REVERSE ENGINEERING A HIDDEN MARKOV MODEL FOR COMPLEX SOCIAL SYSTEMS

By

Hung-Ching Chen

A Thesis Submitted to the Graduate Faculty of Rensselaer Polytechnic Institute in Partial Fulfillment of the Requirements for the Degree of DOCTOR OF PHILOSOPHY

Major Subject: COMPUTER SCIENCE

Approved by the Examining Committee:

Dr. Malik Magdon-Ismail, Thesis Adviser

Dr. Mark Goldberg, Member

Dr. William (Al) Wallace, Member

Dr. Mohammed Zaki, Member

Rensselaer Polytechnic Institute
Troy, New York
May 2008
(For Graduation August 2008)
REVERSE ENGINEERING A HIDDEN MARKOV MODEL FOR
COMPLEX SOCIAL SYSTEMS

By

Hung-Ching Chen

An Abstract of a Thesis Submitted to the Graduate
Faculty of Rensselaer Polytechnic Institute
in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY
Major Subject: COMPUTER SCIENCE
The original of the complete thesis is on file
in the Rensselaer Polytechnic Institute Library

Examining Committee:

Dr. Malik Magdon-Ismail, Thesis Adviser
Dr. Mark Goldberg, Member
Dr. William (Al) Wallace, Member
Dr. Mohammed Zaki, Member

Rensselaer Polytechnic Institute
Troy, New York

May 2008
(For Graduation August 2008)
CONTENTS

LIST OF TABLES ......................................................... vi
LIST OF FIGURES ....................................................... vii
LIST OF ALGORITHMS .................................................... ix
ACKNOWLEDGMENT ....................................................... x
ABSTRACT ................................................................. xi

1. INTRODUCTION ....................................................... 1
   1.1 Our Contributions .................................................. 3
   1.2 Summary ........................................................... 6

2. BACKGROUND ........................................................ 7
   2.1 Social Network Analysis (SNA) .................................... 7
   2.2 Hidden Markov Model (HMM) ....................................... 8
      2.2.1 Definition ...................................................... 9
      2.2.2 Assumptions .................................................. 10
      2.2.3 Three Basic Problems of HMMs ............................... 11
         2.2.3.1 Evaluation Problem ..................................... 11
         2.2.3.2 Decoding Problem ..................................... 15
         2.2.3.3 Learning Problem ..................................... 18

3. ViSAGE—A HMM FOR VIRTUAL SIMULATION AND ANALYSIS OF GROUP EVOLUTION ......................................................... 20
   3.1 State of The Society .............................................. 22
      3.1.1 Type .......................................................... 24
      3.1.2 Rank .......................................................... 24
      3.1.3 Qualification ............................................... 25
      3.1.4 Resources .................................................... 25
   3.2 State Transitions .................................................. 26
      3.2.1 Actor’s Normative Actions ................................... 26
      3.2.2 Actors’ Real Actions ....................................... 27
      3.2.3 State Update ................................................ 30
   3.3 Communications ................................................... 31
   3.4 Example of ViSAGE ............................................... 35
A. List of Functions and Variables ........................................... 119
  A.1 Actor Properties ....................................................... 119
  A.2 Group Properties ..................................................... 120
  A.3 Resources ............................................................. 120
  A.4 Action Probabilities .................................................. 121
  A.5 Actor and Group Fitting .............................................. 122
  A.6 Association Probabilities ............................................ 124
  A.7 Returned Social Capital Models ................................. 124
B. Configuration ............................................................. 126
  B.1 Parameters ............................................................ 126
  B.2 Default Values ....................................................... 127
C. Example of Learning Reward/Penalty Parameters ................. 128
  C.1 Solving the First Term .............................................. 128
  C.2 Solving the Second Term .......................................... 131
LIST OF TABLES

3.1 The Agency and Structure Table .............................................. 30
5.1 Confusion matrix in learning size-type using Cluster, EM and Learn .... 72
5.2 Confusion matrix in learning ambition-type using A-EM and A-Optimal ... 75
5.3 The mean and standard deviation (S.D.) about the differences between the learned $\omega_\alpha$’s and true $\omega_\alpha$’s .......................................................... 78
5.4 The results of learning actors’ size-type on Movie newsgroup. ............ 90
5.5 The results of learning $\omega_\alpha$’s on Movie newsgroup ......................... 91
5.6 The results of learning actors’ size-type on Enron email ....................... 93
5.7 The results of learning $\omega_\alpha$’s on Enron email ................................ 93
6.1 Some statistical properties of the $\{D_{ij}\}$ .................................... 102
6.2 The dependence of the Accuracy (in %) on $del, wid, run$ ..................... 105
6.3 The average number of messages posted by different types of actors ...... 107
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Modeling of dynamics</td>
<td>3</td>
</tr>
<tr>
<td>3.1</td>
<td>Framework for the step by step evolution in the social group model</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>Simulation using ViSAGE</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>Usage of ViSAGE</td>
<td>21</td>
</tr>
<tr>
<td>3.4</td>
<td>The flowchart of ViSAGE</td>
<td>23</td>
</tr>
<tr>
<td>3.5</td>
<td>An example of ViSAGE</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>Multistage learning process</td>
<td>38</td>
</tr>
<tr>
<td>4.2</td>
<td>An example of data set including only communication dynamics</td>
<td>39</td>
</tr>
<tr>
<td>4.3</td>
<td>An example of data set including group structures</td>
<td>39</td>
</tr>
<tr>
<td>4.4</td>
<td>An example of data set including group evolution</td>
<td>44</td>
</tr>
<tr>
<td>5.1</td>
<td>Multistage learning process with algorithms</td>
<td>69</td>
</tr>
<tr>
<td>5.2</td>
<td>The accuracy (%) of learning size-type using Learn, Cluster, and EM algorithms from group evolution corresponding with different time steps</td>
<td>73</td>
</tr>
<tr>
<td>5.3</td>
<td>The predictive error $E_f$ and $E_p$ for various algorithms for learning stype</td>
<td>74</td>
</tr>
<tr>
<td>5.4</td>
<td>The accuracy (%) of learning ambition-type using algorithm A-Optimal and A-EM from group evolution with different time steps of data set</td>
<td>75</td>
</tr>
<tr>
<td>5.5</td>
<td>The accuracy (%) of learning ambition-type using algorithm A-EM from group evolution with different learning iterations</td>
<td>76</td>
</tr>
<tr>
<td>5.6</td>
<td>The results of learning one $\omega_\alpha$ under different conditions</td>
<td>77</td>
</tr>
<tr>
<td>5.7</td>
<td>The accuracy (%) of learning size-type from group structure using different algorithms with different time steps of data set</td>
<td>79</td>
</tr>
<tr>
<td>5.8</td>
<td>The comparison of the results of learning size-type between learning from group evolution and group structure</td>
<td>79</td>
</tr>
<tr>
<td>5.9</td>
<td>The accuracy (%) of learning ambition-type from group structure using different algorithms with different time steps of data set</td>
<td>81</td>
</tr>
<tr>
<td>5.10</td>
<td>The comparison of the results of learning ambition-type between learning from group evolution and group structure</td>
<td>81</td>
</tr>
</tbody>
</table>
## LIST OF ALGORITHMS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 The process of an HMM</td>
<td>10</td>
</tr>
<tr>
<td>2.2 Forward Algorithm</td>
<td>13</td>
</tr>
<tr>
<td>2.3 Backward Algorithm</td>
<td>15</td>
</tr>
<tr>
<td>2.4 Viterbi Algorithm</td>
<td>17</td>
</tr>
<tr>
<td>3.1 Compute the number of friendship and the probabilities of communications</td>
<td>33</td>
</tr>
<tr>
<td>3.2 The communication model</td>
<td>34</td>
</tr>
<tr>
<td>4.1 Finding matchings using a greedy algorithm</td>
<td>45</td>
</tr>
<tr>
<td>4.2 Maximum log-likelihood learning algorithm for independent discrete parameters</td>
<td>59</td>
</tr>
<tr>
<td>4.3 Clustering Algorithm</td>
<td>60</td>
</tr>
<tr>
<td>4.4 EM algorithm for learning actors’ size-type</td>
<td>61</td>
</tr>
<tr>
<td>4.5 EM algorithm for learning dependent discrete parameters</td>
<td>62</td>
</tr>
<tr>
<td>4.6 The algorithm for learning reward/penalty parameters $\omega_a$</td>
<td>66</td>
</tr>
<tr>
<td>6.1 Multi-user Identification Algorithm</td>
<td>103</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENT

I should like to express my heartfelt gratitude to a number of people who have helped me to complete my doctoral dissertation. First of all, this dissertation could not have been written without continuous guidance and support from my advisor, Dr. Malik Magdon-Ismail. Professor Magdon-Ismail’s insightful advice has inspired my desire to further investigate and explore matters of importance in the scholarly field of. Moreover, he offered me support not only as I faced intellectual challenges but also as I faced challenges to my health and underwent cancer treatment. The understanding of such adversity in my life and encouragement from Professor Magdon-Ismail certainly has made the continuation of my studies possible.

Secondly, I should also like to extend my appreciation to my committees. I thank Dr. Mark Goldberg for his assistance, constructive reminders and vital encouragement throughout the journey. I thank Dr. William (Al) Wallace, Professor of Decision Sciences and Engineering System, for his patient, kindness and useful resources. I thank Dr. Mohammed Zaki for his trust and confidence in giving me his unreserved challenges, intriguing and constructive discussions which allow me to reflect more profoundly and metacognitively in all possibilities outside of my confined concepts. Let me express additional gratitude to Jeffrey Baumes for helping me in overlapping clustering algorithm; to Matthew Francisco for engaging me with diverse discussions and his inputs in the ramifications of social science; to Yingjie Zhou for the help on Enron email data.

At last but not least, I give special thanks to my family and loved ones who have accompanied me and provided me with their endless unconditional love throughout this seemly impossible journey (particularly under precarious health conditions) – my grandmother; my mothers and fathers; my sisters, Linda and Mary; and to a person who I didn’t mention . . . I love you!
ABSTRACT

We present a machine learning methodology (models, algorithms, and experimental data) to discover the agent dynamics that drive the evolution of the social groups in a community. Agent dynamics are governed by micro-laws, which determines an agent’s behaviors. There are a number of challenges, the first and foremost being the complex nature of the micro-laws needs to represent even a very simple society. Agents may have discrete attributes together with continuous parameters, which inevitably lead to a mixed optimization problem, and each agent has its own attributes, which also interact with other agents’ attributes, suffering from combinatorial and dimensionality curses. Another challenge is that the data upon which to answer the question is not available—typically social groups (especially online groups) do not announce their memberships, and a researcher has to infer groups from observable macro-quantities such as communication statistics. We set up the problem by introducing an agent-based hidden Markov model for the agent dynamics: an agent’s actions are determined by micro-laws. Nonetheless, we learn the agent dynamics from the observed communications without knowing state transitions. Our approach is to identify the appropriate micro-laws corresponding to an identification of the appropriate parameters in the model. The model identification problem is then formulated as a mixed optimization problem. To solve the problem, we develop a multi-stage learning process for determining the group structure, the group evolution, and the micro-laws of a community based on the observed set of communications among actors, without knowing the semantic contents. Finally, to test the quality of our approximations and the feasibility of the approach, we present the results of extensive experiments on synthetic data as well as the results on real communities, such as Enron email and Movie newsgroups. Insight into agent dynamics helps us to understand the driving forces behind social evolution.
CHAPTER 1
INTRODUCTION

The social evolution of a community can be captured by the dynamics of its social groups. A social group is a collection of agents or actors¹ who share some common context [1]. The dynamics of the social groups (global behaviors) are governed by the actor dynamics (local behaviors)—actors join new groups, leave groups, and do nothing. An actor’s actions are governed by micro-laws [2] which may be personal attributes (e.g., some people like to go out with a bunch of people, but some would prefer one-on-one), the actions of other actors (e.g., the actor may join a group because his/her friend is a member of that group), and the influence of the community (e.g., some people take an action because it is expected by some communities). Sometimes, the same actors operating within different communities perform different behaviors or are governed by different micro-laws. In summary, any reasonable model for an agent based evolving community must necessarily involve complex interactions between actors’ attributes and the social group structure itself.

Each day, individuals from all over the world receive and respond to the information from other individuals of the society. Over the past decades, technology has been integrated aggressively into our daily life. The rapid exchanges of communication among individuals have gone from surfing online for information to providing information, building individual Space/Blogs as well as getting connected through various instant messaging communities. It is apparent that online communities have become one of the most influential mediums in the process of social evolution. Yet, regardless of the impact of online communities, the role of a social value continues to play an imperative factor in the dynamics of the online communities as it has for the rapid growth of offline communities, sudden emergence or hasty dissipation due to changes of demands, needs, and values of the existing society. Thus, there has been a transfer of social capital from real societies to online communities. Therefore, it is essential to acquire ranges of more comprehensive and objective social factors that might have propelled the evolution of the society.

¹An actor generically refers to an individual entity in the society.
What makes one community different from another? Consider the community of college students as opposed to the community of bloggers. The same sorts of people form a large fraction of both of these communities (young adults between the ages of 16-24), yet the group structure in both of these communities is quite different. For example, an actor typically belongs to one college, but may belong to many blog-communities. It is intuitively clear why such significant differences exist between these two communities. For example the process of gaining admission into a social group is considerably harder and more selective for the college community than a typical online community. The micro-laws (e.g., actors’ group size preferences) which govern actors’ dynamics are different in these two cases, and hence the resulting communities look quite distinct. The time commitment for college is much longer than a blog community.

Recently, the explosion of online communities has provided an ideal pasture on which to groom and test social science theories. And the most natural question is: what are the micro-laws which govern a particular society? It is an interesting hunt as to how the micro-laws govern the actors’ behaviors and make a society tick. Furthermore, an efficient approach to answering this question for a given community yields a powerful tool for a sociologist that can be used for discovering the dominant factors that determine how a community evolves.

There are a number of challenges, the first and foremost being the complex nature of the micro-laws needs to represent even a very simple society. Actors may have discrete attributes together with continuous parameters, which inevitably lead to a mixed optimization problem, and each actor has his/her own attributes, which also interact with other actors’ attributes, suffering from combinatorial and dimensionality curses. Another challenge is that the data upon which to answer the question is not available—typically social groups (especially online groups) do not announce their memberships, and one has to infer groups from observable macro-quantities such as communication statistics. Given the recent explosion in online communities, it is now recognized that understanding how communities evolve is a task of rising importance. We take a machine learning approach to these challenges.
1.1 Our Contributions

We present a machine learning methodology (models, algorithms, and experimental data) for determining the appropriate micro-laws of a community (appropriate parameters in the model) based on either the observed social group evolution or observed set of communications among actors without knowing semantic contents. There are several benefits of using not content based approach:

1. *Computationally tractable*: in a online community, communication data is vast. (For instance, Myspace.com has more than 100 million active members in 2003. If we collect all the content of messages, the amount of data is huge.);

2. *More general*: such as language independent; and

3. *Less intrusive*: we can avoid the privacy, ethical and legal issues to data.

Our approach uses an agent-based hidden Markov model (HMM) of social systems for the agent dynamics, which includes a parameterized probabilistic micro-law based model of social group evolution and a communication model, which are justified through social capital theories, \[3, 4, 5, 6, 7\]. Figure 1.1 shows the general framework of the modeling of dynamics. The micro-laws are governed by the parameters, actors’ historic behaviors, and the global behaviors, and then the micro-laws influence an actor’s next action.

![Figure 1.1: Modeling of dynamics.](image-url)
There are three kinds of micro-laws (local behaviors):

1. *Actor micro-laws*: probabilistically specify actor decisions;

2. *Group micro-laws*: probabilistically specify group decisions; and


We configure the local behaviors in order to simulate, observe, and analyze the global behaviors of the society such as the number of groups and the distribution of the size of groups in a society, etc.

We have developed a multistage learning process to learn the group structure, the group evolution, and the appropriate micro-laws in the agent-based hidden Markov model. The multistage learning process is a modular design; therefore, we can develop, run, test, and improve each module independently. The benefits of multistage learning process are fourfold.

1. Each module have different outputs, and we can extract different kinds of information from each module.

2. Each module is a checkpoint. We can test the accuracy and check the performance of the algorithms for each module, and it is also easier to improve the accuracy and the performance of the learning process in each individual module.

3. Reduce the impact from the data noise in the learning process. If we use only one stage learning process, then the impact of data noise is very large, and it is difficult to control and improve the learning process. On the other hand, in the multistage learning process, each module only extra certain part of information from the data set, so it is easier to reduce the impact of data noise in the learning process.

4. The data sets are normally very huge due to the time steps and the number of communications, actors and groups. Modular design is easier to cooperate with distributed and parallel computing techniques to increase the efficiency of performance.
We develop algorithms to identify the appropriate micro-laws which correspond to identifying the appropriate parameters in the model. We identify the appropriate micro-laws by solving a mixed optimization problem because our model contains discrete parameters as well as continuous parameters. Furthermore, actors’ attributes interact with others’ attributes, and to avoid the resulting combinatorial explosion, we appropriately approximate and optimize the objective with some efficient algorithms. Most of time, we are only able to observe communications among actors instead of group evolution. So we build up a communication model to create communications among actors based on group structure, and then we can develop algorithms to identify micro-laws with appropriate parameters from communication data. In a hidden Markov model, the observations are different from the states in a Markov process. The reason that our models are hidden Markov models is that we are able to observe communication among actors, but we are not able to observe group structures which follow Markov process. We develop efficient learning algorithms to learn the appropriate parameters in our model, instead of using the traditional learning algorithms for HMMs which cannot work very well with our models.

Sometimes actors in communication networks, such as newsgroups and chatrooms, like to post messages using different actor IDs. We denote such actors as multi-ID actors or multi-ID users. As a rule, these actors attempt to hide the fact that one person is operating multiple IDs. The reasons for an actor to use multiple IDs are varied. Sometimes an actor who has become a pariah of a certain public forum may try to regain his/her status using a different ID; other times actors may post messages under different IDs in order to instigate debate or controversy; and still other times actors may pose as multiple IDs in order to sway democratic processes in their favor in certain voting procedures on internet forums, such as a leadership election. In general, the identification of an actor who posts under several IDs should have important forensic value. At a very least, flagging IDs as possibly belonging to the same actor of a public forum can justify further investigation of the posts under those IDs. Therefore, we develop a model and algorithms to classify who are multi-ID actors.

Finally, to test the quality of our approximations and the feasibility of the approach, we present the results of extensive experiments on simulated data as well as some results on real data (e.g., Enron email and Movie newsgroups).
1.2 Summary

In Chapter 2, we give the background about our work. We discuss some works on social network analysis, and the hidden Markov models (HMMs), and some traditional algorithms to solve some problems in HMMs. In Chapter 3, we give the details of the agent-based hidden Markov model, called ViSAGE, which include the probabilistic social group evolution model and the probabilistic communication model. The default functions, variables and configuration are described in Appendixes A and B. Our model is extendable and very flexible, and users can change the default setting as necessary. We also give an example to describe some micro-laws with the related parameters. Some related papers about the model and the justification has been presented in [3, 8, 9, 10, 11].

In Chapter 4, we present our approach, the multistage learning process, to learning appropriate micro-laws in the model from communications data, group structure data, and group evolution data. We describe the details of each learning stage showing the concept and the learning algorithms. We show the algorithms for learning dependent and independent discrete parameters and continuous parameters. We also give some examples about general learning framework on certain parameters in the model, and the detail of an example about learning a continuous parameter is shown in Appendix C. The related works have been presented in [12, 13, 14, 15, 16]. Chapter 5 shows some experiments and results on learning some micro-laws—some parameters in the model, including independent and dependent discrete parameters and continuous parameters. We also present the learning results of real data, e.g., Movie newsgroups and Enron email. We also discuss the impact of learning on prediction to tell which parameters are important to learn but which are not.

In Chapter 6, we present a model and algorithms to identify users who employ multiple IDs on public communication forums. We describe a model for realtime communication exchange in public forums such as newsgroups and chatrooms. We use this model to develop an efficient algorithm which identifies the users that post their messages under different IDs, multi-ID users, and then, we show the learning results. The related work has been presented in [17]. In Chapter 7, we give some discussions and conclusions.
CHAPTER 2
BACKGROUND

2.1 Social Network Analysis (SNA)

Most of the traditional methods of social network studies are labor intensive. People interview and analyze the data manually in most of the cases. They usually spend a lot of time and can only work on a small amount of data. The results can more easily be subjective and less accurate, or even misleading. Authors in [29, 30] address some of the traditional research methods in the social sciences, but when there is a large amount of data or a lot of research objects, it is really difficult for sociologists to use those traditional research methods. In [20], the author analyzes newsgroup data by hand and only works on a small amount of data. Our approach applies social theories to the model as a tool to automatically find the appropriate micro-laws, so it is a powerful tool for sociologists to use for research.

Due to the growing popularity and interest in social network analysis (SNA), especially because of the booming exposure of online communities, researchers have started to use different methods to help them collect and study the structure of social networks and analyze the ranges/factors of social dynamics. There are significant literatures on modeling and analysis of social networks, [1, 4, 18, 19, 21, 22, 24, 25, 26, 23, 27, 28]. Most of this work focuses on modeling of evolving social networks. In [27], the authors address very simple models that fit in a very limited setting. Our work addresses the problem using a much more general setting. While we present our methodology in the context of a specific model, it can be appropriately extended to any parameterized model. In [28], the authors use Markov chain Monte Carlo (MCMC) methods to simulate how the links among actors evolve. Other works also focus on the links among actors: Latent Space model [31], Dynamic Latent Space model [32], Probabilistic Relational Models (PRMs) [33, 34]. In our model, the actors’ actions are governed by micro-laws justified by social theories. And the changes of the relationships (links) among actors represent the actors’ actions. Based on the paths of the actors’ actions, we can use reverse engineering techniques to discover the appropriate micro-laws in the model.
A lot of literatures use the graph theory and transfer social networks into a graph. Authors in [35, 36, 37, 38, 39, 40] try to solve graph clustering problems and find the community or group structures. They only consider which node should belong to the same groups or display more similarity. In [41, 42], authors present some works on finding the properties of dynamic networks, and they analyze the communication networks based on either the edges of graph or the graph structures. On the other hand, our approach, in addition, considers the interaction among actors (nodes) and between actors and groups (graph structures). In [43], the authors use a decision-tree approach to determine some properties of the social network. The decision-tree approach is a deterministic process, which is different from our approach which uses the stochastic process to determine actors’ behaviors. The reason is that in social networks, even under the same environment, actors do not necessarily possess or reflect the same behaviors. In [44], the authors model the dynamics of customer relationships using HMMs, but there are only three relationship states in the model. Our HMMs include hundreds and thousands of states, and our approaches can more generally fix into any HMMs with a large number of states.

2.2 Hidden Markov Model (HMM)

A hidden Markov model is a statistical model which is assumed to be a Markov process, but the observations are different from the states in Markov process because the states are hidden behind the observations. Leonard E. Baum and others first introduced hidden Markov models in statistical papers [45, 46, 47, 48] in the second half of the 1960s. One of the first applications using hidden Markov models was speech recognition in the mid-1970s. After that, hidden Markov models became more popular, and there are a lot of applications using a hidden Markov model, e.g., bioinformatics, genomics, gesture and body motion recognition, and optical character recognition. In this section, we give a brief discussion about HMMs; for the details about HMMs, please refer to [49, 50, 51].
2.2.1 Definition

We first introduce some notation. The number of states of the model is $N$, and we use $S = \{S_1, S_2, \cdots, S_N\}$ to indicate the individual state, and $q_t$ the state at time $t$. The number of possible observations\(^2\) of the model is $M$, and we denote the individual possible observation as $V = \{v_1, v_2, \cdots, v_M\}$ and the observation at time $t$ as $O_t$. The transition probability distribution of the state is $A = \{a_{ij}\}$, where

$$a_{ij} = P(q_t = S_j|q_{t-1} = S_i), \quad 1 \leq i, j \leq N, \tag{2.1}$$

$a_{ij} \geq 0$ and $\sum_{j=1}^{N} a_{ij} = 1$; sometimes, we also use $a_{q_{t-1}q_t}$ to indicate $a_{ij}$. The probability distribution of the observation in state $j$ is $B = \{b_j(k)\}$, where

$$b_j(k) = P(O_t = v_k|q_t = S_j), \quad 1 \leq j \leq N, 1 \leq k \leq M, \tag{2.2}$$

$b_j(k) \geq 0$ and $\sum_{k=1}^{M} b_j(k) = 1$. The initial state distribution is $\pi = \{\pi_i\}$, where

$$\pi_i = P(q_1 = S_i), \quad 1 \leq i \leq N, \tag{2.3}$$

$\pi_i \geq 0$ and $\sum_{i=1}^{N} \pi_i = 1$. We use the following expression:

$$\lambda = (S, V, A, B, \pi) \tag{2.4}$$

to denote the complete parameter set of an HMM. The process of an HMM is as follows:

\(^2\)Sometimes, the observations are continuous, but here we use discrete observations as example to describe HMMs. The basic concept about HMMs is the same on both.
**Algorithm 2.1**: The process of an HMM

**Input**: $\lambda = (S, V, A, B, \pi)$, and time step $T$

**Output**: Observations $O = O_1O_2 \cdots O_T$

1. Setup the initial state $q_1 = S_i$ based on $\pi$
2. Decide the observation $O_1 = v_k$ based on $b_i(k)$
3. $O \leftarrow \{O_1\}$
4. for $t \leftarrow 2$ to $T$ do
   5. Decide next state $q_t = S_j$ based on $A$
   6. Decide the observation $O_t = v_l$ based on $b_j(l)$
   7. $O \leftarrow O \cup \{O_t\}$
5. endfor
6. return $O$

### 2.2.2 Assumptions

There are three assumptions, *Markov Assumption*, *Stationarity Assumption*, and *Observations Independence Assumption*, made in the theory of HMMs, and we briefly describe them as follows:

**Assumption 2.2.1 (Markov Assumption).** This is the assumption that the current state only depends on the previous state,

$$P(q_t|q_{t-1}, q_{t-2}, \cdots, q_1) = P(q_t|q_{t-1}), \quad (2.5)$$

where the probability of state $q_t$ only depends on the state $q_{t-1}$. The more general case is that the current state depends on the previous $k$ states, called $k$th order HMM,

$$P(q_t|q_{t-1}, q_{t-2}, \cdots, q_1) = P(q_t|q_{t-1}, q_{t-2}, \cdots, q_{t-k}), \quad (2.6)$$

where $t > k$. The first order HMM is the most common, and we use it as the case for the following discussions.
Assumption 2.2.2 (Stationarity Assumption). This is the assumption that the transition probability of states is independent of when it occurs:

\[ P(q_{t_1+1}|q_{t_1}) = P(q_{t_2+1}|q_{t_2}), \quad (2.7) \]

where \( q_{t_1} = q_{t_2} = S_i \) and \( q_{t_1+1} = q_{t_2+1} = S_j \) for any \( i, j, t_1 \) and \( t_2 \).

Assumption 2.2.3 (Observations Independence Assumption). This is the assumption that the current observation is independent of the previous observations. Let \( O = O_1O_2 \cdots O_T \) be a sequence of observations, then according to this assumption, we arrive at the following:

\[ P(O|q_1, q_2, \ldots, q_T, \lambda) = \prod_{t=1}^{T} P(O_t|q_t, \lambda). \quad (2.8) \]

2.2.3 Three Basic Problems of HMMs

When people have HMMs, they used to solve three basic problems\(^3\), evaluation problem, decoding problem, and learning problem. In the following three sections, we describe the three basic problems and the common algorithms to solve them.

2.2.3.1 Evaluation Problem

What is the probability that observations are generated by the model? Given a sequence of observations \( O = O_1O_2 \cdots O_T \) and an HMM \( \lambda = (S, V, A, B, \pi) \), then what is \( P(O|\lambda) \)? We can use the Forward Algorithm, [46, 47, 49], to solve this problem. Define forward variable as we see here:

\[ \alpha_t(i) = P(O_1O_2 \cdots O_t, q_t = S_i|\lambda), \quad 1 \leq i \leq N, \quad (2.9) \]

\(^3\)Jack Ferguson of IDA (Institute for Defense Analysis) characterized the theoretical aspects of HMMs into solving three basic problems.
then,

$$\alpha_{t+1}(j) = P(O_1O_2\cdots O_{t+1}, q_{t+1} = S_j|\lambda) \quad (2.10)$$
$$= \sum_{i=1}^{N} P(O_1O_2\cdots O_tO_{t+1}, q_t = S_i, q_{t+1} = S_j|\lambda) \quad (2.11)$$
$$= \sum_{i=1}^{N} P(O_1O_2\cdots O_t, q_t = S_i|\lambda)a_{ij}b_j(O_{t+1}) \quad (2.12)$$
$$= b_j(O_{t+1}) \sum_{i=1}^{N} \alpha_t(i)a_{ij}, \quad (2.13)$$

and then,

$$P(O|\lambda) = \sum_{i=1}^{N} P(O, q_T = S_i|\lambda) \quad (2.14)$$
$$= \sum_{i=1}^{N} \alpha_T(i). \quad (2.15)$$

We can use dynamic programming as Algorithm 2.2 to compute $P(O|\lambda)$.
Algorithm 2.2: Forward Algorithm

```
\begin{algorithm}
\begin{align*}
\text{Input} & : \lambda = (S, V, A, B, \pi), \text{ and } O = O_1O_2 \cdots O_T \\
\text{Output:} & \ P(O|\lambda) \\
\text{for} & \ i \leftarrow 1 \ \text{to} \ N \ \text{do} \\
\alpha_1(i) & \leftarrow \pi_i b_i(O_1) \\
\text{endfor} \\
\text{for} & \ t \leftarrow 2 \ \text{to} \ T \ \text{do} \\
\text{for} & \ j \leftarrow 1 \ \text{to} \ N \ \text{do} \\
\alpha_t(j) & \leftarrow 0 \\
\text{for} & \ i \leftarrow 1 \ \text{to} \ N \ \text{do} \\
\alpha_t(j) & \leftarrow \alpha_t(j) + \alpha_{t-1}(i) \times a_{ij} \\
\text{endfor} \\
\alpha_t(j) & \leftarrow \alpha_t(j) \times b_j(O_t) \\
\text{endfor} \\
\text{endfor} \\
\text{out} & \leftarrow 0 \\
\text{for} & \ i \leftarrow 1 \ \text{to} \ N \ \text{do} \\
\text{out} & \leftarrow \text{out} + \alpha_T(i) \\
\text{endfor} \\
\text{return} & \ \text{out}
\end{align*}
\end{algorithm}
```

The complexity between lines 1 and 3 in Algorithm 2.2 is $O(N)$, and the complexity between lines 14 and 16 is also $O(N)$. Most calculations in Algorithm 2.2 are between lines 4 and 12, where there are three for-loop’s, and the complexities of each loop are $O(T)$, $O(N)$, and $O(N)$, respectively. Then, the complexity of the Forward Algorithm is $O(TN^2 + 2N) \approx O(TN^2)$.

In the similar argument, we introduce the Backward Algorithm, [46, 47, 49]. Define backward variable as shown here:

$$
\beta_t(j) = P(O_{t+1}O_{t+2} \cdots O_T|q_t = S_j, \lambda), \quad 1 \leq j \leq N, \quad (2.16)
$$
and when $t = T$, $\beta_T(j) = 1$. Then,

$$\beta_{t-1}(i) = P(O_t O_{t+1} \cdots O_T| q_{t-1} = S_i, \lambda)$$

$$= \sum_{j=1}^{N} P(O_t O_{t+1} \cdots O_T| q_{t-1} = S_i, q_t = S_j, \lambda)$$

$$= \sum_{j=1}^{N} P(O_{t+1} O_{t+2} \cdots O_T| q_t = S_j, \lambda) a_{ij} b_j(O_t)$$

$$= \sum_{j=1}^{N} \beta_t(j) a_{ij} b_j(O_t).$$

From Eq. (2.9) and (2.16), we can see what is shown here:

$$\alpha_t(i) \beta_t(i) = P(O, q_t = S_i| \lambda),$$

and then another way to compute $P(O|\lambda)$ can be seen here:

$$P(O|\lambda) = \sum_{i=1}^{N} P(O, q_t = S_i| \lambda)$$

$$= \sum_{i=1}^{N} \alpha_t(i) \beta_t(i).$$

The easy way is to set $t = 1$, then

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_1(i) \beta_1(i)$$

$$= \sum_{i=1}^{N} \pi_i b_i(O_1) \beta_1(i).$$

We can also use dynamic programming for the Backward Algorithm as Algorithm 2.3 to compute $P(O|\lambda)$:
Algorithm 2.3: Backward Algorithm

\[
\text{Input : } \lambda = (S, V, A, B, \pi), \text{ and } O = O_1O_2 \cdots O_T \\
\text{Output: } P(O|\lambda)
\]

1. for \( i \leftarrow 1 \) to \( N \) do
2. \hspace{1em} \( \beta_T(i) \leftarrow 1 \)
3. endfor
4. for \( t \leftarrow T-1 \) to 1 do
5. \hspace{1em} for \( i \leftarrow 1 \) to \( N \) do
6. \hspace{2em} \( \beta_t(i) \leftarrow 0 \)
7. \hspace{2em} for \( j \leftarrow 1 \) to \( N \) do
8. \hspace{3em} \( \beta_t(i) \leftarrow \beta_t(i) + \beta_{t+1}(j) \times a_{ij} \times b_j(O_{t+1}) \)
9. \hspace{2em} endfor
10. endfor
11. endfor
12. \( \text{out} \leftarrow 0 \)
13. for \( i \leftarrow 1 \) to \( N \) do
14. \hspace{1em} \( \text{out} \leftarrow \text{out} + \pi_i \times b_i(O_1) \times \beta_1(i) \)
15. endfor
16. return \( \text{out} \)

The complexity between lines 1 and 3 in Algorithm 2.3 is \( O(N) \), and also is the complexity between lines 13 and 15. Most computation in Algorithm 2.3 are between lines 4 and 11 where includes three for-loop’s, and the complexities of each loop are \( O(T) \), \( O(N) \), and \( O(N) \), respectively. Therefore, the complexity of the Backward Algorithm is also \( O(TN^2 + 2N) \approx O(TN^2) \).

2.2.3.2 Decoding Problem

What is the most likely state sequence in the model that produces the observations? Given a sequence of observations \( O = O_1O_2 \cdots O_T \) and an HMM \( \lambda = (S, V, A, B, \pi) \), then what is most likely state sequence \( Q = q_1q_2 \cdots q_T \)? We can use the Viterbi
Algorithm, [52, 53, 49], to solve this problem. Let

\[
\delta_t(i) = \max_{q_1q_2\cdots q_{t-1}} P(q_1q_2\cdots q_{t-1}, q_t = S_i, O_1O_2\cdots O_t|\lambda), \quad 1 \leq i \leq N, \tag{2.26}
\]

which show the maximum probability for partial state sequence, which ends at state \(S_i\), corresponding to the partial observation sequence. When \(t = 1\), \(\delta_1(i) = \pi_ib_i(O_1)\). Then,

\[
\delta_{t+1}(j) = \max_{q_1q_2\cdots q_{t}} P(q_1q_2\cdots q_{t}, q_{t+1} = S_j, O_1O_2\cdots O_{t+1}|\lambda) \tag{2.27}
\]

\[
= \max_i \max_{q_1q_2\cdots q_{t-1}} P(q_1q_2\cdots q_{t-1}, q_t = S_i, O_1O_2\cdots O_t|\lambda)a_{ij}b_j(O_{t+1}) \tag{2.28}
\]

\[
= \max_i \delta_t(i)a_{ij}b_j(O_{t+1}) \tag{2.29}
\]

\[
= b_j(O_{t+1})\max_i \delta_t(i)a_{ij}. \tag{2.30}
\]

We can use dynamic programming as Algorithm 2.4 to compute \(\delta_t(i)\) and also output the most likely state sequence \(Q\). The complexities from lines 2 to 4 and from lines 21 to 24 in Algorithm 2.4 are \(O(2N)\). When we use the same arguments as Algorithms 2.2 and 2.3, the complexity between lines 5 and 18 is \(O(TN^2)\). Then, from the above, we know the complexity of the Viterbi Algorithm is also \(O(TN^2 + 2N) \approx O(TN^2)\).
Algorithm 2.4: Viterbi Algorithm

**Input**: \( \lambda = (S, V, A, B, \pi) \), and \( O = O_1O_2 \cdots O_T \)

**Output**: \( Q = q_1q_2 \cdots q_T \)

1. Let \( Q(t, i) \) as a table to store the best hidden state ending with state \( S_i \) at time \( t \).

2. for \( i \leftarrow 1 \) to \( N \) do

3. \( \delta_1(i) \leftarrow \pi_i b_i(O_1) \)

4. endfor

5. for \( t \leftarrow 2 \) to \( T \) do

6. for \( j \leftarrow 1 \) to \( N \) do

7. \( \max_{\text{prob}} \leftarrow -1 \)

8. for \( i \leftarrow 1 \) to \( N \) do

9. \( \text{prob} = \delta_{t-1}(i) a_{ij} \)

10. if \( \text{prob} > \max_{\text{prob}} \) then

11. \( \max_{\text{prob}} \leftarrow \text{prob} \)

12. \( \max_{\text{state}} \leftarrow S_i \)

13. endif

14. endfor

15. \( \delta_t(j) \leftarrow \max_{\text{prob}} \times b_j(O_t) \)

16. \( Q(t, j) \leftarrow \max_{\text{state}} \)

17. endfor

18. \( i^* \leftarrow \arg\max_i \delta_T(i) \)

19. \( \text{out} \leftarrow S_{i^*} \), where \( \text{out} \) is a stack structure for output.

20. for \( t \leftarrow T \) to \( 2 \) do

21. \( \text{out} \leftarrow \text{out} + Q(t, i^*) \)

22. \( i^* \leftarrow Q(t, i^*) \)

23. endfor

24. return \( \text{out} \)
2.2.3.3 Learning Problem

How should we adjust the model parameters? Given a sequence of observations \( O = O_1O_2 \cdots O_T \), then how do we adjust \( A, B, \pi \) to maximize \( P(O|S, V, A, B, \pi) \)? So far, there is no any algorithm which is able to optimize \( \lambda = (S, V, A, B, \pi) \) and find the global maximum of \( P(O|\lambda) \). However, we can use the Baum-Welch Algorithm, [52, 53, 49] or gradient techniques [54] to find the local maximum of \( P(O|\lambda) \). Here, we briefly present the Baum-Welch Algorithm and discuss the complexity. In the Baum-Welch Algorithm, it defines the following:

\[
\xi_t(i, j) = P(q_t = S_i, q_{t+1} = S_j, O|\lambda), \quad 1 \leq i,j \leq N, 1 \leq t < T, \tag{2.31}
\]

use the forward variable, Eq. (2.9), and the backward variable, Eq. (2.16), then

\[
\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)}. \tag{2.32}
\]

It also defines the following:

\[
\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i, j), \quad 1 \leq i \leq N, 1 \leq t < T. \tag{2.33}
\]

Then the estimations of \( \pi, A, \) and \( B \) are as follows:

\[
\bar{\pi}_i = \gamma_1(i), \tag{2.34}
\]

\[
\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}, \tag{2.35}
\]

\[
\bar{b}_j(k) = \frac{\sum_{t=1}^{T-1} \text{s.t. } O_t = v_k \gamma_t(j)}{\sum_{t=1}^{T-1} \gamma_t(j)}. \tag{2.36}
\]

We can use Algorithms 2.2 and 2.3 to calculate \( P(O|\lambda) \) and all \( \alpha_t(i) \) and \( \beta_t(i) \) where \( 1 \leq t \leq T \) and \( 1 \leq i \leq N \), and store all those values. We know the complexities for both Algorithms 2.2 and 2.3 are the same, \( O(TN^2) \). After we know the values of \( P(O|\lambda) \), \( \alpha_t(i) \) and \( \beta_t(i) \), the complexity for computing one \( \xi_t(i, j) \) in Eq. (2.32) is only constant,
$O(1)$, and the complexity for computing all $\xi_t(i, j)$ where $1 \leq t < T$ and $1 \leq i, j \leq N$ is $O(TN^2)$. When we know the values of $\xi_t(i, j)$, the complexity for calculating one $\gamma_t(i)$ in Eq. (2.33) is $O(N)$, and the complexity for calculating all $\gamma_t(i)$ where $1 \leq t < T$ and $1 \leq i \leq N$ is also $O(TN^2)$. After we know all the values of $\xi_t(i, j)$ and $\gamma_t(i)$, the complexities for computing all $\bar{\pi}_i$, $\bar{a}_{ij}$, and $\bar{b}_j(k)$, where $1 \leq i, j \leq N$, $1 \leq k \leq M$, in Eq. (2.34), (2.35), and (2.36) are $O(N^2)$, $O(TN^2)$, and $O(TN^2)$, respectively. So the complexity for the Baum-Welch Algorithm is also $O(TN^2)$. 
CHAPTER 3

ViSAGE—A HMM FOR VIRTUAL SIMULATION AND ANALYSIS OF GROUP EVOLUTION

We give the details of the agent-based hidden Markov model, ViSAGE\(^4\) (Virtual Simulation and Analysis of Group Evolution), which includes the probabilistic evolving social group model and the probabilistic communication model. The justification of ViSAGE through social capital theories, [4, 5, 6, 7], can be found in [3, 8, 9, 10, 11].

The foundation of the evolving social group model is a Markov process, and Figure 3.1 shows the general framework for the step-by-step evolution in the model. There are actors and social groups, and the state of the society is defined by properties of the actors and groups. There are three kinds of actions—join a group, leave a group, and do nothing. Based on the current state of the society, each actor decides which action he/she most likely wants to execute, which is known as the Normative Action. Nonetheless, under the influence of the present communities, actors’ actions could be affected. After being influenced, each actor decides to perform the Choice Action and eventually performs the Real Action. Depending on the feedbacks from actors’ Normative Action and Choice Action, the properties of actors and groups are updated accordingly.

We also develop a probabilistic communication model to simulate communication

\(^{4}\)The structure of ViSAGE is derived from the model described in [55]

---

**Figure 3.1:** Framework for the step by step evolution in the social group model.

---

20
networks based on the *state* of the society. In the communication model, we only consider the communication edges\(^5\) between actors and do not look at the communication content. Combining the evolving social group model with the communication model, the whole process represents a hidden Markov model (HMM). Figure 3.2 shows the use of ViSAGE to generate group evolution and communications data. In the real world, it is easy to observe communication network, but the *states* of the society is difficult to observe and remains the hidden driver of the communication network.

People can use ViSAGE as a tool for studying emergent behavior in social networks, and Figure 3.3 shows the general framework of using ViSAGE. To illustrate the

---

\(^5\)A communication edge indicates a communication link between two actors, which includes the information about the sender, receiver, and date.
value of ViSAGE, consider the case of designing an educational program to instill the values and skills of entrepreneurship and technology management in ethnic minorities. These values and skills could be taught to students through classroom learning and a team business plan project over the course of a multi-week, immersive educational program. The purpose of using ViSAGE would be to study changes in the values of the students by modeling the dynamics of the social groups and their networks. In so doing, we would gather communications and social network data among students and faculty using direct observation, participant reporting, interviews, and automated logging of online communications. We would use this data to identify social groups and the characteristics of these groups as they change over time. This data is then reverse engineered using ViSAGE to categorize actors into two behavioral categories called ambition and leadership. The distribution and population of actors with different ambition and leadership configurations serve as our distributed theories of practice for the community from which our data was gathered. Micro-laws drive the evolution of the social group data and are a hypothesis about the real human practices that are a major component of how social groups develop. The result of our work would be presented to those responsible for the design and implementation of the educational program.

Thus, there are three modes in which ViSAGE could be used: (1) as a tool for understanding the behavior of a society given its micro-laws; (2) as a tool to validate postulated micro-laws against observed evolution by simulating forward and comparing with the data; and (3) as a tool for determining the micro-laws by searching for those parameters that, through simulation, can be validated against the data.

In this chapter, we present the details of the hidden Markov process and the general framework of ViSAGE. Figure 3.4 shows the flowchart of the hidden Markov process, where the solid lines show the deterministic process and the dash lines show the stochastic process, and the details of the functions, variables, and configuration in the default setting can be found in Appendixes A and B.

3.1 State of The Society

In the evolving social group model, many parameters govern how actors decide whether to join a group, to leave a group, or to do nothing, and also which group the actor
Figure 3.4: The flowchart of ViSAGE.

desires to join or to leave; i.e., parameters such as the group memberships, how long each actor has been in a group, the ranks of the actors, and the amount of resources available to an actor (in excess of what is currently being used to maintain the actor’s current group
memberships). Thus, the state of the society at time \( t \) can be defined as we see here:

\[
S_t = \{ \text{type}_i, r_t^i, R_t^i, \{ G_t^l \}_{i \in G_t^l} \}_{i=1}^N,
\]

where \( N \) indicates the number of actors, \( \text{type}_i \) is a vector and shows actor \( i \)'s types, \( r_t^i \) is a vector and indicates actor \( i \)'s rank in each group at time \( t \), \( R_t^i \) is actor \( i \)'s resources at time \( t \), \( G_t^l \) is a social group \( l \) at time \( t \), and \( \{ G_t^l \}_{i \in G_t^l} \) is a set of groups actor \( i \) joined at time \( t \). In the following sections, we describe the properties of actors and groups, and the default setting can be found in Appendixes A.1, A.2, and A.3.

### 3.1.1 Type

Each actor has a private set of attributes, which we refer to as its type. In the model, there are two parameters defining an actor’s type: the size-type (stype) controls the actor’s group size preferences, and the ambition-type (atype) controls actor’s “ambition” (how quickly his/her rank increases in a group). There are three values for an actor’s size-type: \( Type_S \) prefers small groups, \( Type_L \) prefers large groups, and \( Type_M \) prefers the group size in the middle. And there are also three values for an actor’s ambition-type: \( Type_A \) is most ambitious, \( Type_C \) is least ambitious, and \( Type_B \) is in the middle.

### 3.1.2 Rank

Each actor has a rank in each group to represent the actor’s position in the group. As actors spend more time in a group, their positions in the group change. There is a tendency for more senior members of a group to have a higher position within the group than junior members. Also, \( Type_A \) actors have a highest tendency to increase their positions within the group, and \( Type_C \) actors lowest. The notation \( r_{t(i,l)}^i \) is for actor \( i \)'s rank in group \( G_t^l \) at time \( t \), and the vector \( r_t^i \) is the set of ranks of actor \( i \) in all groups at time \( t \). The definition of rank is as follows:

\[
\text{time}_{t(i,l)}^i \delta_i \sum_{j \in G_t^l} \text{time}_{t(j,l)}^j \delta_j,
\]

where we use notation \( G_t^l \) for a social group \( l \) at time \( t \), \( \text{time}_{t(i,l)}^i \) for the amount of time that actor \( i \) has spent in group \( G_t^l \) at time \( t \), and \( \delta_i \) indicates how quickly actor \( i \)'s rank increases in a group for each time unit. \( \delta_i \) is a function of actor \( i \)'s ambition-type—\( Type_A \)
has the biggest value of $\delta_i$, $Type_C$ has the smallest value, and $Type_B$ has the value in the middle. In addition, when an actor leaves a group, his/her rank in that group will gradually decrease, as shown here:

\[
time_{(i,l)}^t = \max \left\{ 0, \time_{(i,l)}^{t-1} - \rho \right\},
\]

where $\rho$ is an amount of time an actor lost in the total joining time for each time step when an actor leaves a group.

### 3.1.3 Qualification

Each actor has a qualification ($q_i^t$) to represent an actor’s prestige. It is determined as the average rank of an actor among the groups in which he/she has been a member, and the rank is weighted to give a stronger weight to ranks from larger groups. The qualification of an actor is used for an actor to determine which group he/she is more likely to join or leave. The definition of qualification is as follows:

\[
q_i^t = \frac{\sum_{i \in G_l^t} r_{(i,l)}^t |G_l^t|}{\sum_{i \in G_l^t} |G_l^t|},
\]

where $|G_l^t|$ denotes the size of group $l$ at time $t$.

Similarly, each group has its qualification ($Q_l^t$) defined as the average qualification of actors currently participating in the group. The higher a group’s qualification, the more attractive it will appear to other actors looking for a group to join. The qualification of group $G_l^t$ is defined by this formula:

\[
Q_l^t = \sum_{i \in G_l^t} q_i^t \times r_{(i,l)}^t.
\]

### 3.1.4 Resources

Each actor has a certain amount of available resources ($R_i^t$) at each time step, and the initial available resources is $R_i^0$. The available resources mean the internal mental and physical resources or capacity each actor has to engage in social activities, and they are
used to maintain a membership in a group. We also define the excess resources \( (e_i) \) as

\[
e_i = R_i - \phi_i L_{br}^i - \psi_i L_{bo}^i,
\]

(3.6)

where \( \phi_i L_{br}^i \) and \( \psi_i L_{bo}^i \) indicate the Bridging Cost and Bonding Cost [8], respectively—how many resources an actor needs to maintain a membership in a group. Bridging Cost is related to Bridging Social Capital—the number of groups actor \( i \) is in,

\[
L_{br}^i = |G : i \in G|,
\]

(3.7)

and Bonding Cost is related to Bonding Social Capital—the sum of the ranks of actor \( i \),

\[
L_{bo}^i = \sum_{G_1: i \in G_1} r_{(i,l)}^t.
\]

(3.8)

The parameters \( \phi_i \) and \( \psi_i \) are weight parameters, and they are functions of \( R_i \). The excess resources influence what kind of action an actor is going to perform at the next time step, described in Section 3.2.1.

### 3.2 State Transitions

At each time step, every actor needs to decide whether to leave one group, join one group, or remain in the same groups. Then, based on actors’ actions, the state of the society transitions. In the following section, we describe the Markov process, and the default setting can be found in Appendixes A.4, A.5, A.6, and A.7.

#### 3.2.1 Actor’s Normative Actions

In the first step, each actor needs to decide his/her Normative Action, the action he/she most likely wants to take. The decision depends on an actor’s excess resources \( (e_i) \). If \( e_i \) is positive, the actor will tend to use the excess resources in joining another group, if \( e_i \) is negative, the actor will tend to leave a group in order to lessen the cost needed, and if \( e_i \) is near zero, the actor may decide to remain in all the same groups for the current time step. Based on the amount of excess resources, we can determine the probabilities for actor \( i \) to join, to leave, or to do nothing, and we use the notations \( P_{+1}(e_i) \), \( P_0(e_i) \),
and $P_{-1}(e^t_i)$ to denote the probabilities, respectively, where

$$
\sum_{k=-1}^{+1} P_k(e^t_i) = 1. \tag{3.9}
$$

The definition of the *Normative* action is what we see here:

$$
Act_{norm} = \arg\max_{\text{action}} \left( P_{\text{action}}(e^t_i) \right), \tag{3.10}
$$

where $P_{\text{action}} \in \{P_{+1}, P_0, P_{-1}\}$.

### 3.2.2 Actors’ Real Actions

Ideally, the actor would always choose to perform the *Normative* action, since this creates a state of stability. However, we assume that the actors sometimes make non-rational decisions, regardless of the amount of excess resources they have. An actor chooses an action he/she is going to perform based on a stochastic process as follows:

$$
Act_{\text{choice}} = \text{random} \left( P_{+1}, P_0, P_{-1} \right), \tag{3.11}
$$

where $\text{random} \left( P_{+1}, P_0, P_{-1} \right)$ means randomly choose an action based on the probabilities of actions, $P_{+1}(e^t_i)$, $P_0(e^t_i)$, and $P_{-1}(e^t_i)$.

After an actor has chosen which action he/she would like to perform, he/she needs to decide which group to “apply to” (want to join) or to leave. If the action is to leave, the actor selects one group from the groups he/she has joined to leave. And for joining a group, an actor learns from his/her neighbors which groups out there he/she is able to join, and this process is denoted by “joining by reference”. Specifically, the set of groups known to an actor are the groups to which the actor’s neighbors belong.

**Definition 3.2.1 (Join by reference).** Let $a_i$ and $a_j$ denote actor $i$ and actor $j$, and the set of candidate groups actor $i$ knows and would like to join at time $t$ is what is shown here:

$$
D^r_e = \left\{ G^t_k \mid \text{for some } j, l, a_i, a_j \in G^t_i, a_j \in G^t_k, a_i \notin G^t_k \right\}, \tag{3.12}
$$

where $k$ indexes the groups in the reference set.
In addition, there is a small chance that an actor would like to adventure the whole world; the candidate groups are all the groups in the world, as seen here:

\[ D^r_c = \{ \text{All groups in the world.} \} \]  

(3.13)

We use \( P_r \) to denote the probability that an actor would like to join by reference and \( P_e \) to denote the probability that an actor would like to adventure the world, and

\[ P_r + P_e = 1. \]  

(3.14)

After identifying the set of candidate groups, we need to determine the probability to join or to leave each candidate group. An actor takes into account the size of the group during decision making based on his/her group size preference. The size preference is defined by a function of the actor’s size-type and the group’s size as follows:

\[ \text{SizeAff}_t^{i,l} = \text{Function}(\text{size-type}_i, |G_t^l|), \]  

(3.15)

which indicates the size preference for actor \( i \) to group \( l \) at time \( t \). The probability of an actor joining or leaving a group also depends on the qualification of the group and the actor’s qualifications and ranks. Actor \( i \)’s qualification affinity for group \( l \) is defined as follows:

\[ \text{QualAff}_t^{i,l} = \text{Function}(q_t^i, Q_t^l). \]  

(3.16)

The idea is that an actor would like to join a group which has a higher qualification than his/her qualification, and he/she is more likely to select a group which he/she has lower rank to leave. Then, the probability that actor \( i \) would like to select group \( l \) to join or to leave at time \( t \) is defined as follows:

\[ P_{\text{Join}}_t^{i,l} = \text{Function}(\text{SizeAff}_t^{i,l}, \text{QualAff}_t^{i,l}) , \]  

(3.17)

\[ P_{\text{Leave}}_t^{i,l} = \text{Function}(r_t^i, \text{SizeAff}_t^{i,l}) , \]  

(3.18)

where \( r_t^i \) denotes the set of ranks of actor \( i \) in all his/her groups at time \( t \). If actor \( i \) chooses group \( l \) to join, the group can either accept or reject the actor’s application based a
stochastic process, which is related to the group’s qualification and the actor’s qualification. Intuitively, a group would like to accept an actor with higher qualification and to reject an actor with lower qualification. Group $l$’s actor affinity for actor $i$ is defined by a function of group $l$’s qualification and actor $i$’s qualification as follows:

$$\text{ActorAff}^t_l(i) = (Q^t_l, q^t_i).$$

(3.19)

The probability of group $l$ rejecting actor $i$ at time $t$ is defined as follows:

$$P\text{Reject}^t_l(i) = \text{Function (ActorAff}^t_l(i)).$$

(3.20)

Thus, the probability actor $i$ really joins group $l$ is what is shown here:

$$\text{PRJoin}^t_l(i) = \text{PJoin}^t_l(i) 	imes (1 - \text{PReject}^t_l(i)).$$

(3.21)

All the above stochastic processes result in the Real action of the actor. This process mimics the real life process of using one’s energy/resources to join groups, apply to groups and possibly be rejected.

**Lemma 3.2.2 (The probabilities of Real action).** Let $\tilde{P}^t_{+1}(e_i^t)$, $\tilde{P}^t_{-1}(e_i^t)$, and $\tilde{P}^t_0(e_i^t)$ indicate the probabilities of actor $i$’s Real action at time $t$ to join a group, to leave a group, or to do nothing, respectively. Then,

$$\tilde{P}^t_{+1}(e_i^t) = P^t_{+1}(e_i^t) \times \left(1 - P_r \sum_{G_l^i \in D^e_r} P\text{Join}^t_l(i) \times \text{PReject}^t_l(i)\right)$$

(3.22)

$$\tilde{P}^t_{-1}(e_i^t) = \begin{cases} 0, & \text{if actor } i \text{ is in no group} \\ P^t_{-1}(e_i^t), & \text{otherwise} \end{cases}$$

(3.23)

$$\tilde{P}^t_0(e_i^t) = 1 - \tilde{P}^t_{+1}(e_i^t) - \tilde{P}^t_{-1}(e_i^t)$$

(3.24)

where $D^e_r$ and $D^e_c$ denote the sets of two kinds of candidate groups, and $P_r$ and $P_{\bar{r}}$ are the
probabilities for one kind of candidate group selected respectively. \( G_i^t \) denotes the groups in candidate groups to which actor \( i \) would like to apply.

### 3.2.3 State Update

The final step of the process at each time step is to update the properties of actors and groups including Rank, Qualification, and Resources. To update Rank and Qualification of actors and groups, we use Eq. (3.2), (3.4), and (3.5). The change of available resources is related to an actor’s Normative action, Choice action, and the society reward/penalty parameters \( \theta_{reward} \). Intuitively, an actor is rewarded or penalized based on how closely his/her Choice action is to the action expected by society (the Normative action). The reward/penalty parameters \( \theta_{reward} \) are used in the Agency and Structure Table to determine how to update an actor’s resources. In Table 3.1, \( \theta_{reward} = \{\omega_\alpha | \alpha = 1, 2, \ldots, 9\} \), based on \( Act_{norm} \) and \( Act_{choice} \), we decide which \( \omega_\alpha \) is used to update the available resources.

| “Agency” \( Act_{choice} \) | “Structure” \( Act_{norm} \) |
|---|---|---|
| join | \( \omega_1 \) | \( \omega_4 \) | \( \omega_7 \) |
| stay | \( \omega_2 \) | \( \omega_5 \) | \( \omega_8 \) |
| leave | \( \omega_3 \) | \( \omega_6 \) | \( \omega_9 \) |

The formula for updating the available resources is summarized heuristically by

\[
\Delta R_i^t = f_{\Delta R} \left( actions, R_i^t, \theta_{action}, \theta_{reward} \right),
\]

(3.25)

where \( \theta_{action} \) indicates some parameters related to actors’ actions, and \( f_{\Delta R} \) is the returned social capital function indicating the formulas of updating resources, and \( f_{\Delta R} \) is shown as follows:

\[
f_{\Delta R} = PenaltyW \left( actions, \theta_{action}, \theta_{reward} \right) \times \text{Penalty}(R_i^t)
+ RewardW \left( actions, \theta_{action}, \theta_{reward} \right) \times \text{Reward}(R_i^t),
\]

(3.26)
where functions $\text{Reward}()$ and $\text{Penalty}()$ return positive valuables for the quantity of reward or penalty, function $\text{RewardW}()$ returns a positive valuable for the weight of reward, and function $\text{PenaltyW}()$ returns a negative valuable for the weight of penalty. The choice of $\omega_\alpha$ in $\theta_{\text{reward}}$ using in $f_{\Delta R}$ is according to an actor’s Normative action and Choice action. Then, the available resources at next time step is shown here:

$$R_{i+1}^t = R_i^t + \Delta R_i^t. \tag{3.27}$$

### 3.3 Communications

We also developed a probabilistic communication model to simulate social networks and to produce the communication edges among actors. The communication model is based on the state of the society without considering the semantics of the messages. The basic idea is that the more groups two actors have in common, the higher probability these two actors should communicate with each other; however, if two actors have no group in common, they still have a small chance to communicate with each other. A more general model also considers actors’ friends; if two actors are not in the same group but they have a common friend (2nd level friendship), then there is another probability for this kind of communication. We can also consider how many levels of the friendship, e.g., friend of friends in common (3rd level friendship), or friend of friend of friends in common (4th level friendship).

**Definition 3.3.1 (The $n$th level friendship).** Consider that a group structure is an unweighted graph, $G_g = (\mathcal{A}, E_g)$, where $\mathcal{A}$ is a set of nodes (actors), and $E_g$ is a set of edges—an edge between two actors $a_i$ and $a_j$, denoted by $\varepsilon_g(i, j)$, indicates that $a_i$ and $a_j$ are in the same group. Two actors $a_i$ and $a_j$ have a $n$th level friendship if and only if there exists a distinct path\(^6\) between $a_i$ and $a_j$, and the number of edges (length) of that path is $n$.

**Definition 3.3.2 (The number of $n$th level friendship).** Let’s consider a group structure, an unweighted graph $G_g = (\mathcal{A}, E_g)$, where $\mathcal{A}$ is a set of nodes (actors), and $E_g$ is a set of edges. An edge between two actors $a_i$ and $a_j$ indicates that $a_i$ and $a_j$ are in the same

\(^6\)In a distinct path, each edge cannot duplicate.
group. Then, the number of n-th level friendship is the number of distinct paths between a_i and a_j, and the lengths of those paths are n.

From Definitions 3.3.1 and 3.3.2, we know a_i and a_j can have different levels and different numbers of friendship. And based on our communication model, each friendship creates a chance that there is a communication between a_i and a_j. Therefore, we can compute the the probabilities of the communication between any pair of actors.

**Definition 3.3.3 (The probability of communication).** Let \( P_l \) denote the probability of communications in an l-th level friendship; \( P_0 \) refers to the probability of communications between two actors who have no friendship. Let \( F_l(i,j) \) be the number of l-th level friendships between a_i and a_j, \( P_e(i,j) \) the probability of communication between a_i and a_j, and \( \bar{P}_e(i,j) \) the probability of no communication between a_i and a_j. Then, we have

\[
P_e(i,j) + \bar{P}_e(i,j) = 1.
\]

(3.28)

Assume the maximum level of friendship is \( L \). Then, the probability of communication between a_i and a_j

\[
P_e(i,j) = \begin{cases} 
P_0 & \text{, if } a_i, a_j \text{ have no friendship}, \\
1 - \prod_{l=1}^{L} (1 - P_l)^{F_l(i,j)} & \text{, otherwise}. 
\end{cases}
\]

(3.29)

Let |\( \mathcal{A} \)| be the number of actors, and |\( \mathcal{A} \)| = \( N \). The algorithm to compute the number of each level, [1st, \( \cdots \), Lth] level, of friendship and the probabilities of communication between any pair of actors (\( P_e(i,j) \)), is shown in Algorithm 3.1. The algorithm uses depth-first search (DFS), [56], to find the friendships among actors and the probabilities to communicate.
Algorithm 3.1: Compute the number of friendship and the probabilities of communications

Input : $G_g = (A, E_g)$, where $|A| = N$
Output: $F_l(i, j)$ and $P_e(i, j)$ for all $l, i, and j$

1 function Friendship ($l, i, k, \tilde{E}_g$) {
2 if $l > L$ then return
3 for $j = 1$ to $N$ do
4     if $j \neq i, j \neq k$ and $\varepsilon_g(j, k) \in \tilde{E}_g$ then
5         $F_l(i, j) \leftarrow F_l(i, j) + 1$
6         $\tilde{P}_e(i, j) \leftarrow \tilde{P}_e(i, j) \times (1 - P_l)$
7         Friendship ($l+1, i, j, \tilde{E}_g - \varepsilon_g(j, k)$)
8     endif
9 endfor
10 }endfunc

11 forall the $i$ and $j$ do
12     $\tilde{P}_e(i, j) \leftarrow 1$
13     forall the $l$ do
14         $F_l(i, j) \leftarrow 0$
15     endfall
16 endfall

17 for $i = 1$ to $N$ do
18     Friendship ($1, i, i, E_g$)
19 endfor

20 forall the $i$ and $j$ do
21     if $\tilde{P}_e(i, j) = 1$ then $P_e(i, j) = P_0$
22     else $P_e(i, j) \leftarrow 1 - \tilde{P}_e(i, j)$
23 endfall
24 return $F_l(i, j)$ and $P_e(i, j)$

Lines 1 to 10 present a function to find the number of $l$th level friendship and compute the $\tilde{P}_e(i, j)$. In a path, each edge in $E_g$ can’t be repeated. The argument $\tilde{E}_g$ is a set of
edges which have not appeared in the path from \( a_i \) to \( a_k \), and \( \tilde{E}_g \subseteq E_g \). Line 7 calls the function itself and goes to next level of friendship. Lines 11 to 16 do the initialization, and lines 17 to 19 find the friendship related to actor \( i \). Lines 20 to 23 compute \( P_e(i, j) \), and line 24 returns the number of friendship in each level \( (F_l(i, j)) \) and the probabilities of communications between \( a_i \) and \( a_j \) \( (P_e(i, j)) \). We know the function \( \text{Friendship}() \), lines 1 to 10, called itself \( L \) times, and the time complexity for each time is \( O(N) \). So, the time complexity for the function \( \text{Friendship}() \) is \( O(N^L) \). In the main loop, lines 17 to 19, calls the function \( \text{Friendship}() \) \( N \) times, so the time complexity of Algorithm 3.1 is \( O(N^{L+1}) \).

Based on those probabilities of communication \( (P_e(i, j)) \), we create communication edges among actors. The Algorithm 3.2 shows the communication model creating communication edges, where \( G_c \) is a communication graph, \( E_c \) is a set of communication edges, and \( \varepsilon_c(i, j) \) denotes the communication edge between \( a_i \) and \( a_j \).

**Algorithm 3.2: The communication model**

**Input**: actors \( \mathcal{A} \) and \( P_e(i, j) \), where \( |\mathcal{A}| = N \)

**Output**: \( G_c = (\mathcal{A}, E_c) \)

1. \( E_c \leftarrow \emptyset \)

2. for \( i=1 \) to \( N \) do

3. 

4. 

5. \( E_c \leftarrow E_c \cup \varepsilon_c(i, j) \)

6. endfor

7. endfor

8. return \( G_c = (\mathcal{A}, E_c) \)

The time complexity of Algorithm 3.2 is \( O(N^2) \) because there are two for loop between lines 2 and 8. Combining Algorithms 3.1 and 3.2, the time complexity of the whole communication model is \( O(N^{L+1} + N^2) \).
3.4 Example of ViSAGE

Figure 3.5 gives an illustration of an evolving community from time $t$ to $t + 1$. We use this example to describe the essential process of ViSAGE, which easily extends to an arbitrary number of actors and groups. In this example (Figure 3.5), there are five actors, $a_1, a_2, a_3, a_4$ and $a_5$, and three social groups $G_1, G_2$ and $G_3$ at time $t$ and $t + 1$. We use $G^t_l$ for social group $l$ at time $t$. Let's focus on actor $a_1$ at time $t$. Some of the properties of actor $a_1$ have been indicated: $type_1$, $r^t_1$, and $R^t_1$. As indicated, $r^t_{(1,1)}$ depends on $f_{\Delta R}(JOIN, R^t_1, \theta_{action}, \theta_{reward})$.

Enter Group $(Q, a_2, a_4, \theta_{group})$, where $Q = \{Q^t_2, Q^t_3, |G^t_2|, |G^t_3|, \{r^t_2, r^t_3\}, \{r^t_4, r^t_5\}, r^t_1, type_1\}$. 

Figure 3.5: An example of ViSAGE.
ambition-type$_1$ through how long the actor $a_1$ has been in the group $G_1$, and $r_{(1,1)}^t$ also depends on the ranks of the other actors, $a_2$ and $a_4$, in the group, through the fact that the sum of all ranks of actors in a group is 1,

$$\sum_{i \in G_1^t} r_{(i,t)}^t = 1.$$  

(3.30)

Thus, $r_{(1,1)}^t$ indirectly depends on ambition-type$_2$ and ambition-type$_4$. Based on the properties of actor $a_1$, he/she decides to join a new group through the stochastic process denoted by $Action()$ in Figure 3.5, which depends on a set of parameters $\theta_{action}$; two other possible actions are to leave a group or to do nothing.

Having decided to join a group, actor $a_1$ must now decide which specific group to join. This is accomplished by a stochastic hand-shaking process in which $a_1$ decides to which group to “apply”, $G_2$ in this case, and $G_2$ decides whether or not to accept $a_1$ into the group. This process is indicated by $Group()$ in Figure 3.5 and is governed by its own set of parameters $\theta_{group}$, together with the properties of some of the other actors ($a_2, a_4$ in this case) and the group structure. Actor $a_1$ learns about which other groups are out there to join through his/her neighbors $a_2, a_4$ (i.e., the groups they belong to, and thus only applies to other groups by reference; the potential groups to join by reference are $G_2$ and $G_3$ in this case). Actor $a_1$ then decides to which group to apply based on his/her qualification, as measured by his/her average rank in all his/her groups, and the qualification thresholds and sizes of the groups. A group also decides whether to accept $a_1$ based on similar properties. In the example, $a_1$ applied to join $G_2$ and in this particular case was accepted.

The resources of $a_1$ now get updated by some reward, through a stochastic process denoted by $f_{\Delta R}()$ in Figure 3.5, which additionally depends on the actual action, available resources, and some parameters $\theta_{reward}$. This process is analogous to society encouragement or penalty for doing expected or unexpected things. The abstract quantity we denote by resources combines things like social capital, time, energy, money, etc. After all actors go through a similar process in batch mode and independently of each other, the entire state of the society is updated in a feedback loop as indicated in Figures 3.1 and 3.4 to obtain the state at time $t + 1$. 
CHAPTER 4
REVERSE ENGINEERING

Our approach uses an agent-based hidden Markov model, and the common learning algorithms for solving problems within a hidden Markov model are the Forward-backard Algorithm [47], Viterbi Algorithm [52], and Baum-Welch Algorithm [46], briefly described in Section 2.2. The time complexities of these three algorithms are the same, $O(TM^2)$, where $T$ is the total time steps, and $M$ is the number of states.

**Lemma 4.0.1 (The number of states in ViSAGE).** In ViSAGE, if there are $N$ actors and the maximum number of groups in a society is $K$, then, at each time step, the number of possible states is $\Omega(2^{NK})$.

**Proof:** From Eq. (3.1), we know each state includes all actors’ types, ranks, available resources, and which groups they are in. Consider the term $\left\{G_i \mid i \in G_t\right\}_{i=1}^N$ in Eq. (3.1). We know there are a maximum of $K$ groups, and each actor has two possibilities for each group: to be either in a group or not in a group. Thus, there are $2^K$ possible group structures for each actor. There are total $N$ actors, so the number of possible group structure is $(2^K)^N = 2^{NK}$. Without considering other factors in the state, the number of the possible state is already $2^{NK}$.

**Lemma 4.0.2 (The number of observed states in ViSAGE).** In ViSAGE, if there are $N$ actors, then, at each time step, the number of possible observations is $O(N^2 - N)$.

**Proof:** The observations in ViSAGE are presented by the communication dynamics. Therefore, we need to count how many communication links are connected. If there are $N$ actors, then there are $\binom{N}{2}$ possible pairs of actors in the communication dynamics, and each pair has two possible of communication states—connected or disconnected. Hence, the number of possible observations from a communication dynamics is as we see here:

$$\binom{N}{2} \times 2 = N^2 - N.$$
If we have a data set for $T$ time steps, the complexities of using the above algorithms to solve the three basic problems in ViSAGE are $\Omega(T \times 2^{NK})$, which is exponential computation time and not practically feasible. Therefore, we have developed a multistage learning process (see Figure 4.1) to learn the group structure, the group evolution, and the appropriate parameters in the model in a more efficient way. In our multistage learning process, we divide the whole learning process into three stages. In the first stage, we find the group structure at each time step based on the communication networks. In the second stage, we discover the group evolution using the data of group structure. In the last stage, we learn from the group evolution to identify the appropriate parameters (micro-laws) in ViSAGE.

4.1 Learning from Communications

The challenge with real data is that the group structure and the group evolution are not known, especially in online communities. Instead, one observes the communication dynamics, and Figure 4.2 illustrates an example of a data set including only communication dynamics. However, the communication dynamics are indicative of the group
dynamics, since a pair of actors who are in many groups together are likely to communicate often. One could place one more probabilistic layer on the model linking the group structure to the communications; however, the state space for this hidden Markov model would be prohibitive. From Lemma 4.0.1 and 4.0.2, we know that the number of states is $\Omega(2^{NK})$ and the number of observed states is $O(N^2 - N)$. We therefore opt for a simpler approach. The first stage in learning is to use the communication dynamics to construct the set of groups. Figure 4.3 shows the same example with the group structure inferred from the communications in Figure 4.2.

In our communication model (see Section 3.3), let $i, j$ refer to actors, and let $P_e(i, j)$ be the probability that two actors $a_i$ and $a_j$ communicate. Let $x_{ij}$ be a boolean value, equal to 1 if the communication between $a_i$ and $a_j$ exists. Then, the probability of observing
the communication graph is what we see here:

\[ P_{\text{rob}} = \prod_{i,j} P_e(i, j)^{x_{ij}}(1 - P_e(i, j))^{(1-x_{ij})}, \quad (4.1) \]

where \( P_e(i, j) \), Eq. (3.29), is

\[
P_e(i, j) = \begin{cases} 
P_0, & \text{if } a_i, a_j \text{ have no friendship.} \\
1 - \prod_{l=1}^{L} (1 - P_l)^{F_l(i, j)} , & \text{otherwise.}
\end{cases} \quad (4.2)
\]

We would like to infer \( P_0, P_1, \cdots, P_L \) so as to maximize the probability of observing the graph; in fact, we need to find appropriate \( P_e(i, j) \) to maximize Eq. (4.1) where \( x_{ij} \) for all \( i \) and \( j \) are known from communication data. Solving for \( P_e(i, j) \) means that we need to find the optimal solution for \( P_0, \{P_l\} \), and \( \{F_l(i, j)\} \), where \( l = 1, \cdots, L \). Imagine that communications among the actors are aggregated over some time period \( \tau \) to obtain a weighted communication graph \( G_c(\tau) \). The actors \( \mathcal{A} \) is a set of nodes in \( G_c(\tau) \) and the edge weight \( w_{ij} \) between two actors is the communication intensity (number of communications) between \( a_i \) and \( a_j \). In the unweighted graph, all non-zero \( w_{ij} \)’s are 1. The sub-task we would like to solve is to infer the group structure from the communication graph \( G_c(\tau) \). Any reasonable formulation of this problem is NP-hard, [35], and it only solves the value of \( \{F_l(i, j)\} \). So, we need some efficient heuristic for finding the appropriate solution for Eq. (4.1).

In our approach, we have developed a two-step method to find and verify the approximate solution. In the first step, we find the clusters in a graph that correspond to the social groups. In particular, the clusters should be allowed to overlap, as is natural for social groups. This excludes most of the traditional clustering algorithms, which partition the graph. There are some papers presenting the overlapping clustering algorithms, [35, 57, 58, 59, 60]. We use the algorithms developed by Jeffery Baumes et al., [35], which efficiently find overlapping communities in a communication graph. We consider time periods \( \tau_1, \tau_2, \cdots, \tau_{T+1} \) and the corresponding communication graphs \( G_c(\tau_1), G_c(\tau_2), \cdots, G_c(\tau_{T+1}) \). The time periods need not be disjointed; in fact, we have used overlapping time periods to cluster the communication data into groups. Since there
is considerable noise in the communications—aggregation, therefore, we use overlapping
time periods to smoothen the time series of communication graphs. This is a way of ob-
taining more stable and more reliable group structure estimates for learning on real data.
Given a single graph $G_c(\tau_t)$, the algorithm in [35] outputs a set of overlapping clusters,
$D_t$ (a set of groups at time step $t$).

After knowing the group structure ($D_t$), we can find $x_{ij}$ and $F_l(i, j)$ for all $l$, $i$, and $j$. In the second step, we can solve for $P_0$ and $\{P_l\}$ to maximize Eq. (4.1). In this way,
we can verify how well the overlapping algorithm works with the communication model.
Instead of maximizing Eq. (4.1), we maximize the logarithm of Eq. (4.1), as shown here:

$$\log \text{Prob} = \sum_{i,j} (x_{ij} \log P_e(i,j) + (1-x_{ij}) \log(1 - P_e(i,j))) .$$  (4.3)

In the Eq. (4.2), we know there are two cases for $P_e(i,j)$. Let $i = j$ indicate that there is
a friendship between actor $i$ and actor $j$; contrarily, $i \neq j$ indicates no friendship between
them. Let

$$\Upsilon_{ij} = x_{ij} \log P_e(i,j) + (1-x_{ij}) \log(1 - P_e(i,j)),$$  (4.4)

and then, Eq. (4.3) becomes

$$\log \text{Prob} = \sum_{i \neq j} \Upsilon_{ij} + \sum_{i = j} \Upsilon_{ij}. \quad (4.5)$$

We know that $P_0$ is independent of $P_l$’s, and in Eq. (4.5), the first term, $i \neq j$, is related
to $P_0$, and the second term, $i = j$, is related to $P_l$’s; therefore, we can work on them
separately to maximize $\log \text{Prob}$. Here, we are going to work on both cases, $i \neq j$
(without friendship) and $i = j$ (with friendship).

**Without friendship.** When $i \neq j$, we need work on the first term in Eq. (4.5), and
$P_e(i,j) = P_0$. Then, we get

$$\sum_{i \neq j} \Upsilon_{ij} = \sum_{i \neq j} (x_{ij} \log P_0 + (1-x_{ij}) \log(1 - P_0)). \quad (4.6)$$

We know $x_{ij}$ is a boolean value which indicates if there is a communication between
actor $i$ and actor $j$. Let $N_e$ be the number of pair of actors who have a communi-
cation without friendship, and $N_n$ be the number of pair of actors with neither a communication nor friendship. Then, Eq. (4.6) becomes

$$\sum_{i \neq j} \Upsilon_{ij} = N_e \times \log P_0 + N_n \times \log(1 - P_0).$$  \hspace{1cm} (4.7)

We take a derivative of Eq. (4.7) with respect to $P_0$ to find the solution maximizing Eq. (4.7),

$$d \sum_{i \neq j} \Upsilon_{ij} \frac{d}{dP_0} = N_e \times \frac{1}{P_0} - N_n \times \frac{1}{1 - P_0}.$$  \hspace{1cm} (4.8)

Setting this expression to 0, and we get the following equation:

$$P_0 = \frac{N_e}{N_e + N_n}.$$  \hspace{1cm} (4.9)

**With friendship.** When $i = j$, we need to work on the second term in Eq. (4.5), and it is related to the $P_l$’s. From Eq. (4.2) and (4.4), we know that:

$$\sum_{i=j} \Upsilon_{ij} = \sum_{i=j} (x_{ij} \log P_e(i, j) + (1 - x_{ij}) \log(1 - P_e(i, j))),$$  \hspace{1cm} (4.10)

where

$$P_e(i, j) = 1 - \prod_{l=1}^{L} (1 - P_l)^{F_l(i, j)}.$$  \hspace{1cm} (4.11)

Let $P_l = 1 - \bar{P}_l$, and then, Eq. (4.11) becomes

$$P_e(i, j) = 1 - \prod_{l=1}^{L} \bar{P}_l^{F_l(i, j)}.$$  \hspace{1cm} (4.12)

We use gradient method to maximize the Eq. (4.10). Taking a derivative of Eq. (4.10) with respect to $\bar{P}_k$, and we get the following equation:

$$\frac{\partial}{\partial \bar{P}_k} \sum_{i=j} \Upsilon_{ij} = \sum_{i=j} \left( x_{ij} \frac{\partial}{\partial \bar{P}_k} \log \left( 1 - \prod_{l=1}^{L} \bar{P}_l^{F_l(i, j)} \right) + (1 - x_{ij}) \frac{\partial}{\partial \bar{P}_k} \log \prod_{l=1}^{L} \bar{P}_l^{F_l(i, j)} \right).$$  \hspace{1cm} (4.13)
where

\[
\frac{\partial \log \left(1 - \prod_{l=1}^{L} \bar{P}_{l}^{F_l(i,j)}\right)}{\partial \bar{P}_k} = \frac{1}{1 - \prod_{l=1}^{L} \bar{P}_{l}^{F_l(i,j)}} \times \frac{\partial \left(1 - \prod_{l=1}^{L} \bar{P}_{l}^{F_l(i,j)}\right)}{\partial \bar{P}_k},
\]

(4.14)

\[
= \frac{- \prod_{l \neq k} \bar{P}_{l}^{F_l(i,j)}}{1 - \prod_{l=1}^{L} \bar{P}_{l}^{F_l(i,j)}} \times \frac{\partial \bar{P}_{k}^{F_k(i,j)}}{\partial \bar{P}_k},
\]

(4.15)

\[
= \frac{- \prod_{l \neq k} \bar{P}_{l}^{F_l(i,j)}}{1 - \prod_{l=1}^{L} \bar{P}_{l}^{F_l(i,j)}} \times \bar{F}_{k}(i,j) \bar{P}_{k}^{F_k(i,j)-1},
\]

(4.16)

and

\[
\frac{\partial \log \prod_{l=1}^{L} \bar{P}_{l}^{F_l(i,j)}}{\partial \bar{P}_k} = \frac{\partial \sum_{l=1}^{L} \log \bar{P}_{l}^{F_l(i,j)}}{\partial \bar{P}_k}
\]

(4.17)

\[
= \frac{\bar{F}_{k}(i,j)}{\bar{P}_k}.
\]

(4.18)

From the group structure, we know the values of \(x_{ij}\) and \(F_l(i,j)\) for all \(i, j,\) and \(l,\)
and then we can compute the derivative of Eq. (4.10) within the reasonable range of \(\bar{P}_k, [0.0, 1.0].\)

### 4.2 Learning from Group Structure

From the previous stage, we have a set of group structures \(D_t, t = 1, \ldots, T,\) like Figure 4.3. However, in order to use the learning method in the next stage, one needs to construct the paths of each actor. This means we need the correspondence between groups of time step \(t\) and \(t + 1,\) in order to determine the actors’ actions. Formally, we need a match between the groups in \(D_t\) and \(D_{t+1}\) for \(t = 1, \ldots, T - 1,\) (see Figure 4.4): for each group in \(D_t,\) we must identify the corresponding group in \(D_{t+1}\) to which it evolved. If there are more groups in \(D_{t+1},\) then some new groups arose. If there are fewer groups in \(D_{t+1},\) then some of the existing groups disappeared.
We define the matching problem as follows:

**Definition 4.2.1 (Finding matchings).** Let $\mathcal{X} = \{X_1, \ldots, X_n\}$ and $\mathcal{Y} = \{Y_1, \ldots, Y_n\}$ be two collections of sets, and we allow some of the sets in $\mathcal{X}$ or $\mathcal{Y}$ to be empty. We use the symmetric set difference shown here:

$$d(X_i, Y_j) = 1 - \frac{|X_i \cap Y_j|}{|X_i \cup Y_j|}$$

(4.19)

as a measure of error between two sets. Then, we consider the complete bipartite graph on $(\mathcal{X}, \mathcal{Y})$ and would like to construct a match of minimum total weight, where the weight on the edge $(X_i, Y_j)$ is $d(X_i, Y_j)$.

This problem can be solved in cubic time using max-flow techniques, [61]. However, for our purposes, this is too slow, so we use a simple greedy heuristic. First find the best match, the pair $(i^*, j^*)$ which minimizes $d(X_i, Y_j)$ over all pairs $(i, j)$. This pair is removed from the sets and the process continues. An efficient implementation of this greedy approach can be done in $O(n^2 \log n)$, after $d(X_i, Y_j)$ has been computed for each pair $(i, j)$. The algorithm is shown as Algorithm 4.1.
Algorithm 4.1: Finding matchings using a greedy algorithm

**Input**: $\mathcal{X} = \{X_1, \ldots, X_n\}$ and $\mathcal{Y} = \{Y_1, \ldots, Y_n\}$

**Output**: Pairs of matching groups.

1. $S \leftarrow \emptyset$
2. $M \leftarrow \emptyset$
3. foreach $X_i \in \mathcal{X}$ do
4.   foreach $Y_j \in \mathcal{Y}$ do
5.     Compute $d(X_i, Y_j)$
6.     $S \leftarrow S \cup \{i, j, d(X_i, Y_j)\}$
7.   endfch
8. endfch
9. Sort $S$ based on $d(X_i, Y_j)$
10. while $S \neq \emptyset$ do
11.     Find pair $(i^*, j^*)$ which minimizes $d(X_i, Y_j)$
12.     $M \leftarrow M \cup \{i^*, j^*, d(X_{i^*}, Y_{j^*})\}$
13.     $S \leftarrow S - \{k, l, d(X_k, Y_l)\}$ where $k = i^*$ or $l = j^*$
14. endw
15. return $M$

4.3 Learning from Group Evolution

In this section, we discuss the learning process from data sets with group evolution (see Figure 4.4). We first introduce some notation. The set of actors is $\mathcal{A}$; we use $i, j, k$ to refer to actors. The data $\mathcal{D} = \{\mathcal{D}_t\}_{t=1}^{T+1}$ is the set of social groups at each time step, where each $\mathcal{D}_t$ is a set of groups, $\mathcal{D}_t = \{G^t_i\}_i$, $G^t_i \subseteq \mathcal{A}$; we use $l, m, n$ to refer to groups. We collect all the parameters which specify the model as $\Theta_M$, which includes all the parameters specific to an actor (e.g., type, rank, qualification) and all the global parameters in the model (e.g., $\theta_{\text{action}}, \theta_{\text{reward}}, \theta_{\text{group}}$). We would like to maximize the likelihood,

$$L(\Theta_M) = \text{Prob}(\mathcal{D}|\Theta_M).$$

(4.20)
We define the path of actor $i$, $p_i^T = (p_i(1), \ldots, p_i(T))$, as the set of actions he/she took over the time steps $t = 1, \ldots, T$. The action at time $t$, $p_i(t)$, constitutes a decision to join, to leave or to do nothing, and also which group actor $i$ joins or leaves. Given $D$, we can construct $p_i^T$ for every actor $i$, and conversely, given $\{p_i^T\}_{i=1}^{|A|}$, we can reconstruct $D$. Therefore, we can alternatively maximize the following equation:

$$L(\Theta_M) = Prob \left( p_1^T, \ldots, p_{|A|}^T | \Theta_M \right).$$

(4.21)

It is this form of the likelihood that we manipulate. Typical ways to break up this combinatorial optimization is to iteratively improve the discrete parameters and the continuous parameters. The main problem we face is an algorithmic one—namely that typically, the number of actors, $|A|$ is very large (thousands or tens of thousands), as is the number of time steps, $T$, (hundreds). From the viewpoint of actor $i$, we break down $\Theta_M$ into three types of parameters:

1. $\theta_i$: the parameters specific to actor $i$, e.g., types, ranks, qualification and resources;
2. $\theta_i$: the parameters specific to other actors; and
3. $\theta_G$: the parameters of the society, global to all the actors.

The optimization style is iterative in the following sense. Fixing parameters specific to actors, one can optimize with respect to $\theta_G$. Since $\theta_G$ is a fixed number of parameters, this process is algorithmically feasible. We now consider optimization with respect to $\theta_i$ with fixing $\theta_i$ and $\theta_G$. This is the task which is algorithmically non-trivial, since there are $\Omega(|A|)$ such parameters; therefore, we need to develop some efficient algorithms to optimize $\theta_i$.

In ViSAGE, actors at each time step take independent actions. At time $t$, the state of the society $S_t$ (see Eq. (3.1)) can be summarized by actors’ types, actors’ ranks in each group, actors’ available resources, and the group structure. Given $S_t$, each actor acts independently at time $t$. Thus, we can write the following equation:

$$L(\Theta_M) = Prob \left( p_1^{T-1}, \ldots, p_{|A|}^{T-1} | \Theta_M \right) \times \prod_{i=1}^{|A|} Prob \left( p_i(T) | \Theta_M, S_t \right)$$

(4.22)
Continuing in this way, by induction, Eq. (4.22) becomes

\[ \mathcal{L}(\Theta_M) = \prod_i \prod_t \text{Prob} (p_i(t)|\Theta_M, S_t). \]  

(4.23)

Instead, we maximize the log-likelihood \( \ell \),

\[ \ell(\Theta_M) = \sum_i \sum_t \log \text{Prob} (p_i(t)|\Theta_M, S_t). \]  

(4.24)

From the evolution of groups, we can observe actors’ Real actions (\( Act_{real} \)) and which groups they join and leave. We use \( Act^i_{real}(t) \) to denote actor \( i \)'s Real action at time \( t \), then we have the following equation:

\[ \text{Prob} (p_i(t)|\Theta_M, S_t) = \text{Prob} (Act^i_{real}(t)|\Theta_M, S_t). \]  

(4.25)

In the following arguments, all the conditional probabilities are given \( \Theta_M \) and \( S_t \). So we omit \( \Theta_M \) and \( S_t \) and simplify the expression as shown here:

\[ \text{Prob} (p_i(t)) = \text{Prob} (Act^i_{real}(t)). \]  

(4.26)

We know there are three possible actions, and let \( J, L, \) and \( S \) denote the actions, Join, Leave, and Do Nothing, respectively. We also use \( J_l \) and \( L_l \) to denote the actions, join group \( l \) and leave group \( l \), respectively. \( Act^i_{choice}(t) \) denotes actor \( i \)'s Choice action at time \( t \) (see Section 3.2.2), which means the action actor \( i \) is going to perform at time \( t \).

**Lemma 4.3.1.** The conditional probability of actor \( i \)'s observed action to join group \( l \) at time \( t \), given \( Act^i_{choice}(t) = J \), is as follows:

\[ \text{Prob} (Act^i_{real}(t) = J_l|Act^i_{choice}(t) = J) = P_r \times PRJoin^i_{(i,l)} + P_t \times \bar{PRJoin}^i_{(i,l)}, \]  

(4.27)

where the \( P_r \) and \( P_t \) denote the probabilities of selecting the set of candidate groups, \( D^r \) and \( D^t \), respectively, and \( PRJoin^i_{(i,l)} \) and \( \bar{PRJoin}^i_{(i,l)} \) are the probabilities that actor \( i \) really join group \( l \) selected from different set of candidate groups.
**Proof:** Applying the law of total probability [62], we can write the following equation:

\[ Prob(\text{Act}_i^{\text{real}}(t) = J_l|\text{Act}_i^{\text{choice}}(t) = J) = \sum_{G_k^i \in D_c}(Prob(\text{Act}_i^{\text{choice}}(t) = J_k|\text{Act}_i^{\text{choice}}(t) = J) \times Prob(\text{Act}_i^{\text{real}}(t) = J_l|\text{Act}_i^{\text{choice}}(t) = J_k, \text{Act}_i^{\text{choice}}(t) = J)) \]  

(4.28)

where \(D_c\) is the set of candidate groups actor \(i\) would like to join, and \(k\) indexes the groups in the set. From Section 3.2.2, we know there are two kinds of candidate groups, \(D_c^r\) and \(D_c^p\), which are selected with the probabilities \(P_r\) and \(P_p\), respectively. Then, Eq. (4.28) becomes the following equation:

\[ Prob(\text{Act}_i^{\text{real}}(t) = J_l|\text{Act}_i^{\text{choice}}(t) = J) = P_r \sum_{G_k^i \in D_c^r}(Prob(\text{Act}_i^{\text{choice}}(t) = J_k|\text{Act}_i^{\text{choice}}(t) = J) \times Prob(\text{Act}_i^{\text{real}}(t) = J_l|\text{Act}_i^{\text{choice}}(t) = J_k, \text{Act}_i^{\text{choice}}(t) = J)) + P_p \sum_{G_k^i \in D_c^p}(Prob(\text{Act}_i^{\text{choice}}(t) = J_k|\text{Act}_i^{\text{choice}}(t) = J) \times Prob(\text{Act}_i^{\text{real}}(t) = J_l|\text{Act}_i^{\text{choice}}(t) = J_k, \text{Act}_i^{\text{choice}}(t) = J)) . \]

(4.29)

We know when actor \(i\) would like to join group \(l\), it is impossible to observe that actor \(i\) joins a group other than group \(l\). Therefore, in Eq. (4.29), when \(k \neq l\),

\[ Prob(\text{Act}_i^{\text{real}}(t) = J_l|\text{Act}_i^{\text{choice}}(t) = J_k \neq l, \text{Act}_i^{\text{choice}}(t) = J) = 0. \]

(4.30)

Then, Eq. (4.29) becomes

\[ Prob(\text{Act}_i^{\text{real}}(t) = J_l|\text{Act}_i^{\text{choice}}(t) = J) = P_r \times Prob(\text{Act}_i^{\text{choice}}(t) = J_k, G_k^t \in D_c^r|\text{Act}_i^{\text{choice}}(t) = J) \times Prob(\text{Act}_i^{\text{real}}(t) = J_l|\text{Act}_i^{\text{choice}}(t) = J_k, \text{Act}_i^{\text{choice}}(t) = J) + P_p \times Prob(\text{Act}_i^{\text{choice}}(t) = J_k, G_k^t \in D_c^p|\text{Act}_i^{\text{choice}}(t) = J) \times Prob(\text{Act}_i^{\text{real}}(t) = J_l|\text{Act}_i^{\text{choice}}(t) = J_k, \text{Act}_i^{\text{choice}}(t) = J) . \]

(4.31)

And from Section 3.2 and Eq. (3.17), we know the probabilities of actor \(i\) who would like to join and joins group \(l\) are shown here:

\[ Prob(\text{Act}_i^{\text{choice}}(t) = J_l, G_k^t \in D_c^r|\text{Act}_i^{\text{choice}}(t) = J) = P\text{Join}_{(i,l)}^t, \]

(4.32)
where $PJoin^i_{(i,l)}$ and $\bar{P}Join^i_{(i,l)}$ are the probabilities that actor $i$ would like to join group $l$ selected from a different set of candidate groups, $D^c_r$ or $D^\bar{c}_r$. However, when actor $i$ decides to join group $l$, there is a probability that group $l$ may reject actor $i$. Therefore, the probability of actor $i$ really to join group $l$ is what we see here:

$$\text{Prob} (\text{Act}^i_{real}(t) = J_l | \text{Act}^i_{choice}(t) = J) = 1 - P\text{Reject}^i_{(i,l)}.$$

(4.34)

$$\text{Prob} (\text{Act}^i_{real}(t) = J_l | \text{Act}^i_{choice}(t) = J) = 1 - \bar{P}\text{Reject}^i_{(i,l)}.$$

(4.35)

Then, Eq. (4.31) becomes the following equation:

$$\text{Prob} (\text{Act}^i_{real}(t) = J | \text{Act}^i_{choice}(t) = J) = P_r \times PJoin^i_{(i,l)} \times (1 - P\text{Reject}^i_{(i,l)}) + P_\bar{r} \times \bar{P}Join^i_{(i,l)} \times (1 - \bar{P}\text{Reject}^i_{(i,l)}).$$

(4.36)

Lemma 4.3.2. The conditional probability of actor $i$’s observed action to leave group $l$ at time $t$, given $\text{Act}^i_{choice}(t) = L$, is as follows:

$$\text{Prob} (\text{Act}^i_{real}(t) = L_l | \text{Act}^i_{choice}(t) = L) = P\text{Leave}^i_{(i,l)},$$

(4.37)

where $P\text{Leave}^i_{(i,l)}$ is the probability actor $i$ leaves group $l$.

Proof: Applying the law of total probability, we can write the following equation:

$$\text{Prob} (\text{Act}^i_{real}(t) = L_l | \text{Act}^i_{choice}(t) = L) = \sum_{G_k \in D_r} (\text{Prob} (\text{Act}^i_{choice}(t) = L_k | \text{Act}^i_{choice}(t) = L) \\ \times \text{Prob} (\text{Act}^i_{real}(t) = L_l | \text{Act}^i_{choice}(t) = L_k, \text{Act}^i_{choice}(t) = L)),

(4.38)

where $D_r$ is the set of groups in which actor $i$ is a member, and $k$ indexes the groups in the set. When actor $i$ would like to leave group $l$, we won’t observe that actor $i$ leaves a
group other than group \( l \). Therefore, in Eq. (4.38), when \( k \neq l \),

\[
Prob (\text{Act}_\text{real}^i(t) = L_l | \text{Act}_\text{choice}^i(t) = L_{k \neq l}, \text{Act}_\text{choice}^i(t) = L) = 0. \tag{4.39}
\]

Then, we can write Eq. (4.38) as follows:

\[
Prob (\text{Act}_\text{real}^i(t) = L_l | \text{Act}_\text{choice}^i(t) = L)
= Prob (\text{Act}_\text{choice}^i(t) = L_l | \text{Act}_\text{choice}^i(t) = L)
\times Prob (\text{Act}_\text{real}^i(t) = L_l | \text{Act}_\text{choice}^i(t) = L_l, \text{Act}_\text{choice}^i(t) = L), \tag{4.40}
\]

and from Section 3.2 and Eq. (3.18), we know the probability of actor \( i \) who would like to leave and leaves group \( l \) is what we see here:

\[
Prob (\text{Act}_\text{choice}^i(t) = L_l | \text{Act}_\text{choice}^i(t) = L) = \text{PLeave}_t^i(i, l), \tag{4.41}
\]

and the probability of actor \( i \) who decides to leave group \( l \) and his/her observed action is also to leave group \( l \) is the following equation:

\[
Prob (\text{Act}_\text{real}^i(t) = L_l | \text{Act}_\text{choice}^i(t) = L_l, \text{Act}_\text{choice}^i(t) = L) = 1. \tag{4.42}
\]

Then Eq. (4.40) becomes what is shown here:

\[
Prob (\text{Act}_\text{real}^i(t) = L_l | \text{Act}_\text{choice}^i(t) = L) = \text{PLeave}_t^i(i, l). \tag{4.43}
\]

**Lemma 4.3.3.** The conditional probability of actor \( i \)'s observed action as Do Nothing at time \( t \), given \( \text{Act}_\text{choice}^i(t) = J \), is as follows:

\[
Prob (\text{Act}_\text{real}^i(t) = S | \text{Act}_\text{choice}^i(t) = J)
= P_r \sum_{G' \in D'^c} P_{\text{Join}}^t(i, k) \times P_{\text{Reject}}^t(i, k) + P_f \sum_{G' \in D'^f} P_{\text{Join}}^t(i, k) \times P_{\text{Reject}}^t(i, k), \tag{4.44}
\]

where the \( P_r \) and \( P_f \) denote the probabilities of selecting the set of candidate groups, \( D'^c \) and \( D'^f \), respectively. \( P_{\text{Join}}^t(i, j) \) and \( P_{\text{Join}}^t(j, i) \) are the probabilities that actor \( i \) would like to join group \( l \). \( P_{\text{Reject}}^t(i, j) \) and \( P_{\text{Reject}}^t(j, i) \) are the probabilities that group \( l \) rejects
actor $i$.

**Proof:** Applying the *law of total probability* and similar to Lemma 4.3.1, we can write the following equation:

$$
\text{Prob}(\text{Act}_i^{\text{real}}(t) = S | \text{Act}_i^{\text{choice}}(t) = J) \\
= P_r \sum_{G_t^i \in D_r^c} (\text{Prob}(\text{Act}_i^{\text{choice}}(t) = J_k | \text{Act}_i^{\text{choice}}(t) = J) \\
\times \text{Prob}(\text{Act}_i^{\text{real}}(t) = S | \text{Act}_i^{\text{choice}}(t) = J_k, \text{Act}_i^{\text{choice}}(t) = J)) \\
+ P_r \sum_{G_t^i \in D_r^c} (\text{Prob}(\text{Act}_i^{\text{choice}}(t) = J_k | \text{Act}_i^{\text{choice}}(t) = J) \\
\times \text{Prob}(\text{Act}_i^{\text{real}}(t) = S | \text{Act}_i^{\text{choice}}(t) = J_k, \text{Act}_i^{\text{choice}}(t) = J)) .
$$

(4.45)

When $\text{Act}_i^{\text{real}}(t) = S$, we don’t know which group rejects actor $i$, so we need to compute the probabilities for all groups in $D_r^c$ and $D_r^c$. From Eq. (4.32) and (4.33), we know:

$$
\text{Prob}(\text{Act}_i^{\text{choice}}(t) = J_k | \text{Act}_i^{\text{choice}}(t) = J) = \begin{cases} 
P_{\text{Join}}^{t, (i,k)}(J), & \text{if } G^i_t \in D_r^c. \\
\bar{P}_{\text{Join}}^{t, (i,k)}, & \text{if } G^i_t \in D_r^c.
\end{cases}
$$

(4.46)

and from Section 3.2 and Eq. (3.20), the probability of actor $i$ who would like to join group $k$ but is rejected is what is shown here:

$$
\text{Prob}(\text{Act}_i^{\text{real}}(t) = S | \text{Act}_i^{\text{choice}}(t) = J_k, \text{Act}_i^{\text{choice}}(t) = J) = \begin{cases} 
P_{\text{Reject}}^{t, (i,k)}(J), & \text{if } G^i_t \in D_r^c. \\
\bar{P}_{\text{Reject}}^{t, (i,k)}, & \text{if } G^i_t \in D_r^c.
\end{cases}
$$

(4.47)

**Lemma 4.3.4.** The conditional probability of actor $i$’s observed action as Do Nothing at time $t$, given $\text{Act}_i^{\text{choice}}(t) = L$, is as follows:

$$
\text{Prob}(\text{Act}_i^{\text{real}}(t) = S | \text{Act}_i^{\text{choice}}(t) = L) = \begin{cases} 
1, & \text{if actor } i \text{ is not in any group.} \\
0, & \text{otherwise.}
\end{cases}
$$

(4.48)

**Proof:** If actor $i$ is in group $l$ and $\text{Act}_i^{\text{choice}}(t) = L_l$, then actor $i$ definitely leaves group $l$. On the other hand, if actor $i$ is not in any group, then the observed action of actor $i$ is Do Nothing.
Theorem 4.3.5 (The probability of the observed action). The probability of the observed action of actor \( i \) at time \( t \) is as follows:

\[
\begin{align*}
\text{Prob}(\text{Act}^i_{\text{real}}(t)) &= \begin{cases} 
\text{Prob}(\text{Act}^i_{\text{real}}(t) = J_l), & \text{if Act}^i_{\text{real}}(t) \text{ is to join group } l \\
\text{Prob}(\text{Act}^i_{\text{real}}(t) = L_l), & \text{if Act}^i_{\text{real}}(t) \text{ is to leave group } l \\
\text{Prob}(\text{Act}^i_{\text{real}}(t) = S), & \text{if Act}^i_{\text{real}}(t) \text{ is to do nothing},
\end{cases}
\end{align*}
\]

where

\[
\begin{align*}
\text{Prob}(\text{Act}^i_{\text{real}}(t) = J_l) &= P_{+1}(e^i_t) \times (P_r \times \text{PRJoin}^t_{(i,l)} + P_{\bar{r}} \times \text{PRJoin}^t_{(i,l)}), \\
\text{Prob}(\text{Act}^i_{\text{real}}(t) = L_l) &= P_{-1}(e^i_t) \times \text{PL} \text{ave}^t_{(i,l)}, \\
\text{Prob}(\text{Act}^i_{\text{real}}(t) = S) &= \text{ProbSJ}^t_i + \text{ProbSL}^t_i + \text{ProbSS}^t_i.
\end{align*}
\]

In Eq. (4.52),

\[
\begin{align*}
\text{ProbSJ}^t_i &= P_{+1}(e^i_t) \times \left( P_r \sum_{G^i_k \in D^i} \text{PJoin}^t_{(i,k)} \times \text{PReject}^t_{(i,k)} \\
&\quad + P_{\bar{r}} \sum_{G^i_k \in D^i} \bar{P} \text{Join}^t_{(i,k)} \times \bar{P} \text{Reject}^t_{(i,k)} \right), \\
\text{ProbSL}^t_i &= \begin{cases} 
P_{-1}(e^i_t), & \text{if actor } i \text{ is not in any group} \\
0, & \text{otherwise}
\end{cases}, \\
\text{and} \\
\text{ProbSS}^t_i &= P_0(e^i_t).
\end{align*}
\]

Proof: Applying the law of total probability, we can write this equation:

\[
\begin{align*}
\text{Prob}(\text{Act}^i_{\text{real}}(t)) &= \text{Prob}(\text{Act}^i_{\text{choice}}(t) = J) \text{Prob}(\text{Act}^i_{\text{real}}(t) | \text{Act}^i_{\text{choice}}(t) = J) \\
&\quad + \text{Prob}(\text{Act}^i_{\text{choice}}(t) = L) \text{Prob}(\text{Act}^i_{\text{real}}(t) | \text{Act}^i_{\text{choice}}(t) = L) \\
&\quad + \text{Prob}(\text{Act}^i_{\text{choice}}(t) = S) \text{Prob}(\text{Act}^i_{\text{real}}(t) | \text{Act}^i_{\text{choice}}(t) = S).
\end{align*}
\]
Based on actor $i$’s three possible $Act_{real}$, we discuss each case as follows:

1. When $Act_{real}^i(t) = J$,

$$
Prob (Act_{real}^i(t) = J | Act_{choice}^i(t) = L) = 0,
Prob (Act_{real}^i(t) = J | Act_{choice}^i(t) = S) = 0.
$$

Then, from Eq. (4.56), we can write:

$$
Prob (Act_{real}^i(t) = J) = Prob (Act_{choice}^i(t) = J) \\
\times Prob (Act_{real}^i(t) = J | Act_{choice}^i(t) = J).
$$

(4.57)

From Section 3.2, we know that:

$$
Prob (Act_{choice}^i(t) = J) = P_{+1}(e^i_t),
$$

(4.58)

where $P_{+1}(e^i_t)$ is the probability that actor $i$ would like to join a group at time $t$, and from Lemma 4.3.1, we can write:

$$
Prob (Act_{real}^i(t) = J) = P_{+1}(e^i_t) \times (P_r \times PRJoin^i_{(i,l)} + P_r \times \bar{PRJoin}^i_{(i,l)}).
$$

(4.59)

2. When $Act_{real}^i(t) = L$,

$$
Prob (Act_{real}^i(t) = L | Act_{choice}^i(t) = J) = 0,
Prob (Act_{real}^i(t) = L | Act_{choice}^i(t) = S) = 0.
$$

Then, Eq. (4.56) becomes the following equation:

$$
Prob (Act_{real}^i(t) = L) = Prob (Act_{choice}^i(t) = L) \\
\times Prob (Act_{real}^i(t) = L | Act_{choice}^i(t) = L).
$$

(4.60)
From Section 3.2, we know:

\[
Prob( Act_{\text{choice}}^i(t) = L ) = P_{-1}(e_i^t), \tag{4.61}
\]

where \( P_{-1}(e_i^t) \) is the probability that actor \( i \) is going to leave a group, and from Lemma 4.3.2, Eq. (4.60) becomes the following equation:

\[
Prob( Act_{\text{real}}^i(t) = L_i ) = P_{-1}(e_i^t) \times P_{\text{Leave}}^{t(i,l)}, \tag{4.62}
\]

where \( P_{\text{Leave}}^{t(i,l)} \) is the probability that actor \( i \) chooses group \( l \) to leave.

3. When \( Act_{\text{real}}^i(t) = S \), each possible \( Act_{\text{choice}} \) has the possibility to observe the real action as \( Do \text{ Nothing} \). Here, we discuss all three possible cases.

(a) When an actor’s \( Act_{\text{choice}} = J \), but the group rejects the actor, and then we will observe the actor’s real action as \( Do \text{ Nothing} \). Let \( ProbSJ_i^t \) indicate the probability that actor \( i \) would like to join a group, but the group rejects actor \( i \). Then,

\[
ProbSJ_i^t = Prob( Act_{\text{choice}}^i(t) = J ) \times Prob( Act_{\text{real}}^i(t) = S | Act_{\text{choice}}^i(t) = J ). \tag{4.63}
\]

From Eq. (4.58) and Lemma 4.3.3, we can write:

\[
ProbSJ_i^t = P_{+1}(e_i^t) \times \left( P \sum_{G_k^i \in D^i_e} P_{\text{Join}}^{t(i,k)} \times P_{\text{Reject}}^{t(i,k)} + P \sum_{G_k^i \in D^i_e} \bar{P}_{\text{Join}}^{t(i,k)} \times \bar{P}_{\text{Reject}}^{t(i,k)} \right) \tag{4.64}
\]

(b) When an actor’s \( Act_{\text{choice}} = L \), but he/she is not in any group, and then we will also observe the real action as \( Do \text{ Nothing} \). Let \( ProbSL_i^t \) indicate the probability that actor \( i \), who is not in any group, would like to leave a group.
Then,

\[
\text{Prob}_t^{SL_i} = \text{Prob} \left( \text{Act}_\text{choice}^i(t) = L \right) \times \text{Prob} \left( \text{Act}_\text{real}^i(t) = S | \text{Act}_\text{choice}^i(t) = L \right). \tag{4.65}
\]

From Eq. (4.61) and Lemma 4.3.4, we can write:

\[
\text{Prob}_t^{SL_i} = \begin{cases} 
P_{-1}(e_i^t) & \text{if actor } i \text{ is not in any group.} \\
0 & \text{otherwise.} 
\end{cases} \tag{4.66}
\]

(c) When an actor’s \( \text{Act}_\text{choice}^i = S \), his/her observed action is definitely \textit{Do Nothing}. Let \( \text{Prob}_t^{SS_i} \) indicate the probability that actor \( i \) would like to do nothing and his/her observed action is \textit{Do Nothing}. Then,

\[
\text{Prob}_t^{SS_i} = \text{Prob} \left( \text{Act}_\text{choice}^i(t) = S \right) \times \text{Prob} \left( \text{Act}_\text{real}^i(t) = S | \text{Act}_\text{choice}^i(t) = S \right), \tag{4.67}
\]

where

\[
\text{Prob} \left( \text{Act}_\text{real}^i(t) = S | \text{Act}_\text{choice}^i(t) = S \right) = 1. \tag{4.68}
\]

Hence, from Section 3.2, we can write:

\[
\text{Prob}_t^{SS_i} = \text{Prob} \left( \text{Act}_\text{choice}^i(t) = S \right) \times 1 = P_0(e_i^t). \tag{4.69}
\]

From the above arguments, we can write the probability of actor \( i \)’s real action as \textit{Do Nothing} as follows:

\[
\text{Prob} \left( \text{Act}_\text{real}^i(t) = S \right) = \text{Prob}_i^{SJ_t} + \text{Prob}_t^{SL_i} + \text{Prob}_t^{SS_i}. \tag{4.70}
\]

\[\blacksquare\]

**Theorem 4.3.6 (The complexities of computing the probabilities of observed actions).**

\textit{If the maximum number of groups in a society is } K, \textit{then the complexity of computing
\( \text{Prob}(\text{Act}_{\text{real}}^i(t) = J_i) \) is \( O(K) \), the complexity of computing \( \text{Prob}(\text{Act}_{\text{real}}^i(t) = L_i) \) is \( O(K) \), and the complexity of computing \( \text{Prob}(\text{Act}_{\text{real}}^i(t) = S) \) is \( O(K^2) \).

**Proof:** In Theorem 4.3.5, we need to compute \( P_{+1}(e_i^t) \), \( P_0(e_i^t) \), \( P_{-1}(e_i^t) \), \( P^{\text{Join}}_{(i,t)} \), \( P^{\text{Leave}}_{(i,t)} \), \( P^{\text{Reject}}_{(i,t)} \), \( \bar{P}^{\text{Join}}_{(i,t)} \), \( \bar{P}^{\text{Join}}_{(i,t)} \), and \( \bar{P}^{\text{Reject}}_{(i,t)} \) in order to compute the probabilities of an actor’s observed actions. We know

\[
\sum_{k=-1}^{+1} P_k(e_i^t) = 1, \tag{4.71}
\]

so when we compute one of the probability, we need to also compute other probabilities in order to satisfy Eq. (4.71). We compute the complexity based the number of the computation of a probability. Then, the complexity of computing \( P_{+1}(e_i^t) \), \( P_0(e_i^t) \) and \( P_{-1}(e_i^t) \) is \( O(3) \). We use the same argument in computing the complexities of \( P^{\text{Join}}_{(i,t)} \) and \( \bar{P}^{\text{Join}}_{(i,t)} \), and we know that

\[
\sum_{k=1}^{D_c^t} P^{\text{Join}}_{(i,t)} = 1, \tag{4.72}
\]

and

\[
\sum_{k=1}^{D_c^t} \bar{P}^{\text{Join}}_{(i,t)} = 1. \tag{4.73}
\]

Then, the complexities of computing \( P^{\text{Join}}_{(i,t)} \) and \( \bar{P}^{\text{Join}}_{(i,t)} \) are \( O(|D_c^t|) \) and \( O(|D_c^t|) \), respectively. Using the same argument, we know that the complexity of \( P^{\text{Leave}}_{(i,t)} \) and \( \bar{P}^{\text{Join}}_{(i,t)} \) are \( O(|D_c^t|) \) and \( O(|D_c^t|) \), respectively. The complexity of \( P^{\text{Reject}}_{(i,t)} \) is \( O(|D_c|) \), and the complexities of \( P^{\text{Reject}}_{(i,t)} \) and \( P^{\text{Reject}}_{(i,t)} \) are all \( O(1) \). We know that if the maximum number of groups in a society is \( K \), then \( |D_c^t| = K, |D_c^t| < K \), and \( |D_c| < K \).

From Eq. (4.50), we know the complexity of computing \( \text{Prob}(\text{Act}_{\text{real}}^i(t) = J_i) \) is \( O(3 \times (|D_c^t| + |D_c^t|)) \approx O(K) \). And from Eq. (4.51), we know the complexity of computing \( \text{Prob}(\text{Act}_{\text{real}}^i(t) = L_i) \) is \( O(3 \times (|D_c|)) \approx O(K) \). Also, based on Eq. (4.52), (4.53), (4.54), and (4.55), we know the complexity of \( \text{Prob}(\text{Act}_{\text{real}}^i(t) = S) \) is \( O(3 \times (|D_c^t|^2 + |D_c^t|^2) + 3 + 3) \approx O(K^2) \). \( \blacksquare \)
In the following sections, we present the basic concepts and the details of the algorithms about learning different types of parameters in ViSAGE. We classify parameters into three types—independent discrete parameters (e.g., actors’ size-type), dependent discrete parameters (e.g., actors’ ambition-type), and continuous parameters (e.g., reward/penalty parameters $\theta_{reward}$). The definition of dependent and independent parameters is as follows:

**Definition 4.3.7 (Dependent & Independent Parameters).** Let $\theta$ be an actor’s parameter—$\theta_i$ is specific to actor $i$, and $\theta_{\bar{i}}$ is specific to other actors. Also $\Theta_M = \{\theta_i, \theta_{\bar{i}}, \Theta_{\bar{M}}\}$. Then, $\theta$ is an independent parameter if and only if for all $i$,

$$Prob(p_i(t)|\theta_i, \Theta_{\bar{M}}, S_t) = Prob(p_i(t)|\theta_{\bar{i}}, \Theta_{\bar{M}}, S_t).$$  \hspace{1cm} \text{(4.74)}

Otherwise, $\theta$ is a dependent parameter.

### 4.3.1 Learning Independent Discrete Parameters

The basic idea is to maximize the log-likelihood $\ell(\Theta_M)$ (see Eq. (4.24)). Now consider optimization with respect to a particular actor’s parameters $\theta_i$. Then, Eq. (4.24) becomes this equation:

$$\ell(\theta_i) = \sum_t \log Prob(p_i(t)|\Theta_M, S_t) + \sum_{\bar{i} \neq i} \sum_t \log Prob(p_{\bar{i}}(t)|\Theta_M, S_t).$$ \hspace{1cm} \text{(4.75)}

When the actions of $\bar{i} \neq i$ are independent of $\theta_i$, or in some cases, the actions of $\bar{i} \neq i$ depends on $\theta_i$ only through $S_t$, which is a second order dependence, we treat $\theta_i$ as an independent parameter. From the definition of independent parameter (see Eq. (4.74)), we can write Eq. (4.75) as follows:

$$\ell(\theta_i) = \sum_t \log Prob(p_i(t)|\theta_i, \Theta_{\bar{M}}, S_t) + \sum_{\bar{i} \neq i} \sum_t \log Prob(p_{\bar{i}}(t)|\theta_{\bar{i}}, \Theta_{\bar{M}}, S_t).$$ \hspace{1cm} \text{(4.76)}
We ignore the second term in optimizing the parameters specific to actor $i$ because only the first term is related to $\theta_i$. Then

$$
\theta^*_i \leftarrow \arg\max_{\theta_i} \sum_t \log \text{Prob} \left( p_i(t) | \theta_i, \Theta_{\tilde{M}}, S_t \right). \tag{4.77}
$$

Hence, the maximization over a single actor’s parameters only involves that actor’s path and is a factor of $|\mathcal{A}|$ which is more efficient to compute than if we looked at all actors’ paths. Therefore, the entire learning process can be summarized by maximizing over each parameter successively, where in order to maximize over the parameters specific to an actor, we use only that actor’s path.

We present in detail the algorithm for learning actors’ size-types (see Section 3.1.1) in order to illustrate the process of learning independent discrete parameters. An actor’s size-type is an independent parameter because it only influences an actor’s own group size preference. Since, we have a data set which includes the information about group evolution (see Figure 4.4), and the group evolution gives us the information about actors’ actions (join a group, leave a group, or do nothing) and which groups actors join or leave in each time step. In our model, we know that an actor’s size-type controls which group he/she most likely joins or leaves (see Section 3.2.2). Therefore, if we have the data of the group evolution, we can use the path performed by an actor (which groups an actor joins or leaves) to learn that actor’s size-type. Assume that we know all the values of parameters except the actors’ size-types, then based on Eq. (3.15), (3.17), (3.18), (3.20), and (3.21), we compute the probability of the path for each actor. From Eq. (4.25) and (4.77) and Theorem 4.3.5, we get the following equation:

$$
\text{Prob} \left( p_i(t) | \theta_i, \Theta_{\tilde{M}}, S_t \right) = \text{Prob} \left( \text{Act}^i_{\text{real}}(t) \right), \tag{4.78}
$$

where

$$
\text{Prob} \left( \text{Act}^i_{\text{real}}(t) \right) = \begin{cases} 
\text{Prob} \left( \text{Act}^i_{\text{real}}(t) = J_l \right), & \text{if } p_i(t) \text{ is Join.} \\
\text{Prob} \left( \text{Act}^i_{\text{real}}(t) = L_l \right), & \text{if } p_i(t) \text{ is Leave.} \\
\text{Prob} \left( \text{Act}^i_{\text{real}}(t) = S \right), & \text{if } p_i(t) \text{ is Do nothing.}
\end{cases} \tag{4.79}
$$
From Theorem 4.3.6, we know the complexities $\text{Prob}(\text{Act}_{\text{real}}^i(t) = J_l)$ and $\text{Prob}(\text{Act}_{\text{real}}^i(t) = L_l)$ are linear, and the complexity of $\text{Prob}(\text{Act}_{\text{real}}^i(t) = S)$ is quadratic for one action at one time step. So, if the data set has $N$ actors and $T$ time steps, then the complexities need to time $NT$. Usually, the $N$ and $T$ are very huge, and for the efficiency of computation, we skip computing $\text{Prob}(\text{Act}_{\text{real}}^i(t) = S)$; however, we can still get the information about actors’ size-types from computing $\text{Prob}(\text{Act}_{\text{real}}^i(t) = J_l)$ and $\text{Prob}(\text{Act}_{\text{real}}^i(t) = J_l)$. We select an actor’s best size-type based on which size-type has the highest probability of generating the observed path. The learning algorithm is shown in Algorithm 4.2.

**Algorithm 4.2: Maximum log-likelihood learning algorithm for independent discrete parameters**

<table>
<thead>
<tr>
<th>Input</th>
<th>The set of matching social groups at each time step, $\mathcal{D} = {\mathcal{D}<em>t}</em>{t=1}^{T+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>The best value for $\theta_i$ for each actor $i$</td>
</tr>
</tbody>
</table>

1. foreach actor $i$ do
2.  | foreach possible value $x$ for $\theta_i$ do
3.  | $\text{Prob}_x \leftarrow \sum_t \log \text{Prob}(p_i(t)|\theta_i = x, \hat{\Theta}_M, S_t)$
4. endfch
5. $\theta_i^* \leftarrow \arg\max_x (\text{Prob}_x)$
6. endfch
7. return all $\theta_i^*$

Sometimes, we don’t know actors’ probability distributions over group size corresponding to each size-type used in Eq. \((3.15)\). We develop an expectation-maximization (EM) algorithm, [63], to learn the actors’ size-types and also the probability distributions over group size. We know that actors with different size-types have different probability distributions over group size, and these probability distributions over group size influence which group an actor is most likely to join or to leave. Using the group evolution, we are able to find out the sizes of groups an actor joins and leaves, and we can use this information for all actors with the same size-type to determine the probability distribution over group size for that size-type.

There are two steps in the EM algorithm, Expectation step and Maximization step. In the $E$ step, we learn the probability distribution over group size for each size-type; subsequently in the $M$ step, we optimize the actors’ size-types based on the learned prob-
ability distributions. Which kind of probability distribution should be used in the learning process is based on the knowledge or experiences about the data set; however, the general learning framework is the same. Here, we assume the probability distribution over group size for each size-type is a Gaussian distribution, which is different from our default model, and then we need to determine the means and variances for those Gaussian distributions. We use standard K-means algorithm, [64, 65], to cluster actors based on the average size of groups each actor joined into three clusters (there are three values for an actor’s size-type). We, then, compute the mean and the variance of the average group size for each cluster. This is a simple heuristic based on the observation that Type_S actors join small groups, Type_L actors join large groups, and Type_M actors join median groups. The clustering algorithm is shown in Algorithm 4.3. For different probability distributions over group size, the general learning process in Algorithm 4.3 is the same, except the steps 5 through 8. We need to learn properties of different probability distributions from steps 5 through 8.

Algorithm 4.3: Cluster Algorithm

<table>
<thead>
<tr>
<th>Input</th>
<th>The set of social groups at each time step, $D = {D_t}_{t=1}^{T+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Means and variances of the distribution of the group size preference</td>
</tr>
</tbody>
</table>

1. foreach actor $i$ do
   2.  $size_i \leftarrow$ the average size of groups actor $i$ joined
3. endforeach
4. Cluster $\{size_i\}_{i=1}^{|A|}$ into 3 clusters using the standard 3-means algorithm.
5. foreach cluster $c$ do
   6.  $\mu_c \leftarrow E(size_k)$ where $k \in$ cluster $c$
   7.  $\sigma_c \leftarrow Var(size_k)$ where $k \in$ cluster $c$
8. endforeach
9. return all $\mu_c$ and $\sigma_c$

In $M$ step, we use the same algorithm as Algorithm 4.2 to optimize the actors’ size-types based on the means ($\mu_c$’s) and variances ($\sigma_c$’s) (learned from Algorithm 4.3) of the Gaussian distributions. After Algorithm 4.2, we are able to obtain a new set of clusters, from which we can calculate the new means and variances. The whole process is repeated until an error threshold is reached to obtain a set of clusters which have maximum probability with respect to all paths of the actors. The full EM algorithm is shown in
Algorithm 4.4.

**Algorithm 4.4:** EM algorithm for learning actors’ size-type

<table>
<thead>
<tr>
<th>Input</th>
<th>The set of social groups at each time step, $D = {D_t}_{t=1}^{T+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>The optimized value of $\theta_i$</td>
</tr>
</tbody>
</table>

1. Compute $\mu_c$’s and $\sigma_c$’s using Algorithm 4.3, Cluster algorithm
2. repeat
3. Apply $\mu_c$’s and $\sigma_c$’s to Algorithm 4.2
4. Calculate $\mu_c$ and $\sigma_c$ for each actor type based on new $\theta_i^*$
5. until exceed threshold
6. return all $\theta_i^*$

Since size-type controls an actor’s group size preference, the clustering algorithm, Algorithm 4.3, can be used as a benchmark to test the value added by the EM algorithm. However, in our model, when an actor decides to join or to leave a group, he/she considers not only the group size preference but also his/her friendships, his/her qualification, and groups’ qualifications. The reason is that in ViSAGE, an actor joins a group by reference (see Definition 3.2.1), and so an actor’s friendships influence which groups he/she would like to join. In addition, an actor’s qualification and groups’ qualifications also influence which group is selected to join or to leave, and whether the the group accepts the actor. Therefore, the EM algorithm is a better approach to learn actors’ size-types, and we show the results in Section 5.

### 4.3.2 Learning Dependent Discrete Parameters

For dependent discrete parameters, the idea is also to maximize the log-likelihood $\ell(\Theta_M)$ (see Eq. (4.24)). However, for the optimal solution, we need to compute $\ell(\Theta_M)$ for all possible values. For instance, let $\theta$ be a particular type of actor parameter, and $\theta_i$ is specific to actor $i$. If there are $N$ actors, then $\Theta_N$ is a set of the $\theta$ values for all actors; $\Theta_N = \{\theta_1, \theta_2, \cdots, \theta_N\}$ and $\Theta_M = \{\Theta_N, \Theta_X\}$. If each $\theta_i$ have $K$ possible values, then there are $K^N$ possible combinations for $\Theta_N$. If the data set has $T$ time steps, the time complexity for computing the maximum of $\ell(\Theta_M)$ is $O(TK^N)$ which is exponential. Therefore, we have developed a more efficient approximation algorithm. We also use expectation-maximization (EM) algorithm, [63], in order to solve this problem.
Algorithm 4.5: EM algorithm for learning dependent discrete parameters

**Input**: The set of social groups at each time step, \( D = \{ D_t \}_{t=1}^{T+1} \)

**Output**: The optimized value of \( \theta_i \)

1. **function** `ComputeProb()` {

2.     **foreach** actor \( i \) do

3.         **foreach** possible value \( x \) for \( \theta_i \) do

4.             \( \text{Prob}_x \leftarrow \sum_t \log \text{Prob}(p_i(t) | \theta_i = x, \Theta_N, \Theta_{\bar{N}}, S_t) \)

5.             \( \mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{x, \text{Prob}_x\} \)

6.         endfunc

7.     endfunc

8. **foreach** actor \( i \) do \( \mathcal{E}_i \leftarrow \emptyset \)

9.     for \( k = 1 \) to \( H \) do

10.        \( \Theta_N \leftarrow \text{randomly assign} \)

11.        ComputeProb()

12.     endfor

13. **repeat**

14.     **until** exceed threshold

15.     **foreach** actor \( i \) do \( \theta_i^* \leftarrow \arg\max_{\text{Prob}_x} (\mathcal{E}_i) \)

16. **return** all \( \theta_i^* \)

17. **endfunc**

Algorithm 4.5 shows the EM algorithm for learning dependent discrete parameters.

The function `ComputeProb()` is to compute the \( \text{Prob}_x \) for all possible \( \theta_i \) for each actor in one iteration under the parameters \( \Theta_N \), and then to save all of the \( \theta_i \) and \( \text{Prob}_x \) into \( \mathcal{E}_i \) for each actor. The main routine starts from line 9 which initializes the \( \mathcal{E}_i \) as an empty set. In the \( E \) step, between lines 10 to 13, we randomly assign each actor’s \( \theta_i \) based on normal distribution to construct \( \Theta_N \), and call `ComputeProb()` to compute and save \( \theta_i \) and \( \text{Prob}_x \) for each actor. Then, run this routine \( H \) iterations to generate the probability distribution for \( \theta_i \) saved in \( \mathcal{E}_i \). The lines 14 to 17 represent the \( M \) step and also update the probability distribution for \( \theta_i \) at the same time. Based on the probability distribution saved in \( \mathcal{E}_i \), we construct the \( \Theta_N \) and call `ComputeProb()` to compute and save \( \theta_i \) and \( \text{Prob}_x \) for each actor and also update the probability distribution, \( \mathcal{E}_i \). Line 18 decides the optimal solution for each \( \theta_i \) based on the \( \mathcal{E}_i \). The complexity of the function `ComputeProb()` is \( O(TKN) \): the loop from line 2 runs \( N \) times, the loop from line 3 runs \( K \) times, and
the summation in line 4 is for \( T \) time steps. The function \texttt{ComputeProb()} is called in lines 12 and 16 for \( H \) times and \( I \) times, respectively, where \( I \) indicates the number of iterations before exceeding the threshold. Therefore, the complexity of Algorithm 4.5 is \( O((H + I) \times TKN) \).

We learn actors’ \textit{ambition-types} to demonstrate the performance of our approach for learning dependent discrete parameters. From Section 3.1.2, we know an actor’s \textit{ambition-type} influences an actor’s ranks, and in Eq. (3.2), actor \( i \)’s rank in a certain group depends on all other members in that group; therefore, an actor’s \textit{ambition-type} is a dependent parameter. For learning actors’ \textit{ambition-types}, we compute the same probabilities of actors’ paths described in Eq. (4.78) and (4.79). Furthermore, for the efficiency reason, we only compute the probabilities of actors’ \textit{Join} and \textit{Leave} actions. We apply the Algorithm 4.5 for learning an actor’s \textit{ambition-type}, and then select the \textit{ambition-type} having maximum probability of the actor’s observed path as the optimal solution.

\subsection*{4.3.3 Learning Continuous Parameters}

The continuous parameters, regardless of whether they are dependent or independent, can be optimized using a gradient based approach, which involves taking derivatives of the log-likelihood \( \ell(\Theta_M) \), Eq. (4.24),

\[
\ell(\Theta_M) = \sum_i \sum_t \log Prob(p_i(t)|\Theta_M, S_t). \quad (4.80)
\]

Here, as an example, we use the optimization with respect to the reward/penalty parameters \( \omega_\alpha \) (see Section 3.2.3) for learning the Agency and Structure Table to demonstrate learning continuous parameters. We can write Eq. (4.80) as the following equation:

\[
\ell(\omega_\alpha) = \sum_i \sum_t \log Prob(p_i(t)|\Theta_M, S_t). \quad (4.81)
\]

From Eq. (4.25) and (4.26), we can write:

\[
\ell(\omega_\alpha) = \sum_i \sum_t \log Prob\left(Act^i_{\text{real}}(t)\right). \quad (4.82)
\]
Taking the derivative of Eq. (4.82) with respect to $\omega_\alpha$, we get what is shown here:

$$\frac{\partial \ell(\omega_\alpha)}{\partial \omega_\alpha} = \sum_i \sum_t \frac{\partial \log \text{Prob}(\text{Act}_{\text{real}}^i(t))}{\partial \omega_\alpha} \frac{\partial \text{Prob}(\text{Act}_{\text{real}}^i(t))}{\partial e^t_i \partial \omega_\alpha} \times \frac{\partial e^t_i}{\partial \omega_\alpha}.$$ (4.83)

The first step is to solve the first term in Eq. (4.84). There are three kinds of observed actions ($\text{Act}_{\text{real}}^i(t)$), *Join*, *Leave*, and *Do Nothing*. Based on Theorem 4.3.5, we do the derivative on the probability of each action with respect to $e^t_i$, as shown in the following three cases:

1. When $\text{Act}_{\text{real}}^i(t) = \text{Join}$, based on Eq. (4.57),

$$\frac{\partial \log \text{Prob}(\text{Act}_{\text{real}}^i(t) = J_i)}{\partial e^t_i} = \frac{\partial \log \text{Prob}(\text{Act}_{\text{choice}}^i(t) = J)}{\partial e^t_i} + \frac{\partial \log \text{Prob}(\text{Act}_{\text{real}}^i(t) = J_i|\text{Act}_{\text{choice}}^i(t) = J)}{\partial e^t_i}.$$ (4.85)

2. When $\text{Act}_{\text{real}}^i(t) = \text{Leave}$, based on Eq. (4.60),

$$\frac{\partial \log \text{Prob}(\text{Act}_{\text{real}}^i(t) = L_i)}{\partial e^t_i} = \frac{\partial \log \text{Prob}(\text{Act}_{\text{choice}}^i(t) = L)}{\partial e^t_i} + \frac{\partial \log \text{Prob}(\text{Act}_{\text{real}}^i(t) = L_i|\text{Act}_{\text{choice}}^i(t) = L)}{\partial e^t_i}.$$ (4.86)

3. When $\text{Act}_{\text{real}}^i(t) = \text{Do Nothing}$, based on Eq. (4.63), (4.65), and (4.67),

$$\frac{\partial \log \text{Prob}(\text{Act}_{\text{real}}^i(t) = S)}{\partial e^t_i} = \frac{\partial \log (\text{Prob}SJ^i_t + \text{Prob}SL^i_t + \text{Prob}SS^i_t)}{\partial e^t_i}.$$ (4.87)

The derivatives of the functions are based on the formulas. Here, we just present the general framework of doing the derivatives, and the details of the derivatives on the default setting can be found in Appendix C of this dissertation.

To solve the second term in Eq. (4.84), we know from Eq. (3.6) that

$$\frac{\partial e^t_i}{\partial \omega_\alpha} = \frac{\partial R^t_i}{\partial \omega_\alpha} - \frac{\partial \psi_1 L^t_{br}}{\partial \omega_\alpha} - \frac{\partial \psi_2 L^t_{bo}}{\partial \omega_\alpha}.$$ (4.88)
We know both the Bridging Cost \( \phi_i L_{br}^i \) and Bonding Cost \( \psi_i L_{bo}^i \) are functions of \( R_t^i \), and \( R_t^i \) is a function of \( \omega_\alpha \). Applying the chain rule, we get the following equation:

\[
\frac{\partial e_t^i}{\partial \omega_\alpha} = \left( 1 - \frac{\partial \phi_i L_{br}^i}{\partial R_t^i} - \frac{\partial \psi_i L_{bo}^i}{\partial R_t^i} \right) \times \frac{\partial R_t^i}{\partial \omega_\alpha}.
\]

(4.89)

From Eq. (3.25), (3.26), and (3.27), we know that

\[
\frac{\partial R_t^i}{\partial \omega_\alpha} = \frac{\partial R_t^{i-1}}{\partial \omega_\alpha} + \frac{\partial \Delta R_t^{i-1}}{\partial \omega_\alpha},
\]

(4.90)

where \( \Delta R_t^{i-1} \) is a function of \( R_t^{i-1} \) and \( \omega_\alpha \). After doing the derivatives and some rearrangements, we have what we show here:

\[
\frac{\partial R_t^i}{\partial \omega_\alpha} = \Phi_t^{i-1} \times \frac{\partial R_t^{i-1}}{\partial \omega_\alpha} + \Psi_t^{i-1},
\]

(4.91)

where \( \Phi_t^{i-1} \) and \( \Psi_t^{i-1} \) are functions of \( R_t^{i-1} \). When \( t = 0 \), \( R_t^0 \) is a constant, and so we get what is show here:

\[
\frac{\partial R_0^i}{\partial \omega_\alpha} = 0.
\]

(4.92)

So we can use dynamic programming to solve \( \partial R_t^i / \partial \omega_\alpha \) at each time step, and then to calculate the derivatives of \( \ell(\omega_\alpha) \), Eq. (4.83). The example of learning the reward/penalty parameters \( \omega_\alpha \) and the details of solving Eq. (4.83) in the default setting can be found in Appendix C.

Algorithm 4.6 shows the pseudo code for the learning the reward/penalty parameters \( \omega_\alpha \) where \( R_t^i \) denotes \( \partial R_t^i / \partial \omega_\alpha \). There are four loops in the algorithm, and the complexity of this gradient based algorithm is \( O(ITKN|A|) \), where \( |A| \) is the number of actors, \( T \) is the number of time steps, \( K \) is the number of \( \omega_\alpha \)'s, and \( I \) is the number of iterations during the gradient learning process.
Algorithm 4.6: The algorithm for learning reward/penalty parameters $\omega_\alpha$

**Input**: The set of social groups at each time step, $D = \{D_t\}_{t=1}^{T+1}$

**Output**: The optimized value of $\omega_\alpha$

repeat
  foreach $\omega_\alpha$ do
    foreach time step $t$ do
      foreach actor $i$ do
        if $t=0$ then
          $R_0^i = 0$
        else
          compute $\Phi^{t-1}$ and $\Psi^{t-1}$.
          $R_t^i \leftarrow \Phi^{t-1} \times R_{t-1}^i + \Psi^{t-1}$.
          compute $\partial e^t_i / \partial \omega_\alpha$ based on Eq. (4.89).
          compute Eq. (4.85), (4.86), and (4.87) based actor $i$’s observed action.
          update $\partial \ell(\omega_\alpha) / \partial \omega_\alpha$ based on Eq. (4.84).
        endif
      endfch
    endfch
  endfch
  foreach $\omega_\alpha$ do
    update $\omega_\alpha$ according to $\partial \ell(\omega_\alpha) / \partial \omega_\alpha$
  endfch
until exceed threshold
return $\omega_\alpha$
4.4 Predictions

Unlike in a traditional supervised learning task, where the quality of the learner can be measured by its performance on a test set, the learned function in our setting is a stochastic process, and the test data are a realization of the stochastic process. Specifically, assume we have training data $D_{train} = \{D_t\}_{t=1}^{T+1}$ and test data $D_{test} = \{D_t\}_{t=T+2}^{T+K}$. We learn the parameters governing the micro-laws using $D_{train}$, and use multi-step prediction to test on the test data. Specifically, starting from the social group structure $D_{T+1}$ at time $T + 1$, we predict the actions of the actors—the actor paths into the future. Based on these paths and using some metric, we can construct the evolving social group structure and compare these predicted groups with the observed groups on the test data. We can compare global behaviors such as the distribution of groups size, the distribution of the number of group an actor joins, the distribution of the number of each level of friendship. Here, we use a comparison of the distribution of groups’ sizes to demonstrate the prediction.

4.4.1 Distribution of Group Sizes

We compare the difference of the distribution of group sizes to measure our performance. Specifically, let $N_k^{pred}(t)$ and $N_k^{true}(t)$ be the number of groups which size is $k$ at time $t$ for the predicted and true societies respectively. We report our results using the squared error measure (normalized) between the frequencies as well as the squared error difference between the probabilities,

$$E_f(t) = \sqrt{\sum_k \left( \frac{N_k^{pred}(t)}{N_k^{pred}(t)} - \frac{N_k^{true}(t)}{N_k^{true}(t)} \right)^2},$$  \hspace{1cm} (4.93)$$

$$E_p(t) = \sqrt{\sum_k \left( \frac{N_k^{pred}(t)}{N_k^{pred}(t)} - \frac{N_k^{true}(t)}{N_k^{true}(t)} \right)^2},$$ \hspace{1cm} (4.94)$$

where $N_k^{pred}(t) = \sum_k N_k^{pred}(t)$ and $N_k^{true}(t) = \sum_k N_k^{true}(t)$. $E_p$ measures the difference in the shape of the histograms, whereas $E_f$ takes into account the actual number of groups.
4.4.2 Power Law Networks

Articles have shown that a lot of networks in the real world are power-law networks, as we see in social networks [66, 67, 68], Internet topologies [69], the Web [70, 71], neural networks [72], and power grids [73]. Authors in [67] define power-law networks as follows:

Definition 4.4.1 (Power-law networks). Power-law networks are networks where the probability that a node has degree $k$ is proportional to $k^{-\gamma}$, where $\gamma$ is called the power-law coefficient and $\gamma > 1$.

We examine whether the distribution of the number of communication link among actors exhibits power-law distribution on predictions using ViSAGE, and we use the maximum likelihood method described in [74] to estimate the power-law coefficient, $\gamma$. 
CHAPTER 5
EXPERIMENTS AND RESULTS

In our model, there are a lot of parameters which can be learned by using the approach described in Section 4. Here, we focus on the following parameters: the actor’s size-type and ambition-type (\text{type}_i, see Section 3.1.1), the actor’s initial resources ($E^0_i$, see Section 3.1.4), available resources ($E^t_i$, see Section 3.1.4), the society reward/penalty parameters ($\theta_{\text{reward}}$, see Section 3.2.3), and the probabilities of communications ($P_l$, see Section 3.3). We show the results of learning from the synthetic data and also from real data such as Enron email and Movie newsgroups.

In the real world, most of time, people are only able to collect communication data without knowing the group structure or group evolution. We have developed a multistage learning process to learn micro-laws from communication step-by-step (see Figure 5.1). There are three learning stages; in the first stage, we use overlapping clustering algorithm (see Section 4.1) to learn group structure from communication data; in the second stage, we learn group evolution from group structure using matching algorithm.

![Figure 5.1: Multistage learning process with algorithms.](image)
(see Section 4.2); in the last stage, we use learning algorithms (see Section 4.3) to learn appropriate micro-laws from group evolution. From synthetic data, we can collect all three kinds of data—communications, group structure, and group evolution. Each learning stage is a research project; here, we develop algorithms for last two stages, and we adopt the overlapping clustering algorithm developed by Baumes et al., [35], for the first stage. In the following sections, we are going to show the influence on the accuracy of learning parameters from each stage in synthetic data, and then we present the learning results from the real data—notably, Movie newsgroup and Enron email.

5.1 Results on Synthetic Data

To evaluate our performance, we use an instance of the model to generate synthetic data for training and testing. Since we know the values of parameters in the model, we can compare the true values with the learned values to compute the accuracy. We simulate 50, 100, 150, 200 and 250 time steps of training data (averaged over 20 data sets) to test our learning algorithms.

5.1.1 Learning from Group Evolution

From the synthetic data, we are also able to collect the information about group evolution which avoids the first two learning stages mentioned in Sections 4.1 and 4.2. In the following sections, we are going to present the results of learning from group evolution using the algorithms in Section 4.3.

5.1.1.1 Learning Actors’ Size-type

We evaluate the learning results of the actor’s size-type from the following seven different algorithms (described in Section 4.3.1):

1. **Learn**: Use only maximum log-likelihood algorithm, *Algorithm 4.2*, with true probability distributions over group size for the three types.

2. **Cluster**: For each actor $i$, let $size_i$ be the average size of groups actor $i$ joined. We cluster $\{size_i\}_{i=1}^{|A|}$ into three groups using the standard 3-means algorithm, *Algorithm 4.3*. This is a simple heuristic based on the observation that Type $S$ actors
like to join small groups, \textit{Type}_{L} actors join large groups, and \textit{Type}_{M} actors join medium sized groups.

3. **EM**: With unknown probability distributions over group size. We use the expectation-maximization (EM) algorithm, Algorithm 4.4, to learn the actor’s \textit{size-type} as well as the probability distributions over group size.

4. **Optimal**: The ground truth model. (For comparison and only available on synthetic data.)

5. **Leader, Socialite, Follower**: Benchmarks which assign all actors’ \textit{size-types} to \textit{Type}_{S}, \textit{Type}_{M} or \textit{Type}_{L}, respectively.

Each data set is created by randomly assigning about 1/3 of the actors to each \textit{size-type} and simulating for 50, 100, 150, 200, and 250 time steps. All other parameters except \textit{size-type} and probability distributions over group size are held fixed. Table 5.1 shows the accuracies (%) of the Learn, EM, and Cluster algorithms using 250 time steps of training data (averaged over 20 data sets). The accuracies are 98.6\%, 70.19\%, and 55.58\%, respectively, and for comparison, the accuracy of randomly assigning \textit{size-type} is 33.33\%, the accuracies of the Leader, Socialite, and Follower algorithms are 33.33\%, and the accuracy of the Optimal algorithm is 100\%. Figure 5.2 shows the accuracies of the three algorithms corresponding with the number of time steps of training data. As expected, when the length of the time period of training data increases, we obtain better results from all three algorithms.

The results indicate that the accuracy for the Learn algorithm is the best because it uses the true probability distribution over group sizes and only needs to learn the actor’s \textit{size-type}. The Cluster algorithm has the worst result, and the reason is that it only considers the group size preference and ignores the interactions between actors. The EM algorithm learn the actor’s \textit{size-type} and the probability distributions over group size. Table 5.1 and Figure 5.2 show that the EM algorithm does improve the results from the Cluster algorithm. In Section 3.2, we know which group is selected by an actor to apply to and finally join depends on the actor’s \textit{size-type} as well as the actor’s ranks and qualification, the group’s qualification, and the mechanism “join by reference.” The Cluster algorithm is only based on the average size of groups the actor joined which can be detected
Table 5.1: Confusion matrix in learning size-type using Cluster, EM and Learn

(a) Learn algorithm

<table>
<thead>
<tr>
<th>Accuracy 98.6%</th>
<th>Learned Actors’ size-type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T_{typeS}$</td>
</tr>
<tr>
<td>True Actors’ size-type</td>
<td>137.35</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
</tr>
</tbody>
</table>

(b) EM algorithm

<table>
<thead>
<tr>
<th>Accuracy 70.19%</th>
<th>Learned Actors’ size-type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T_{typeS}$</td>
</tr>
<tr>
<td>True Actors’ size-type</td>
<td>114.65</td>
</tr>
<tr>
<td></td>
<td>15.5</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
</tr>
</tbody>
</table>

(c) Cluster algorithm

<table>
<thead>
<tr>
<th>Accuracy 55.58%</th>
<th>Learned Actors’ size-type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T_{typeS}$</td>
</tr>
<tr>
<td>True Actors’ size-type</td>
<td>62.65</td>
</tr>
<tr>
<td></td>
<td>3.35</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
</tr>
</tbody>
</table>

from observable group structure. On the other hand, the Learn and EM algorithms learn the actor’s hidden curriculum, which cannot be observed from the data directly, based on the interactions with other actors and the influences of the environment.

The predictive performance of the algorithms on test data according to the metrics in Eq. (4.93) and (4.94) is shown in Figure 5.3. The figure clearly shows that the Learn algorithm surpasses all the algorithms because the prediction from Learn is closest to Optimal (which is unattainable in reality), and the worst cases are the Leader and Follower algorithms which assign all actors to leader or follower types.

5.1.1.2 Learning Actors’ ambition-type

We evaluate the learning results of the actor’s ambition-type from the following two algorithms (see Section 4.3.2):

1. A-EM: Using the expectation-maximization (EM) algorithm, described in Algorithm 4.4, to learn the actor’s ambition-type.
Figure 5.2: The accuracy (%) of learning size-type using Learn, Cluster, and EM algorithms from group evolution corresponding with different time steps.

2. **A-Optimal:** When learn an actor’s ambition-type, all other actors’ ambition-types use the true values. (For comparison and only available on synthetic data.)

Each data set is created by randomly assigning about 1/3 of the actors to each ambition-type and to simulate for 50, 100, 150, 200, 250, and 1000 time steps. All other parameters except ambition-type use true value. In the data set, if there are 500 actors and each actor has three possible ambition-types, and then, the number of possible combinations of all actors’ ambition-types is $3^{500} > 10^{238}$ which is exponential complexity. Table 5.2 shows the accuracies (%) of the A-EM and A-Optimal algorithms using 1000 time steps of training data (averaged over 20 data sets), and using 100 combinations of all actors’ ambition-types as initial value in the learning process. The accuracies are 58.24% and 74.2%, respectively. Figure 5.4 shows the accuracies of the two algorithms corresponding with the different number of time steps of training data. As expected, when the length of the time period of training data increases, we obtain better results from all two algorithms. Figure 5.5 shows the accuracy with different learning iterations using
The difference between the distribution of group size

Figure 5.3: The predictive error $E_f$ and $E_p$ for various algorithms for learning $s_{type}$.
Table 5.2: Confusion matrix in learning ambition-type using A-EM and A-Optimal

(a) A-EM algorithm

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Learned Actors’ ambition-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>58.24%</td>
<td>( T_{type_A} )</td>
</tr>
<tr>
<td>True Type A</td>
<td>19.71</td>
</tr>
<tr>
<td>Type B</td>
<td>43.71</td>
</tr>
<tr>
<td>Type C</td>
<td>35.59</td>
</tr>
</tbody>
</table>

(b) A-Optimal algorithm

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Learned Actors’ ambition-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>74.2%</td>
<td>( T_{type_A} )</td>
</tr>
<tr>
<td>True Type A</td>
<td>39.3</td>
</tr>
<tr>
<td>Type B</td>
<td>19.95</td>
</tr>
<tr>
<td>Type C</td>
<td>40.6</td>
</tr>
</tbody>
</table>

Figure 5.4: The accuracy (%) of learning ambition-type using algorithm A-Optimal and A-EM from group evolution with different time steps of data set.

the A-EM algorithm. It clearly shows that we can learn better learning results from more learning iterations, which means we learn from more combinations of actors’ ambition-type.
Figure 5.5: The accuracy (%) of learning \textit{ambition-type} using algorithm A-EM from group evolution with different learning iterations.

5.1.1.3 Learning Reward/Penalty Parameters

In the Agency and Structure table, there are 9 parameters, $\theta_{\text{reward}} = \{\omega_{\alpha} | \alpha = 1, 2, \cdots, 9\}$. Here, we show the results for learning the reward/penalty parameters from the synthetic data. From Table 3.1, we know which $\omega$ is used based on $\text{Act}_{\text{norm}}$ and $\text{Act}_{\text{choice}}$. In the group evolution data, we can tell which $\text{Act}_{\text{real}}$ an actor performs based on an actor’s observable action—\textit{join a group}, \textit{leave a group}, or \textit{do nothing}. However, there is no information about $\text{Act}_{\text{norm}}$ from the training data set without knowing an actor’s available resources, $R_t^i$. Therefore, during the learning process, we can only calculate $\text{Act}_{\text{norm}}$ based on the properties of the current state. We can only compute the probabilities of possible $\text{Act}_{\text{choice}}$’s from the observed $\text{Act}_{\text{real}}$.

Figure 5.6 shows the results for learning one $\omega_{\alpha}$ under different conditions. In this training data set, the values of $\omega_{\alpha}$’s are between -2.0 and 2.0. The solid line, \textit{Grid Search 1}, indicates that we know the true $\text{Act}_{\text{norm}}$ and use a grid search to compute $\ell(\omega_{\alpha})$ with different values of $\omega_{\alpha}$. The dash line, \textit{Grid Search 2}, indicates that we don’t know the true $\text{Act}_{\text{norm}}$, and that we compute what $\text{Act}_{\text{norm}}$ should be based on the current environment,
and also do the grid search. The line Grid Search 2 has more unknown parameters than the line Grid Search 1 has; therefore, the maximum of $\ell(\omega_\alpha)$ on line Grid Search 2 is smaller than the one on line Grid Search 1. In this experiment, the true value of $\omega_\alpha = -1$, and the pentagram marker (True Max1) and the square marker (True Max2) indicate the points with true $\omega_\alpha$ on the line Grid Search 1 and the line Grid Search 2, respectively; both markers are around the maxima of the both lines. The circle marker (Learned Max1) and the diamond marker (Learned Max2) show the results about learning $\omega_\alpha$ by using the gradient learning algorithm, described in Section 4.3.3, with and without the true $\text{Act}_\text{norm}$, respectively. Both results show the gradient learning algorithm can learn the $\omega_\alpha$ very accurately, even though the model is very complex and the data is very noisy. The plus sign marker (Learned Max3) is the result of learning all of the night $\omega_\alpha$’s together without knowing the true $\text{Act}_\text{norm}$’s. The $\ell(\omega_\alpha)$ of the marker (Learned Max3) is the smallest value among five makers because the other four markers (True Max1, True Max2, Learned Max1, and Learned Max2) have only one unknown $\omega_\alpha$; but for the marker (Learned Max3), we have night $\omega_\alpha$’s as the degrees of freedom, and the learned $\omega_\alpha$ is still very close to true $\omega_\alpha$. 

Figure 5.6: The results of learning one $\omega_\alpha$ under different conditions.
Table 5.3: The mean and standard deviation (S.D.) about the differences between the learned $\omega_\alpha$'s and true $\omega_\alpha$'s

<table>
<thead>
<tr>
<th></th>
<th>$\omega_1$</th>
<th>$\omega_4$</th>
<th>$\omega_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>-0.5417</td>
<td>0.0880</td>
<td>-0.0981</td>
</tr>
<tr>
<td>S.D.</td>
<td>0.1235</td>
<td>0.1405</td>
<td>0.2849</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$\omega_2$</th>
<th>$\omega_5$</th>
<th>$\omega_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>-0.2173</td>
<td>-0.1113</td>
<td>0.0028</td>
</tr>
<tr>
<td>S.D.</td>
<td>0.1277</td>
<td>0.0490</td>
<td>0.1451</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$\omega_3$</th>
<th>$\omega_6$</th>
<th>$\omega_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.3643</td>
<td>-0.1127</td>
<td>0.1343</td>
</tr>
<tr>
<td>S.D.</td>
<td>0.2323</td>
<td>0.1349</td>
<td>0.1858</td>
</tr>
</tbody>
</table>

We simulate 50, 100, 150, 200 and 250 time steps of training data (averaged over 3 data sets). All other parameters except the reward/penalty parameters ($\omega_\alpha$) and initial resources ($R^0_i$) were held fixed. Table 5.3 shows the mean and standard deviation for the differences between the learned $\omega_\alpha$'s and true $\omega_\alpha$'s. In the default setting, the values of $\omega_\alpha$’s in the functions $PenaltyW()$ and $RewardW()$ used in Eq. (3.26) are between -2.0 and 2.0. The absolute values of the mean of errors for $\omega_4$, $\omega_5$, $\omega_6$, $\omega_7$, $\omega_8$, and $\omega_9$ range from 0.0028 to 0.1343 which registers a measurement ranging from 0.07% to 3.36% of the possible range. The mean of errors for $\omega_1$, $\omega_2$, and $\omega_3$ are larger than others because these three parameters are used when Act$_{norm}$ is Join a group, which means that an actor would like to join a group. However, under this condition, there is a possibility that the group rejects the actor’s application, after which we observe the Act$_{real}$ as doing nothing. So, when Act$_{norm}$ is Join a group, there is more “noise” in the data set, and hence the error of learning is larger.

5.1.2 Learning from Group Structure

If a data set only has the information about group structure, we need to have one more learning stage to find the matchings between groups and get the group evolution. We use Algorithm 4.1 described in Section 4.2 to find the matchings, and then we apply the algorithms in Section 4.3 to learn the micro-laws in the model.

Figure 5.7 shows the accuracy of learning size-type from group structure using the Learn, Cluster, and EM algorithms. The results show that the Learn algorithm is still the best, and the result from the EM algorithm is also better than the result from the
Figure 5.7: The accuracy (%) of learning size-type from group structure using different algorithms with different time steps of data set.

Figure 5.8: The comparison of the results of learning size-type between learning from group evolution and group structure.
Cluster algorithm. The results also indicate that the matching algorithm have done a nice job, so algorithms Learn and EM have some improvement from the Cluster algorithm. Figure 5.8 shows the comparison of the results of learning size-type between learning from group evolution and learning from group structure. As expected, the learning result from group structure is worse than the one from group evolution. The reason is that we need to have one more learning step (learning group evolution from group structure) which will increase the learning error. We also can tell that the learning results on the Learn algorithm are most sensitive to the accuracy of group evolution because the accuracy of learning results from group structure on the Learn algorithm decreases most from the one learned from group evolution. On the other hand, the Cluster algorithm is less sensitive to the accuracy of group evolution, and the learning results between learning from group structure and group evolution are very close.

Figure 5.9 shows the accuracy of learning ambition-type from group structure using the A-EM and A-Optimal algorithms. We know each data set is created by randomly assigning 1/3 of the actors to each ambition-type, and so, if we assign all actors to be one ambition-type, the accuracy is 33.33%. Both algorithms still improve the accuracy from assigning all actors to one ambition-type, even though we have one more learning stage—learning group evolution from group structure. Figure 5.10 shows a comparison of the results of learning ambition-type between learning from group evolution and learning from group structure. We can tell the A-EM algorithm is more sensitive to the accuracy of group evolution than the A-Optimal algorithm because the change of accuracy of the A-EM algorithm is more than the A-Optimal algorithm.

5.1.3 Learning from Communication

In the real world, most of time, we are only able to collect the communication between actors without knowing the group structure due to, sometimes, privacy, ethical and legal issues. Therefore, we need to apply all three learning stages to learn the micro-laws in the model. In the first stage, we learn the group structure from the communication data; in the second stage, we learn the group evolution from the group structure; in the last stage, we learn the parameters from the group evolution.
Figure 5.9: The accuracy (%) of learning *ambition-type* from group structure using different algorithms with different time steps of data set.

Figure 5.10: The comparison of the results of learning *ambition-type* between learning from group evolution and group structure.
5.1.3.1 Learning Communication Probabilities

In this section, we show the learning results from the first learning stage, described in Section 4.1. Here, we only consider the first level of friendship, and the communication probabilities are $P_0 = P_b$ and $P_1 = P_g$. Figure 5.11 shows the results of learning communication probabilities. When the group structure is known, the algorithm can learn the $P_b$ and $P_g$ very well (dot lines in Figure 5.11). Meanwhile, as we apply the overlapping clustering algorithms in [35] to get the group structure, the upper figure shows the learned $P_b$’s are not influenced by different $P_g$’s very much. However, different $P_b$’s have an impact on the learned $P_g$’s (shown in the lower figure) because some outsiders have been included in the same group and the size of the group is larger than true size. The bottom line of learning communication probabilities is that a monotonic relationship exists: higher learned $P_g$ implies higher true $P_g$, and the same can be said for $P_b$.

5.1.3.2 Results of Learning Parameters

Figure 5.12 shows the accuracy of learning size-type from communication using the Learn, Cluster, and EM algorithms, and Figure 5.13 shows the comparison of the results of learning size-type between learning from group structure and learning from communication. The results show that the Cluster algorithm is the best, and it is least sensitive to the error of learning group structure; the accuracy of learning actor size-type doesn’t decrease too much. The Learn and EM algorithms are more sensitive to the accuracy of group structure; the accuracy of learning actor size-type decrease a lot. The overlapping clustering algorithm developed by Baumes et al., [35] tends to cluster groups into larger size groups (see Section 5.1.3.1). In the Learn algorithm, we know the probability distributions over group size, and if all of the sizes of clustered groups are greater than the group size preference of $Type_L$ actors, the learning results of the Learn algorithm will learn all actors as $Type_L$ actors. Therefore, we can see that the accuracies of the Learn algorithm are around 31%. The EM algorithm is based on the learned group structure and group evolution to learning the probability distributions over group size; hence, the error of learned group structure and group evolution will decrease the accuracy of learning the probability distributions over group size, and then the results of the EM algorithm have worse performances than the Learn algorithm.
Figure 5.11: The accuracy of learning the communication probabilities.
Figure 5.12: The accuracy (%) of learning size-type from communication using different algorithms with different time steps of data set.

Figure 5.13: The comparison of the results of learning size-type between learning from group structure and communication.
Figure 5.14: The accuracy (%) of learning ambition-type from communication using different algorithms with different time steps of data set.

Figure 5.15: The comparison of the results of learning ambition-type between learning from group structure and communication.
Figure 5.14 shows the accuracy of learning ambition-type from communication using both the A-EM and A-Optimal algorithms. Both algorithms still learn some information about actors’ ambition-types even though we have two more learning stages—learning group structure and group evolution. Figure 5.15 shows a comparison of the results of learning ambition-type between learning from communication and learning from group structure. Both algorithms are not too sensitive to the accuracy of learning group structure, because the decrease in the accuracy of learning actor ambition-type is not too much.

5.2 Impact of Learning on Prediction

In Section 4, we have discussed the learning process is time consuming; if in a community, there are $N$ actors and $K$ groups, then, in each time step, there are $2^{NK}$ possible states. If we have data for $T$ time steps, then the complexity of finding the optimal path using dynamic programming is $O(T \times 2^{NK})$. Even though we are able to develop some approximate and efficient algorithms, the learning process is still very time consuming. And so, it is useful to know whether or not a parameter has significant impact on the predictive performance before the learning takes place.

From Figure 5.3, it is clear that an actor’s size-type significantly impacts the predictive performance. While we could learn $\{E_i\}$ and $\theta_{\text{reward}}$, perhaps it is not useful to learn them if they don’t significantly improve predictive performance. Figure 5.16 shows the impact of the optimal predictor (all parameters set to their true values) versus choosing a random $\theta_{\text{reward}}$, and for various choices of $\{E_i\}$ (every actor assigned some fixed $E$ according to max{$E_i$}, min{$E_i$} or average{$E_i$}). We compute the average errors from 20 runs. As can be seen, the wrong $\theta_{\text{reward}}$ does significantly affect the performance, but using the wrong $\{E_i\}$ does not—which might be expected as the effects of initial conditions should equilibrate out. Thus, one can use simulation through ViSAGE to first investigate which parameters are important to learn before undertaking the time consuming process of learning.

We use the same setting described in Section 5.1 with communication probabilities $p_0 = 0.004\%$ and $p_1 = 4.5\%$ to examine the distribution of the number of communication link. Figure 5.17 shows the complementary cumulative distribution function (CCDF) of
Figure 5.16: The errors, $E_f$ and $E_p$, on the predictions using incorrect $\theta_{\text{reward}}$ and $\{E_i\}$. 
Figure 5.17: The complementary cumulative distribution function (CCDF) of the number of communication link on synthetic data.

Figure 5.18: CCDF of the number of communication link with different communication probabilities on synthetic data.
the number of communication link. Using the maximum likelihood method [74], it shows that the estimated power-law coefficient is 9.094 along with the Kolmogorov-Smirnov goodness-of-fit metric 0.0188 when $k \geq 9$. We know the number of communication link is influenced by the communication probabilities. Figure 5.18 shows few examples of CCDF of the communication link with different probabilities, and it shows that the number of communication link with larger communication probabilities has larger maximum of the number of communication link. It also shows that the power-law distribution on the number of communication link from the output of our model tends to have a bigger power-law coefficient.

5.3 Results on Real Data

Our results on real data are based on communications between actors without knowing the communication contents, because it is difficult to collect data that includes the group evolution and the content of the messages from the real world. Hence, we use the multistage learning process and algorithms in Section 4 to learn the parameters. For the first stage, we need to use the algorithms in Section 4.1 to obtain the group structure from communication dynamics—using the overlapping clustering algorithm [35] to obtain the group information at each time step. Then, we apply the matching algorithm in Section 4.2 to obtain out the group evolution. After having the data about the group evolution, we apply the techniques mentioned in Section 4.3 to learn the parameters from the group evolution.

5.3.1 Movie Newsgroup

We collected the communication data from a Movie newsgroup, which includes 1528 active actors and has total 103 time steps. In the first stage, we learn the group structure and the communication probabilities. Figure 5.19 shows the results of learning the communication probabilities ($P_g = P_1$ and $P_b = P_0$) at different time steps. A comparison from both figures show that people in the same group communicate more frequently than people not in the same group ($P_b$ is much smaller than $P_g$). In addition, from the upper figure, we see two ranges of obvious activities—more active communications (time step 0 to 60) and much reduced communications (time step 60 to 103).
Figure 5.19: The results of learning the communication probabilities on Movie newsgroup.

Table 5.4: The results of learning actors’ size-type on Movie newsgroup.

(a) Cluster algorithm

<table>
<thead>
<tr>
<th></th>
<th>$Type_S$</th>
<th>$Type_M$</th>
<th>$Type_L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Actor</td>
<td>822</td>
<td>550</td>
<td>156</td>
</tr>
<tr>
<td>Percentage</td>
<td>53.8%</td>
<td>36.0%</td>
<td>10.2%</td>
</tr>
</tbody>
</table>

(b) EM algorithm

<table>
<thead>
<tr>
<th></th>
<th>$Type_S$</th>
<th>$Type_M$</th>
<th>$Type_L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Actor</td>
<td>532</td>
<td>368</td>
<td>628</td>
</tr>
<tr>
<td>Percentage</td>
<td>34.8%</td>
<td>24.1%</td>
<td>41.1%</td>
</tr>
</tbody>
</table>

In the second learning stage, we use Algorithm 4.1 in Section 4.2 to obtain the group evolution, and then apply the learning algorithms, Cluster (Algorithm 4.3) and EM (Algorithm 4.4) in Section 4.3 to learn the appropriate values of actors’ size-types. Table 5.4 compares the classifications from the EM and Cluster algorithms. Based on the Cluster algorithm, the majority of actors are $Type_S$ which only meant that they joined the small groups—yet, this does not represent these actors’ preferences in group size. This result of
Table 5.5: The results of learning $\omega_\alpha$’s on Movie newsgroup

<table>
<thead>
<tr>
<th>“Structure” Act norm</th>
<th>join</th>
<th>stay</th>
<th>leave</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Agency” Act choice</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>join</td>
<td>0.11869</td>
<td>2.00142</td>
<td>2.00062</td>
</tr>
<tr>
<td>stay</td>
<td>0.00099</td>
<td>0.09080</td>
<td>1.38086</td>
</tr>
<tr>
<td>leave</td>
<td>-0.06444</td>
<td>-0.017090</td>
<td>1.68060</td>
</tr>
</tbody>
</table>

the Cluster algorithm matches the finding that was done by hand in Brian Butler’s social analysis in the newsgroups data [20]. However, the result of the EM algorithm shows that the number of TypeL increases 30.9%, the number of TypeS decreases 19%, and the number of TypeM decreases 11.9%.

According to the research data shown as above, there is a significant difference between both results: in Butler’s finding (Cluster algorithm), it is easy for one manually to locate which size of group an actor joined, but it is difficult for one manually to detect an actor’s actual group size preferences with considering social interactions, which can play an influential role in the actor’s decision making, between actors and groups. By applying the EM algorithm approach, one can not only consider the observable groups size that an actor joined but also the social interactions between the actors, such as, Join by reference, Qualification of actors and groups, and so on. As can be seen from the above data, the approach in using the EM algorithm yields similar results by showing that majority of actors are more likely to read news (TypeL) than to post news (TypeS) in a Movie newsgroup community. Actors belonging to TypeM and TypeL are easily attracted by TypeS to join small groups (observed actions). The reason is that classes with a higher qualification (TypeS) attract the lower classes (TypeM and TypeL), while it is less likely for classes with lower qualifications to influence the classes with higher qualifications.

Next, we use the learning algorithm in section 4 to learn $\omega_\alpha$’s. Table 5.5 shows the results about learning $\omega_\alpha$’s on the Movie newsgroup. The learning results show that the Movie newsgroup data set has much more friendly environment than the Enron email data set (shown in Section 5.3.2). In the Movie newsgroup, even though an individual is more likely to leave ($Act_{norm} = Leave$), the community would still reward the individual ($\omega_\alpha > 0$) to encourage higher interaction rate regardless of his/her choice ($Act_{choice}$).

We use the micro-laws which have been learned to setup the model using ViSAGE.
The CCDF of communication links

Figure 5.20: The complementary cumulative distribution functions (CCDF) of the number of communication link on Movie newsgroup.

to simulate 100 time steps, and then we use the techniques in Section 4.4.2 to examine the distribution of the number of communication link. Figure 5.20 shows the comparison of the complementary cumulative distribution functions (CCDF) of the number of communication link between training data and simulation data. The complementary cumulative distribution function (CCDF) of the number of communication link on the training data has estimated power-law coefficient 2.55 along with the Kolmogorov-Smirnov goodness-of-fit metric 0.0303 when $k \geq 10$. The simulation data has the complementary cumulative distribution function (CCDF) of the number of communication link with estimated power-law coefficient 6.822 along with the Kolmogorov-Smirnov goodness-of-fit metric 0.069 when $k \geq 27$.

5.3.2 Enron Email

Before using our approach to extract the information from Enron email data set, we use the strategies in [75] for cleaning Enron email from 13 November 1998 to 21 June 2002. Then, we obtain the communication network for 154 active actors. Table 5.6 shows the learning results of both EM and Cluster algorithms. The results from the EM and
Table 5.6: The results of learning actors’ size-type on Enron email

(a) Cluster algorithm

<table>
<thead>
<tr>
<th>Learn Actor’s Types</th>
<th>Leader</th>
<th>Socialite</th>
<th>Follower</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Actor</td>
<td>28</td>
<td>50</td>
<td>76</td>
</tr>
<tr>
<td>Percentage</td>
<td>18.2%</td>
<td>32.5%</td>
<td>49.3%</td>
</tr>
</tbody>
</table>

(b) EM algorithm

<table>
<thead>
<tr>
<th>Learn Actor’s Types</th>
<th>Leader</th>
<th>Socialite</th>
<th>Follower</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Actor</td>
<td>24</td>
<td>62</td>
<td>68</td>
</tr>
<tr>
<td>Percentage</td>
<td>15.6%</td>
<td>40.2%</td>
<td>44.2%</td>
</tr>
</tbody>
</table>

Table 5.7: The results of learning $\omega_\alpha$’s on Enron email

<table>
<thead>
<tr>
<th>“Structure” $\text{Act}_\text{norm}$</th>
<th>join</th>
<th>stay</th>
<th>leave</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Agency” $\text{Act}_\text{choice}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>join</td>
<td>-0.10073</td>
<td>0.71158</td>
<td>1.12028</td>
</tr>
<tr>
<td>stay</td>
<td>-0.01653</td>
<td>0.24063</td>
<td>0.05618</td>
</tr>
<tr>
<td>leave</td>
<td>0.00787</td>
<td>-0.14780</td>
<td>-1.01433</td>
</tr>
</tbody>
</table>

Cluster algorithms are very similar. The reason being that in a company, an individual’s preference is usually masked because the employees cannot change their “jobs” as freely as their responsibilities will change. Yet, in the Movie news group, actors can change groups anytime according to one’s desire. Therefore, the communications within the Enron email network are based upon work constraints, and employees ($\text{Type}_M$ or $\text{Type}_L$) cannot just join a group due to the attraction of the manager ($\text{Type}_S$) of that group.

Table 5.7 represents the results of learning $\omega_\alpha$’s from Enron email data set, and it shows an interesting result. In the Enron email, loyalty to the environment seems to be a major factor according to the end result—if both the individual’s original intent was to leave ($\text{Act}_\text{norm} = \text{Leave}$), and her $\text{Choice}$ action also showed the individual has actually left the group ($\text{Act}_\text{choice} = \text{Leave}$), this individual will be penalized ($\omega_\alpha < 0$). Yet, as for the ones who might have showed an intent to leave but his/her $\text{Choice}$ action was to join the group instead, the result showed that the individual was rewarded for not leaving. Therefore, from the results, we can also suspect that ”reward or penalty” from the environment can influence the decisions ($\text{Choice}$ actions) of the individuals.
Figure 5.21: The complementary cumulative distribution functions (CCDF) of the number of communication link on Enron email.

We use the same setting as Movie newsgroup, described in Section 5.3.1 to simulate the society’s future and to examine the distribution of the number of communication link. We compare the complementary cumulative distribution functions (CCDF) of the number of communication link between training data and simulation data on the Enron email data set (see Figure 5.21). We also use the maximum likelihood method [74] to estimated power-law coefficient; the training data has estimated power-law coefficient 5.115 along with the Kolmogorov-Smirnov goodness-of-fit metric 0.0624 when $k \geq 25$, comparing to the simulation data having power-law coefficient 5.509 along with the Kolmogorov-Smirnov goodness-of-fit metric 0.0993 when $k \geq 21$. From the complementary cumulative distribution functions (CCDF) of the number of communication link on Movie newsgroup and Enron email, it shows that our model represents Enron email better than Movie newsgroup.
CHAPTER 6
REVERSE ENGINEERING OF MULTI-ID USERS

In this chapter, we present some works on identifying users who employ multiple IDs on public communication forums. We describe a model for realtime communication exchange in public forums such as newsgroups and chatrooms. Next, we use this model to develop an efficient algorithm that identifies the users who post their messages under different IDs, multi-ID users, and to show the learning results.

A variety of approaches could be adopted to identify multi-ID actors. Though tracing the source of an Internet packet is not trivial, it is technically possible to identify the IP address of the packets sent by different IDs. While this certainly helps with the identification, this information is often insufficient, since a single IP address could represent a cluster of computers on a local network. Furthermore, access to those computers could be available to different people, as is the case for computer laboratories in universities. Another approach would be to analyze the semantics of the messages. For example, a particular actor may use a particular phrase in all of the IDs that it operates. Semantic analysis would attempt to discover stylistic similarities between posts of the same actor using different IDs. This entails sophisticated linguistic analysis that is generally not efficient, and may not be easy to automate, engendering a serious obstacle if the task is to identify multi-ID users operating in several communication networks, if not many thousands of them. Below is a sample log from a chatroom, illustrating the difficulty of any form of linguistic analysis.
In this dissertation, we present an altogether different approach, based upon statistical properties of the posts. To take the chatroom as an example, each post has three tags associated with it, \(<t, id, message>\): \(t\) is the time of the post; \(id\) is the ID posting the message; and \(message\) is the message that was posted. The question we ask is: *Is it possible to identify the multi-ID users based only on the times and IDs of the posts, and without paying attention to the actual message texts?*

We describe a model that provides a realistic emulation of a live public forum such as a chatroom or a newsgroup. This model is based on viewing each actor as a queue of “posts-in-waiting.” Based upon the messages that are delivered by the server and the list of its “friends,” every actor builds up its queue of jobs—the replies to messages received by the actor. The actor processes each of these messages one-by-one and submits each reply to the server; these messages then generate reply-jobs in queues of other IDs (the friends), and so on. Using such a model as a foundation, we discover statistical parameters that differentiate between multi-ID actors and single-ID actors. These parameters are the result of numeric and combinatorial analysis of the sequence of posts which (the analysis) does not use semantic information regarding the texts of the posts. The main observation, which forms the basis of our algorithm, is that the posts tagged with an ID
operated by a multi-ID actor do not appear as frequently as do the posts of single-ID users. Furthermore, all posts of multi-ID users are correlated; in particular, they do not occur too close together. Our algorithm detects the IDs whose posts display such statistical anomalies and identify them as coming from multi-ID users.

Our experiments based upon the model of an open forum establish the feasibility of the statistical identification of multi-ID users. The accuracy of our algorithm depends on the length of time over which data is collected, and, as expected, the more data is collected, the more accurate the results of the algorithm. Our error rates over long time periods are under 1%.

Very little work exists on determining the multi-ID users from open forum logs such as chat rooms. However, a number of researchers have mined for various other information on chat rooms, instant messaging forums and internet relay forums, [76, 77, 78, 79, 80, 81, 82]

6.1 Preliminaries

In order to make the discussion more precise, we will introduce some definitions here. We use small letters $i, j, k, \ldots$ to denote specific IDs that post on the open forum, and capital letters $A, B, C, \ldots$ to denote specific actors. There is a (many to one) mapping $\mathcal{A}$ that associates IDs with actors, thus $\mathcal{A}(i) = A$ means that actor $A$ is operating ID $i$. We use $\text{id}$ to denote the inverse of $\mathcal{A}$, thus $\text{id}(A) = \{i, j, k, \ldots\}$ is the set of IDs that actor $A$ is operating. The number of IDs that $A$ is operating is given by $|\text{id}(A)|$. If $|\text{id}(A)| > 1$, then $A$ is a multi-ID actor; otherwise, if $|\text{id}(A)| = 1$, $A$ is a single-ID actor.

We assume that there are relationships among the IDs, i.e., an ID $i$ can be the “friend” of one or more other IDs, $j_1, \ldots, j_k$. All these relationships are represented by the friendship-graph $G$, which is a graph in which all the IDs correspond to a vertex. There is an edge between two IDs if they are friends. We assume that, on the open forum, messages are exchanged only among friends. Two IDs that are not friends do not communicate during the time-period in question. We do not have access to the friendship graph, even though this graph governs the communication dynamics. We assume that every ID knows its friends.

A message posted by an ID has four attributes $\{t, \text{id}_S, \text{txt}, \text{id}_R\}$. Here $t$ is the time at
that the message was posted; id\(_S\) is the source-ID that is posting the message; txt is the
text of the message; and id\(_R\) is the set of IDs for the intended receivers of the message.
For simplicity, assume that id\(_R\) always contains exactly one intended receiver. Our results,
however, apply to the more general case. The time stamp is given by the server at the time
it posts the message onto the screen.

We assume that the receiver-ID, upon seeing the message posted, knows that the
message was intended for that ID. Not all four attributes are necessarily posted on the
open forum. For example, in chatrooms, id\(_R\) is not posted and the receiver-ID knows
implicitly if a message is meant for him/her from context of the text. In newsgroups, the
id\(_R\) is often included in the post. We define the forum log \(\mathcal{L}\) as the sequence of posts in
the form \(\{< t, id\_S >\}_i^{N}\), where \(N\) is the number of posts that were made. Note that we
ignore the possible information that is present in the message texts, even though in some
cases, the message reveals the receiver ID.

Using only the information in the forum log, our goal is to construct an algorithm
to determine multi-ID actors.

### 6.2 A Model of an Open Forum

We assume that the messages appear on a virtual screen in a sequence, and that
they are accessible to all actors participating in the forum. In the reality, though, these
messages may appear on different physical screens. We do not address the motivation or
the semantics of the messages; we do address the stochastic process that generates the
messages.

Underlying the communication is the friendship graph. For an illustration, consider
the friendship graph illustrated in Figure 6.1. In (a), we show the graph after mapping
each ID to its corresponding actor, thus for example, \( A(i,j) = B \). In (b), we show a second scenario, in which now \( A \) is a multi-ID actor, thus \( \text{id}(A) = \{i,j\} \). Our main concern is to determine how the forum log will be different when the IDs are operated by actors as in (a) versus (b). The fundamental observation that will aid us toward this main goal is that an actor has a finite bandwidth, \( i.e., \), it takes a user some amount of time to process messages onto the server. Thus, in (a), the messages for IDs \( i \) and \( j \) are processed by different actors, whereas in (b), a single actor has to divide her bandwidth between the two IDs. In particular, we will investigate the statistical consequences of this division of bandwidth among the IDs of a multi-ID actor.

To make this notion of bandwidth division more formal, we associate to each actor \( A \), a processing queue, \( Q_A \). The messages that this actor wishes to post are placed in the queue and processed in FIFO order. Further, after completing a message, the actor submits it to the server, which is also implementing a queue. We assume that messages arriving at the same time are placed in any queues in a random order. To illustrate, notice that according to the friendship graph, three conversations are going on, namely \((i,k), (j,l), (k,l)\). Suppose that \( i, j \) and \( k \) choose to initiate these conversations. When the actor graph is as in the (a) of Figure 6.1, the initial queue status is shown in the (i) of Figure 6.2. The (ii) of Figure 6.2 shows the server messages after each actor has processed one message and the (iii) of Figure 6.2 shows the resulting queues after the actors see the messages and initiate replies into their queues. Figure 6.3 shows the same evolution for the actor graph in the (b) of Figure 6.1 with one multi-ID user. Notice how the forum log will be different solely on account of the fact that actor \( A \) is now operating more than one ID. We
Figure 6.3: Example of processing messages using queue with one multi-ID user.

now describe a forum log generator that implements such a model. Our generator can be made very general, but, as an illustration, we present one of the simplest versions. There are three components:

**Initialization.** Let \( \{e_1, e_2, \ldots, e_m\} \) be a sequence of the edges in \( G \) (randomly ordered).

For every actor \( A \), a queue \( Q_A \) is defined; initially, all \( Q_A \)'s are empty. For every edge, one of its endpoints is randomly selected as the source \( s \) and the other endpoints is determined as the recipient \( r \). Note that \( s \) and \( r \) are IDs rather than actors. A reply message to \( r \) is pushed onto the queue \( Q_A(s) \), i.e., onto the queue of the actor corresponding to the source ID. The result of the initialization step is an array of queues, \([Q_{A_1}, Q_{A_2}, \ldots, Q_{A_n}]\), where each queue corresponds to unique actor operating on the forum. A queue may correspond to multiple IDs in the friendship graph \( G \). Note that some queues may be empty.

**Processing by Actors.** Every actor \( A_i \):

1. Processes and removes the first message on its queue. The time to process this message \( \tau \) could be set randomly to simulate long and short messages. After processing this message, the actor submits it to the forum.

2. Scans the forum postings for any messages that are addressed to any ID in \( \text{id}(A_i) \). Each such message generates a reply to the poster of the message. This reply is pushed onto \( Q_{A_i} \).

**Processing by Forum.** The forum has a global queue \( Q_F \) of messages to be posted. The messages arrive according to the times they were submitted to the forum by actors;
if posts arrive at the same time, the forum sorts them into an arbitrary (random) order. The forum processes its queue using FIFO order, taking a time of 1 unit to post a message.

### 6.3 Multi-user Identification Algorithm

The input to the multiuser-identification algorithm is the forum log \( \mathcal{L} = \{< t, id_s >\}_{i=1}^{N} \), the times of the postings and the IDs that made the postings. To design an identification algorithm, we need to determine a statistical property of the communication exchange which separates multi-users from users that employ one ID only. The intuition behind our algorithm is that since a user has only one queue, it can only process messages sequentially. This is independent of whether she is operating one ID or multiple IDs. Suppose that, on average, it takes an actor \( \tau_0 \) to complete a message. Then, the time gap between two messages posted by the same actor who is using different IDs will on average be \( \tau_0 \) time units. On the other hand, if two different actors are posting messages for a pair of IDs, then this restriction does not hold. In fact, over a long enough time period, one expects that the posts of these two IDs may arrive arbitrarily close to each other.

Let \( i, j \) be two IDs, and consider the two subsequences of the forum log consisting only of the posts of each of these IDs: \( \{< t_i, i >\} \) and \( \{< t_j, j >\} \). Define the set of separation times, \( \{D_{ij}\} \), as the set of time differences between consecutive posts of the two IDs—i.e., for every pair of times \( t_i, t_j \) at which \( i \) posts followed by \( j \) or \( j \) followed by \( i \), with no posts made in between these two posts, then \( |t_j - t_i| \in \{D_{ij}\} \). We define two separation indices, the mean separation index \( M(i, j) \) for IDs \( i, j \), and the minimum separation index \( \min D(i, j) \):

\[
M(i, j) = \text{mean}\{D_{ij}\}
\]

\[
\min D(i, j) = \text{min}\{D_{ij}\}
\]

We expect that if \( \mathcal{A}(i) = \mathcal{A}(j) \), then these separation indices will be significantly larger than if \( \mathcal{A}(i) \neq \mathcal{A}(j) \). More specifically, if \( M(i, j) \) and \( \min D(i, j) \) are small, then it is
not possible that $A(i) = A(j)$. On the other hand, if they are large, then it would be extremely unlikely that $A(i) \neq A(j)$ on account of the independence of the actors behind the IDs, and hence it is likely that $A(i) = A(j)$. The intuition we have described would hold for any model of the forum log that assumes a finite bandwidth for not only the actors but also the sequential processing of messages. The quantities that would vary from model to model would be exactly how large the separation indices would have to get before one could declare that a pair of IDs is suspicious and is probably from a multi-ID actor. In order to test these hypotheses, we simulate a forum log according to the model in the previous section and compute the statistics $M(i,j)$ and $\min D(i,j)$ for every pair of IDs. In Table 6.1 below, we give the averages of these statistics over pairs of IDs from the same actor versus pairs from different actors separately. The statistics in Table 6.1 were obtained assuming that the time $\tau$ that an actor takes to prepare a post is 250 units. It is clear that the separation indices are drastically different depending on whether the pair of IDs is on the same actors versus on different actors. Further, the histograms in Figure 6.4 indicate not only that the separation indices are different on average, but also that the distributions are well separated.

We are thus led to the Algorithm 6.1 for identifying multi-ID actors:

Table 6.1: Some statistical properties of the $\{D_{ij}\}$

<table>
<thead>
<tr>
<th></th>
<th>$A(i) = A(j)$</th>
<th>$A(i) \neq A(j)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean ${M(i,j)}$</td>
<td>382.79</td>
<td>264.63</td>
</tr>
<tr>
<td>mean ${\min D(i,j)}$</td>
<td>156.36</td>
<td>3.22</td>
</tr>
</tbody>
</table>
The Histogram of \( \min D(i,j) \)

(a) 

(b) 

Figure 6.4: Histograms of \( \min D(i,j) \). In (a), \( A(i) = A(j) \); in (b), \( A(i) \neq A(j) \).

Algorithm 6.1: Multi-user Identification Algorithm

<table>
<thead>
<tr>
<th>Input</th>
<th>Forum Log ( \mathcal{L} = {&lt;t, id_s&gt;_{i=1}^N } )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Pairs of IDs on the same actor</td>
</tr>
</tbody>
</table>

1. for every pair of IDs \( i, j \) do
   2. compute \( \min D(i,j) \)
   3. endfor

4. Cluster the values of \( \min D(i,j) \) into two groups, \( G_{\text{large}} \) containing the large values, and \( G_{\text{small}} \) containing the smaller. Every pair \( (i,j) \) belongs to one of these groups.

5. return all the pairs \( (i,j) \in G_{\text{large}} \)

Defining the equivalence relation \( i \equiv j \) iff \( (i,j) \in G_{\text{large}} \), the equivalence classes of the IDs then partitions the IDs into sets, each of which are operated by one actor. Thus, we can identify all the IDs common to a given actor from the algorithm above.

6.4 Experiments

The model for the dynamics of the forum log has the following seven parameters:

- \textit{del}: The average time for an actor to compose a message and submit it to the forum server. Although, this average composition time can be, generally, time and/or
actor dependent, for our simplified model, we assumed it to be a constant.

\( wid \): A parameter specifying how much the actual time to compose a message can vary from the average time \( del \). If \( \tau \) is the time to compose a message, then we assume that \( \tau \) is uniformly distributed in \([del - \frac{1}{2} wid, del + \frac{1}{2} wid]\). In general, \( wid \) can be time and/or actor dependent, but for our simplified model, we keep \( wid \) fixed. \( wid \) can be viewed as a noise parameter that introduces some non-determinism into the forum log.

\( len \): the length of a post by a actor; it is assumed that \( len \) determines the minimal time-period needed for a actor to compose the message; in general, \( len \) is time- and actor-dependent, but for this model, \( len \) is a constant.

\( run \): the total number of time units for which the forum log is generated.

\( nID \): the total number of IDs participating in the communication exchange.

\( maxID \): the maximum number of IDs employed by a single actor—a \( k \)-ID actor operates \( k \) IDs.

\( nFriend \): the average number of friends an ID has in the friendship graph.

In our simulations, we fixed \( nID = 500 \) with about an equal number of 1-ID actors, 2-ID actors, 3-ID actors and 4-ID actors, thus \( maxID = 4 \). We fixed \( nFriend = 5 \), and used different values of \( del, wid \) and \( run \) to determine how these three parameters influence the accuracy of the detection algorithm, which we define as the percentage of pairs of IDs that are assigned to the correct group (multi or single).

\[ Accuracy = \frac{\text{The number of pairs (i,j) assigned into correct group}}{\text{The total number of (i,j) pairs}} \quad (6.1) \]

To implement step 4 of the Algorithm 6.1, we used a standard \( K \)-means algorithm \([64, 65]\), with \( K \) set to 2.

The details of the simulation are as follows. First, randomly generate a friendship graph with average degree 5. Using this friendship graph, we run the forum generator for \( run \) timesteps, to generate a forum log. We then implement the multi-user identification
algorithm, Algorithm 6.1, to determine which ID’s are from single-ID actors and which IDs are on the same actor. We then compute the accuracy, and repeat this entire simulation over 10 times to get a more accurate estimate of the average accuracy.

Table 6.2 illustrates how the accuracy depends on \( wid \) and \( del \) when \( run = 1,000,000 \) and when \( run = 10,000 \). When the observation sequence is long enough (\( run = 1,000,000 \)), the accuracy is almost 100% and is not influenced much by \( wid \), i.e., the fact that messages take random amounts of time to compose does not seem to heavily affect the algorithm’s accuracy. However, there is a slight decrease in performance when \( del \) increases. This is mostly due to the fact that there are fewer posts (data) when \( del \) increases, as the observation period is fixed. Shown in Figure 6.5 is the dependence of the accuracy on \( del \), the time to compose a message, for different value of the noise parameter \( wid \).

### Table 6.2: The dependence of the Accuracy (in %) on \( del, wid, run \)

<table>
<thead>
<tr>
<th>( del )</th>
<th>( wid )</th>
<th>( 0 )</th>
<th>( 50 )</th>
<th>( 100 )</th>
<th>( 250 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td></td>
<td>99.9992</td>
<td>99.9998</td>
<td>99.9999</td>
<td>99.9996</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>99.9995</td>
<td>99.9997</td>
<td>99.9993</td>
<td>99.9993</td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td>99.9976</td>
<td>99.9972</td>
<td>99.9979</td>
<td>99.9972</td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td>99.7129</td>
<td>99.7452</td>
<td>99.7740</td>
<td>99.7695</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( del )</th>
<th>( wid )</th>
<th>( 0 )</th>
<th>( 50 )</th>
<th>( 100 )</th>
<th>( 250 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td></td>
<td>98.1425</td>
<td>97.98</td>
<td>97.8906</td>
<td>98.0964</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>96.2923</td>
<td>96.634</td>
<td>96.6173</td>
<td>96.8459</td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td>91.6025</td>
<td>92.3209</td>
<td>92.3017</td>
<td>92.2991</td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td>—</td>
<td>84.8614</td>
<td>82.3213</td>
<td>79.4296</td>
</tr>
</tbody>
</table>

Notice that when \( run \) is small (i.e., the observation period is small), the accuracy considerably drops. This is because of the significant drop in the amount of data available for the classification into single versus multi-ID actors. This is illustrated in Table 6.3, where we show the number of posts that the various types of actors make. What is also
Figure 6.5: The dependence of the Accuracy (in %) on \( del \) and \( wid \).
Table 6.3: The average number of messages posted by different types of actors

(a) run = 1,000,000

<table>
<thead>
<tr>
<th>del</th>
<th># of IDs operated by an actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>2876.6 1910.8 1326.4 1015.9</td>
</tr>
<tr>
<td>500</td>
<td>1436.6 954.8 663.3 507.3</td>
</tr>
<tr>
<td>1000</td>
<td>717.1   477.2 331.5 253.8</td>
</tr>
<tr>
<td>5000</td>
<td>142.9   95.7  66.6  50.7</td>
</tr>
</tbody>
</table>

(b) run = 10,000

<table>
<thead>
<tr>
<th>del</th>
<th># of IDs operated by an actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>23.95 18.89 13   9.75</td>
</tr>
<tr>
<td>500</td>
<td>12.41 9.18  6.31 4.75</td>
</tr>
<tr>
<td>1000</td>
<td>5.39  4.23  2.98 2.25</td>
</tr>
<tr>
<td>5000</td>
<td>0.38  0.49  0.33 0.25</td>
</tr>
</tbody>
</table>

illustrated in Table 6.3 is the effect of the limited bandwidth assumption that we place on the actors. The 4-ID actors post less frequently than the 1-ID actors. While one would expect the frequency of posting to have dropped by a factor of 4 in going from the 1-ID actors to the 4-ID actors, it is not quite a factor of 4. The reason is that some of the 1-ID actors are friends with IDs belonging to 4-ID actors, and this reason means that the frequency of posting of these 1-ID actors will be slowed down by the fact that they have to wait longer for the responses from the 4-ID actors.

In principle the average number of posts could be used to discriminate between multi-ID actors and single-ID actors; however the distributions of this statistic are not as separated as with minD.
6.5 Conclusions

We have presented an algorithm to identify multi-ID users, the justification of which is based on a novel and reasonable model of communication exchange in a public forum. Along this direction, the model could be considerably expanded to include more realistic and nondeterministic phenomena such as multi-party exchange, server delays, and different composition time distributions for each actor. Further, one could allow the friendship graphs and the general communication dynamics on the forum to be time varying. It is also possible to incorporate statistical parameters of the real communication forum into the model.

What we have demonstrated is under the broad assumption of a finite processing power for each actor and the fact that messages processed sequentially. An actor operating multiple IDs will give his/her posts away, and those IDs operated by single-ID actors will have different posting statistics to the normal. In particular, the posts of these multiple IDs will be neither independent nor as frequent. We see that introducing randomness into the time to compose a message does not have much effect on the algorithm, nor did changing the average time to compose a message—i.e., the algorithm is quite robust to the specific details of the model, which is comforting. Our simulations show that, if the time to compose a message is 5000 units (a unit being the time for the server to process a message), then with 1,000,000 time units of observation, we can essentially obtain 100% accuracy. To put this in perspective, if it takes about 10 seconds to compose a message in a chatroom, then we need under 1 hour of observation.

Of course, it is important to realize that every identification algorithm can potentially be deceived by a skillful actor who intends to hide its multiple IDs. However, the attempt to deceive our algorithm will likely cause a time delay in the postings of the user (or other irregularities). A systematic delay in a user’s activity may in turn be employed for the identification purposes.
We have presented a parameterized agent-based hidden Markov model for learning the micro-laws that govern actors’ dynamics and a society’s social group dynamics. Parameterized stochastic models in machine learning are not uncommon; however, in the context of learning social laws, they have scarcely been applied. Since the model is sufficiently general and grounded in social science theory, some instance of the model should be appropriate for any given society. The model can be integrated with different micro-laws in a society that one wishes to find. Furthermore, by applying the same learning techniques, one will be able to find the appropriate parameters within the micro-laws that make a society tick. Therefore, under this scenario, almost any general model of this form which is founded in social science theory will yield outputs that can serve as productive reference to one’s decision making or stimulating triggers to new research studies.

We have also developed a multistage learning process to learn micro-laws from group evolution, group structure, and communication data. The benefits of the multistage learning process are, at each learning stage, to extract different information about actors and society dynamics, to reduce the impact of data noise on learning, to develop more efficient algorithms, and to setup a framework for more easily evaluating the performance of algorithms. We need to develop efficient algorithms to achieve the object in each learning stage. We have developed algorithms to discover appropriate parameters for the micro-laws, including independent discrete parameters, dependent discrete parameters, and continuous parameters. We have also developed a greedy matching algorithm to find the group evolution more efficiently. To learn the group structure, we adopt an overlapping clustering algorithm, and then we learn the communication probabilities among actors to verify the performance of the overlapping clustering algorithm. Our main contributions are the application of efficient algorithms and heuristics toward learning the parameters in the specific application of modeling social groups and communications.

Within our model, one of the interesting conclusions from a social science point of view is the ability to identify which parameters have a significant impact on the future
evolution of the society. In particular, the initial resource allocation (a fixed intellectual resource of the actor) does not seem to be a dominant factor; but the actor’s types, and the penalty/reward structure for the society have significant impacts on the evolution. These types of conclusions are exactly the types of information a social scientist is trying to discern from observable data. Our results are not completely surprising. A community which excessively penalizes traitors (say by sever dismemberment), does necessarily see far fewer changes in its social group structure than one that doesn’t. Our observations indicate that learning effort should be focused on certain parameters and perhaps not as importantly on others, which can further enhance the efficiency of the learning. Having determined what the important parameters are, one can then go ahead to learn their values for specific communities, and to make conclusions regarding the nature and functioning of the community under consideration.

The results also show some interesting findings. People sometimes change their behavior because of the influence from the outside environment, and how they act is different from how they really want to behave. If only based on the observed behavior such as the Cluser algorithm, people can only capture the explicit behaviors. However, through the Learn algorithm, people not only can capture actors’ observable behaviors, but also the influence from the society (e.g., friendship) which enable them to discover actors’ implicit behaviors. The learning results on the real data, Movie newsgroup and Enron email data, distinguished some different actors behaviors (micro-laws) between different communities. In online communities (e.g., Movie newgroup), people are usually more able to follow their preferences and have less pressures to do things from the communities than they are in a cooperator (e.g., Enron email data).

From the learning results on synthetic data, we know that the last two learning stages, learning from both group evolution and group structure, have a nice performance and the learning accuracy is good. In the first learning stage, learning from communications, the overlapping clustering algorithm tends to cluster actors into larger groups which causes the learning is underperformance. There is still a space of improvement for the overlapping clustering algorithm: setting a threshold to limit some actors classed into a group, and so the algorithm can cluster smaller size of overlapping groups. People can set a threshold to limit the outsider be included into the group so it can decrease
the clustered group size. For simulating the power-law networks, the results show our model has a certain sharp of the probability distribution of the number of the communication link. In the future, we can implement a more complicated communication model to improve the outputs and also determine how the communication model effects the probability distribution of the number of the communication link. Nevertheless, in this doctoral dissertation, we present a framework using machine learning methodology (models, algorithms, and experimental data) to discover the agent dynamics that drive the evolution of the social groups in a community and provide another method to solve the problems of hidden Markov models.
REFERENCES


Our model is very flexible and extendable, and users can change any function and variable as necessary. We describe the functions and variables used in the default model in the following sections.

A.1 Actor Properties

- $\text{isclass}_i$: An integer number representing actor $i$’s size-type, ($\S$3.1.1).

- $\text{raclass}_i$: A real number representing actor $i$’s ambition-type, ($\S$3.1.1).

- $\chi_i$: The bounded size preference of actor $i$, 
  \[ \chi_i = \frac{e^{+\text{raclass}_i} - e^{-\text{raclass}_i}}{e^{+\text{raclass}_i} + e^{-\text{raclass}_i}} \]  
  (A.1)

- $\text{iaclass}_i$: An integer number representing actor $i$’s ambition-type, ($\S$3.1.1),
  \[ \text{iaclass}_i = \begin{cases} 
  1 & \text{: Type A, if } \chi_i < -0.5 \\
  5 & \text{: Type B, if } \chi_i > 0.5 \\
  3 & \text{: Type C, otherwise} 
\end{cases} \]  
  (A.2)

- $\delta_i$: The amount how quickly actor $i$’s rank increases in a group for each time unit, ($\S$3.1.2), 
  \[ \delta_i = \zeta - \chi_i, \]  
  (A.3)
  where $\zeta$ is a constant.

- $time^t_{(i,l)}$: The amount of time actor $i$ spends in a group $l$ at time $t$, ($\S$3.1.2).

- $r^t_{(i,l)}$: Indicates the rank of actor $i$ in group $l$ at time $t$, ($\S$3.1.2),
  \[ r^t_{(i,l)} = \frac{time^t_{(i,l)} \delta_i}{\sum_{j \in G_l} time^t_{(j,l)} \delta_j}, \]  
  (A.4)
where $G^t_l$ is group $l$ at time $t$, and actor $j$ denotes each actor in $G^t_l$.

- $q^t_i$: Indicates the qualification of actor $i$ at time $t$, (§3.1.3),

\[
q^t_i = \frac{\sum_{i \in G^t_l} r^t_{(i,j)} |G^t_l|}{\sum_{i \in G^t_l} |G^t_l|},
\]  

(A.5)

where $G^t_l$ is any group actor $i$ is a member, and $|G^t_l|$ is the size of group $l$; the number of actors in group $l$.

### A.2 Group Properties

- $Q^t_l$: The qualification of group $l$ at time $t$, (§3.1.3),

\[
Q^t_l = \sum_{i \in G^t_l} q^t_i \times r^t_{(i,i)}.
\]  

(A.6)

### A.3 Resources

- $R^t_i$: The available resources actor $i$ has in time step $t$, and the default initial value ($R^0_i$) is $InitSocCap$, (§3.1.4),

\[
R^t_i = \begin{cases} 
R^{t-1}_i + \Delta R^{t-1}_i - \eta, & \text{when } ExcessType = 0 \\
1 - \eta, & \text{when } ExcessType = 1 
\end{cases},
\]  

(A.7)

where $\Delta R^{t-1}_i$ indicates the amount of changes of actor $i$’s available resources at time $t - 1$, and

\[
\eta = \begin{cases} 
0, & \text{when } t \leq ChangeBegin \\
(t - ChangeBegin) \times ChangeAmount, & \text{when } t > ChangeBegin 
\end{cases}.
\]  

(A.8)

See Section A.7 for the specification of returned social capital about $\Delta R^{t-1}_i$.

- $L^i_{br}$: Bridging Social Capital—the number of groups actor $i$ is in, (§3.1.4),

\[
L^i_{br} = |G : i \in G|.
\]  

(A.9)
• $L_{bo}^i$ : Bonding Social Capital—the sum of the ranks of actor $i$, (§3.1.4),

$$L_{bo}^i = \sum_{G_i; i \in G_i} r_{(i,l)}^i.$$  \hspace{1cm} (A.10)

• $\phi_i$ : Bridging Coefficient, (§3.1.4),

$$\phi_i = \begin{cases} 
  \text{BridgingParam1} \\
  \frac{\text{BridgingParam1}}{(\text{BridgingParam2} + R_i^t)^{\text{BridgingParam3}}} 
\end{cases} \text{, when BridgingCostType} = 0$$

$$\phi_i = \begin{cases} 
  \text{BridgingParam1} \\
  \frac{\text{BridgingParam1}}{(\text{BridgingParam2} + R_i^t)^{\text{BridgingParam3}}} 
\end{cases} \text{, when BridgingCostType} = 1.$$  \hspace{1cm} (A.11)

• $\psi_i$ : Bonding Coefficient, (§3.1.4),

$$\psi_i = \begin{cases} 
  \text{BondingParam1} \\
  \frac{\text{BondingParam1}}{(\text{BondingParam2} + R_i^t)^{\text{BondingParam3}}} 
\end{cases} \text{, when BondingCostType} = 0$$

$$\psi_i = \begin{cases} 
  \text{BondingParam1} \\
  \frac{\text{BondingParam1}}{(\text{BondingParam2} + R_i^t)^{\text{BondingParam3}}} 
\end{cases} \text{, when BondingCostType} = 1.$$  \hspace{1cm} (A.12)

• $e_i^t$ : The excess resources for actor $i$ at time $t$, (§3.1.4),

$$e_i^t = R_i^t - \phi_i L_{br}^i - \psi_i L_{bo}^i.$$  \hspace{1cm} (A.13)

### A.4 Action Probabilities

• $P_{+1}(e_i^t)$, $P_0(e_i^t)$, and $P_{-1}(e_i^t)$ : The probabilities of actor $i$ at time $t$ to join a group, do nothing, or leave a group, respectively, (§3.2.1),

$$P_{+1}(e_i^t) = \frac{P'_{+1}}{P'_{+1} + P'_{-1} + P'_{0}}.$$  \hspace{1cm} (A.14)

$$P_0(e_i^t) = \frac{P'_{0}}{P'_{+1} + P'_{-1} + P'_{0}}.$$  \hspace{1cm} (A.15)

$$P_{-1}(e_i^t) = \frac{P'_{-1}}{P'_{+1} + P'_{-1} + P'_{0}}.$$  \hspace{1cm} (A.16)
where

\[ P'_{+1} = \frac{A_{\text{plus}}}{1 + e^{-Rho_{\text{plus}} \times \left(\frac{e^{i_t}}{\text{Threshold}} - 1\right)}}, \quad (A.17) \]

\[ P'_{0} = A_{\text{zero}} \times e^{-Rho_{\text{zero}} \times \left|e^{i_t}\right|/\text{Threshold}}, \quad (A.18) \]

\[ P'_{-1} = \frac{A_{\text{minus}}}{1 + e^{Rho_{\text{minus}} \times \left(\frac{e^{i_t}}{\text{Threshold}} + 1\right)}}, \quad (A.19) \]

- \textit{Act}_n\text{orm} : Normative Action, (§3.2.1),

\[ \text{Act}_n\text{orm} = \arg\max_{\text{action}} \left( P_{\text{action}}(e^{i_t}) \right), \quad (A.20) \]

where \( P_{\text{action}} \in \{P_{+1}, P_{0}, P_{-1}\} \).

- \textit{Act}_c\text{hoice} : Choice Action, (§3.2.2),

\[ \text{Act}_c\text{hoice} = \text{random} \left( P_{+1}, P_{0}, P_{-1} \right), \quad (A.21) \]

where \text{random} \left( P_{+1}, P_{0}, P_{-1} \right) means randomly choose an action based on the probabilities of actions, \( P_{+1}(e^{i_t}), P_{0}(e^{i_t}), \) and \( P_{-1}(e^{i_t}) \).

### A.5 Actor and Group Fitting

- \textit{SizeAff}_{(i,l)}^t : The group size affinity actor \( i \) has for group \( l \) at time \( t \), (§3.2.2),

\[ \text{SizeAff}_{(i,l)}^t = \frac{1}{\Gamma(is\text{class}_i + 1) \times \Theta is\text{class}_i} \times \left( |G_{i}^t| + 2 \right)^{(is\text{class}_i - 1)} \times e^{-(|G_{i}^t| + 2)/\Theta} \]

\[ \times e^{-(|G_{i}^t| + 2)/\Theta}, \quad (A.22) \]
where

\[
\Gamma(isclass_i + 1) = \begin{cases} 
\text{GammaLeader}, & \text{if the actor is a Type}_S \\
\text{GammaSocialite}, & \text{if the actor is a Type}_M \\
\text{GammaFollower}, & \text{if the actor is a Type}_L 
\end{cases}
\]  
(A.23)

- \textit{QualAff}^{t}_{(i,l)}: The qualification affinity actor \textit{i} has for group \textit{l} at time \textit{t}, based on a comparison between the actor’s qualifications and the group’s qualifications, (§3.2.2),

\[
\text{QualAff}^{t}_{(i,l)} = \text{QualRange} + (1 - \text{QualRange}) \times (1 + \tanh(\xi_q))/2, \quad (A.24)
\]

where

\[
\xi_q = \begin{cases} 
\text{QualSensitivity} \times (Q^t_l - q^t_i)/q^t_i, & \text{if } q^t_i \neq 0 \\
\text{QualSensitivity}, & \text{otherwise}
\end{cases}
\]  
(A.25)

- \textit{ActorAff}^{t}_{(l,i)}: The group’s actor affinity group \textit{l} has for actor \textit{i} at time \textit{t}, based on a comparison between the group’s qualifications to the actor’s qualifications, (§3.2.2),

\[
\text{ActorAff}^{t}_{(l,i)} = \text{QualRange} + (1 - \text{QualRange}) \times (1 + \tanh(\xi_a))/2, \quad (A.26)
\]

where

\[
\xi_a = \begin{cases} 
\text{QualSensitivity} \times (q^t_i - Q^t_l)/Q^t_l, & \text{if } Q^t_l \neq 0 \\
\text{QualSensitivity}, & \text{otherwise}
\end{cases}
\]  
(A.27)

- \textit{Rep}^{t}_{(i,l)}: The repulsion actor \textit{i} has for group \textit{l} at time \textit{t}, (§3.2.2),

\[
\text{Rep}^{t}_{(i,l)} = \left(1 - \frac{r^t_{(i,l)}}{\sum_{G^t_k \in D_m} r^t_{(i,k)}}\right) \times (1 - \text{SizeAff}^{t}_{(i,l)}), \quad (A.28)
\]

where \(D_m\) is a set of groups actor \textit{i} is currently associated with.
A.6 Association Probabilities

- \( P_{\text{Join}}^t(i,l) \): The Group Join Probability—the probability for actor \( i \) to join group \( l \) at time \( t \), (§3.2.2),

\[
P_{\text{Join}}^t(i,l) = \frac{\text{SizeAff}^t(i,l) \times \text{QualAff}^t(i,l)}{\sum_{G_k \in D_c} \text{SizeAff}^t(i,k) \times \text{QualAff}^t(i,k)}, \tag{A.29}
\]

where \( D_c \) is the set of candidate groups that actor \( i \) can possibly apply to.

- \( P_{\text{Leave}}^t(i,l) \): The Group Leave Probability—the probability for actor \( i \) to leave group \( l \) at time \( t \), (§3.2.2),

\[
P_{\text{Leave}}^t(i,l) = \frac{\text{Rep}^t(i,l)}{\sum_{G_k \in D_m} \text{Rep}^t(i,k)}, \tag{A.30}
\]

where \( D_m \) is the set of groups of which the actor has a membership.

- \( P_{\text{Reject}}^t(i,l) \): The Group Reject Probability—the probability for group \( l \) to reject actor \( i \)'s application for joining the group, (§3.2.2),

\[
P_{\text{Reject}}^t(i,l) = \text{ActorAff}^t(l,i). \tag{A.31}
\]

A.7 Returned Social Capital Models

The returned social capital function, \( f_{\Delta R}(R_i^t, \text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) \), specifies how an actor’s returned social capital (available resources) changes at each time step, (§3.2.3),

\[
f_{\Delta R}(R_i^t, \text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) = \text{PenaltyW}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) \times \text{Penalty}(R_i^t) + \text{RewardW}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) \times \text{Reward}(R_i^t), \tag{A.32}
\]
where

\[
\text{Penalty}(R^t_i) = \begin{cases} 
-0.35 \ast (0.01R^t_i + (1.99R^t_i)^{1.05}) & \text{, if } \text{PenaltyFun} = 0 \\
-0.35 \ast (R_i^{t0.5} + 100R^t_i) & \text{, if } \text{PenaltyFun} = 1 \\
-0.35 \ast (R_i^{t2} + 100R^t_i) & \text{, if } \text{PenaltyFun} = 2 \\
-1/(1 + e^{(5.0 - 10R^t_i)}) & \text{, if } \text{PenaltyFun} = 3
\end{cases}
\]  

(A.33)

\[
\text{Reward}(R^t_i) = -\text{Penalty}(1.0 - R_i^t)
\]  

(A.34)

\[
\text{PenaltyW}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) = \frac{1}{(1 + e^{((1+\text{SCTable}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}))×5)})}
\]  

(A.35)

\[
\text{RewardW}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) = \frac{1}{(1 + e^{((1-\text{SCTable}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}))×5)})}
\]  

(A.36)

\[
\text{SCTable}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) = \begin{cases} 
\text{SCTableR0}[1], \omega_1 & \text{when } \text{Act}_{\text{norm}} = \text{join and } \text{Act}_{\text{choice}} = \text{join} \\
\text{SCTableR0}[2], \omega_2 & \text{when } \text{Act}_{\text{norm}} = \text{join and } \text{Act}_{\text{choice}} = \text{stay} \\
\text{SCTableR0}[3], \omega_3 & \text{when } \text{Act}_{\text{norm}} = \text{join and } \text{Act}_{\text{choice}} = \text{leave} \\
\text{SCTableR1}[1], \omega_4 & \text{when } \text{Act}_{\text{norm}} = \text{stay and } \text{Act}_{\text{choice}} = \text{join} \\
\text{SCTableR1}[2], \omega_5 & \text{when } \text{Act}_{\text{norm}} = \text{stay and } \text{Act}_{\text{choice}} = \text{stay} \\
\text{SCTableR1}[3], \omega_6 & \text{when } \text{Act}_{\text{norm}} = \text{stay and } \text{Act}_{\text{choice}} = \text{leave} \\
\text{SCTableR2}[1], \omega_7 & \text{when } \text{Act}_{\text{norm}} = \text{leave and } \text{Act}_{\text{choice}} = \text{join} \\
\text{SCTableR2}[2], \omega_8 & \text{when } \text{Act}_{\text{norm}} = \text{leave and } \text{Act}_{\text{choice}} = \text{stay} \\
\text{SCTableR2}[3], \omega_9 & \text{when } \text{Act}_{\text{norm}} = \text{leave and } \text{Act}_{\text{choice}} = \text{leave}
\end{cases}
\]  

(A.37)

where \(\{\omega_\alpha|\alpha = 1, 2, \cdots, 9\}\) are the society reward/penalty parameters in the Agency and Structure Table, see Table 3.1.
APPENDIX B
Configuration

B.1 Parameters

1. Actors: the total number of actors in the population.

2. Groups: the maximum number of groups during the simulation.

3. TimeSteps: the simulation time.

4. LeaderPercent: the fraction of population that is $Type_S$.

5. SocialitePercent: the fraction of population that is $Type_M$.

6. MostAmbPercent: the fraction of population that is $Type_A$.

7. MedAmbPercent: the fraction of population that is $Type_B$.

8. SCTableR0, SCTableR1, SCTableR2: the values of the Agency and Structure Table, see Table 3.1 and Eq. A.37.

9. Aplus, Azero, Aminus, Rhoplus, Rhzero, Rhominus and Threshold: the parameters for computing the action probabilities $P_{+1}(e_i^t)$, $P_b(e_i^t)$, and $P_{-1}(e_i^t)$, see Eq. (A.16).

10. BridgingCostType, BridgingParam1, BridgingParam2, BridgingParam3: the parameters for computing the bridging cost, see Eq. (A.11).

11. BondingCostType, BondingParam1, BondingParam2, BondingParam3: the parameters for computing the bonding cost, see Eq. (A.12).

12. ExcessType, ChangeBegin, ChangeAmount: the parameters for computing the available resources, see Eq. (A.7) and Eq. (A.8).

13. InitSocCap: the initial value of available resources.
14. **Theta, GammaLeader, GammaSocialite, GammaFollower**: the parameters for computing $SizeAf f_{(i,l)}^t$, see Eq. (A.22) and Eq. (A.23).

15. **QualRange, QualSensitivity**: the parameters for computing $QualAf f_{(i,l)}^t$ and $Actor Af f_{(l,i)}^t$, see Eq. (A.24), (A.25), (A.26) and (A.27).

16. **PenaltyFun**: The switch of using different penalty function, see Eq. (A.33).

### B.2 Default Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Name</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>500</td>
<td>BridgingCostType</td>
<td>1</td>
</tr>
<tr>
<td>Groups</td>
<td>2000</td>
<td>BridgingParam1</td>
<td>0.3</td>
</tr>
<tr>
<td>TimeSteps</td>
<td>1000</td>
<td>BridgingParam2</td>
<td>1.0</td>
</tr>
<tr>
<td>LeaderPercent</td>
<td>0.0</td>
<td>BridgingParam3</td>
<td>1.0</td>
</tr>
<tr>
<td>SocialitePercent</td>
<td>1.0</td>
<td>BondingCostType</td>
<td>1</td>
</tr>
<tr>
<td>SCTableR0</td>
<td>2.0, 1.0, 0.0</td>
<td>BondingParam1</td>
<td>0.8</td>
</tr>
<tr>
<td>SCTableR1</td>
<td>1.0, 0.0, -1.0</td>
<td>BondingParam2</td>
<td>1.0</td>
</tr>
<tr>
<td>SCTableR2</td>
<td>0.0, -1.0, -2.0</td>
<td>BondingParam3</td>
<td>1.0</td>
</tr>
<tr>
<td>Aplus</td>
<td>1.0</td>
<td>InitSocCap</td>
<td>0.25</td>
</tr>
<tr>
<td>Azero</td>
<td>0.75</td>
<td>Theta</td>
<td>4.0</td>
</tr>
<tr>
<td>Aminus</td>
<td>1.0</td>
<td>GammaLeader</td>
<td>1.0</td>
</tr>
<tr>
<td>Rhoplus</td>
<td>1.0</td>
<td>GammaSocialite</td>
<td>6.0</td>
</tr>
<tr>
<td>Rhozero</td>
<td>1.0</td>
<td>GammaFollower</td>
<td>120.0</td>
</tr>
<tr>
<td>Rhominus</td>
<td>1.0</td>
<td>QualRange</td>
<td>0.9</td>
</tr>
<tr>
<td>Threshold</td>
<td>0.5</td>
<td>QualSensitivity</td>
<td>0.75</td>
</tr>
<tr>
<td>ExcessType</td>
<td>1</td>
<td>PenaltyFun</td>
<td>0</td>
</tr>
<tr>
<td>ChangeBegin</td>
<td>500</td>
<td>MostAmbPercent</td>
<td>0.1</td>
</tr>
<tr>
<td>ChangeAmount</td>
<td>0.0</td>
<td>MedAmbPercent</td>
<td>0.25</td>
</tr>
<tr>
<td>MaxFriendship</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX C
Example of Learning Reward/Penalty Parameters

We use gradient based approach to learn reward/penalty parameters ($\omega$). In this section, we show the details of the learning process (see Section 4.3.3) based on the default setting in the model (see Appendixes A and B). We would like to maximize Eq. (4.82),

$$\ell(\omega) = \sum_i \sum_t \log \text{Prob}(\text{Act}_{\text{real}}^i(t)).$$

where $\text{Act}_{\text{real}} \in \{\text{Join}, \text{Leave}, \text{Do Nothing}\}$. Using gradient based approach, we need to solve the derivative of Eq. (C.1) with respect to $\omega$, Eq. (4.84),

$$\frac{\partial \ell(\omega)}{\partial \omega} = \sum_i \sum_t \frac{\partial \log \text{Prob}(\text{Act}_{\text{real}}^i(t))}{\partial e^i_t} \times \frac{\partial e^i_t}{\partial \omega}.$$  

(C.2)

C.1 Solving the First Term

There are three cases about the first term in Eq. (C.2),

$$\frac{\partial \log \text{Prob}(\text{Act}_{\text{real}}^i(t))}{\partial e^i_t}.$$

- When $\text{Act}_{\text{real}}^i(t) = \text{Join}$, based on Eq. (4.50) and (4.85), we get what is shown here:

$$\frac{\partial \log \text{Prob}(\text{Act}_{\text{real}}^i(t) = J_l)}{\partial e^i_t} = \frac{\partial \log P_{+1}(e^i_t)}{\partial e^i_t} + \frac{\partial \log \left(P_r \times PR_{\text{Join}}^i_{(i,l)} + P_{\bar{r}} \times \bar{PR}_{\text{Join}}^i_{(i,l)} \right)}{\partial e^i_t}.$$  

(C.3)

From Sections 3.2, A.5, and A.6, we know $P_r$, $P_{\bar{r}}$, $PR_{\text{Join}}^i_{(i,l)}$, and $\bar{PR}_{\text{Join}}^i_{(i,l)}$ are not functions of $e^i_t$, then the second term in Eq. (C.3) is 0. Hence, we can write:

$$\frac{\partial \log \text{Prob}(\text{Act}_{\text{real}}^i(t) = J_l)}{\partial e^i_t} = \frac{\partial \log P_{+1}(e^i_t)}{\partial e^i_t}.$$  

(C.4)
When $Act^i_{\text{real}}(t) = \text{Leave}$, based on Eq. (4.51) and Eq. (4.86), we can write the following equation:

$$\frac{\partial \log \text{Prob}(Act^i_{\text{real}}(t) = L_i)}{\partial e^t_i} = \frac{\partial \log P_{-1}(e^t_i)}{\partial e^t_i} + \frac{\partial \log P_{\text{Leave}^i_{(i,l)}}}{\partial e^t_i}. \quad (C.5)$$

From Sections 3.2, A.5, and Eq. (A.30), we know $P_{\text{Leave}^i_{(i,l)}}$ is not a function of $e^t_i$, then the second term in Eq. (C.5) is 0. Then, we get the following equation:

$$\frac{\partial \log \text{Prob}(Act^i_{\text{real}}(t) = L_i)}{\partial e^t_i} = \frac{\partial \log P_{-1}(e^t_i)}{\partial e^t_i} \quad (C.6)$$

When $Act^i_{\text{real}}(t) = \text{Do Nothing}$, based on Eq. (4.87), we can write:

$$\frac{\partial \log \text{Prob}(Act^i_{\text{real}}(t) = S)}{\partial e^t_i} = \frac{\partial \log (\text{Prob}SJ^i_t + \text{Prob}SL^i_t + \text{Prob}SS^i_t)}{\partial e^t_i} \quad (C.7)$$

$$= \frac{1}{\text{Prob}SJ^i_t + \text{Prob}SL^i_t + \text{Prob}SS^i_t} \frac{\partial (\text{Prob}SJ^i_t + \text{Prob}SL^i_t + \text{Prob}SS^i_t)}{\partial e^t_i} \quad (C.8)$$

$$= \frac{1}{\text{Prob}SJ^i_t + \text{Prob}SL^i_t + \text{Prob}SS^i_t} \left( \frac{\partial \text{Prob}SJ^i_t}{\partial e^t_i} + \frac{\partial \text{Prob}SL^i_t}{\partial e^t_i} + \frac{\partial \text{Prob}SS^i_t}{\partial e^t_i} \right) \quad (C.9)$$

From Eq. (4.53), (4.54), and (4.55), we have the following equations:

$$\frac{\partial \text{Prob}SJ^i_t}{\partial e^t_i} = \frac{\partial P_{+1}(e^t_i)}{\partial e^t_i} \times \left( P_{r} \sum_{G^t_k \in D^e_c} P_{\text{Join}^i_{(i,k)}} \times P_{\text{Reject}^i_{(i,k)}} + P_{\bar{r}} \sum_{G^t_k \in D^e_c} \bar{P}_{\text{Join}^i_{(i,k)}} \times \bar{P}_{\text{Reject}^i_{(i,k)}} \right) \quad (C.10)$$

$$\frac{\partial \text{Prob}SL^i_t}{\partial e^t_i} = \begin{cases} \frac{\partial P_{-1}(e^t_i)}{\partial e^t_i} & \text{if actor } i \text{ is not in any group} \\ 0 & \text{otherwise} \end{cases} \quad (C.11)$$

and

$$\frac{\partial \text{Prob}SS^i_t}{\partial e^t_i} = \frac{\partial P_{0}(e^t_i)}{\partial e^t_i}. \quad (C.12)$$
Next, we need to solve the derivatives of $P_{+1}(e^t_i), P_0(e^t_i),$ and $P_{-1}(e^t_i)$ with respect to $e^t_i$. From Section A.4, we know that:

$$P_{Actr}(e^t_i) = \frac{P'_{Actr}}{P'_{+1} + P'_{0} + P'_{-1}},$$ \hspace{1cm} (C.13)

where $Actr \in \{+1, 0, -1\}$, and $P'_{Actr} \in \{P'_{+1} + P'_{0} + P'_{-1}\}$. We take logarithm on both sides of (C.13), and then we have:

$$\log P_{Actr}(e^t_i) = \log P'_{Actr} - \log(P'_{+1} + P'_{0} + P'_{-1})$$ \hspace{1cm} (C.14)

$$= \log P'_{Actr} - \log \sum_{k=-1}^{+1} P'_k.$$ \hspace{1cm} (C.15)

We know that $P'_{+1}, P'_{0},$ and $P'_{-1}$ are functions of $Excess Resources (e^t_i).$ and $e^t_i$ is a function of $\omega_\alpha$. Then, from (C.15), we know that:

$$\frac{\partial \log P_{Actr}(e^t_i)}{\partial e^t_i} = \frac{\partial}{\partial e^t_i} \left( \log P'_{Actr} - \log \sum_{k=-1}^{+1} P'_k \right)$$ \hspace{1cm} (C.16)

$$= \frac{1}{P'_{Actr}} \frac{\partial P'_{Actr}}{\partial e^t_i} - \frac{1}{\sum_{k=-1}^{+1} P'_k} \sum_{k=-1}^{+1} \frac{\partial P'_k}{\partial e^t_i}.$$ \hspace{1cm} (C.17)

From Eq. (A.17), we can write:

$$\frac{\partial P'_{+1}}{\partial e^t_i} = \frac{\partial}{\partial e^t_i} \left( \frac{Aplus}{1 + e^{-Rholus \times \left( \frac{e^t_i}{Threshold} - 1 \right)}} \right)$$ \hspace{1cm} (C.18)

$$= \frac{-Aplus}{\left(1 + e^{-Rholus \times \left( \frac{e^t_i}{Threshold} - 1 \right)}\right)^2} \times \frac{\partial}{\partial e^t_i} \left( 1 + e^{-Rholus \times \left( \frac{e^t_i}{Threshold} - 1 \right)} \right) \hspace{1cm} (C.19)$$

$$= \frac{Rholus \times Aplus \times e^{-Rholus \times \left( \frac{e^t_i}{Threshold} - 1 \right)}}{Threshold \times \left(1 + e^{-Rholus \times \left( \frac{e^t_i}{Threshold} - 1 \right)}\right)^2}.$$ \hspace{1cm} (C.20)
from Eq. (A.18), we can write:

\[
\frac{\partial P'_0}{\partial e'_i} = \frac{\partial}{\partial e'_i} \left( Azero \times e^{-Rhozero \times \frac{|e'_i|}{Threshold}} \right) \tag{C.21}
\]

\[
= Azero \times e^{-Rhozero \times \frac{|e'_i|}{Threshold}} \times \frac{\partial}{\partial e'_i} \left( -Rhozero \times \frac{|e'_i|}{Threshold} \right) \tag{C.22}
\]

\[
= \frac{-Rhozero \times Azero}{Threshold} \times e^{-Rhozero \times \frac{|e'_i|}{Threshold}} \times \frac{\partial |e'_i|}{\partial e'_i} \tag{C.23}
\]

and from Eq. (A.19), we can write:

\[
\frac{\partial P'_{-1}}{\partial e'_i} = \frac{\partial}{\partial e'_i} \left( Aminus \times \frac{1}{1 + e^{-Rhominus \times \left( \frac{e'_i}{Threshold} + 1 \right)}} \right) \tag{C.24}
\]

\[
= \frac{-Aminus}{\left( 1 + e^{-Rhominus \times \left( \frac{e'_i}{Threshold} + 1 \right)} \right)^2} \times \frac{\partial}{\partial e'_i} \left( 1 + e^{Rhominus \times \left( \frac{e'_i}{Threshold} + 1 \right)} \right) \tag{C.25}
\]

\[
= \frac{-Rhominus \times Aminus \times e^{Rhominus \times \left( \frac{e'_i}{Threshold} + 1 \right)}}{Threshold \times \left( 1 + e^{Rhominus \times \left( \frac{e'_i}{Threshold} + 1 \right)} \right)^2} \tag{C.26}
\]

After computing Eq. (C.20), (C.23), and (C.26), we have the value of Eq. (C.17). Then we can use Eq. (C.17) to compute Eq. (C.4), (C.6), and (C.9) and to solve the first term in Eq. (C.2).

### C.2 Solving the Second Term

For solving the second term in Eq. (C.2), \( \frac{\partial e'_i}{\partial \omega_\alpha} \), from Eq. (4.89), we know that:

\[
\frac{\partial e'_i}{\partial \omega_\alpha} = \left( 1 - \frac{\partial \psi_i L_{bi}}{\partial R^i} - \frac{\partial \psi_i L_{bo}}{\partial R^i} \right) \times \frac{\partial R^i}{\partial \omega_\alpha}. \tag{C.27}
\]
From Eq. (A.9) and (A.11), we can write:

\[ \frac{\partial \phi_i L_i^{br}}{\partial R_i^{t}} = \frac{\partial}{\partial R_i^{t}} \left( L_i^{br} \times \frac{BridgingParam1}{(BridgingParam2 + R_i^{t})^{BridgingParam3}} \right). \]  

(C.28)

We use \( B_{r1}, B_{r2}, \) and \( B_{r3} \) to indicate BridgingParam1, BridgingParam2, and BridgingParam3, respectively, then

\[ \frac{\partial \phi_i L_i^{br}}{\partial R_i^{t}} = \frac{\partial}{\partial R_i^{t}} \left( L_i^{br} \times \frac{B_{r1}}{(B_{r2} + R_i^{t})^{B_{r3}}} \right) \]

(C.29)

\[ = L_i^{br} \times B_{r1} \times \frac{\partial}{\partial R_i^{t}} (B_{r2} + R_i^{t})^{-B_{r3}} \]

(C.30)

\[ = -L_i^{br} \times B_{r1} \times B_{r3} \times (B_{r2} + R_i^{t})^{-B_{r3}-1} \]

(C.31)

From Eq. (A.10) and (A.12), we can write:

\[ \frac{\partial \psi_i L_i^{bo}}{\partial R_i^{t}} = \frac{\partial}{\partial R_i^{t}} \left( L_i^{bo} \times \frac{BondingParam1}{(BondingParam2 + R_i^{t})^{BondingParam3}} \right). \]  

(C.32)

We use \( B_{o1}, B_{o2}, \) and \( B_{o3} \) to indicate BondingParam1, BondingParam2, and BondingParam3, respectively, and the same argument as Eq. (C.28), then

\[ \frac{\partial \psi_i L_i^{bo}}{\partial R_i^{t}} = \frac{\partial}{\partial R_i^{t}} \left( L_i^{bo} \times \frac{B_{o1}}{(B_{o2} + R_i^{t})^{B_{o3}}} \right) \]

(C.33)

\[ = -L_i^{bo} \times B_{o1} \times B_{o3} \times (B_{o2} + R_i^{t})^{-B_{o3}-1} \]

(C.34)

From Eq. (A.7) and Section A.7, we know the default setting of \( R_i^{t} \) is what we see here:

\[ R_i^{t} = R_i^{t-1} + \Delta R_i^{t-1} - \eta, \]

(C.35)
Then doing the derivative of $R^t_i$ with respect to $\omega_\alpha$:

$$\frac{\partial R^t_i}{\partial \omega_\alpha} = \frac{\partial}{\partial \omega_\alpha} \left( R^t_i - 1 + \Delta R^t_i - \eta \right)$$  \hspace{1cm} (C.36)

$$= \frac{\partial}{\partial \omega_\alpha} \left( R^t_i - 1 + f_{\Delta R}(R^t_i - 1, \text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) - \eta \right)$$  \hspace{1cm} (C.37)

$$= \frac{\partial R^t_i - 1}{\partial \omega_\alpha} + \frac{\partial}{\partial \omega_\alpha} f_{\Delta R}(R^t_i - 1, \text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}).$$  \hspace{1cm} (C.38)

From Eq. (A.32), we know that:

$$f_{\Delta R}(R^t_i, \text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) = \text{Penalty}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) \times \text{Penalty}(R^t_i)$$

$$+ \text{Reward}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) \times \text{Reward}(R^t_i),$$

then doing the derivative with respect to $\omega_\alpha$:

$$\frac{\partial}{\partial \omega_\alpha} f_{\Delta R}(R^t_i, \text{Act}_{\text{norm}}, \text{Act}_{\text{choice}})$$

$$= \frac{\partial}{\partial \omega_\alpha} \left( \text{Penalty}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) \times \text{Penalty}(R^t_i) \right)$$

$$+ \frac{\partial}{\partial \omega_\alpha} \left( \text{Reward}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) \times \text{Reward}(R^t_i) \right)$$  \hspace{1cm} (C.39)

$$= \frac{\partial \text{Penalty}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}})}{\partial \omega_\alpha} \times \text{Penalty}(R^t_i)$$

$$+ \text{Penalty}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) \times \frac{\partial \text{Penalty}(R^t_i)}{\partial \omega_\alpha}$$

$$+ \frac{\partial \text{Reward}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}})}{\partial \omega_\alpha} \times \text{Reward}(R^t_i)$$

$$+ \text{Reward}(\text{Act}_{\text{norm}}, \text{Act}_{\text{choice}}) \times \frac{\partial \text{Reward}(R^t_i)}{\partial \omega_\alpha}. \hspace{1cm} (C.40)$$
Act\textsubscript{norm} and Act\textsubscript{choice} decide which \( \omega \) in the Agency and Structure table (Table 3.1) is used. Let \( \omega_\beta \) indicate the selected \( \omega \), and from Eq. (A.35), we know that:

\[
\frac{\partial \text{Penalty} W(\text{Act}\textsubscript{norm}, \text{Act}\textsubscript{choice})}{\partial \omega_\alpha} = \frac{\partial}{\partial \omega_\alpha} \frac{1}{1 + e^{5(1+\omega_\beta)}} \tag{C.41}
\]

\[
= \frac{-1}{(1 + e^{5(1+\omega_\beta)})^2} \times \frac{\partial}{\partial \omega_\alpha} (1 + e^{5(1+\omega_\beta)}) \tag{C.42}
\]

\[
= \frac{-e^{5(1+\omega_\beta)}}{(1 + e^{5(1+\omega_\beta)})^2} \times \frac{\partial}{\partial \omega_\alpha} (5 + 5\omega_\beta) \tag{C.43}
\]

\[
= \frac{-5e^{5(1+\omega_\beta)}}{(1 + e^{5(1+\omega_\beta)})^2} \times \frac{\partial \omega_\beta}{\partial \omega_\alpha}. \tag{C.44}
\]

There are two cases in Eq. (C.44):

\[
\frac{\partial \text{Penalty} W(\text{Act}\textsubscript{norm}, \text{Act}\textsubscript{choice})}{\partial \omega_\alpha} = \begin{cases} 
-5e^{5(1+\omega_\alpha)} & \text{if } \beta = \alpha \\
0 & \text{otherwise}
\end{cases} \tag{C.45}
\]

From Eq. (A.36), we can write:

\[
\frac{\partial \text{Reward} W(\text{Act}\textsubscript{norm}, \text{Act}\textsubscript{choice})}{\partial \omega_\alpha} = \frac{\partial}{\partial \omega_\alpha} \frac{1}{1 + e^{5(1-\omega_\beta)}} \tag{C.46}
\]

\[
= \frac{-1}{(1 + e^{5(1-\omega_\beta)})^2} \times \frac{\partial}{\partial \omega_\alpha} (1 + e^{5(1-\omega_\beta)}) \tag{C.47}
\]

\[
= \frac{-e^{5(1-\omega_\beta)}}{(1 + e^{5(1-\omega_\beta)})^2} \times \frac{\partial}{\partial \omega_\alpha} (5 - 5\omega_\beta) \tag{C.48}
\]

\[
= \frac{5e^{5(1-\omega_\beta)}}{(1 + e^{5(1-\omega_\beta)})^2} \times \frac{\partial \omega_\beta}{\partial \omega_\alpha}. \tag{C.49}
\]

There are also two cases in Eq. (C.49):

\[
\frac{\partial \text{Reward} W(\text{Act}\textsubscript{norm}, \text{Act}\textsubscript{choice})}{\partial \omega_\alpha} = \begin{cases} 
\frac{5e^{5(1-\omega_\alpha)}}{(1 + e^{5(1-\omega_\alpha)})^2} & \text{if } \beta = \alpha \\
0 & \text{otherwise}
\end{cases} \tag{C.50}
\]
From Eq. (A.33), we can write:

\[
\frac{\partial \text{Penalty}(R_t)}{\partial \omega_\alpha} = -0.35 \times (0.01R_t + (1.99R_t)^{1.05})
\]  
(C.51)

\[
= -0.35 \times (0.01R_t + (1.99R_t)^{1.05}) \times \frac{\partial R_t}{\partial \omega_\alpha} 
\]  
(C.52)

\[
= -0.0035 \frac{\partial R_t}{\partial \omega_\alpha} - 0.35 \times 1.99^{1.05} \times (R_t)^{0.05} \times \frac{\partial R_t}{\partial \omega_\alpha}
\]  
(C.53)

\[
= - \left( 0.0035 + 0.3675 \times 1.99^{1.05} \times (R_t)^{0.05} \right) \frac{\partial R_t}{\partial \omega_\alpha}
\]  
(C.54)

From Eq. (A.34), we know that:

\[
\frac{\partial \text{Reward}(R_t)}{\partial \omega_\alpha} = -\frac{\partial R_t \text{Penalty}(1 - R_t^t)}{\partial \omega_\alpha}
\]  
(C.55)

\[
= - \frac{\partial \text{Penalty}(1 - R_t)}{\partial (1 - R_t)} \times \frac{\partial (1 - R_t)}{\partial \omega_\alpha}
\]  
(C.56)

\[
= - \left( 0.0035 + 0.3675 \times 1.99^{1.05} \times (1 - R_t)^{0.05} \right) \frac{\partial R_t}{\partial \omega_\alpha}
\]  
(C.57)

Let \( a = 0.0035 \) and \( b = 0.3675 \times 1.99^{1.05} \), then Eq. (C.54) and (C.57) become what is shown here:

\[
\frac{\partial \text{Penalty}(R_t)}{\partial \omega_\alpha} = - \left( a + b(R_t)^{0.05} \right) \frac{\partial R_t}{\partial \omega_\alpha}
\]  
(C.58)

\[
\frac{\partial \text{Reward}(R_t)}{\partial \omega_\alpha} = - \left( a + b(1 - R_t)^{0.05} \right) \frac{\partial R_t}{\partial \omega_\alpha}
\]  
(C.59)
From Eq. (C.38), (C.40), (C.45), (C.50), (C.58), and (C.59), we get the following equation:

\[
\frac{\partial R_t^i}{\partial \omega_\alpha} = \frac{\partial R_{t-1}^i}{\partial \omega_\alpha} + \frac{\partial \text{PenaltyW}(\text{Act}\_\text{norm}, \text{Act}\_\text{choice})}{\partial \omega_\alpha} \times \text{Penalty}(R_{t-1}^i) \\
+ \text{PenaltyW}(\text{Act}\_\text{norm}, \text{Act}\_\text{choice}) \times - \left( a + b(R_{t-1}^i)^{0.05} \right) \frac{\partial R_{t-1}^i}{\partial \omega_\alpha} \\
+ \frac{\partial \text{RewardW}(\text{Act}\_\text{norm}, \text{Act}\_\text{choice})}{\partial \omega_\alpha} \times \text{Reward}(R_{t-1}^i) \\
+ \text{RewardW}(\text{Act}\_\text{norm}, \text{Act}\_\text{choice}) \times - \left( a + b(1 - R_{t-1}^i)^{0.05} \right) \frac{\partial R_{t-1}^i}{\partial \omega_\alpha} \quad \text{(C.60)}
\]

We rearrange Eq. (C.60), and let

\[
\Phi_t = 1 - \left( a + b(R_{t-1}^i)^{0.05} \right) \times \text{PenaltyW}(\text{Act}\_\text{norm}, \text{Act}\_\text{choice}) \\
- \left( a + b(1 - R_{t-1}^i)^{0.05} \right) \times \text{RewardW}(\text{Act}\_\text{norm}, \text{Act}\_\text{choice}), \quad \text{(C.61)}
\]

\[
\Psi_t = \frac{\partial \text{PenaltyW}(\text{Act}\_\text{norm}, \text{Act}\_\text{choice})}{\partial \omega_\alpha} \times \text{Penalty}(R_{t-1}^i) \\
+ \frac{\partial \text{RewardW}(\text{Act}\_\text{norm}, \text{Act}\_\text{choice})}{\partial \omega_\alpha} \times \text{Reward}(R_{t-1}^i). \quad \text{(C.62)}
\]

Then, Eq. (C.60) becomes what we show here:

\[
\frac{\partial R_t^i}{\partial \omega_\alpha} = \Phi^{-1} \times \frac{\partial R_{t-1}^i}{\partial \omega_\alpha} + \Psi_t. \quad \text{(C.63)}
\]

We know when \( t = 0 \), \( R_t^i = R_0^i \) which is a constant, and then

\[
\frac{\partial R_0^i}{\partial \omega_\alpha} = 0. \quad \text{(C.64)}
\]
Therefore,
\[
\frac{\partial R_t^i}{\partial \omega_\alpha} = \Phi^{t-1} \times \frac{\partial R_{t-1}^i}{\partial \omega_\alpha} + \Psi^{t-1} \tag{C.65}
\]
\[
= \Phi^{t-1} \left( \Phi^{t-2} \times \frac{\partial R_{t-2}^i}{\partial \omega_\alpha} + \Psi^{t-2} \right) + \Psi^{t-1} \tag{C.66}
\]
\[
= \sum_{i=0}^{t-2} \left( \Psi^t \prod_{j=i+1}^{t-1} \Phi^j \right) + \Psi^{t-1}. \tag{C.67}
\]

We can use recursive and dynamic programming algorithm to solve (C.65). Then, from Eq. (C.31), (C.34), and (C.63), we are able to solve (C.27):
\[
\frac{\partial e_t^i}{\partial \omega_\alpha} = \left( 1 - \frac{\partial \phi_i L_{br}}{\partial R_t^i} - \frac{\partial \psi_i L_{bo}}{\partial R_t^i} \right) \times \frac{\partial R_t^i}{\partial \omega_\alpha}. \tag{C.68}
\]
\[
= \left( 1 + L_{br}^i \times B_{r1} \times B_{r3} \times (B_{r2} + R_t^i)^{-B_{r3}-1} + L_{bo}^i \times B_{o1} \times B_{o3} \times (B_{o2} + R_t^i)^{-B_{o3}-1} \right) \times \frac{\partial R_t^i}{\partial \omega_\alpha}. \tag{C.69}
\]