International Journal of Neural Systems
Vol. ?, No. ? (2008) 1–23

REVERSE ENGINEERING AN AGENT-BASED HIDDEN MARKOV MODEL FOR COMPLEX SOCIAL SYSTEMS

HUNG-CHING (JUSTIN) CHEN
Department of Computer Science, Rensselaer Polytechnic Institute
Troy, New York 12180, USA
chenh3@cs.rpi.edu

MARK GOLDBERG
Department of Computer Science, Rensselaer Polytechnic Institute
Troy, New York 12180, USA
goldberg@cs.rpi.edu

MALIK MAGDON-ISMAIL
Department of Computer Science, Rensselaer Polytechnic Institute
Troy, New York 12180, USA
magdon@cs.rpi.edu

WILLIAM A. WALLACE
Department of Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute
Troy, New York 12180, USA
wallaw@rpi.edu

We present a machine learning approach to discover the agent dynamics that drives the evolution of the social groups in a community. We set up the problem by introducing an agent-based hidden Markov model for the agent dynamics: the acts of an agent 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* which corresponds to identifying 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 multistage learning process for determining the group structure, the group evolution, and the *micro-laws* of a community based on the observed set of communications between actors, without knowing the semantic contents. Finally, We present the results of extensive experiments on synthetic data as well as some results on real communities, *e.g.*, Enron email and Movie newsgroups. Insight into agent dynamics helps us understand the driving forces behind social evolution.

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 *actors*[a] who share some common context[30]. The dynamics of the social groups are governed by the actor dynamics – actors join new

[a]An actor generically refers to an individual entity in the society.

groups, leave groups, and do nothing. An actor's actions are governed by *micro-laws*[17] which may be: personal attributes (*e.g.*, some people like to go out with a bunch of people, but some would prefer one to 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 from the expectation of the community, but some are not). Sometimes, the same actors, but 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.

What makes one community different from another? Consider the community of college students versus 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.

Each day, individuals from all over the world receive and respond to the information from other individuals of the society. In the past decades, high tech has been integrated aggressively into our daily life. The rapid exchanges of communication between individuals have gone from surfing online for information to providing information, building individual Space / Blog as well as getting connected through various instant messaging communities. It is apparent that online communities have become one of the influential medium to the journey of social evolution. Yet, regardless of the impact of the online communities; the role of social value continue to play an imperative factor on the dynamics of the online communities as it has been for the offline communities rapid growth, sudden emergence or hastily dissipated due to changes of demands, needs, and values of the existing society. Therefore, it is essential to acquire ranges of more comprehensive and objective social factors that might have propelled the evolution of the society.

Recently, the explosion of online communities provides an ideal pasture on which to groom and test social science theories, in particular the most natural question is: *what are the micro-laws which govern a particular society*? It is an interesting hunt as how the *micro-laws* govern the actors behaviors and make a society ticks. 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 needed to represent even a very simple society – actors may have discrete attributes together with continuous parameters, which inevitably leads to a mixed optimization problem, and each actor has his/her own attributes, which also interact with others' 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 membership 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.

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 between actors without knowing semantic contexts. 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. A complete description of the model together with its justification through social capital theories[23, 25] is beyond the scope of this presentation, the detail of which can be found in Ref. 6. We have developed a multistage learning process to learn the group structure, the group evolution, and the appropriate *micro-lows* 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 benefit of multistage learning process is as follows:

(1) Each module is a checkpoint. We can test the accuracy and check the performance of algorithm for each module, and it is also easier to improve the ac-

curacy and the performance of the learning process in each individual module.
(2) Each module have different outputs, and we can extra different kinds of information from each 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 identify the appropriate *micro-laws* by solving a mixed optimization problem. And to avoid the resulting combinatorial explosion, we appropriately approximate and optimize the objective within a coordinate-wise learning setting. 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 some results on real data (Enron email and Movie newsgroups).

Related Work. There is significant literature on modeling and analysis of social networks, Ref. 2, 10, 12, 13, 18, 20, 22, 23, 24, 26, 28, 30. Most of this work focuses on modeling of evolving social networks. In Ref. 26 the authors address model fitting in a very limited setting using very simple models. 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 addition, due to the growing popularity and interests of social network analysis (SNA), especially of the booming exposure of online communities, researchers have started to use different methods to help them collect and study the structure of the social network as well as analyze the ranges / factors of social dynamics, Ref. 1, 7, 8, 19. In Ref. 1, the authors use decision-tree approach to determine some properties of the social network. The decision-tree approach is a deterministic process. This is different from our approach in using the stochastic process to determine actors' behaviors. The reason is that in the social network, even under the same environment, actors do not necessary possess or reflect the same behaviors.

Paper Organization. Next, we give an overview of the agent-based hidden Markov model in Sec. 2. Then, we present our multistage learning process to learning the group structure and evolution from the observed communications and also learning the appropriate parameters of the model in Sec. 3. We discuss what we are going to predict in Sec. 4. Then, we give experimental results on synthetic data as well as real data in Sec. 5 and conclude in Sec. 6.

2. Overview of Agent-Based Hidden Markov Model

We give a brief overview of the agent-based hidden Markov Model, which includes the probabilistic evolving social group model ViSAGE[b], (Virtual Simulation and Analysis of Group Evolution), and the probabilistic communication model. The foundation of ViSAGE is a Markov process, and Fig. 1 shows the general framework for the step by step evolution in the model.

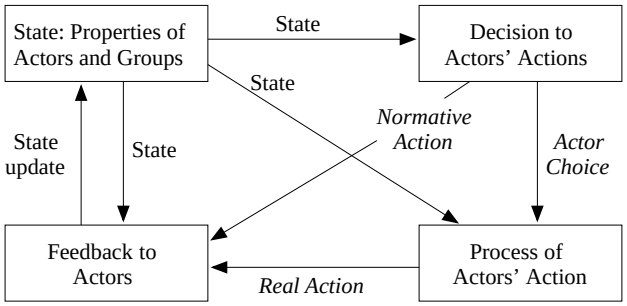


Fig. 1. Framework for the step by step evolution in the social group model.

There are actors, groups, the *state* of the society which 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 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 eventually performs the *Real Action*. Depending on the feedbacks from actors' *Normative Action* and *Real Action*, properties of actors and groups are updated accordingly. We also develop a probabilistic communication model to simulate communication networks based on the *state* of the society. In the

[b] ViSAGE is derived from Ref. 27, and the details of ViSAGE can be found in Ref. 6.
[c] A communication edge indicate a communication link between two actors.

communication model, we only consider about the communication edges[c] between actors without the semantics of the communications. Combining ViSAGE with the communication model, the whole process represents a hidden Markov model (HMM). In the real world, it is easy to observe the communication networks, but the *states* of the society are difficult to be determined and are hidden behind the communication networks.

2.1. *State of the society*

In ViSAGE, many parameters govern how actors decide whether to join or leave a group, or do nothing, and also which group the actor desires to join or 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

$$\mathcal{S}_t = \{\boldsymbol{type}_i, \boldsymbol{r}_i^t, R_i^t, \{G_l^t\}_{i \in G_l^t}\}_{i=1}^N, \tag{1}$$

where N indicates the number of actors, $\boldsymbol{type}_i$ is a vector and shows actor i's types, $\boldsymbol{r}_i^t$ is a vector and indicates actor i's ranks in each group at time t, R_i^t is actor i's resources at time t, G_l^t is a social group l at time t, and $\{G_l^t\}_{i \in G_l^t}$ is a set of groups actor i joined at time t. In the following sections, we briey describe the properties of actors and groups.

2.1.1. *Type*

Each actor has a private set of attributes, which we refer to as its *type*. In the model, there are two categories of actor's type: simply one, *size-type* or *stype*, controls the actor's group size preferences, and the other, *ambition-type* or *atype*, controls actor's "ambition" (how quickly her rank increases in a group). There are 3 subcategories for 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 3 subcategories for actor's *ambition-type*: $Type_A$ is most ambitious, $Type_C$ is least ambitious, and $Type_B$ is in the middle.

2.1.2. *Rank*

Each actor has a rank in each group to present the actor's position in the group. As actors spend more time in a group, their position in the group changes. There is a tendency for more senior members of a group to have a higher position within the group than junior members. Also, $Type_A$ have a highest tendency to increase their position within the group, and $Type_C$ lowest. The notation $r_{(i,l)}^t$ is for actor i's rank in group G_l at time t, and $\boldsymbol{r}_i^t$ is the set of ranks of actor i in all groups at time t. The definition of rank is as follow:

$$r_{(i,l)}^t = \frac{time_{(i,l)}^t \delta_i}{\sum_{j \in G_l^t} time_{(j,l)}^t \delta_j}, \tag{2}$$

where we use notation G_l^t for a social group l at time t, $time_{(i,l)}^t$ for the amount of time that actor i has spent in group G_l at time t, and δ_i indicates how quickly actor i's rank increases in a group for each time step. δ_i is a function of actor i's *ambition-type* – $Type_A$ has the biggest value of δ_i, $Type_C$ has the smallest value, and $Type_B$ has the value in the middle. In addition, when an actor leaves a group, her rank in that group will gradually decrease.

2.1.3. *Qualification*

Each actor has a qualification (q_i^t) to represent an actor's prestige. It is determined as the average rank of the actor among the groups in which the actor 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 she more likely join or leave. The definition of qualification is as follow:

$$q_i^t = \frac{\sum_{i \in G_l^t} r_{(i,l)}^t |G_l^t|}{\sum_{i \in G_l^t} |G_l^t|}, \tag{3}$$

where we use $|G_l^t|$ to denote the size of group l at time t, and it means how many actors are in that group.

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 the formula

$$Q_l^t = \sum_{i \in G_l^t} q_i^t r_{(i,l)}^t. \tag{4}$$

2.1.4. *Resources*

Each actor has certain amount of available resources (R_i^t) at each time step. The available resources are used to maintain a membership in a group. We also define the excess resources (e_i^t) as

$$e_i^t = R_i^t - \phi_i L_{br}^i - \psi_i L_{bo}^i, \tag{5}$$

where $\phi_i L_{br}^i$ and $\psi_i L_{bo}^i$ indicate *Bridging Cost*[5] and *Bonding Cost*[5], respectively – how many resources an

actor needs to maintain a membership in a group. *Bridging Cost* is related to the number of groups the actor is in –

$$L^i_{br} = |G : i \in G|, \tag{6}$$

and *Bonding Cost* is related to the actors' ranks –

$$L^i_{bo} = \sum_{G_l : i \in G_l} r^t_{(i,l)}. \tag{7}$$

ϕ_i and ψ_i are weight parameters and are functions of R^t_i. The excess resources influence what kind of action an actor will perform at next time step, described in §2.2.1.

2.2. *State transitions*

At each time step, every actor needs to decide on leaving one group, joining one group, or remaining in the same groups. And then based on actors' actions, the *state* of the society transfer to next state.

2.2.1. *Actor's Normative actions*

In the first step, each actor needs to decide her *Normative* Action, the action she most likely wants to act. The decision depends on an actor's excess resources (e^t_i). If e^t_i is positive, the actor will tend to use the excess resources in joining another group, if e^t_i is negative, the actor will tend to leave a group in order to lessen the cost needed, and if e^t_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, leave, or do nothing, and we use the notations $P_{+1}(e^t_i)$, $P_0(e^t_i)$, and $P_{-1}(e^t_i)$ to denote the probabilities, respectively; where

$$\sum_{k=-1}^{+1} P_k(e^t_i) = 1. \tag{8}$$

The definition of the $Normative$ action is

$$Act_{normative} = \underset{action}{\text{argmax}} \left(P_{action}(e^t_i)\right), \tag{9}$$

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

2.2.2. *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 she is going to perform based on a stochastic process as follow:

$$Act_{choice} = random\left(P_{+1}, P_0, P_{-1}\right), \tag{10}$$

where $random\left(P_{+1}, P_0, P_{-1}\right)$ 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 she would like to perform, she needs to decide which group to "apply" (want to join) or leave. An actor selects one group she would like to leave from the groups she has joined. And for joining a group, an actor learns from her *neighbors* to know which groups out there she is able to join; called join by *reference*.

Definition 1 (Join by *reference*). *Let a_i and a_j denote actor i and actor j, and the set of groups actor i knows and would like to join at time t is $\{G^t_k\}$, then*

$$\left\{G^t_k | \forall j, a_i, a_j \in G^t_l, a_j \in G^t_k, a_i \notin G^t_k\right\}. \tag{11}$$

Then, we need to determine the probability of each candidate group to join or leave. Each actor has a size preference; therefore, the actor takes into account the size and the qualification of the group during decision making. The size preference is defined by a function of actor's *size-type* and the group's size as follow:

$$SizeAff^t_{(i,l)} = Function\left(\textit{size-type}_i, |G^t_l|\right), \tag{12}$$

which indicates the size preference for actor i to group l at time t. Then the probability of which group the actor decides to join or leave depends on the qualification of the group and the actor's ranks, size preference and qualification. The probabilities actor i would like to select group l to join or leave at time t are defined as follows:

$$PJoin^t_{(i,l)} = Function\left(SizeAff^t_{(i,l)}, q^t_i, Q^t_l\right), \tag{13}$$

$$PLeave^t_{(i,l)} = Function\left(\boldsymbol{r}^t_i, SizeAff^t_{(i,l)}\right), \tag{14}$$

where $\boldsymbol{r}^t_i$ denotes the set of ranks of actor i in all her groups at time t. Then, if actor i chooses group l to join, the group can accept or reject the actor's application based a stochastic process, which is related to the group's qualification and the actor's qualification. The probability of group l rejecting actor i at time t is defined as follow:

$$PReject^t_{(i,l)} = Function\left(q^t_i, Q^t_l\right). \tag{15}$$

Then, after actor i would like to join group l, the probability actor i really joins group l is

$$PRJoin^t_{(i,l)} = PJoin^t_{(i,l)} \times \left(1 - PReject^t_{(i,l)}\right). \tag{16}$$

After all the above stochastic processes, the action the actor performs is called *Real action*.

Lemma 2 (The probabilities of *Real action*). *Let $\tilde{P}_{+1}(e_i^t)$, $\tilde{P}_{-1}(e_i^t)$, and $\tilde{P}_0(e_i^t)$ indicate the probabilities of actor i's* Real action *at time t to join a group, leave a group, or do nothing, respectively. Then,*

$$\tilde{P}_{+1}(e_i^t) = P_{+1}(e_i^t) \times \left(1 - \sum_l PJoin^t_{(i,l)} \times PReject^t_{(i,l)}\right) \quad (17)$$

$$\tilde{P}_{-1}(e_i^t) = \begin{cases} 0 & \text{, if actor } i \text{ is in no group.} \\ P_{-1}(e_i^t) & \text{, otherwise.} \end{cases} \quad (18)$$

$$\tilde{P}_0(e_i^t) = 1 - \tilde{P}_{+1}(e_i^t) - \tilde{P}_{-1}(e_i^t). \quad (19)$$

2.2.3. *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 is based on Eq. (2), (3), and (4). The change of available resources is related to an actor's $Normative$ action, real action, and the society reward/penalty parameters θ_{reward}. The reward/penalty parameters θ_{reward} are used in the Agency and Structure table (see Table 1 where $\theta_{reward} = \{\omega_\alpha | \alpha = 1, 2, \cdots, 9\}$) to determine how to update an actor's resources, and it is summarized heuristically by

$$\Delta R_i^t = f_{\Delta R}\left(action, R_i^t, \theta_{action}, \theta_{reward}\right), \quad (20)$$

where θ_{action} indicates some parameters related to actors' actions, and $f_{\Delta R}$ indicates the formulas of updating resources as follow:

$$\begin{aligned} f_{\Delta R} = {} & PenaltyW\left(action, \theta_{action}, \theta_{reward}\right) \times \\ & Penalty(R_i^t) + \\ & RewardW\left(action, \theta_{action}, \theta_{reward}\right) \times \\ & Reward(R_i^t). \end{aligned} \quad (21)$$

Table 1. The Agency and Structure Table

		"Structure" $Act_{normative}$		
		`join`	`stay`	`leave`
"Agency" Act_{real}	`join`	ω_1	ω_4	ω_7
	`stay`	ω_2	ω_5	ω_8
	`leave`	ω_3	ω_6	ω_9

The model uses which ω_α in $f_{\Delta R}$, Eq. (21) according to an actor's $Normative$ action and real action. Then the available resources at next time step is

$$R_i^{t+1} = R_i^t + \Delta R_i^t. \quad (22)$$

2.3. *Communications*

We also developed a probabilistic communication model to simulate social networks and to produce the communication edges between 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 joined groups two actors have in common, the higher probability these two actors should communicate with each other; however, if two actors have no any joined group in common, they still have a chance to communicate with each other. A more general model also consider actors' friends; if two actors are not in a 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's friends in common (3rd level friendship), or friend's friends' friends in common (4th level friendship), etc.

Definition 3 (*n*th level friendship). *Consider that the group structure is a unweighed graph, $\mathcal{G}_g = (\mathcal{A}, E_g)$, 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 as $\varepsilon_g(i,j)$, indicates that a_i and a_j are in the same group. Two actors a_i and a_j have a nth level friendship if and only if there exists a path between a_i and a_j, and the number of edges (length) of that path is n.*

From *Definition 3*, 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.

Lemma 4 (The probability of communication). *Let P_l denotes the probability of communications in a lth level friendship; P_0 refers to the probability of communications between two actors who have no any friendship. $F_l(i,j)$ shows the number of lth level friendship between a_i and a_j, $P_e(i,j)$ indicates the probability of communication between a_i and a_j, and $\bar{P}_e(i,j)$ is the probability of no communication between a_i and a_j;*

$$P_e(i,j) + \bar{P}_e(i,j) = 1. \quad (23)$$

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

$$P_e(i,j) = \begin{cases} P_0 & \text{, if } a_i, a_j \text{ have no friendship.} \\ 1 - \prod_{l=1}^{L}(1 - P_l)^{F_l(i,j)} & \text{, otherwise.} \end{cases} \quad (24)$$

Let $|\mathcal{A}|$ be the number of actors, and $|\mathcal{A}| = N$. The algorithm to compute the number of each level

$[1st, \cdots, Lth]$ of friendship and the probabilities of communication between any pair of actors ($P_e(i,j)$) is shown in *Algorithm 1*.

Algorithm 1: Compute the number of friendship and the probabilities of communications

Input : $\mathcal{G}_g = (\mathcal{A}, E_g)$, where $|\mathcal{A}| = N$
Output: $F_l(i,j)$ and $P_e(i,j)$ for all l, i, and j

```
1  function Friendship(l,i,k,Ẽ_g){
2  if l > L then return
3  for j=1 to N do
4  |  if j ≠ i, j ≠ k and ε_g(j,k) ∈ Ẽ_g then
5  |  |  F_l(i,j) ⟵ F_l(i,j) + 1
6  |  |  P̄_e(i,j) ⟵ P̄_e(i,j) × (1 − P_l)
7  |  |  Friendship(l+1,i,j,Ẽ_g − ε_g(j,k))
8  |  endif
9  endfor
10 }endfunc
11 forall i and j do
12 |  P̄_e(i,j) ⟵ 1
13 |  forall l do
14 |  |  F_l(i,j) ⟵ 0
15 |  endfall
16 endfall
17 for i=1 to N do
18 |  Friendship(1,i,i,E_g)
19 endfor
20 forall i and j do
21 |  if P̄_e(i,j) = 1 then P_e(i,j) = P_0
22 |  else P_e(i,j) ⟵ 1 − P̄_e(i,j)
23 endfall
24 return F_l(i,j) and P_e(i,j)
```

The algorithm uses depth-first search (DFS)[14] to find the friendship between actors and the probabilities to communicate. Line 1 to 10 present a function to find the number of lth level friendship and compute the $\bar{P}_e(i,j)$. 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. Line 11 to 16 do the initialization, and line 17 to 19 find the friendship related to actor i. Line 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 $Friendship()$, line 1 to 10, called itself L times, and the time complexity for each time is $O(N)$. So, the time complexity for the function $Friendship()$ is $O(N^L)$. In the main loop, line 17 to 19, calls the function $Friendship()$ N times, so the time complexity of *Algorithm 1* is $O(N^{L+1})$.

Based on those probabilities of communication ($P_e(i,j)$), we create communication edges between actors. The *Algorithm 2* shows the communication model creating communication edges, where $\mathcal{G}_c$ 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 2: The communication model

Input : actors $\mathcal{A}$ and $P_e(i,j)$, where $|\mathcal{A}| = N$
Output: $\mathcal{G}_c = (\mathcal{A}, E_c)$

```
1 E_c ⟵ ∅
2 for i=1 to N do
3 |  for j=i+1 to N do
4 |  |  if Based on the probability P_e(i,j) and
  |  |  decide there is an edge between a_i and a_j
  |  |  then
5 |  |  |  E_c ⟵ E_c ∪ ε_c(i,j)
6 |  |  endif
7 |  endfor
8 endfor
9 return G_c = (A, E_c)
```

The time complexity of *Algorithm 2* is $O(N^2)$ because there are two for loop between line 2 and 8. Combine *Algorithm 1* and *Algorithm 2*, the time complexity of the whole communication model is $O(N^{L+1} + N^2)$.

2.4. *Example of ViSAGE*

Figure 2 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 (Fig. 2), 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_l^t for social group l at time t. Focus on actor a_1 at time t. Some of the properties of actor a_1 have been indicated: $\boldsymbol{type}_1$, $\boldsymbol{r}_1^t$, and R_1^t. As indicated, $r_{(1,1)}^t$ depends on *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,

$$\sum_{i \in G_l^t} r_{(i,l)}^t = 1. \tag{25}$$

Thus $r_{(1,1)}^t$ indirectly depends on *ambition-type*$_2$ and *ambition-type*$_4$. Based on her properties, a_1 decides to join a new group through the stochastic process denoted

by $Action()$ in Fig. 2, which depends on a set of parameters θ_{action}; two other possible actions are to leave a group or to do nothing.

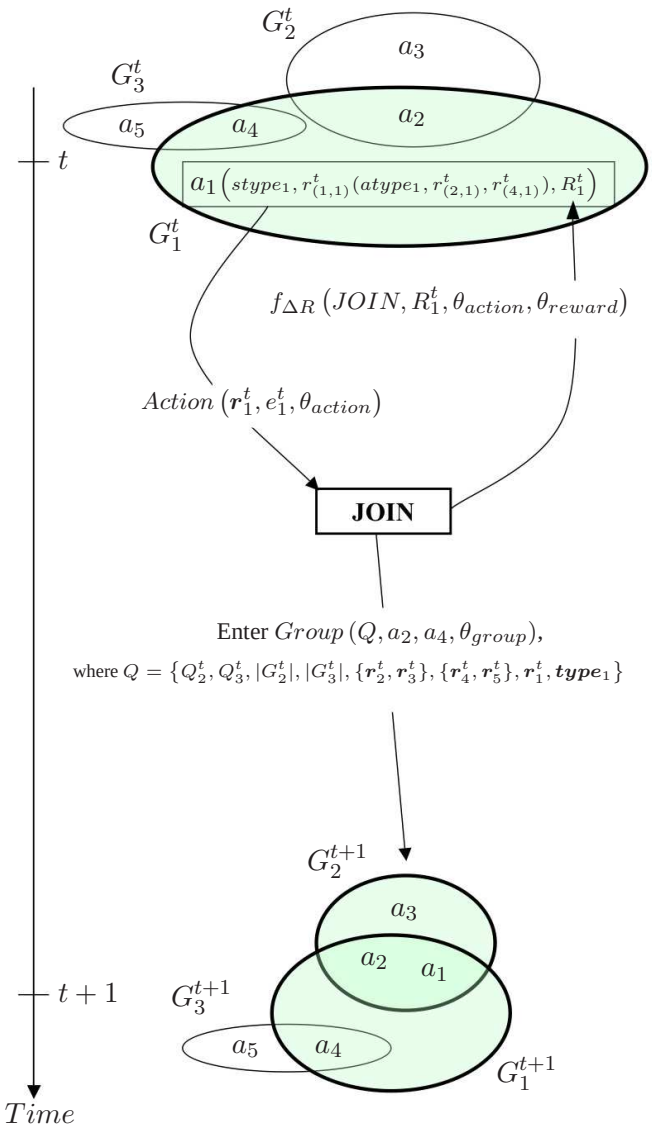


Fig. 2. An example of the probabilistic social group evolution model

Having decided to join a group, a_1 must now decide which specific group to join. This is accomplished by a stochastic hand-shaking process in which a_1 decides which group to "apply" to, 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 Fig. 2 and is governed by her own set of parameters θ_{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 her *neighbors* a_2, a_4 – i.e., the groups they belong to, and thus only applies to other groups by *reference*; the potential joining groups are G_2 and G_3 in this case. Actor a_1 then decides which group to apply to based on her *qualification*, as measured by her average rank in all 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 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 Fig. 2, which additionally depends on the actual action, available resources, and some parameters θ_{reward}. This process is analogous to society encouragement or penalty for doing expected versus unexpected things. 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 Fig. 1 to obtain the state at time $t+1$.

3. Learning Process

The common learning algorithms for solving the problems in a hidden Markov model are like *forward-backard algorithm*[4], *Viterbi algorithm*[29], and *Baum-Welch algorithm*[3]. 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.

Theorem 5 (The number of *state* in ViSAGE). *In ViSAGE, if there are N actors and maximum K groups in a society, then, in each time step, the number of possible state is $\Omega\left(2^{NK}\right)$.*

Proof. From Eq. (1), we know each *state* includes all actors' types, ranks, available resources, and which groups they are in. Consider the term $\left\{\{G_l^t\}_{i\in G_l^t}\right\}_{i=1}^{N}$ in Eq. (1). We know there are maximum K groups, and each actor has two possible condition in each group, in or not in the group. Then, there are 2^K possible group structure for each actor. There are total N actors, so the number of possible group structure is $\left(2^K\right)^N = \left(2^{NK}\right)$. Without consider other factors in the *state*, the number of the possible *state* is already $\left(2^{NK}\right)$. So, we can claim that the number of possible *state* is $\Omega\left(2^{NK}\right)$ in ViSAGE. ■

If we have a data set for T time steps, the complexities of using the above algorithms are $\Omega(T \times 2^{2NK})$, which is exponential computation time and is very time consuming. Therefore, we have developed a multistage learning process to learn the group structures, 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 structures at each time step based on the communication networks. Then, in the second stage, we discover the group evolution using the data of group structures. In the last stage, we learn from the group evolution to identify the appropriate parameters (*micro-laws*) in ViSAGE.

3.1. *Learning from communications*

The challenge with real data is that the groups structure and their evolution are not known, especially in online communities. Instead, one observes the communication dynamics. Fig. 3 illustrates an example of data set with 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. We thus opt for a simpler approach. The first stage in learning is to use the communication dynamics to construct the set of groups. Fig. 4 shows an example of data set with group structures.

In our communication model (see §2.3), let i, j refer to actors, and we know $P_e(i,j)$ is the probability that two actors a_i and a_j would like to communicate. Let x_{ij} be a boolean value presenting the communication between a_i and a_j. Then the problem can be define as maximizing

$$Prob = \prod_{i,j} P_e(i,j)^{x_{ij}} (1 - P_e(i,j))^{(1-x_{ij})}, \quad (26)$$

where $P_e(i,j)$, Eq. (24), is

$$P_e(i,j) = \begin{cases} P_0 & \text{, if } a_i, a_j \text{ have no friendship.} \\ 1 - \prod_{l=1}^{L} (1 - P_l)^{F_l(i,j)} & \text{, otherwise.} \end{cases} \quad (27)$$

To solve this problem, we need to find appropriate $P_e(i,j)$ to maximize Eq. (26) where x_{ij} for all i and j are known from communication data. Solving $P_e(i,j)$ 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 between the actors are aggregated over some time period τ to obtain a weighted communication graph $\mathcal{G}_c(\tau)$. The actors $\mathcal{A}$ is a set of nodes in $\mathcal{G}_c(\tau)$ and the edge weight w_{ij} between two actors is the communication intensity (number of communications) between a_i and a_j. In unweighed graph, all w_{ij}'s are the same. The sub-task we would like to solve is to infer the group structure from the communication graph $\mathcal{G}_c(\tau)$. Any reasonable formulation of this problem is NP-hard, and it only solve the value of $F_l(i,j)$. So, we need some efficient heuristic for finding the appropriate solution for Eq. (26).

In our approach, we have developed 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. We use the algorithms developed by Baumes et al., Ref. 7, which efficiently find overlapping communities in a communication graph. We consider time periods $\tau_1, \tau_2, \ldots, \tau_{T+1}$ and the corresponding communication graphs $\mathcal{G}_c(\tau_1), \mathcal{G}_c(\tau_2), \ldots, \mathcal{G}_c(\tau_{T+1})$. The time periods need not be disjoint, and in fact that we have used the overlapping time periods to cluster the communication data into group, since there is considerable noise in the communications – aggregation, together with ovelap smoothens the time series of communication graphs. This is a way of obtaining more stable and more reliable group structure estimates for learning on real data. Given a single graph $\mathcal{G}_c(\tau_t)$, the algorithms in Ref. 7 output a set of overlapping clusters, $\mathcal{D}_t$ (a set of groups at time step t).

After knowing the group structure ($\mathcal{D}_t$), we can find x_{ij} and $F_l(i,j)$ for all l, i, and j. In the second step, we can solve the P_0 and P_l to maximize Eq. (26). In this way, we can verify how good the overlapping algorithm works with the communication model. Instead of maximizing Eq. (26), we maximize the logarithm of Eq. (26)

$$\log Prob = \sum_{i,j} (x_{ij} \log P_e(i,j) + (1 - x_{ij}) \log(1 - P_e(i,j))) . \quad (28)$$

In the Eq (27), we know there are two cases about $P_e(i,j)$. 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

$$\Upsilon_{ij} = x_{ij} \log P_e(i,j) + (1 - x_{ij}) \log(1 - P_e(i,j)). \quad (29)$$

Then

$$\log Prob = \sum_{i \neq j} \Upsilon_{ij} + \sum_{i=j} \Upsilon_{ij}. \quad (30)$$

P_0 is independent of P_l's, so we can work on them separately.

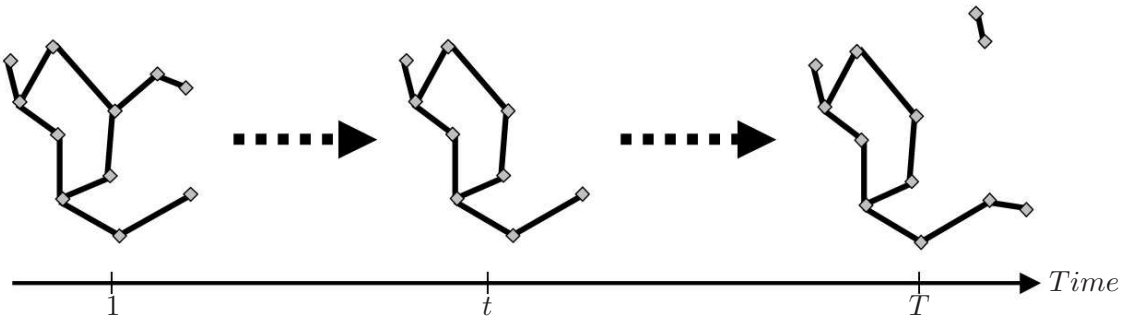


Fig. 3. An example of data set with only communication dynamics

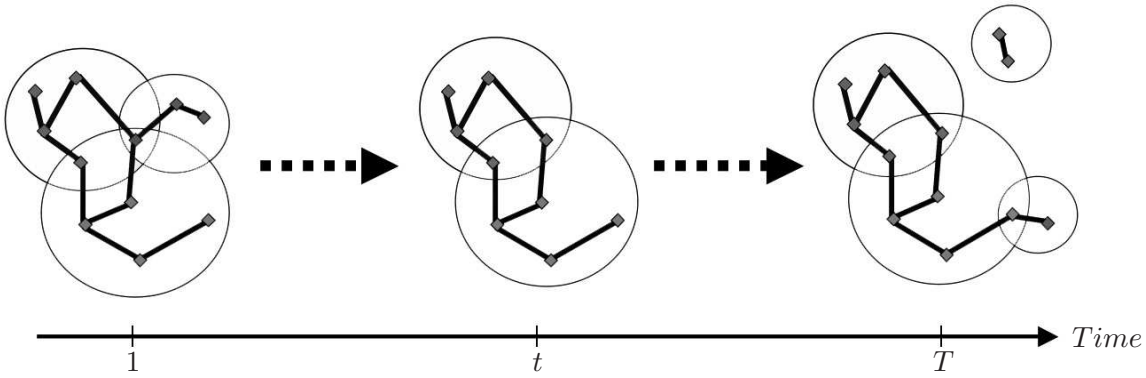


Fig. 4. An example of data set with group structures

Without friendship. We need work on the first term in Eq. (30), and $P_e(i,j) = P_0$. Then

$$\sum_{i\neq j} \Upsilon_{ij} = \sum_{i\neq j} \left(x_{ij} \log P_0 + (1 - x_{ij}) \log(1 - P_0)\right). \tag{31}$$

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 communication without friendship, and N_n be the number of pair of actors without a communication and friendship. Then

$$\sum_{i\neq j} \Upsilon_{ij} = N_e \times \log P_0 + N_n \times \log(1 - P_0). \tag{32}$$

We take a derivative of Eq. (32) with respect to P_0 to find the solution maximizing Eq. (32),

$$\frac{d\sum_{i\neq j} \Upsilon_{ij}}{dP_0} = N_e \times \frac{1}{P_0} - N_n \times \frac{1}{1 - P_0}. \tag{33}$$

Let Eq (33) = 0, and we get

$$P_0 = \frac{N_e}{N_e + N_n}. \tag{34}$$

With friendship. The second term in Eq. (30) is related to P_l's. From Eq. (27), we know

$$\sum_{i=j} \Upsilon_{ij} = \sum_{i=j} (x_{ij} \log P_e(i,j) + (1 - x_{ij}) \log(1 - P_e(i,j))), \tag{35}$$

where

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

Let $\bar{P}_l = 1 - P_l$, then

$$P_e(i,j) = 1 - \prod_{l=1}^{L} \bar{P}_l^{F_l(i,j)}. \tag{36}$$

Take a derivative on Eq. (35) with respect to $\bar{P}_k$ to find the optimal solution to maximize Eq. (35). Then,

$$\frac{\partial \sum_{i=j} \Upsilon_{ij}}{\partial \bar{P}_k} = \sum_{i=j} \left(\frac{x_{ij} \partial \log\left(1 - \prod_{l=1}^{L} \bar{P}_l^{F_l(i,j)}\right)}{\partial \bar{P}_k} + \frac{(1 - x_{ij}) \partial \log \prod_{l=1}^{L} \bar{P}_l^{F_l(i,j)}}{\partial \bar{P}_k} \right), \tag{37}$$

where

$$\frac{\partial \log\left(1-\prod_{l=1}^{L}\bar{P}_l^{F_l(i,j)}\right)}{\partial \bar{P}_k} = \frac{-\prod_{l\neq k}\bar{P}_l^{F_l(i,j)}}{1-\prod_{l=1}^{L}\bar{P}_l^{F_l(i,j)}} \times F_k(i,j)\bar{P}_k^{F_k(i,j)-1}, \quad (38)$$

and

$$\frac{\partial \log \prod_{l=1}^{L}\bar{P}_l^{F_l(i,j)}}{\partial \bar{P}_k} = \frac{F_k(i,j)}{\bar{P}_k^{F_k(i,j)}}. \quad (39)$$

The optimal solution is when Eq. (37) = 0, and we know the reasonable range of $\bar{P}_k$ is $[0.0, 1.0]$. Therefore, the easy way to find optimal $\bar{P}_k$ is using binary search.

3.2. *Learning from group structure*

From the previous stage, we have a set of group structures $\mathcal{D}_t$, $t = 1, \cdots, T$, like Fig. 4. However, in order to use the learning method in 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 actors' actions. Formally, we need a matching between the groups in $\mathcal{D}_t$ and $\mathcal{D}_{t+1}$ for $t = 1, \ldots, T-1$, (see Fig. 5): for each group in $\mathcal{D}_t$, we must identify the corresponding group in $\mathcal{D}_{t+1}$ to which it evolved. If there are more groups in $\mathcal{D}_{t+1}$, then some new groups arose. If there are fewer groups in D_{t+1}, then some of the existing groups disappeared. We define the finding matchings problem as follow:

Definition 6 (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*

$$d(x,y) = 1 - \frac{|x \cap y|}{|x \cup y|} \quad (40)$$

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 matching 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[15]. However, for our purposes, this is too slow, so we use a simple greedy heuristic. First find the best match, i.e. 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 3*.

Algorithm 3: 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.

```
S ⟵ ∅
M ⟵ ∅
foreach X_i ∈ X do
    foreach Y_j ∈ Y do
        Compute d(X_i, Y_j)
        S ⟵ S ∪ {i, j, d(X_i, Y_j)}
    endfch
endfch
Sort S based on d(X_i, Y_j)
while S ≠ ∅ do
    Find pair (i*, j*) which minimizes d(X_i, Y_j)
    M ⟵ M ∪ {i*, j*, d(X_i*, Y_j*)}
    S ⟵ S − {k, l, d(X_k, Y_l)} where k = i* or l = j*
endw
return M
```

3.3. *Learning from group evolution*

In this section, we discuss the learning stage about learning from data sets with group evolution, Fig. 5. 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, $\mathcal{D}_t = \{G_l^t\}_l$, $G_l^t \subseteq \mathcal{A}$; we use l, m, n to refer to groups. Collect all the parameters which specify the model as $\mathbf{\Theta}_M$, which includes all the parameters specific to an actor (*e.g.,* $\boldsymbol{type}$) and all the global parameters in the model (*e.g.,* $\theta_{action}, \theta_{reward}, \theta_{group}$). We would like to maximize the likelihood,

$$\mathcal{L}(\mathbf{\Theta}_M) = Prob\left(\mathcal{D}|\mathbf{\Theta}_M\right). \quad (41)$$

We define the path of actor i, $\mathbf{p}_i^T = (p_i(1), \ldots, p_i(T))$, as the set of actions it took over the time steps $t = 1, \ldots, T$. The actions at time t, $p_i(t)$, constitute deciding to join, leave or stay in groups, as well as which groups were left or joined. Given $\mathcal{D}$, we can construct $\mathbf{p}_i^T$ for every actor i, and conversely, given $\{\mathbf{p}_i^T\}_{i=1}^{|\mathcal{A}|}$, we can reconstruct $\mathcal{D}$. Therefore, we can alternatively maximize

$$\mathcal{L}(\mathbf{\Theta}_M) = Prob\left(\mathbf{p}_1^T, \ldots, \mathbf{p}_{|\mathcal{A}|}^T \| \mathbf{\Theta}_M\right). \quad (42)$$

It is this form of the likelihood that we manipulate. Typical ways to break up this combinatorial optimization is

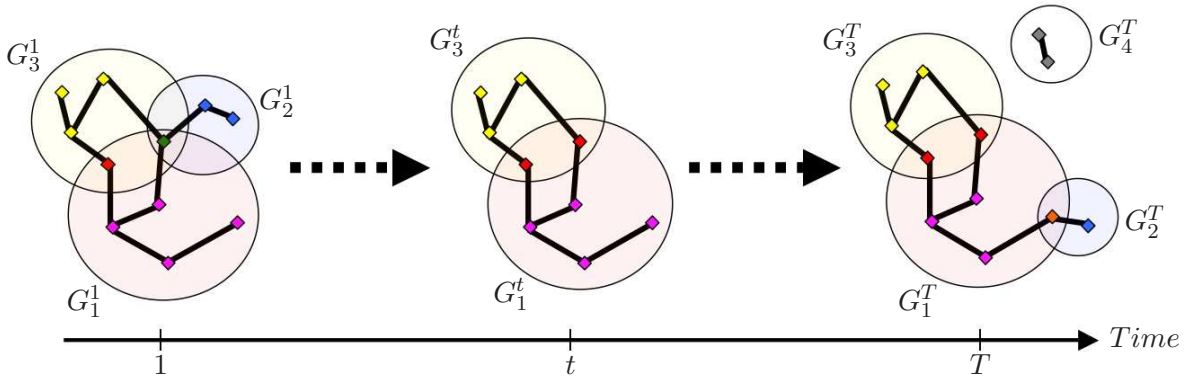


Fig. 5. An example of data set with group evolution

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, $|\mathcal{A}|$ 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 $\mathbf{\Theta}_M$ into three types of parameters:

- θ_i: the parameters specific to actor i, *e.g.*, *types*, *ranks*, *qualification* and *resources*.
- $\theta_{\bar{i}}$, the parameters specific to other actors.
- θ_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 θ_G. Since this is a fixed number of parameters, this process is algorithmically feasible. We now consider optimization with respect to θ_i, fixing $\theta_{\bar{i}}, \theta_G$. This is the task which is algorithmically nontrivial, since there are $\Omega(|\mathcal{A}|)$ such parameters.

In ViSAGE, the actors at each time step take independent actions. At time t, the state of the society $\mathcal{S}_t$, (1), can be summarized by actors' types, actors' ranks in each group, actors' available resources, and the group structure. Given $\mathcal{S}_t$, each actor acts independently at time t. Thus, we can write

$$\mathcal{L}(\mathbf{\Theta}_M) = Prob\left(\mathbf{p}_1^{T-1}, \ldots, \mathbf{p}_{|\mathcal{A}|}^{T-1} | \mathbf{\Theta}_M\right) \times \prod_{i=1}^{|\mathcal{A}|} Prob\left(p_i(T) | \mathbf{\Theta}_M, \mathcal{S}_T\right) \quad (43)$$

Continuing in this way, by induction,

$$\mathcal{L}(\mathbf{\Theta}_M) = \prod_i \prod_t Prob\left(p_i(t) | \mathbf{\Theta}_M, \mathcal{S}_t\right). \quad (44)$$

Instead, we maximize the log-likelihood ℓ,

$$\ell(\mathbf{\Theta}_M) = \sum_i \sum_t \log Prob\left(p_i(t) | \mathbf{\Theta}_M, \mathcal{S}_t\right). \quad (45)$$

Definition 7 (Dependent & Independent Parameters). *Let θ be a particular type of actors parameter – θ_i is specific to actor i, and $\theta_{\bar{i}}$ is specific to other actors. Also $\mathbf{\Theta}_M = \left\{\theta_i, \theta_{\bar{i}}, \mathbf{\Theta}_{\tilde{M}}\right\}$. Then, θ is a independent parameter if and only if $\forall i$*

$$Prob\left(p_i(t) | \theta_i, \theta_{\bar{i}}, \mathbf{\Theta}_{\tilde{M}}, \mathcal{S}_t\right) = Prob\left(p_i(t) | \theta_i, \mathbf{\Theta}_{\tilde{M}}, \mathcal{S}_t\right). \quad (46)$$

Otherwise, θ is a dependent parameter.

In the following sections, we present the basic concept and the detail of algorithms about different types of parameters in ViSAGE, which includes independent discrete parameters, *e.g.*, actors' *size-type*, dependent discrete parameters, *e.g.*, actors' *ambition-type*, and continuous parameters, *e.g.*, reward/penalty parameters θ_{reward}.

3.3.1. *Learning independent discrete parameters*

The basic ideal is to maximize the log-likelihood $\ell(\mathbf{\Theta}_M)$, Eq. (45). Now consider optimization with respect to a particular actors parameters θ_i, and then Eq. (45) becomes

$$\ell(\theta_i) = \sum_t \log Prob\left(p_i(t) | \mathbf{\Theta}_M, \mathcal{S}_t\right) + \sum_{\bar{i} \neq i} \sum_t \log Prob\left(p_{\bar{i}}(t) | \mathbf{\Theta}_M, \mathcal{S}_t\right). \quad (47)$$

When the actions of $\bar{i} \neq i$ independent on θ_i, or in some cases, the actions of $\bar{i} \neq i$ depends on θ_i only through $\mathcal{S}_t$, which is a second order dependence, therefore we

treat θ_i as an independent parameter. From the definition of independent parameter, Eq. (46), then

$$\ell(\theta_i) = \sum_t \log Prob\left(p_i(t)|\theta_i, \mathbf{\Theta}_{\tilde{M}}, \mathcal{S}_t\right) + \sum_{\tilde{i}\neq i}\sum_t \log Prob\left(p_{\tilde{i}}(t)|\theta_{\tilde{i}}, \mathbf{\Theta}_{\tilde{M}}, \mathcal{S}_t\right). \quad (48)$$

we ignore the second term in optimizing the parameters specific to actor i because only the first term is related to θ_i. Then

$$\theta_i^* \leftarrow \underset{\theta_i}{\text{argmax}} \sum_t \log Prob\left(p_i(t)|\theta_i, \mathbf{\Theta}_{\tilde{M}}, \mathcal{S}_t\right). \quad (49)$$

Thus 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 the actor paths. Therefore, the entire learning process can be summarized by maximizing over each parameter successively, where to maximize over the parameters specific to an actor, we use only that actor's path.

We present learning actors' *size-type* (§2.1.1) to illustrate the process of learning independent discrete parameters. *size-type* is a type of independent parameter because it only influences an actor its own group size preference. We have a data set which includes the information about group evolution, Fig. 5. 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 an actor's *size-type* controls which group she most likely joins or leaves (§2.2.2). Therefore, if we have the data of the group evolution, we can use the path of which groups an actor joins or leaves to learn the actor's *size-type*. Assume that we know all the values of parameters except the actors' *size-type*, then, based on Eq. (12), (13), (14), (15), and (16), we compute the probability of the path for each actor with different *size-type*. Then in Eq. (49),

$$Prob\left(p_i(t)|\theta_i, \mathbf{\Theta}_{\tilde{M}}, \mathcal{S}_t\right) = Prob_{real}, \quad (50)$$

where

$$Prob_{real} = \begin{cases} PRJoin^t_{(i,l)} & \text{, if } p_i(t) \text{ is } \textit{Join}. \\ PLeave^t_{(i,l)} & \text{, if } p_i(t) \text{ is } \textit{Leave}. \\ Skip & \text{, if } p_i(t) \text{ is } \textit{Do nothing}. \end{cases} \quad (51)$$

When $p_i(t)$ is *Do nothing*, we don't have any information about *size-type*; therefore, we skip computing $Prob_{real}$. So, what is an actor's best *size-type* is based on which actor's *size-type* have the highest probability of the path. The learning algorithm is shown in *Algorithm 4*.

Algorithm 4: Maximum log-likelihood learning algorithm for independent discrete parameters

Input : The set of matching social groups at each time step, $\mathcal{D} = \{\mathcal{D}_t\}_{t=1}^{T+1}$
Output: The best value for θ_i

```
foreach actor i do
    foreach possible value x for θ_i do
        Prob_x ⟵ Σ_t log Prob(p_i(t)|θ_i = x, Θ_M̃, S_t)
    endfch
    θ_i* ⟵ argmax_x (Prob_x)
endfch
return all θ_i*
```

Sometimes, we don't know the probability distributions of actors' group size preference used in Eq. (12). We develop an expectation-maximization[16] (EM) algorithm to learn the actors' *size-type* and also the probability distributions of actors' group size preference. We know actors with different *size-type* have different probability distribution of group size preference, and we also know the group size preference influence which group an actor most likely joins or leaves. Based on the group evolution, we are able to find out which size of groups an actor joins and leaves, and we can use this information for all actors with same *size-type* to determine the group size preference distribution for that *size-type*.

There are two steps in the EM algorithm, Expectation step and Maximization step. In the E step, we learn the distribution of group size preference for each *size-type*, and then in the M step, we optimize actors' *size-type* based on what learned from E step to maximize the probability of all actors' paths. 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, assume the group size preference for each *size-type* is a Gaussian distribution, which is different from our default model, then we need to determine the means and variances for those Gaussian distributions. We use standard K-means algorithm[21, 9] to cluster actors based on the average size of groups each actor joined into three clusters (there are three subcategories in *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 large groups, and $Type_M$ actors median groups. The clustering algorithm is shown in *Algorithm 5*.

Algorithm 5: Cluster Algorithm

Input : The set of social groups at each time step, $\mathcal{D} = \{\mathcal{D}_t\}_{t=1}^{T+1}$
Output: Means and variances of the distribution of the group size preference

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

For different probability distributions of group size preference, the general learning process in *Algorithm 5* is the same, except the step 5 to step 8. We need to learn different properties from step 5 to step 8 for different probability distributions.

In M step, we use the same algorithm as *Algorithm 4* to optimize the actors' *size-type* based on the means (μ_c's) and variances (σ_c's) learned in *Algorithm 5* as means and variances for those Gaussian distributions of group size preference. After *Algorithm 4*, 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 certain error threshold to obtain a set of clusters which have maximum probability for all paths of actors' actions. The process of EM algorithm is shown in *Algorithm 6*.

Algorithm 6: EM algorithm for learning actors' *size-type*

Input : The set of social groups at each time step, $\mathcal{D} = \{\mathcal{D}_t\}_{t=1}^{T+1}$
Output: The optimized value of θ_i

1 Compute μ_c's and σ_c's using *Algorithm 5*, Cluster algorithm
2 **repeat**
3 | Apply μ_c's and σ_c's to *Algorithm 4*
4 | Calculate μ_c and σ_c for each *actor type* based on new θ_i^*
5 **until** *exceed threshold*
6 **return** all θ_i^*

Based on *size-type* controlling an actor's group size preference, some people may think that it is sufficient to use only clustering algorithm, *Algorithm 5*, to find the appropriate *size-type*. However, in our model, when an actor decides to join or leave a group, she do not only consider the group size preference but also her friendships and qualification and groups' qualifications. The reason is that in ViSAGE, an actor joins a group by reference, *Definition 1*, so, an actor's friendships influence which groups she would like to join. In addition, an actor's qualification and groups' qualifications also influence which group is selected to join or leave, and if the group rejects the actor's join. Therefore, the EM algorithm is a better approach to learn *size-type*, and we show the results in §5.

3.3.2. *Learning dependent discrete parameters*

For dependent discrete parameters, the same ideal is to maximize the log-likelihood $\ell(\mathbf{\Theta}_M)$, Eq. (45). However, for the optimal solution, we need to compute $\ell(\mathbf{\Theta}_M)$ for all possible values. For instance, let θ be a particular type of actors parameter, θ_i is specific to actor i. If there are N actors, $\mathbf{\Theta}_N$ is a set of the θ values for all actors; $\mathbf{\Theta}_N = \{\theta_1, \theta_2, \cdots, \theta_N\}$ and $\mathbf{\Theta}_M = \{\mathbf{\Theta}_N, \mathbf{\Theta}_{\bar{N}}\}$. If each θ_i have K possible values, then, there are K^N possible combinations for $\mathbf{\Theta}_N$. If the data set have T time steps, the time complexity for computing the maximum of $\ell(\mathbf{\Theta}_M)$ is $O(TK^N)$ which is exponential computation time. Therefore, we have developed a more efficient approximation algorithm. We also use expectation-maximiztion[16] (EM) algorithm approach to solve this problem.

Algorithm 7 shows the EM algorithm for learning dependent discrete parameters. The function ComputeProb() is to compute the $Prob_x$ for all possible θ_i for each actor in one iteration under the parameters $\mathbf{\Theta}_N$, and then save the θ_i and $Prob_x$ into $\mathcal{E}_i$ for each actor. The main routine starts from line 9 which initialize the $\mathcal{E}_i$ as empty set. In the E step, line 10 to 13, we randomly assign each actor's θ_i base on normal distribution to construct $\mathbf{\Theta}_N$, and call ComputeProb() to compute and save θ_i and $Prob_x$ for each actor. Then, run this routine H iterations to generate the probability distribution for θ_i saved in $\mathcal{E}_i$. The line 14 to 17 represent the M step and also update the probability distribution for θ_i at same time. Based on the probability distribution saved in $\mathcal{E}_i$, we construct the $\mathbf{\Theta}_N$ and call ComputeProb() to compute and save θ_i and $Prob_x$ for each actor and also update the probability distribution, $\mathcal{E}_i$. Line 18 decides the optimal solution for each θ_i based on the $\mathcal{E}_i$. The complexity of the function ComputeProb() is $O(TKN)$; the fist for loop (from line 2) runs N times, the second for loop (from line 3) runs K times, and the summa-

tion in line 4 is for T time steps. The function ComputeProb() is called in line 12 and 16 for H times and I times, respectively, where I indicate the number of iterations before exceed the threshold. Therefore, the complexity of *Algorithm 7* is $O((H+I)\times TKN)$.

Algorithm 7: EM algorithm for learning dependent discrete parameters

Input : The set of social groups at each time step, $\mathcal{D}=\{\mathcal{D}_t\}_{t=1}^{T+1}$
Output: The optimized value of θ_i

1 **function** `ComputeProb(){`
2 **foreach** *actor* i **do**
3 **foreach** *possible value* x *for* θ_i **do**
4 $Prob_x \longleftarrow \sum_t \log Prob(p_i(t)|\theta_i = x, \mathbf{\Theta}_N, \mathbf{\Theta}_{\bar{N}}, \mathcal{S}_t)$
5 $\mathcal{E}_i \longleftarrow \mathcal{E}_i \cup \{x, Prob_x\}$
6 **endfch**
7 **endfch**
8 `}`**endfunc**
9 **foreach** *actor* i **do** $\mathcal{E}_i \longleftarrow \emptyset$
10 **for** *k=1* **to** H **do**
11 $\mathbf{\Theta}_N \longleftarrow$ randomly assign
12 `ComputeProb()`
13 **endfor**
14 **repeat**
15 $\mathbf{\Theta}_N \longleftarrow$ assign values based on $\mathcal{E}_i$'s
16 `ComputeProb()`
17 **until** *exceed threshold*
18 **foreach** *actor* i **do** $\theta_i^* \longleftarrow \underset{Prob_x}{\text{argmax}}(\mathcal{E}_i)$
19 **return** all θ_i^*

3.3.3. *Learning continuous parameters*

The continuous parameters, regardless dependent or independent, can be optimized using a gradient based approach, which involves taking derivatives of the log-likelihood $\ell(\mathbf{\Theta}_M)$, Eq. (45),

$$\ell(\mathbf{\Theta}_M) = \sum_i \sum_t \log Prob(p_i(t)|\mathbf{\Theta}_M, \mathcal{S}_t). \quad (52)$$

Here, we use the optimization with respect to the reward/penalty parameters ω_α (see §2.2.3) for learning the Agency and Structure Table to demonstrate learning continuous parameters,

$$\ell(\omega_\alpha) = \sum_i \sum_t \log Prob(p_i(t)|\mathbf{\Theta}_M, \mathcal{S}_t). \quad (53)$$

We know the probabilities of actors' actions are $P_{+1}(e_i^t)$, $P_0(e_i^t)$, and $P_{-1}(e_i^t)$, (see §2.1), then

$$\ell(\omega_\alpha) = \sum_i \sum_t \log P_{Act_{real}}(e_i^t), \quad (54)$$

where $Act_{real} \in \{+1, 0, -1\}$. Then we take the derivative of Eq. (54) with respect to ω_α,

$$\frac{\partial \ell(\omega_\alpha)}{\partial \omega_\alpha} = \sum_i \sum_t \frac{\partial \log P_{Act_{real}}(e_i^t)}{\partial \omega_\alpha} \quad (55)$$

$$= \sum_i \sum_t \frac{\partial \log P_{Act_{real}}(e_i^t)}{\partial e_i^t} \times \frac{\partial e_i^t}{\partial \omega_\alpha}. \quad (56)$$

From Eq. (5), we know

$$\frac{\partial e_i^t}{\partial \omega_\alpha} = \frac{\partial R_i^t}{\partial \omega_\alpha} - \frac{\partial \phi_i L_{br}^i}{\partial \omega_\alpha} - \frac{\partial \psi_i L_{bo}^i}{\partial \omega_\alpha}. \quad (57)$$

We know both *Bridging Cost* ($\phi_i L_{br}^i$) and *Bonding Cost* ($\psi_i L_{bo}^i$) are functions of R_i^t, and R_i^t is a function of ω_α. Apply the chain rule and then

$$\frac{\partial e_i^t}{\partial \omega_\alpha} = \left(1 - \frac{\partial \phi_i L_{br}^i}{\partial R_i^t} - \frac{\partial \psi_i L_{bo}^i}{\partial R_i^t}\right) \times \frac{\partial R_i^t}{\partial \omega_\alpha}. \quad (58)$$

From Eq. (20), (21), and (22), we know

$$\frac{\partial R_i^t}{\partial \omega_\alpha} = \frac{\partial R_i^{t-1}}{\partial \omega_\alpha} + \frac{\partial \Delta R_i^{t-1}}{\partial \omega_\alpha}, \quad (59)$$

where ΔR_i^{t-1} is a function of R_i^{t-1} and ω_α. After doing the derivatives and making some arrangements, we have

$$\frac{\partial R_i^t}{\partial \omega_\alpha} = \Phi^{t-1} \times \frac{\partial R_i^{t-1}}{\partial \omega_\alpha} + \Psi^{t-1}, \quad (60)$$

where Φ^{t-1} and Ψ^{t-1} are functions of R_i^{t-1}. When $t=0$, the R_i^0 is a constant, and then

$$\frac{\partial R_i^0}{\partial \omega_\alpha} = 0. \quad (61)$$

So we can use dynamical programming to solve $\partial R_i^t/\partial \omega_\alpha$ at each time step, and then calculate the derivatives of $\ell(\omega_\alpha)$, Eq. (55).

Algorithm 8 shows the pseudo code of the learning reward/penalty parameters ω_α where $\mathbb{R}_i^t$ indicates $\partial R_i^t/\partial \omega_\alpha$. There are four loop in the algorithm, and then the complexity of this gradient algorithm is $O(IKT|\mathcal{A}|)$, where $|\mathcal{A}|$ is the number of actors, T is the number of time steps, I is the number of iterations during the gradient learning process, and K is the number of ω_α.

Algorithm 8: The algorithm for learning reward/penalty parameters ω_α

Input : The set of social groups at each time step, $\mathcal{D} = \{\mathcal{D}_t\}_{t=1}^{T+1}$
Output: The optimized value of ω_α

repeat
  foreach ω_α **do**
    foreach *time step* t **do**
      foreach *actor* i **do**
        if *t=0* **then**
          $\mathbb{R}_i^0 = 0$
        else
          compute Φ^{t-1} and Ψ^{t-1}.
          $\mathbb{R}_i^t \longleftarrow \Phi^{t-1} \times \mathbb{R}_i^{t-1} + \Psi^{t-1}$.
          compute $\partial e_i^t / \partial \omega_\alpha$ based on Eq. (58).
          update $\partial \ell(\omega_\alpha)/\partial \omega_\alpha$ based on Eq. (56).
        endif
      endfch
    endfch
  endfch
  foreach ω_α **do**
    update ω_α according to $\partial \ell(\omega_\alpha)/\partial \omega_\alpha$
  endfch
until *exceed threshold*
return ω_α

4. Prediction

Unlike in a traditional supervised learning task, where the quality of the learner can be measured by its performance on a test set, in our setting, the learned function is a stochastic process, and the test data are a realization of the stochastic process. Specifically, assume we have training data $\mathcal{D}_{train} = \{\mathcal{D}_t\}_{t=1}^{T+1}$ and test data $\mathcal{D}_{test} = \{\mathcal{D}_t\}_{t=T+2}^{T+K}$. We learn the parameters governing the micro-laws using $\mathcal{D}_{train}$, and use multi-step prediction to test on the test data. Specifically, starting from the social group structure $\mathcal{D}_{T+1}$ at time $T+1$, we predict the actions of the actors, i.e., the actor paths into the furute. Based on these paths, we can construct the evolving social group structure and compare these predicted groups with the observed groups on the test data using some metric. We use 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 of size 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,

$$E_f(t) = \sqrt{\frac{\sum_k \left(N_k^{pred}(t) - N_k^{true}(t)\right)^2}{\sum_k \left(N_k^{true}(t)\right)^2}}, \quad (62)$$

$$E_p(t) = \sqrt{\sum_k \left(\frac{N_k^{pred}(t)}{N^{pred}(t)} - \frac{N_k^{true}(t)}{N^{true}(t)}\right)^2}, \quad (63)$$

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

5. Experiments

In our model, there are a lot of parameters which can be learned using the approach described in Section 3. Here, we focus on following parameters: the actor's types (**type**$_i$ §2.1.1), the actor's initial resources (E_i^0), available resources (E_i^t §2.1.4), the society reward/penalty parameters (θ_{reward} §2.2.3), and the probabilities of communications (P_l §2.3). We show the results of learning from the synthetic data and also from real data, such as Enron email and Movie newsgroups.

5.1. *Results on synthetic data*

To evaluate 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. And from the synthetic data, we are also able to collect the information about group evolution which avoid the first two learning stages mentioned in Section 3.1 and 3.2. 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 communication probabilities*

In this section, we show the learning results from the first learning stage, described in §3.1. Here, we only consider first level of friendship, and the communication probabilities are $P_0 = P_b$ and $P_1 = P_g$. Fig. 6 shows the results of learning communication probabilities. When the group structures are known, the algorithm can learn the P_b and P_g very well (dot lines).

Meanwhile, as we apply the overlapping clustering algorithms in Ref. 7 to get the group structures, the upper figure show the learned P_b's are not influenced by different P_g's. However, different P_b's have an impact on the learned P_g's (shown at the lower figure) because some outsiders have been included in the same group. The bottom line of learning communication probabilities is finding the monotonic relationship; higher learned P_g, higher true P_g, as well as P_b.

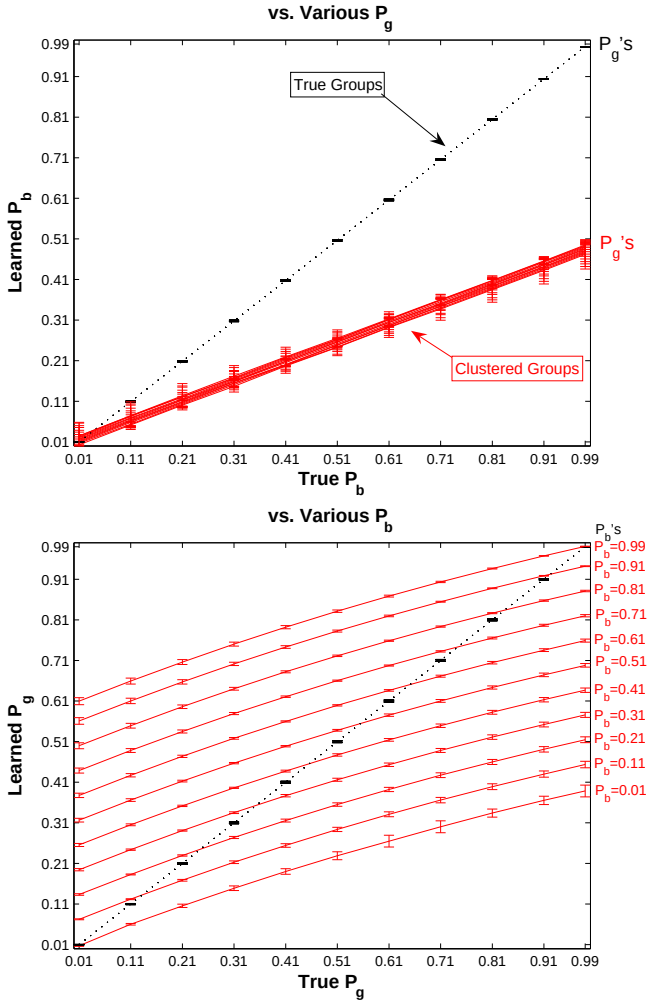


Fig. 6. The accuracy of learning the communication probabilities

5.1.2. *Learning actors'* size-type

We evaluate the learning results on actors' *size-type* from the following seven different algorithms (described in §3.3.1):

- Learn: Use only maximum log-likelihood algorithm, *Algorithm 4*, with true distributions of group size preference.
- Cluster: For each actor i, let $size_i$ be the average size of groups actor i joined. We cluster $\{size_i\}_{i=1}^{|\mathcal{A}|}$ into 3 groups using the standard 3-means algorithm, *Algorithm 5*. This is a simple heuristic based on the observation that $Type_S$ like to join small groups, $Type_L$ large groups, and $Type_M$ median groups.
- EM: With unknown distributions of group size preference, we use expectation-maximization (EM) algorithm, *Algorithm 6*, to learn the actors' *size-type* as well as the distributions of group size preference.
- Optimal: The ground truth model. (For comparison and only available on synthetic data.)
- Leader, Socialite, Follower: Benchmarks which assign all actors' *size-type* to $Type_S$, $Type_M$ or $Type_L$, respectively.

Each data set is created by randomly assigning about 1/3 of the actors to each *size-type* and simulating for 50, 100, 150, 200, and 250 time steps. All other parameters except *size-type* and distributions of group size preference are held fixed. Table 2 shows the accuracies (%) of Learn, Cluster and EM algorithms using 250 time steps of training data (averaged over 20 data sets). The accuracies are (a) 61.64%, (b) 71.59%, and (c) 90.38%, and for comparison, the accuracy of randomly assigning type is 33.33%, the accuracies of Leader, Socialite, and Follower algorithms are 33.33%, and the accuracy of Optimal is 100%.

Table 2. Confusion matrix in learning *size–type* using Cluster, EM and Learn algorithms.

(a) Cluster algorithm

Accuracy 61.64%		**Learned Actors' *size-type***		
		$Type_S$	$Type_M$	$Type_L$
True Actors' *size-type*	$Type_S$	68.25	66.75	2.4
	$Type_M$	0.75	88.5	119.25
	$Type_L$	0.0	2.65	151.45

(b) EM algorithm

Accuracy 71.59%		**Learned Actors' *size-type***		
		$Type_S$	$Type_M$	$Type_L$
True Actors' *size-type*	$Type_S$	126.05	11.24	0.11
	$Type_M$	46.55	125.2	36.75
	$Type_L$	3.47	43.93	106.7

(c) Learn algorithm

Accuracy 90.38%		**Learned Actors' *size-type***		
		$Type_S$	$Type_M$	$Type_L$
True Actors' *size-type*	$Type_S$	137.15	0.25	0.0
	$Type_M$	20.1	185.75	2.65
	$Type_L$	8.3	16.8	129

Fig. 7 shows the accuracies of three algorithms comparing versus different time steps of training data set. We can tell when the length of the time period of training

data set increases, we obtain better results from all three algorithms. The reason is that more data points we can learn from, more accuracy of the results we can achieve.

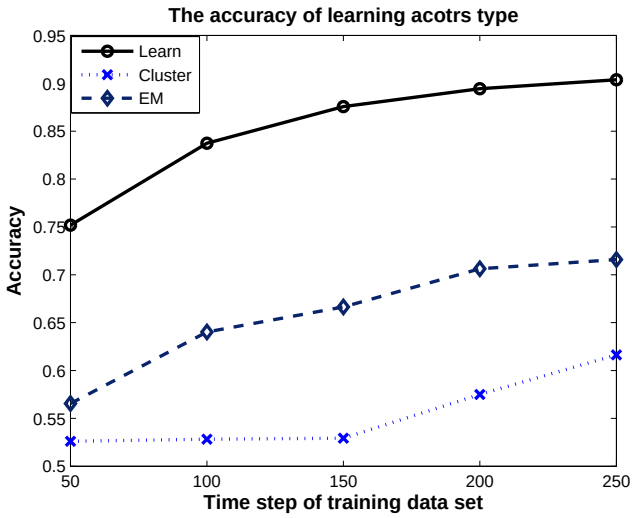


Fig. 7. The accuracy (%) of learning *size-type* using algorithm Learn, Cluster, and EM vs. different time steps of data set

The results tell that the accuracy for Learn algorithm is the best because it uses the true distribution of group size preference and only need to learn the actors' *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 actors' *size-type* also the distribution of group size preference. Table 2 and Figure 7 show that the EM algorithm does improve the results from the Cluster algorithm. In the section 2.2, we know which group is selected by an actor to apply and finally join not only depends on the actor's *size-type* but also 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 from *observable* group structure. On the other hand, the Learn and EM algorithms learn actors' *hidden* curriculum based on the interactions with other actors and the influences of the environment, which cannot be observed from the data.

The multi-step predictive performance of the algorithms on test data according to the metrics in Eq. (62) and (63) is shown in Fig. 8. The figure clearly shows that Learn surpasses all the algorithms because the prediction from Learn is most close to Optimal (which is unattainable in reality), and the worst cases are Leader and Follower which assign all actors to leader or follower types.

5.1.3. *Learning reward/penalty parameters*

In the Agency and Structure table, there are 9 parameters, $\theta_{reward} = \{\omega_\alpha | \alpha = 1, 2, \cdots, 9\}$. Here, we show the results about learning reward/penalty parameter from the synthetic data. From table 1, we know which $omega$ is used based on $Act_{normative}$ and Act_{real}. In the group evolution data, we can tell which Act_{real} an actor performs based on the actor's observable action – *join a group*, *leave a group*, or *do nothing*. However, there is no any information about $Act_{normative}$ from training data set without knowing the actor's available resources, R_i^t. Therefore during the learning process, we can only calculate the $Act_{normative}$ based on the properties of the current state.

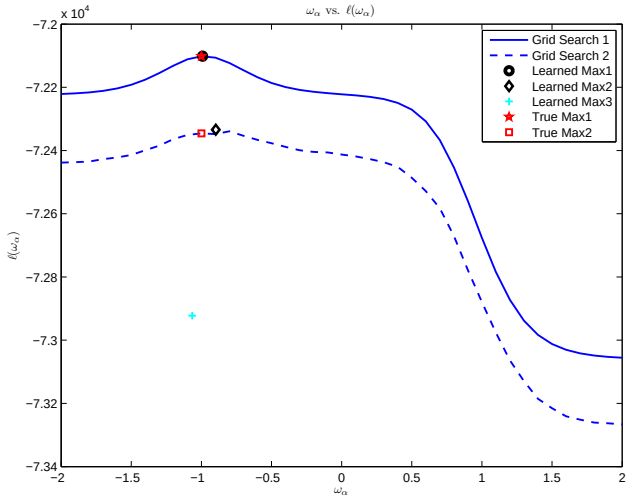
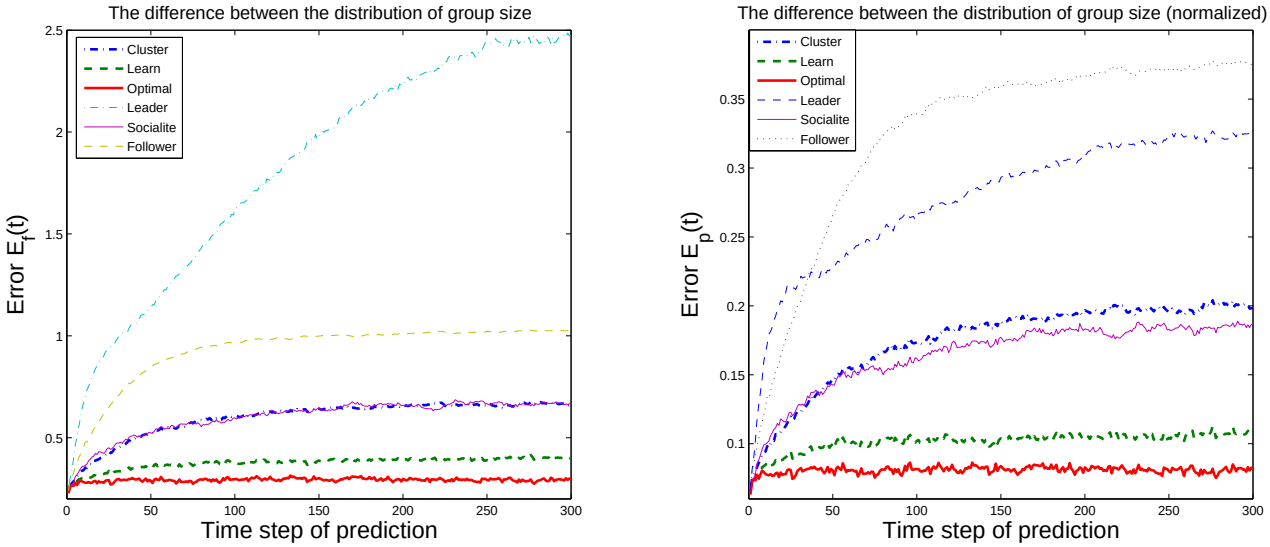


Fig. 9. The results of learning one ω_α under different conditions

Fig. 9 shows the results about learning one ω_α under different conditions. In this training data set, the values of ω_α's are between -2.0 and 2.0. The solid line, *Grid Search 1*, indicates that we know the true $Act_{normative}$ and use grid search to compute $\ell(\omega_\alpha)$ with different ω_α; the dash line, *Grid Search 2*, indicates that we don't know the true $Act_{normative}$ and only compute which $Act_{normative}$ 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 ω_α on the line *Grid Search 1* and the line *Grid Search 2*, respectively; they are around the maxima of the both lines. The circle marker (*Learned Max1*)



(a) E_f (b) E_p

Fig. 8. The predictive error for various algorithms to learn type.

and the diamond marker (*Learned Max2*) show the results about learning ω_α using the gradient learning algorithm, described in §3.3.3. Both results show the gradient learning algorithm can learn the ω_α very close to the true values. The plus sign marker (*Learned Max3*) present the result in learning all of those 9 ω_α's together without knowing the true $Act_{normative}$'s. The $\ell(\omega_\alpha)$ of the marker (*Learned Max3*) is the smallest value among 5 makers because other 4 markers (*True Max1*, *True Max2*, *Learned Max1*, *Learned Max2*) have only one unknown ω_α, but for the marker (*Learned Max3*), none of 9 ω_α's is known; however, the learning ω_α is still very close to true ω_α.

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 (ω_α) and initial resources (R_i^0) were held fixed. Table 3 shows the mean and standard deviation about the differences between the learned ω_α's and true ω_α's.

Table 3. The mean and standard deviation (S.D.) about the differences between the learned ω_α's and true ω_α's

	ω_1	ω_4	ω_7
Mean	-0.5417	0.0880	-0.0981
S.D.	0.1235	0.1405	0.2849
	ω_2	ω_5	ω_8
Mean	-0.2173	-0.1113	0.0028
S.D.	0.1277	0.0490	0.1451
	ω_3	ω_6	ω_9
Mean	0.3643	-0.1127	0.1343
S.D.	0.2323	0.1349	0.1858

The most values of functions $PenaltyW()$ and $RewardW()$ used in Eq. (21) are between -2.0 and 2.0. The absolute values of the mean of errors for ω_4, ω_5, ω_6, ω_7, ω_8, and ω_9 are between 0.0028 to 0.1343 which are between around 0.07% to 3.36% of the possible range. The mean of errors for ω_1, ω_2, and ω_3 are larger than others because these three parameters are using when $Act_{normative}$ 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, and then we observe the Act_{real} as *doing nothing*. So, in this case, there are more noise in the data set, and it can increase the error of learning.

5.2. *Impact of learning on prediction*

In the section 3, 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, the complexity of finding the optimal path using dynamic programming is $O(T\times 2^{2NK})$. Even though, we are able to develop some approximate and efficient algorithms, but when having this complexity, the learning process is still very time consuming. Therefore, it is better to know if the parameter has significantly improved the predictive performance before the learning takes place.

From Fig. 8, it is clear that the types of actors significantly impact the predictive performance as compares to other learning algorithms. While we could learn $\{E_i\}$ and θ_{reward}, perhaps it is not useful to learn them if they

don't significantly improve predictive performance. Figure 10 shows the impact of the optimal predictor (all parameters set to their true values) vs. choosing a random θ_{reward}, and for various choices of $\{E_i\}$ (every actor assigned some fixed E according to $\max\{E_i\}$, $\min\{E_i\}$ or $\text{average}\{E_i\}$). We compute the average errors from 20 runs. As can be seen, the wrong θ_{reward} does significantly affect the performance, but using the wrong $\{E_i\}$ does not (which results mixed with the optimal predictor have the average error around 0.29 in (a) and 0.08 in (b)) – which might be expected as the effects of initial conditions should equilibrate out.

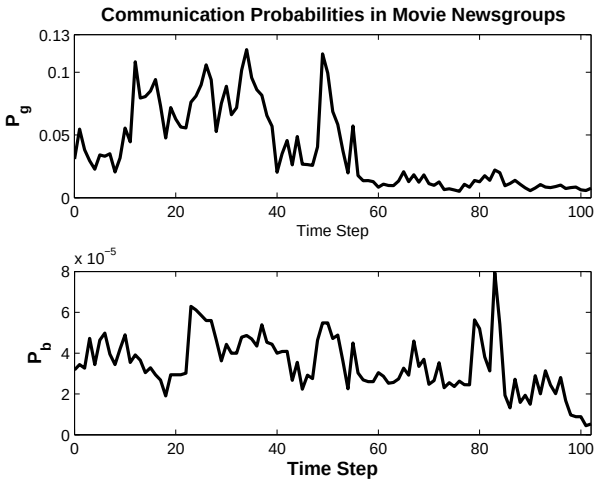


Fig. 11. The results of learning the communication probabilities on Movie newsgroups

5.3. *Results on real data*

Our results on real data are based on communications between actors without knowing the semantic 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 3 to learn the parameters. The first stage, we need to use the algorithms in the section 3.1 to obtain the group structures from communication dynamics – using the overlapping clustering algorithm[7] to obtain the groups information in each time step. Then, we apply the matching algorithm in the section 3.2 to find out the group evolution. After having the data about the group evolution, we apply the techniques mentioned in Section 3.3 to learn the parameters.

5.3.1. *Movie newsgroup.*

We collected the communication data from a Movie newsgroup, which includes 1528 active actors, and there are total 103 time steps. In the first stage, we learn the group structure and the communication probabilities. The figures 11 shows the results of learning the communication probabilities ($P_g = P_1$ and $P_b = P_0$) at different time step. A comparison from both figures show that people in the same group communicate more frequently than people in the different 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).

In the second learning stage, we use *Algorithm 3* in Section 3.2 to obtain the group evolution, and then apply the learning algorithms, Cluster (*Algorithm 5*) and EM (*Algorithm 6*) in Section 3.3 to learn the appropriate values of actors' *size-type*. The table 4 shows the confusion matrix about different classifications between EM algorithm and Cluster algorithms.

Table 4. The results of learning actors' *size-type* on Movie newsgroups.

(a) Cluster algorithm

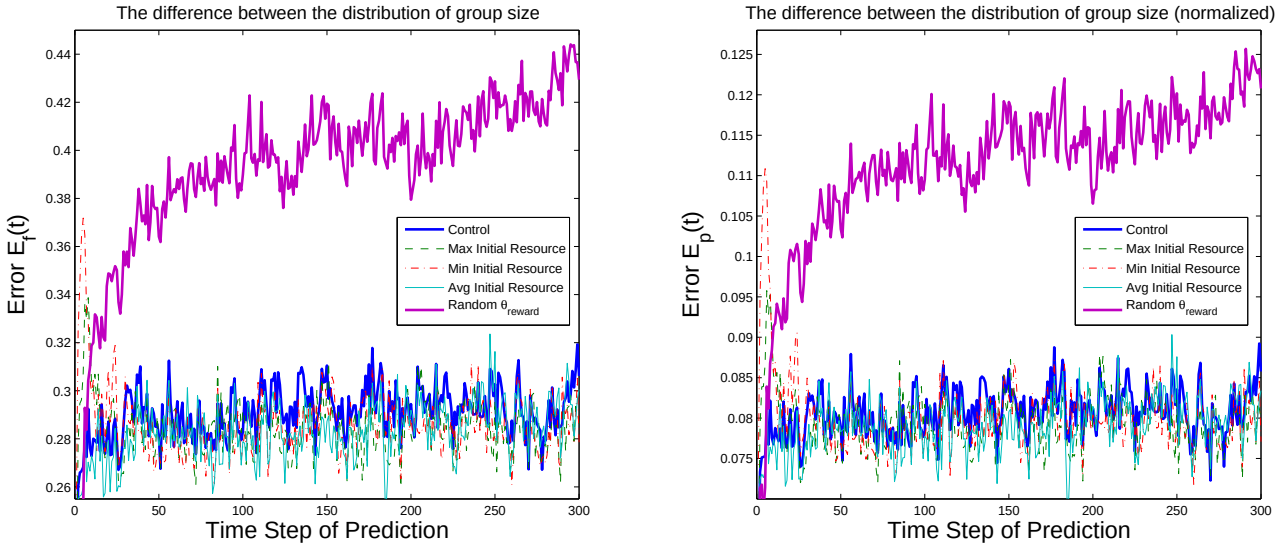
	Learned Actors' *size-type*		
	$Type_S$	$Type_M$	$Type_L$
Number of Actor	822	550	156
Percentage	53.8%	36.0%	10.2%

(b) EM algorithm

	Learned Actors' *size-type*		
	$Type_S$	$Type_M$	$Type_L$
Number of Actor	532	368	628
Percentage	34.8%	24.1%	41.1%

Based on Cluster algorithms, 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 Cluster algorithm in which group size that actors joined proves to match the finding that was done by hands in Butler's social analysis in the newsgroups data[11]. However, the result of EM algorithm shows that the number of $Type_L$ increases 30.9%, the number of $Type_S$ decreases 19%, and the number of $Type_M$ decreases 11.9%.

According to the research data shown as above, there is a significant difference between both results: in



(a) E_f (b) E_p

Fig. 10. Impact of using incorrect θ_{reward} and $\{E_i\}$ on predictive error.

Butler's finding (Cluster algorithm), it is easily for one to locate which size of the groups an actor joined manually but it is difficult for one to detect the actor's actual group size preferences, *i.e.*, social interactions between actors and groups can play an influential role in the actors' decision making. By applying EM algorithm approach, one can not only consider the observable groups size that an actor joined but also the social interactions between the actors, *e.g.*, *Join by reference*, *Qualification* of actors and groups, and so on. As can be seen from above data, the approach in using EM algorithm yields similar result that shows the majority of actors would more likely to read news ($Type_L$) than to post news ($Type_S$) in a movie newsgroup community. Actors belonging to $Type_M$ and $Type_L$ are easily attracted by $Type_S$ to join small groups (observed actions). The reason is that classes with a higher qualification ($Type_S$) attracts the lower classes ($Type_M$ and $Type_L$), 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 3 to learn ω_α's. The table 5 shows the results about learning ω_α's on the Movie Newsgroups. The learning results show that the Movie Newsgroups data set has much more friendly environment than the Enron email data set. In the Movie Newsgroups, even though an individual is more likely to leave ($Act_{normative} = Leave$), the community would still reward the individual ($\omega_\alpha > 0$) to encourage higher interaction rate regardless of their final actions (Act_{real}).

Table 5. The results of learning ω_α's on Movie newsgroups

		"Structure" $Act_{normative}$		
		`join`	`stay`	`leave`
"Agency" Act_{real}	`join`	0.11869	2.00142	2.00062
	`stay`	0.00099	0.09080	1.38086
	`leave`	-0.06444	-0.017090	1.68060

5.3.2. *Enron email.*

Before using our approach to extra the information from Enron email data set, we use the strategies in Ref. 31 to cleaning Enron email from November 13th, 1998 to June 21st, 2002 and obtain the communication network for 154 active actors. Table 6 shows the learning results of EM and Cluster algorithms.

Table 6. The results of learning actors' *size-type* on Enron email

(a) Cluster algorithm

	Learned Actors' Types		
	Leader	*Socialite*	*Follower*
Number of Actor	28	50	76
Percentage	18.2%	32.5%	49.3%

(b) EM algorithm

	Learned Actors' Types		
	Leader	*Socialite*	*Follower*
Number of Actor	24	62	68
Percentage	15.6%	40.2%	44.2%

The results from EM algorithm and 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 accordingly. Yet, in the Movie Newsgroups, actors can change groups anytime according to one's desire. Therefore, the communications within Enron email network are based upon the need of work, and employees ($Type_M$ or $Type_L$) cannot just join a group due to the attraction of the manager ($Type_S$) of that group.

Table 7. The results of learning ω_α's on Enron email

		"Structure" $Act_{normative}$		
		`join`	`stay`	`leave`
"Agency" Act_{real}	`join`	-0.10073	0.71158	1.12028
	`stay`	-0.01653	0.24063	0.05618
	`leave`	0.00787	-0.14780	-1.01433

The table 7 represents the results of learning ω_α's from Enron email data set, and it shows an interesting results. 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 ($Act_{normative} = Leave$), and his final action also showed the individual has actually left the group ($Act_{real} = Leave$), this individual will be penalized ($\omega_\alpha < 0$). Yet, as for the ones that might have showed an intent to leave but his / her final 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 final decisions (actual actions) of the individuals.

6. Conclusions

We have presented a parameterized agent-based hidden Markov model for learning actors' dynamics and the micro-laws governing the society's social group dynamics. The benefits of the multistage learning process are to extracting different information about the actor and the society dynamics, to reduce the learning noise, and to setup the checking point for evaluating the performance of algorithms, at each learning stage. 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. Our results on synthetic data indicate that when the model is well specified, the learning is accurate. Since the model is sufficiently general and grounded in social science theory, any given instance of the model can be appropriate for a given society. Therefore, under this stance, 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.

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**, as well as the penalty/reward structure for the society have significant impact 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 and learn their values for specific communities.

Acknowledgments

This material is based upon work partially supported by the U.S. National Science Foundation (NSF) under Grant Nos. IIS-0621303, IIS-0522672, IIS-0324947, CNS-0323324, NSF IIS-0634875 and by the U.S. Office of Naval Research (ONR) Contract N00014-06-1-0466 and by the U.S. Department of Homeland Security (DHS) through the Center for Dynamic Data Analysis for Homeland Security administered through ONR grant number N00014-07-1-0150 to Rutgers University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the position or policy of the National Science Foundation, Office of Naval Research, Department of Homeland Security, or the U.S. Government, and no official endorsement should be inferred or implied.

References

1. L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: Membership, growth, and evolution. In *ACM international conference on Knowledge discovery and data mining*, pages 44–54, 2006.
2. J. A. Baum and J. Singh, editors. *Evolutionary Dynamics of Organizations*. Oxford Press, New York, 1994.
3. L. E. Baum and J. A. Egon. An inequality with applications to statistical estimation for probabilistic functions of a markov process and to a model for ecology. *Bulletin of the American Mathematical Soc.*, 73:360–363, 1967.

4. L. E. Baum and G. R. Sell. Growth functions for transformations on manifolds. *Pacific Journal of Mathematics*, 27(2):211–227, 1968.
5. J. Baumes, H.-C. Chen, M. Francisco, M. Goldberg, M. Magdon-Ismail, and A. Wallace. Dynamics of bridging and bonding in social groups, a multi-agent model. In *3rd Conf. of North American Association for Computational Social and Organizational Science (NAACSOS 05)*, June 2005.
6. J. Baumes, H.-C. Chen, M. Francisco, M. Goldberg, M. Magdon-Ismail, and A. Wallace. Visage: A virtual laboratory for simulation and analysis of social group evolution. *IEEE Transactions on Autonomous and Adaptive Systems*, to appear.
7. J. Baumes, M. Goldberg, and M. Magdon-Ismail. Efficient identification of overlapping communities. In *IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 27–36, 2005.
8. T. Y. Berger-Wolf and J. Saia. A framework for analysis of dynamic social networks. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 523–528, New York, NY, USA, 2006. ACM Press.
9. C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.
10. H. Bonner. *Group Dynamics*. Ronald Press Company, New York, 1959.
11. B. Butler. The dynamics of cyberspace: Examining and modeling online social structure. Technical report, Carnegie Melon University, Pittsburgh, PA, 1999.
12. K. Carley and M. Prietula, editors. *Computational Organization Theory*. Lawrence Erlbaum associates, NJ, 2001.
13. K. Chopra and W. A. Wallace. Modeling relationships among multiple graphical structures. *Computational and Mathematical Organization Theory*, 6(4), 2000.
14. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, pages 540 – 549. MIT Press and McGraw-Hill, second edition, 2001.
15. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. Mcgraw-Hill, Cambridge, MA, 2nd edition, 2001.
16. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.
17. M. Goldberg, P. Horn, M. Magdon-Ismail, J. Riposo, W. Wallace, and B. Yener. Statistical modeling of social groups on communication networks. In *1st Conference of the North American Association for Computational Social and Organizational Science*, 2003.
18. P. Holland and S. Leinhardt. Dynamic model for social networks. *Journal of Mathematical Sociology*, 5(1):5–20, 1977.
19. R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 611–617, New York, NY, USA, 2006. ACM Press.
20. R. Leenders. Models for network dynamics: A Markovian framework. *Journal of Mathematical Sociology*, 20:1–21, 1995.
21. J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297. University of California Press, 1967.
22. T. F. Mayer. Paries and networks: Stochastic models for relationship networks. *Journal of Mathematical Sociology*, 10:51–103, 1984.
23. P. Monge and N. Contractor. *Theories of Communication Networks*. Oxford University Press, 2002.
24. M. E. J. Newman. The structure and function of complex networks. *SIAM Reviews*, 45(2):167–256, June 2003.
25. R. D. Putnam. *Bowling alone : the collapse and revival of American community*. Simon and Schuster, 2000.
26. A. Sanil, D. Banks, and K. Carley. Models for evolving fixed node networks: Model fitting and model testing. *Journal oF Mathematical Sociology*, 21(1-2):173–196, 1996.
27. D. Siebecker. A hidden markov model for describing the statistical evolution of social groups over communication networks. Master's thesis, Rensselaer Polytechnic Institute, July 2003.
28. T. Snijders. The statistical evaluation of social network dynamics. In M. Sobel and M. Becker, editors, *Sociological Methodology Dynamics*, pages 361–395. Basil Blackwell, Boston & London, 2001.
29. A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2):260–269, April 1967.
30. S. Wasserman and K. Faust. *Social Network Analysis*. Cambridge University Press, 1994.
31. Y. Zhou, M. Goldberg, M. Magdon-Ismail, and A. Wallace. Strategies for cleaning organizational emails with an application to enron email dataset. In *5th Conf. of North American Association for Computational Social and Organizational Science*, 2007.