This is useful for
predicting communication patterns in the future. An example of this would have
the simulation saying that in the future the internet traffic between two areas is
going to substantially increase to the point where the current infrastructure can not
handle it. This would lead to a network upgrade before the bandwidth shortage,
and thus a potential problem can be avoided.

The second use for the methods of generating communications is to validate our
model. In this case communications are not generated but the probability of a set of groups producing a set of communications is calculated. Here the communications have already happened and have been observed and the idea is to tune the model so that the groups output by the model have the highest probability of producing the observed communications. This process is described more in-depth in the section on learning. Section 5 discusses how to use these two uses for communications in more depth.

3.4 Building Applications

Besides just using the communications for the general purpose of validating the model and predicting trends, other models can be built on top of the communication and social network model. Section 7 discusses an example where the spread of a contagion can be modeled on top of communications.
CHAPTER 4
Finding Groups Given Communications

To find the groups given the communications we first treat the communications as a graph, where each communication between nodes corresponds to an edge in the graph. Given the communications graph, areas of dense communications would correspond to groups. Thus the problem of finding groups is reduced to finding dense areas of the graph. Two algorithms are used for doing this, the Kernighan-Lin algorithm and maximizing the cut of the inverse of the graph. Both algorithms work by creating partitions in the graph, and recursively partitioning each cut until appropriately sized groups are found. Each algorithm is seeded with a random partition and each algorithm would be run a number of times to find different groups.

4.1 Kernighan-Lin

The Kernighan-Lin algorithm works by creating an initial partition, then at each step taking the ‘best’ candidate or the one that has the most edges across the partition compared to within the partition in each partition to be flipped, flips it, i.e. moves it to the other partition, and locks it so that it can no longer be a candidate for flipping again. Once all vertices are frozen, the configuration that had the minimum cut is returned to, all vertices are unfrozen and the process is repeated until it converges. The partitions converge when the minimum cut occurs before any flips have happened.

Kernighan-Lin algorithm worked well, but had the drawback that it always created perfectly balanced partitions, which means that all the groups produced were the same size. Having all groups the same size does not work because in practice that never happens. This lead to the algorithm of finding the maximum cut in the complement of the graph.
4.2 Maximizing the Complement

Instead of minimizing the cut as in Kernighan-Lin, an attempt to maximize the complement (which is roughly equivalent, but will ensure that both partitions are non-empty) is made. In an unweighed graph, it is possible to find the complement simply by taking the difference between the number of edges of $v_i$ and the size of the graph, thus making it easy to work on the complement without actually finding it, but in this case it’s not quite as simple as the communications graph is weighted.

The solution to this problem is to use the edges that exist from the vertex to guess what the weight of the other edges would have been. Then at each step, we pick the vertex that would be the best to maximize the cut in the complement, flip it to the other partition and freeze it. We repeat this until all vertices are frozen, and then once again return to the best configuration, unfreeze the variables, and like in Kernighan-Lin the best configuration is returned to. Again the flips occur until the partitions converge.

It should be noted we only really need to record the system of flips and the associated cut size at any step of the algorithm in order to find and return to the best position. This saves on memory because multiple copies of the graph need not be stored to find the best partition.

4.3 Finishing

For both algorithms, after they converge, the resulting partitions are themselves partitioned, and the algorithm is run on those partitions. Thus after recursing a number of times groups are obtained. To try to find other groups that may be hidden in the communications, a further step is added. At the beginning ten different cuts are made as seeding points for the algorithm. After the algorithm is run on these cuts without the recursive step, the three best cuts are used for the recursive part of the algorithm. Since the partitions are random, it can be assumed that they are roughly orthogonal. In practice there is very little overlap between groups, however once group generation is finished, the groups are run through a filter that finds similar groups and discards one if it is too similar to another.
CHAPTER 5
Learning

The group evolution simulator and communications generator taken together formulate a hidden Markov model (HMM), where the group evolution simulator is a Markovian process and the communications are the output symbols of this process. Being that the purpose of this project is the prediction of the nature of communications, we would like to be able to take a set of communications and predict what communications from this source will look like at some point in the future. To accurately predict future communications the appropriate parameters ($\lambda$) of the model need to be found. We do this by learning $\lambda$ from the data that we are given.

For a general HMM [12], $\lambda = (A, B, \pi)$ where $A$ is the state transition probabilities, $B$ is the set of probabilities of a state producing a symbol and $\pi$ is the initial state distribution, i.e. the probability of being in a state at time 0. In the case of the presented model, $A$ would the model described in section 2 and $B$ is the communications as described in section 3. This leaves $\pi$ which will be addressed in section 5.3.

As can been seen in the previous sections, there are a huge number of variables whose optimal values can be learned. So many in fact that learning all at once is nearly impossible. To ease the task, the scope of learning has been narrowed to the point where four classes of variables will be addresses, and only one learned on in this paper. The four most important parameters of the model are $e_i^0$, $\beta$, $\psi$ and $c_i$. To further simplify the problem the assumption is being made that every node has the same initial energy level, i.e. that $e_i^0 = e_j^0$ for every node $i, j$.

Our general approach to learning $\lambda$ is to employ a modified EM algorithm [6] called the Baum-Welch algorithm for parameter estimation [12]. We use this algorithm for the reason that the states of the nodes are not known. Our specific algorithm uses gradient ascent to solve continuous parts of learning and to use simulated annealing for the discrete parts.
5.1 Learning Initial Energy, Energy per Flake and Threshold Values

Now despite the fact that the scope of learning has been narrowed down to only four parameters, learning on three of them is very hard, if not impossible at the moment. Learning on $e_i^0$, $\beta$ and $\psi$ proves to be very hard due to the algorithm for finding groups. The drawback to finding groups based solely on communications is that two nodes may be in a group together but not communicating. This is a problem because the non-communicating nodes exist and are likely to be part of a group, but because they are not communicating the algorithm will not detect them. The non-communicating nodes would be nodes that read the newsgroups but do not post. What this means is that when learning is done on $e_i^0$, $\beta$ and $\psi$ these values will want to go to zero due to the fact that some nodes only communicate for part of the time-series. Because they are assumed to exist for all time, and they are not found to be part of a group for a number of actions, they are assumed to not join a group for each action, despite the fact that they were predicted to, thus causing their energy to go down each step.

What this in turn means is that because for a large number of actions they are doing nothing, the amount of surplus energy they have (the amount of energy that they started with, $e_i^0$) should be smaller, thus causing the gradient on $e_i^0$ to go in the negative direction. As $e_i^0$ goes down, so must $\beta$ and $\psi$ to accommodate for when a node does perform an action. If $\beta$ and $\psi$ are high, but the node has no energy, then the probability of joining is near zero, but reducing them makes the probability of joining higher. Thus $e_i^0$, $\beta$ and $\psi$ all go to zero with the current learning technique. This problem will be fixed when the problem of learning the optimal $FZ^t$ for all $t$ is solved (Section 5.3).

5.2 Learning Node Classifications

The goal with learning the classification of each node is to determine whether learning the classifications for each node will enable the simulation to more accurately evolve groups that are likely to cause the communications observed than randomly guessing the classifications.
5.2.1 Collecting Data for Experimentation

Two different data sets were collected to allow for experiments to be run to test the hypothesis that using learned classifications would yield better results than using guessed classifications. Data was collected from two newsgroups, alt.revisionism and alt.movies. Posts from a time period of 110 days were processed such that a reply to a post was considered to be a communication between two nodes. A moving seven day window was then passed over the data and communications in the window were considered to be part of that time-step. For the 110 days there are 104 distinct seven day windows which will represent the time-steps used for the work. The reason that a seven day window is used is that there are not enough posts per day to reasonably be able to predict groups nor to learn $\lambda$. The 104 time-steps are labeled $t = 0 \ldots 103$.

After the data sets were processed into the communication time-steps, the group finding algorithm was applied to the communications for each time-step, thus producing $FZ^t$ for all $t$. Finding the groups alone does not enable learning however. One of the fundamental principles of the simulation is that groups evolve with time, they do not just exist for one specific time-step. Therefore an algorithm needed to be developed to find a mapping for the groups in $FZ^t$ to the groups in $FZ^{t+1}$.

5.2.1.1 Mapping Groups

The current algorithm used map to groups between time-steps is a best fit algorithm. For each group $j$ in $FZ^{t+1}$ a comparison is made with the groups in $FZ^t$. The group $k$ that is closest to $j$, where closest is defined as $|j \cap k|/|j \cup k|$, is considered to be the group that $j$ evolved from. In the case where $|FZ^{t+1}| < |FZ^t|$ groups are assumed to have disappeared from time $t$ to time $t+1$. When $|FZ^{t+1}| > |FZ^t|$ new groups are assumed to have formed. The new groups would be the ones that match up least well with the groups from time $t$.

5.2.2 Setting up Learning

Now that a group sequence that is likely to produce the communications has been created, it can be used to learn classifications for nodes. For this specific experiment, the data-sets were broken up into two sets each, a training set and a
testing set. The training set is used to learn optimal classifications for each of the nodes, and after learning has been done, the testing set will be used to see how well the learned classifications perform compared to assigning every node to be either all leaders, all socialites or all followers. The specific comparison will be the probability of the groups that the simulation produces have of producing the communications observed in the newsgroups. Also to be compared is performance of the learned classifications on the training set. Looking at performance with the training data tells how well learning worked, and looking at performance on the testing data will show the predictive abilities of the simulation.

5.2.2.1 Experiments

Specifically, the following will be examined,

1. Probability of groups producing the training communications when all nodes are followers.

2. Probability of groups producing the training communications when all nodes are socialites.

3. Probability of groups producing the training communications when all nodes are leaders.

4. Probability of groups producing the training communications when all nodes assigned classifications based on the learned classifications.

5. Probability of groups producing the testing communications when all nodes are followers.

6. Probability of groups producing the testing communications when all nodes are socialites.

7. Probability of groups producing the testing communications when all nodes are leaders.

8. Probability of groups producing the testing communications when all nodes assigned classifications based on the learned classifications.
The first four probabilities test how likely the learned classifications are to produce the communications that they were trained with, while the second four test the predictive abilities of the simulation as well as which method of assigning classifications produces the best results.

5.2.2.2 Learning

The specific algorithm for learning the classifications is gradient ascent. Using $\lambda$ the probability of the specific group evolution ($\Omega$) can be calculated. Thus, by perturbing each nodes classification by a small amount ($h$) the gradient on $\Omega$ can be found for each node. The gradient is calculated for each node’s classification and then a small step in the direction of the gradient is taken ($\eta \times \Delta \Omega$). This serves to increase the probability of seeing the specific group evolution given $\lambda$. Ascending the gradient continues until there is no improvement in the probability of producing the group evolution. At this point a local maxima has been reached and the algorithm is terminated.

5.2.3 Running the Simulation

The learned classifications are then fed into the simulation to determine whether the groups produced by the simulation at each step were more likely to produce the communications than just classifying everyone as the same type. Additionally the groups that were found using the group finding algorithm were used as seed groups for the simulation. This gives the simulation a starting place so that random groups do not evolve, rather groups based on what is known to exist evolve.

The simulation was started in two places, with the groups found at time $t = 0$ and with the groups found at time $t = 50$. This allows for the performance analysis on both the training set and the testing set. Because the simulation is a random process, multiple runs of the simulation were made so that averages could be made of the probability of seeing the observed communications. Specifically the obtained results were from 150 runs of the simulation and corresponding program that checks how likely the simulated groups would have produced the communications. Additionally, the learned classifications were obtained after 7500 iterations of learning. The classifications had not yet converged, but because of time constraints were not
able to. The learning programs continue to run and future tests should show that the more learning that is performed the better the results will be.

5.2.3.1 Unknown Classifications

There is one issue that still needs to be addressed before the simulations can be run. A sizable number of nodes that do not appear in the training data set exist. How then can a classification be learned on these nodes? To accommodate for this, three different experiments were run with the learned data. Three data sets were created with the learned classifications. One with the unknown nodes being leaders, one with them being socialites and one with them being followers. Using these three sets allowed for the unknown nodes to have the spectrum of classifications and allowed for an objective comparison to be made to the assigned classifications.

5.3 Learning the State Sequence

One of the issues that arises with HMMs is that the state sequence producing the observations is generally unknown. It is important to know the state sequence that produced a set of observations because learning $\lambda$ is based on knowing the state sequence. In practice it is very hard to hind the exact state sequence that produces a sequence of output symbols.

Instead what is done is a state sequence that is most likely to have produced the output sequence is found. Generally this would be considered to be a hard (exponential) problem, but the Viterbi algorithm, which makes use of dynamic programming techniques solves the optimal state sequence problem in $O(N^2T)$ time $[12]$. Though Viterbi has been shown to solve the optimal state sequence problem, just applying Viterbi to learning the state sequence that produces communications does not work. This is because Viterbi works on a single state machine while for our model each node is its own state machine, so while Viterbi might work for a single node it will not work for the entire set of nodes because each time-step produces an exponential number of possible states. Because of the exponential number of states that must be analyzed any algorithm that finds an optimal state sequence will either
need to be exponential in nature or make use of some sort of dynamic programming algorithm. Solving this problem is left to be future work in this area of research.

5.3.1 Finding the Initial State

In addition to finding an optimal state sequence finding the initial state $\pi$ is important too. The simulation needs an initial state if it is able to predict what future groups or communications will look like. For the time being the initial state is found using the group finding algorithms, but a better solution will need to be found for the model to be more accurate.
CHAPTER 6

Results

It was expected that using the learned classifications would yield groups with a higher probability of seeing the communications than using the strait classifications of leader, socialite or follower. As the following figures show, the hypothesis that using learned classifications would yield higher probabilities of seeing communications and of producing groups that explain observed communications is true.

6.1 Convergence

One important characteristics of the results is that for three of the four classification schemes, including the learned classifications, the probability of the simulated groups producing the communications converge (figures 6.1, 6.2). This shows that over time the performance will be at some steady level that can be predicted and relied upon. With regard to the follower classification, it has not yet had time to converge, but because it performs so poorly compared to the other classifications schemes, this is not an issue.

The convergence graphs also illustrate a problem mentioned earlier with finding groups with just the density algorithm. For both the learned classifications as well as the socialite classification, there is a drop in probability at the very beginning followed by a quick rise in the probability. Due to the fact that the seed groups only contain communicating nodes, the simulation needs to take a little time to stabilize so that there are groups capable of producing the observed communications, i.e. there are non-communicating nodes in the groups. This stabilization process involves nodes joining the seed groups and increasing the group sizes so that the groups now contain nodes that have a chance of communicating, but currently are not.
Figure 6.1: Convergence Error under Various Classifications in the alt.movies Newsgroup
Figure 6.2: Convergence Error under Various Classifications in the alt.revisionism Newsgroup
Figure 6.3: Comparison of Performance with Various Classifications on
the alt.movies Newsgroup.

6.2 Performance

A direct comparison of the performance of nodes classified as leaders, socialites
and the learned classifications reveals that using the learned classifications yields a
marked improvement in the probability of observing the communications, for both
the training phase and the test phase (Figures 6.3 6.4). Allowing the learning
process to continue will likely yield further improvements.

Note that in these two figures the results of the experiment using the follower
classification are left out because the performance is so poor that the figures become
hard to read. See figures 6.1, 6.2 to see how the follower experiment performed.
Figure 6.4: Comparison of Performance with Various Classifications on the alt.revisionism Newsgroup.

6.3 Learned with Leaders

As noted earlier there was an issue with not being able to learn the classifications of all the nodes due to the fact that they do not appear in the training data. As noted above these unknown nodes were classified as either all leaders, socialites or followers and three separate experiments were run allow for a good comparison.

In the figures, the learned classifications had the unknown nodes classified as being leaders. Classifying them as being leaders provided the best performance. As seen in figures 6.5 6.6, performance with the learned classification/follower was better than just using the follower classification and almost up to the level of just
leaders. Using the learned/socialite classifications yielded performance almost at that of the strait socialite classification, the best of the assigned classification experiments.

Figures 6.5 6.6 compare the performance between the various schemes that were used for the nodes that could not be learned on. Of note is that the learned/leader performed the best.

On the surface it seems odd that the learned/leader would perform better than the learned/socialite because the strait socialite classification performed the best. This is not the case though because of some of the characteristics of the simulation.
Figure 6.6: Comparison of Classification Schemes used with Learned Classifications on alt.revisionism

The nodes that are assigned a classification do not appear in the communications until late in the data. As a result giving them a classification that causes them to not want to join groups, or to join fewer groups at the beginning will give a better probability of seeing the communications. The leader classification spends more energy to be in a group as a function of time. As a result, these nodes are likely to join fewer groups and stay in fewer groups as leaders.

Another question that might be asked is why on the $t = 0 - 103$ experiments is there not a drop in probability at time 50 when the unclassified nodes are now beginning to communicate. The reason for this is that these nodes are really active
for the first 50 time-steps, but just not communicating. Thus when they begin communicating there is no drop in probability because they are active and part of groups and thus are capable of producing the observed communications.

6.4 Distribution of Learned Classifications

The distribution of the learned classifications themselves is interesting to examine. As figures 6.7, 6.8 show, the trend in classifications was toward nodes preferring larger groups. In general the divide between liking medium and large groups occurs when $c_i > 0.6$, but with more learning the classifications may progress to that point, as the classifications had not yet converged. The distributions also show that while medium to large groups are preferred, most nodes like to be in medium sized groups.

6.5 Social Networks

In terms of Social Network theory, having the learned classifications provide the best performance allows for researchers to reverse the process normally used to find theories about the networks. General practice is to develop a theory and go find data to prove or disprove it. Using the algorithms developed here allow for a researcher to take data and find information that fits the data.

An example of this draws directly from the experiments run here. Suppose a

Figure 6.7: Distribution of Learned Classifications on alt.movies.
researcher wants to find out what sized groups a population likes to be members of. The hypothesis could be that the population prefers to be in small groups, or in terms of the simulation, they like to be leaders. The researcher now needs to go find data to support the hypothesis. In doing so the researcher may find the sized groups that the population likes to be in or may only find that the population does not like to be in small groups. Additionally, small is a relative term. What exactly is small? Another question that is hard to answer using this method is what is the distribution of size preferences?

By taking the data and learning on this one attribute the researcher can determine not only a general preference for group size, but also find a distribution for this preference. Furthermore, because learning is already being done on one attribute, why not learn on more than one. Another question that could be asked is how often nodes like to change groups. This question can be answered by learning on $desD, desA$ which help control how much a node likes groups it is in. As these values change, a node is correspondingly going to be more or less likely to want to stay in groups.

A further attribute that could be learned is how long a node likes to stay in a group. The amount of energy spent on a flake as a function of membership time is a function of $\beta$. Increasing beta causes more energy to be spent on the group as time
increases, while decreasing it causes less to be spent. At some point a node could decide that it is spending too much energy and leave the group so that it can gain some surplus energy. Thus by increasing the amount of energy that a node needs to spend as a function of time, it makes the node likely to leave after a shorter interval.
CHAPTER 7
Applications: Contagion Theory

The applications for our model are more than just modeling social networks and communication patterns. Further models can be built over the structure provided by the base model. For example, one could model a disease that spreads via communications between nodes. The stages of disease could be modeled by a finite state machine where different stages could include infection, diagnosed, cured and dead. A state would be associated with a node and at each time-step the state would output a symbol representing what the outside world considers the state of the node to be. Note that much other work has been done in the area of contagious diseases [9, 1] in both the areas of computer [4, 11] and human [7] borne diseases.

7.1 Specifying the Contagion Model

To illustrate how further models can be used with the general model, the contagion model was developed more in-depth. Objectives in using this model are determining whether it is feasible to learn the probabilities governing state transitions as well as if it is possible to learn the specific state sequence evolution that is responsible for the observation symbols. Both of these problems can be solved by applying classic learning techniques for learning the details of HMMs. The contagion model itself is a HMM, with unknown states outputting symbols for the world to observe. Note that though it the contagion model is an HMM, it will be referred to as a state machine.

The contagion model involves six basic states that a node can be in at any time. These states are healthy, infected, diagnosed, cured immune, cured infectious and dead. Healthy individuals remain healthy until they become infected. At this point the disease progresses from infected to any state except for healthy; diagnosed to any but healthy and infected; cured immune is essentially healthy but with some protection rating against getting the disease; cured infectious is like cured immune but the node is still a carrier and may also suffer a relapse; dead can only progress...
to dead. Figure 7.1 depicts the state machine and its output symbols.

Additionally each state produces an output symbol. Healthy produces a $H$, infected the symbol of the state that the node was in prior to infection, diagnosed a $D$, both cured states a $C$ or an $H$ and dead a $K$. The reason that the cured states can emit either a $C$ or an $H$ is because if a node goes from healthy to infected to cured the world may never know that the node had the disease. For the purposes of this example cured will always output $C$. This decision was made because it makes learning the state sequence much easier and will be explained in more depth in that
Note that to the world the two cured states are generally indistinguishable, but by controlling transition probabilities one of the two may be inaccessible, i.e. in the case of AIDS cured immune could not be reached, but in the case of chicken pox the cured infectious state could not be reached.

Nodes become infected when they communicate with an infected node. For every such communication between a healthy node $i$ and an infected node $j$ there is a probability that $i$ becomes infected.

7.1.0.1 Advantages of Contagion Model

There are a few advantages to using the contagion model over other models, such as the SIR (susceptible, infected, recovered) model. One advantage is the infected state, which allows for an incubation period. Another is the fact that cured states can be re-infected, as well as the fact that the cured infectious state still carries the disease. These advantages make the contagion model able to handle a larger variety of diseases.

The infected state is unique to our model compared to the SIR model. We believe that in most cases the it is not known that a person has a disease for some period of time, also known as the incubation period. We feel that it is important to take into account the incubation period when modeling the spread of the disease because in many cases the disease is more widespread than it appears from just the diagnosed cases. Thus if a group, such as the World Health Organization (WHO) wanted to stop a disease from spreading through a quarantine modeling the disease with our contagion model would be appropriate because by learning the state transition probabilities and an optimal state sequence the true extent of the disease can be found and the quarantine could be made more effective.

7.2 Learning Transition Probabilities

Learning the state transition probabilities as well as the probabilities of the contagion spreading can be accomplished with the same algorithm as was used to find the optimal classifications as described in section 5.2.2.2.
Figure 7.2: An example of the network over which the contagion was modeled.

7.2.1 Results of Learning Transition Probabilities

To test whether it is possible to learn the various parameters governing the state machine a state machine was built and a network of nodes was created over which they could communicate and the contagion could spread. The nodes were arranged in a matrix and each node could communicate with its four neighbors, designated as up, down, left and right. Nodes on the edge of the matrix communicated with nodes on the opposite edge so that the network is essentially a sphere as seen in Figure 7.2.

Two experiments were then run to determine the performance of learning the parameters of the contagion model. Experiment 1 involved setting up a 40 node by 40 node network and infecting 2 disjoint nodes. The spread of the contagion was simulated for 200 time-steps during which both the states that the nodes were in and the node output symbols were kept track of. Using the states and output symbols the transition and infection probabilities were learned from some starting point. The learned probabilities were then compared to the original probabilities to see how well the original probabilities could be recovered.

Experiment 2 involved a 100 node by 100 node network that was simulated on for 50 time-steps with 50 random nodes being infected\(^4\). The parameters were then learned using the state and observation symbol data. Using the learned probabilities,

\(^4\)Nodes were selected with replacement.
<table>
<thead>
<tr>
<th>Probability</th>
<th>Seed</th>
<th>Learned</th>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(\text{infected} \Rightarrow \text{diagnosed})$</td>
<td>0.3</td>
<td>0.230089</td>
<td>0.2</td>
</tr>
<tr>
<td>$p(\text{infected} \Rightarrow \text{curedimmune})$</td>
<td>0.02</td>
<td>0.0102662</td>
<td>0.01</td>
</tr>
<tr>
<td>$p(\text{infected} \Rightarrow \text{curedinfectious})$</td>
<td>0.05</td>
<td>0.0114718</td>
<td>0.01</td>
</tr>
<tr>
<td>$p(\text{infected} \Rightarrow \text{dead})$</td>
<td>0.02</td>
<td>0.0103274</td>
<td>0.01</td>
</tr>
<tr>
<td>$p(\text{infected} \Rightarrow \text{infected})$</td>
<td>0.61</td>
<td>0.731846</td>
<td>0.77</td>
</tr>
<tr>
<td>$p(\text{diagnosed} \Rightarrow \text{curedimmune})$</td>
<td>0.5</td>
<td>0.275212</td>
<td>0.1</td>
</tr>
<tr>
<td>$p(\text{diagnosed} \Rightarrow \text{curedinfectious})$</td>
<td>0.1</td>
<td>0.0774411</td>
<td>0.1</td>
</tr>
<tr>
<td>$p(\text{diagnosed} \Rightarrow \text{dead})$</td>
<td>0.01</td>
<td>0.00796995</td>
<td>0.01</td>
</tr>
<tr>
<td>$p(\text{diagnosed} \Rightarrow \text{diagnosed})$</td>
<td>0.39</td>
<td>0.639377</td>
<td>0.79</td>
</tr>
<tr>
<td>$p(\text{curedinfectious} \Rightarrow \text{infected})$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$p(\text{curedinfectious} \Rightarrow \text{curedimmune})$</td>
<td>0.1</td>
<td>0.051998</td>
<td>0.05</td>
</tr>
<tr>
<td>$p(\text{curedinfectious} \Rightarrow \text{curedinfectious})$</td>
<td>0.9</td>
<td>0.948002</td>
<td>0.95</td>
</tr>
<tr>
<td>$p(\text{infection}(\text{infectious})$</td>
<td>0.3</td>
<td>0.388942</td>
<td>0.2</td>
</tr>
<tr>
<td>$p(\text{infection}(\text{diagnosed})$</td>
<td>0.05</td>
<td>0.140356</td>
<td>0.1</td>
</tr>
<tr>
<td>$p(\text{infection}(\text{curedinfectious})$</td>
<td>0.1</td>
<td>0.159219</td>
<td>0.1</td>
</tr>
<tr>
<td>$p(\text{communication})$</td>
<td>0.5</td>
<td>0.587111</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 7.1: Results of Learning on a 1600 Node Contagion Network.

the simulation was restarted at time 50 and run to time 100 to see if the spread of the contagion had similar characteristics to the original parameters.

7.2.2 Experiment One

Table 7.1 shows that in most cases the original parameters were recoverable through learning. The results were obtained taking the average of 50 attempts at learning the data. The only variable for each pass was the seed given to the simulation and learning program for random numbers. The same two nodes were infected each time. The learning algorithm was performed 1500 times for each data set. Using more infected nodes at the start reduces the need for high numbers of gradient ascent iterations as well as lowering the number of time-steps needed for learning as evidenced by experiment 2.

7.2.3 Experiment 2

With this experiment we were seeking to show how well learned probabilities could predict spread patterns. Figure 7.3 shows the initial state where the light gray dots represent the initially infected nodes. As the shade gets darker the nodes
are correspondingly diagnosed, cured or dead. Figure 7.4 shows the difference in the spread of the contagion between using the actual and the learned probabilities. Both were started from the same state (the state at time 50) and simulated for 50 time-steps. Using the learned probabilities results in a much faster spread of the contagion, which is to be expected given that the learned probabilities for the spread of the infection are higher than the real probabilities as seen in table 7.2. Additionally the probability of two nodes communicating is higher then it really should be. Allowing for more iterations of learning would hopefully solve some of these issues, though the problem of the higher infection rates needs to be solved.

7.3 Learning the State Sequence

Given that the transition probabilities can be learned, can the state sequence responsible for the observations be learned? The short answer is yes, but it is very hard. The Viterbi algorithm has been shown to solve the optimal state sequence
### Table 7.2: Results of Learning on a 10,000 Node Contagion Network.

<table>
<thead>
<tr>
<th>Probability</th>
<th>Seed</th>
<th>Learned</th>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p(\text{infected} \Rightarrow \text{diagnosed}) )</td>
<td>0.3</td>
<td>0.23726</td>
<td>0.2</td>
</tr>
<tr>
<td>( p(\text{infected} \Rightarrow \text{curedimmune}) )</td>
<td>0.02</td>
<td>0.00913</td>
<td>0.01</td>
</tr>
<tr>
<td>( p(\text{infected} \Rightarrow \text{curedinfectious}) )</td>
<td>0.05</td>
<td>0.00913</td>
<td>0.01</td>
</tr>
<tr>
<td>( p(\text{infected} \Rightarrow \text{dead}) )</td>
<td>0.02</td>
<td>0.00972</td>
<td>0.01</td>
</tr>
<tr>
<td>( p(\text{infected} \Rightarrow \text{infected}) )</td>
<td>0.61</td>
<td>0.73475</td>
<td>0.77</td>
</tr>
<tr>
<td>( p(\text{diagnosed} \Rightarrow \text{curedimmune}) )</td>
<td>0.5</td>
<td>0.3085</td>
<td>0.1</td>
</tr>
<tr>
<td>( p(\text{diagnosed} \Rightarrow \text{curedinfectious}) )</td>
<td>0.1</td>
<td>0.07584</td>
<td>0.1</td>
</tr>
<tr>
<td>( p(\text{diagnosed} \Rightarrow \text{dead}) )</td>
<td>0.01</td>
<td>0.0071</td>
<td>0.01</td>
</tr>
<tr>
<td>( p(\text{diagnosed} \Rightarrow \text{diagnosed}) )</td>
<td>0.39</td>
<td>0.60855</td>
<td>0.79</td>
</tr>
<tr>
<td>( p(\text{curedinfectious} \Rightarrow \text{infected}) )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( p(\text{curedinfectious} \Rightarrow \text{curedimmune}) )</td>
<td>0.1</td>
<td>0.05449</td>
<td>0.05</td>
</tr>
<tr>
<td>( p(\text{curedinfectious} \Rightarrow \text{curedinfectious}) )</td>
<td>0.9</td>
<td>0.9455</td>
<td>0.95</td>
</tr>
<tr>
<td>( p(\text{infection}(\text{infectious}) )</td>
<td>0.3</td>
<td>0.3933</td>
<td>0.2</td>
</tr>
<tr>
<td>( p(\text{infection}(\text{diagnosed}) )</td>
<td>0.05</td>
<td>0.14272</td>
<td>0.1</td>
</tr>
<tr>
<td>( p(\text{infection}(\text{curedinfectious}) )</td>
<td>0.1</td>
<td>0.14704</td>
<td>0.1</td>
</tr>
<tr>
<td>( p(\text{communication}) )</td>
<td>0.5</td>
<td>0.58667</td>
<td>0.5</td>
</tr>
</tbody>
</table>

problem, though in the case of the contagion model Viterbi will need exponential
time and memory to work as opposed to the normal \( N^2T \) time where \( N \) is the
number of states \([12]\). The reason for the increased complexity is that instead of
having one Markovian process, there are \( n \) Markovian processes, where \( n \) is the
number of nodes in the simulation. Given that a node can be in one of 6 states at
any time there are \( 6^n \) possible states that the system can be in at any given time.
Thus running Viterbi on the entire state space yields a \((6^n)^2T\) worst case algorithm.

#### 7.3.1 Viterbi on Individual Nodes

We theorized that there would be a way around the time and space restriction
by running the Viterbi algorithm on each node separately. As there are \( n \) state
machines, running on individual nodes would yield an \( nN^2T \) algorithm. For this to
work though a seed state sequence must be supplied to that probabilities of infection
can be calculated.
7.3.1.1 Seeding Viterbi

Because most of the states only emit one observation symbol it can be easy to
tell what state the node is in at any given time. Issues arise when a node is infected,
in which case it emits the symbol of the state that it was in before infection, or with
the cured states because based on observations it can be hard to determine if a node
is either cured immune or cured infectious.

More can be inferred about the state sequence based on the fact that if a node
is diagnosed at time $t$ then it must have been in the infected state at time $t - 1$. Given
the information about the order in which state transitions are known to occur,
a preliminary state sequence can be built in $O(n \times t)$ time. The algorithm starts
with the last state that emitted and looks at what states nodes could have been in
at that time. Nodes that appear healthy are considered to be healthy. Cured nodes
are randomly assigned to be cured immune or cured infectious. All other states are
known to be true, i.e. if a node appears dead then it must be dead.

The observation symbols at time $T - 1$ are then examined. If the symbol is
the same at time $T$ the node is assumed to not have made a state change, excepting
that if at time $T$ the node was cured immune there is a small chance that it is
cured infectious at time $T - 1$. In cases where there is a transition from healthy to
diagnosed or cured the node is assigned to be infected at time $T - 1$. This algorithm
is then applied to time $T - 2$ and $T - 1$ with the addition that if a node goes from
being observed as healthy to being infected that it is assumed that the node really
was healthy in time $T - 2$. Recursing backward yields a state sequence which can
be given to the Viterbi algorithm. The assumption that a node remains infected for
only one time-step is the major deficiency of the algorithm and it is for this reason
that the modified Viterbi approach was designed.

Earlier it was mentioned that the cured states always output the $C$ as opposed
to outputting an $H$ when the node transitions to cured from infected. The reason
for this is that it makes it even harder to find the optimal state sequence. If the
cured states could output an $H$ then even a modified Viterbi approach might not
work because it would never detect infected nodes.
Though running Viterbi on individual nodes was found to produce improved state sequences over the seeding algorithm it too suffers from a deficiency, that of producing state transitions with zero probability. The situation of zero probability transitions arises with the following situation. Suppose that at time $t$ a node is observed to be diagnosed (Figure 7.5a) but at time $t-1$ the node was observed to be healthy (Figure 7.5b). This means that at time $t-1$ the node is actually infected (Figure 7.5c). Assuming that the node was not infected at the start of the simulation the node contracted the contagion from one of its neighbors. The issue is that none of the neighbors will claim responsibility for infecting the node. They all assume that one of the other neighbors is responsible. This means that the node became infected as a random event which is not supported by the model. Even running Viterbi on the neighbor nodes first does not solve this problem because the algorithm is looking only at the probability of that one nodes transition, not its neighbor nodes. As a result, Viterbi does not find that the neighbor node became infected earlier than it thinks it did because the probability of this happening is not high. This situation is essentially a version of the free-rider problem in that every node assumes that another node is picking up the slack or claiming responsibility.

Taking into account the neighbor nodes’ transitions would solve this, but leads to a two away problem where what if the contagion spread undetected from a node two nodes away. The neighbors to the neighbors could all assume that the contagion

---

5. The seeding algorithm suffers from the same problem.
spread through someone else.

7.3.1.3 Effective Solution

The only effective solution is to look at how changing a single node’s state affects the probability of the all nodes’ transitions. This is different from considering the whole state transition in the exponential solution in that there we are considering all possible state combinations of all nodes, whereas with this we are only considering all possible state transitions on a single node and noting the affects of changing the transition on the probability of every other nodes’ transition. Now the effects of the free-rider problem have been mitigated, but another problem arises.

Assuming that there is one situation like this in the state sequence, there can easily be more than one, meaning that when we attempt to use the effective solution to fix one situation, the other place that we have an impossible situation will cause the probability of the state transition to be zero, meaning that it would be impossible to determine if changing a node’s state affects the probability because there is another zero probability transition elsewhere, i.e. the probability is always zero for the state transition. Thus even solving the free-rider problem does not solve the problem of finding the optimal state sequence.

7.3.1.4 Possible Solution to Optimal State Sequence

One possible solution to finding an algorithm that is linear with respect to the number of nodes would be for each node to remember the node that it contracted the contagion from. In this manner the contagion could be tracked and the case of zero probability transitions would be eliminated because the node responsible for spreading the contagion would no longer be able to absolve itself of responsibility. While this might seem like cheating in that we are getting state information along with observation information, in the case of diseases such as AIDS or any sexually transmitted disease it is often very easy for the infected person to point the finger at the person responsible for transmitting the disease.
7.3.2 Initial State Issues

There is one more issue that needs to be addressed in any solution to the optimal state sequence problem, that of where the disease first originated. The techniques discussed here all assume that the initial state contained some infected nodes. To accurately calculate the probability of a state sequence, the initially infected nodes must either be guessed or known. This is not a large concern though because other models (reference SIR model...) of contagion spread also make the same assumption and do not worry about random outbreaks. Our temporary solution was to assume that the initial state is known, which it is from the state data that the simulation outputs. Doing this allows for the learning process to not worry about where the disease came from and it can just figure out where it has spread to.

In fact a slight modification to the contagion model allows for random outbreaks of the disease by assigning a small probability for the healthy to infected state transition. It would be small enough such that it is not likely to happen, but every now and then probability will dictate that a node become randomly infected by some outside source. Modeling it as a transition from healthy to infected with all healthy neighbors would accurately account for infection by outside sources.
CHAPTER 8
Conclusion

This project’s goal was to design a model for the statistical evolution of social groups over communication networks. We developed a model for the evolution of these networks, as well as the communications they produced based on a hidden Markov model. Additionally our goal was to be able to take a real data set and learn the parameters of the model for this specific data set, and then to be able to predict more accurately predict the shape of future communications than just guessing the model parameters.

Using the alt.movies and the alt.revisionism newsgroups we were able to accomplish both of the goals of learning the model parameters as well as predicting the shape of future communications. Learning on the size preference characteristic allowed for anywhere between 50-100 percent improvement in the probability of predicting future communications compared to just guessing parameters. This is significant because learning on more parameters would likely yield even better performance.

In addition to showing that the model is viable, we further showed how applications other than just the prediction of communications. The generic contagion model that we built on top of the communications allows for the modeling of both natural diseases as well as computer viruses. By modeling the contagion on top of the communications the communications model can be used to predict what future communications may look like as well as more effectively predict the spread of the contagion.

It is our hope that these models can be extended and some of the problems left to be solved, such as finding an optimal state sequence, can be solved. In particular finding a non-exponential optimal state sequence solution has applications outside of just this work, as it would be an effective solution to any hidden Markov model where multiple dependent state machines are used to produce the overall state and observation sequences.
LITERATURE CITED


47
APPENDIX A
Formula and Variable Reference

1. $FZ^t$: The set of flakes at time $t$.

2. $e_i^t$: The energy for node $i$ at time $t$.

3. $c_i$: The group size preference for node $i$.

4. $\delta_i$: Calculated from $c_i$, used to normalize the classification to $[0,1]$.

5. $\tau_j^i$: Node $i$’s membership in flake $j$ at time $t$.

6. $S_t$: The world state at time $t$.

7. $a_i^t$: The action for node $i$ at time $t$.

8. $F_i^t$: The set of flakes of which node $i$ is a member of at time $t$.

9. $e_i^t$: Node $i$’s surplus energy at time $t$.

10. $\gamma$: Constant used for the calculation in surplus energy.

11. $\beta$: Constant used for the calculation in surplus energy.

12. $p_+$: The probability of having join as an action.

13. $p_0$: The probability of having stay as an action.

14. $p_-$: The probability of having leave as an action.

15. $P_i^t$: The most likely action for node $i$ at time $t$.

16. $F(e_i^t, P_i^t, a_i^t)$: The energy reward/penalty function.

17. $\lambda(x), \lambda_0(x), \mu(x)$ and $\mu_0(x)$: Specific functions used in calculation the amount of energy rewarded or penalized.

18. $q_i^t$: The qualification of node $i$ at time $t$. 
19. $K^t_i$: The set of flakes of which node $i$ has knowledge at time $t$.

20. $C_{il}$: The contact of two nodes. $i$ and $l$ are in contact if and only if they are in a flake together.

21. $P$: Percentage of $FZ^t$ that a node will consider joining.

22. $FZ^t_{m_i}$: Set of flakes, drawn from $FZ^t$ that a node will consider for joining.

23. $D_F$: The set of flakes that a node will consider joining.

24. $gg_i(x)$: A rating of how much a node likes a flake of size $x$.

25. $Aff_i(j)$: The affinity of node $i$ for flake $j$, based on qualification.

26. $\theta$: A constant used in calculating a nodes preference for a group of a specific size.

27. $desD, desA$: Constants used for the calculation of $Aff_i(j)$.

28. $p^t_i(j)$: Relative probability of node $i$ wanting to join flake $j$.

29. $p^n_i(j)$: Normalized probability of node $i$ wanting to join flake $j$.

30. $j_s$: Flake that is chosen by a node to attempt to join.

31. $q^t_j$: The average qualification of all nodes in flake $j$ at time $t$.

32. $Aff_j(i)$: A measure for how much flake $j$ wants to node $i$ to join.

33. $r^t_i(j)$: The rank of node $i$ in flake $j$.

34. $pl_i(j)$: The relative probability of node $i$ wanting to leave flake $j$.

35. $pln_i(j)$: Normalized probability of node $i$ wanting to leave flake $j$.

36. $ps^t_i$: The probability of node $i$ wanting to stay in flake $j$.

37. $\lambda$: Parameters of the HMM.

38. $\rho_*$: Constants used in calculating $p_+, p_0, p_-$.
39. $\psi$: Threshold value for $\epsilon^t_i$ used in calculating $p_+, p_0, p_-$. 

40. $\Omega$: Probability of producing a group evolution given $\lambda$. 

41. $c'_l$: Static leader classification. 

42. $c'_s$: Static socialite classification. 

43. $c'_f$: Static follower classification. 

44. $c'_i$: Static classification of node $i$. 