Detecting and Tracking Communities in Social Networks

BY

CHAYANT TANTIPATHANANANDH
B.Eng., Chiang Mai University, Thailand, 2002
M.S., University of Illinois at Chicago, 2007

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2013

Chicago, Illinois

ACKNOWLEDGMENTS

This work would not have been possible without the help of my advisor Tanya Berger-Wolf who has been very supportive and encouraging throughout regardless of my stubborn ignorance. I am thankful to the members of my thesis committee, Bhaskar DasGupta, David Kempe, Robert Kenyon, Vijay Subramanian, and Philip Yu, for their helpful feedback and advice. I am also grateful to a number of professors, Gyorgy Turan, Druv Mubayi, Robert Sloan, Dan Rubinstein, Clement Yu, and Klaus Miescke, for their teaching and guidance.

I am grateful to my lab mates and friends, Andrew Ring, Arun Maiya, Chihua Ma, Habiba, Heba Basiony, Marco Maggioni, Mayank Lahiri, Paul Varkey, Rajmonda Sulo Cáceres, and Saad Sheikh, for their help and encouragement.

Last but not least, this thesis would not have been possible if not for the love and support from my family. Thank you!

C. T.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF ABBREVIATIONS

DCI	Dynamic community interpretation (problem).
EM	Expectation-Maximization (algorithm).
GPS	Global positioning system.
i.i.d.	independent and identically distributed
LHS	Left hand side (of an equation).
LP	Linear program.
MAP	Maximum a posteriori probability.
PTAS	Polynomial-time approximation scheme.
QP	Quadratic program.
RHS	Right hand side (of an equation).
SDP	Semidefinite program.
TDK	Time-decay (model).
UIC	Unilateral improvement and contraction (algorithm).
VP	Vector program.

SUMMARY

Community detection is an important task in social network analysis. Social interactions exist within some social context and communities are a fundamental form of social contexts. Communities are intuitively characterized as "unusually densely knit" subsets of a social network. This notation becomes more problematic if the social interactions change over time. The interplay between social interacts and social contexts are crucial to understand the evolution of networks. Thus, it is important to both detect communities and track their changes.

My contributions fall into two categories. First, I consider the problem of tracking communities over time, assuming that partitions into communities are already given for all snapshot graphs. The question is which community snapshot becomes which community snapshot at another point in time. My contributions to the first part are models and algorithms for tracking communities. I show a constant-factor approximation algorithm based on path cover, another algorithm based on a state-of-the-art approximation algorithm for the CORRELATION CLUSTERING problem, and a fast heuristic algorithm which produces even better solutions in practice than the two prior algorithms. Tools developed for this task can be used for longitudinal social network analysis, ecological inference, etc. Secondly, I consider the combined problem of detecting communities in network snapshots and simultaneously tracking them. For this second part, I show an algorithm based on the state-of-the-art approximation algorithm, similar to the above. This gives the first algorithm for the dynamic community detection problem which produces a numerical approximation guarantee of a solution.

CHAPTER 1

INTRODUCTION

Many systems in the world can be represented as networks in which network links represent relationships between the interrelating parts (nodes) of the systems. Examples of well-known networks are social media and online social networking sites such as Facebook, Google+, and Twitter. Networks have been used to model systems of interrelating parts in many fields ranging from social sciences to behavioral ecology to molecular biology, from civil engineering to electrical engineering, to computer science. Network links, in different domains, represent various kinds of relationships such as human friendship, organizational structures, physical proximity of animals, interconnectivity of infrastructures, Web hyperlinks, or even more abstract relationships such as similarity of data points.

One of the most important observations about networks in the nature is the existence of communities (15; 43; 46; 100; 101; 102). Communities, also known as modules and clusters, are sets of nodes which are relatively more connected, and are believed to be the intrinsic structures in networks in the nature. Nodes in the same community often share interesting properties such as a common function, interest, or purpose. Thus, community detection is one of the most important problems in network analysis. Among the areas to which network analysis is applicable, my research interest is specifically in developing computational methods for analyzing networks which arise in sociology and behavioral ecology such as those which describe friendship links and animal social interactions. Next, I review important questions in these specific fields which motivate my work.

1.1 Motivations

One of the most important questions in sociology is that about communities. This is known as the Community Question (101) which can be paraphrased as: How does the community structure affect the social interactions among the people? Vice versa, how does the social interactions among people affect the community structure? In terms of social network, this question asks how community structure affects the formation and evolution of links between nodes, and vice versa. The nature of a human community used to be confined by geography. In fact, the term *community* once used to mean a place. As the technology in transportation and communication shifted, the mode of social interactions has changed, and so did the way people interact with their communities (5). Nevertheless, communities still exist and are the foundations of human societies (15; 101; 102). Thus, the question about communities is still important. To answer it, we need to first be able to identify the community structure in a social network. Despite the importance and the long history of the problem of identifying communities, there is still no consensus answer to the problem.

At the heart of my thesis is the one aspect of social networks that has not been sufficiently addressed — *time*. We seem to perceive the world as slowly evolving and essentially *static*. Fifty years ago, Wilbert Moore made an observation (67) that *"social sciences tended to neglect the way the limits and flows of time intersect the persistent and changeful qualities of human enterprises."* He corroborated his observation by noting the dominance of *static* models in social analysis at the time, and further made an observation that *"all analytical sciences tend to perfect their descriptions of elements and observations of combinations before they develop the capacity to observe orderly transformations in the course of time."* In the present days, static models are still dominant in social network analysis. In particular, most work on identifying communities assumes that social networks and the underlying community structures are *static*. Given the pace at which our societies have changed, any conclusions drawn from such static analysis are unlikely to withstand the test of time. Moreover, given this fifty-year-old observation of Moore, I doubt that we would ever perfect the descriptions of elements and proceed to investigate the transformations. This clearly necessitates the development of frameworks and algorithms for identifying the community structures in ever-changing social networks, which is at the heart of my thesis.

Besides humans, other social animals are prevalent in the animal kingdom. Time is still a very important aspect in studying social behaviors of animals. Especially for conservation purposes, it is important to be able to detect and recognize changes as they happen in order to contain the damages or reverse the process before it is too late. The most basic social interactions of many animals are in the form of social groups. Many species (including birds, spiders, fishes, and mammals) live in groups. Group living offers the strength in numbers which helps compensate for the predation risk (17; 57). Group living also offers a place for social learning (58) by which animals learn about foraging areas, dangerous places, food preferences, predator recognition, preferences on mates, just to name a few. Social interactions in animals are intrinsically confined by spatial proximity since the interactions involve an individual seeing, hearing, smelling, or touching other individuals. To juxtapose with human, animal social groups are similar to what human communities used to be in the past when face-to-face meetings were the most common mode of social interaction. Nonetheless, the complexity of animal societies ranges from very simple to fairly complex, from shoals of fish to pair-bonding birds to social hierarchies in primates (24). In a simple society in which interactions mean being in close physical proximity, the corresponding social network is quite easy to construct and study. In a more complex society in which members of a social group form bonds with certain other members, it is difficult to gauge the strength of the bonds (88). Unlike with humans, we cannot interview or use questionnaires with animals (but, thankfully, they do not use email or cellphones). Although there is a way to estimate the strength of the bonds using associate indices (24), it is less than ideal especially when we know that social interactions change over time. These association indices treat the group events as independent sampling events, and as a result, disregard the chronological continuity and ordering. This problem is perhaps most severe in species which live in fission-fusion societies (17; 23; 57). The changes in group compositions are probably the most interesting part to observe and study in such societies. This necessitates the development of methods for analyzing transitions of social groups, which is at the heart of this thesis.

1.2 Organization of the Thesis

In this thesis, I developed computational solutions for the problem of tracking communities especially in social networks which change over time. In particular, the main contribution of my thesis is two-pronged. The first is a well-motivated computational framework. The other is very efficient and accurate algorithms for the task. The rest of the thesis is organized as follows. Chapter 2 gives the definitions of notations and a formal statement of the problems which are addressed in this thesis. Chapter 3 surveys the existing literature on the problem and other related topics. After that, the thesis is conceptually separated into two parts. The first part, which spans two chapters 4 and 5, deals with the problem of tracking communities over time, assuming that community membership in each snapshot of the network is already given. The goal of this part is to develop computational tools which consistently string together these snapshots of community memberships across time. In particular, Chapter 4 presents a solution to a special case in which time is treated as a series of discrete and uniform timesteps. In other words, the snapshots are assumed to represent community memberships at equidistant timesteps. This special case naturally arises in many applications where data can be collected at pre-defined time intervals. In Chapter 5, I relax this assumption about uniform timesteps and consider the general case of the problem of tracking communities over timesteps of variable length. This concludes the first part. The second part is Chapter 6 in which I consider the problem of *identifying* communities in a general dynamic network and simultaneously tracking them over time. Lastly, Chapter 7 gives concluding remarks and future directions.

Lastly, software used in this thesis is or will be made available at the following website: `http://compbio.cs.uic.edu/~chayant/commdy/`.

CHAPTER 2

PRELIMINARIES

In this section, I give the basic definitions which I use throughout the thesis, and formally definite the problem of identifying communities in dynamic social networks.

2.1 Basic Notations

A *static* graph is denoted by $G = (V, E)$ where V is a set of n vertices and $E \subseteq V \times V$ is a set of edges. Although some graphs here contain self loops $(u, u) \in E$, most of them do not. Graphs here also do not have multiple edges between the same pair of vertices. For such a simple graph $G = (V, E)$ with no self loops, G can have as many as $\binom{|V|}{2}$ edges where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is the binomial coefficient, $0 \leq k \leq n$.

In social network terminology, it is more common to refer to the vertices and edges of a graph as nodes and links. I will use the terms interchangeably. For a graph G, I will also use $V(G)$ to denote the vertex set of G and $E(G)$ to denote the edge set of G. Let $A = [A_{ij}]$ denotes the adjacency matrix of the graph G where A_{ij} denotes the element at the i^{th} row and j^{th} column. $A_{ij} = 1$ if (i, j) is an edge in G, otherwise 0. Let $N(v)$ denote the neighborhood of v, which is the set of vertices adjacent to v. Let d_v denote the degree of the vertex v, which is the number of neighbors of v. For a subset of vertices $C \subseteq V$, let $G[C]$ denote the induced subgraph on C. That is, $G[C]$ is the graph on vertex set C and the edges in $E(G)$ with both end points in C. Since non-induced subgraphs arise very rarely in my context, I will use the terms *subgraph* and *induced subgraph*, interchangeably, unless stated otherwise. A *partition* of a set S is a collection of disjoint subsets $\mathcal{P} = \{P_1, \ldots, P_k\}$, called *parts* or *clusters*, whose union $\bigcup_{i=1}^{k} P_i$ equals to S. All partitions in this thesis are partitions of the vertex set of some graph, unless stated otherwise. Each element of S belongs to exactly one part. Thus, a partition of a vertex set can be thought of as an assignment of vertices to communities. The terms *partition*, *clustering*, and *community assignment* will be used interchangeably.

A *dynamic social network*, or simply a *network*, on a vertex set V is a sequence of *snapshot* graphs $\mathcal{G} = \langle G_1, \ldots, G_m\rangle$ over m discrete *timesteps* $t_1 < t_2 < \cdots < t_m$. The *snapshot at time* t_i is the graph $G_i = (V_i, E_i)$ on vertex set $V_i \subseteq V$ which is a subset of all the vertices in the network. The vertices in V_i are *known* to be interacting at timestep t_i and the edges in E_i represent the *observed* interactions at that timestep. For the vertices not in V_i, we do not know anything about their interactions at timestep t_i, or the lack thereof. A *community structure* or a *coloring* is a function $\chi : V \times \{t_1, \ldots, t_m\} \to \mathbb{N}$. Each color represents a unique community. The interpretation is that the color $\chi(v, t)$ represents the community with which the vertex v is affiliated at the timestep t. A vertex v at a timestep t is affiliated with the same community as a vertex v' at a timestep t' if and only if $\chi(v, t) = \chi(v', t')$.

2.2 Problem Statements

In this section, I formally state two problems to be addressed later in this thesis: (a) community detection in dynamic social networks, and (b) tracking communities over time.

2.2.1 Community Identification in Dynamic Social Networks

The problem of identifying communities in a dynamic social network can be generally defined as follows. Given a dynamic social network $\mathcal{G} = \langle G_1, \ldots, G_m \rangle$, we want to find the community structure χ which *best* explains the social interactions in $\mathcal{G}$. I will define what *best* means in later chapters. The problem statement suggests that the problem can be solved in two steps:

1. Finding communities in each snapshot graph G_i,
2. Stringing the communities across snapshots into a community structure χ.

The Step 1 is essentially finding communities in a static graph, which is not the main focus of this thesis (see Chapter 3 for a survey). Step 2 is the main focus of this thesis. Now, I formally define the problem in Step 2. Having done Step 1, we have partitioned the vertex set V_i into community membership $\mathcal{C}_i = \{C_1, \ldots, C_{k_i}\}$ which best explains the social interactions observed at time t_i. Together, $\langle \mathcal{C}_1, \ldots, \mathcal{C}_m \rangle$ is a series of community memberships over all timesteps. The problem is to find a coloring function χ which *best* explains $\langle \mathcal{C}_1, \ldots, \mathcal{C}_m \rangle$. Again, I will define what I mean by *best* later. In addition to solving the problem in two separate steps, I also present how to solve the problem in one step in Chapter 6.

2.2.2 Tracking Communities over Time

In real world, the membership of communities tend to change gradually. Backstrom et al. observe this on the communities of LiveJournal users and communities of conference publications on DBLP (5). So it is important not only to detect communities but also to track the changes in membership over time. In this section, we review other work on tracking communities over time. That is, we suppose that communities with each snapshot of the network are have been detected. Now, the questions are: Which community in one snapshot becomes which community in the next snapshot? What membership changes occur in between?

The problem of tracking communities is motivated by a problem in behavioral ecology in studying animals which live in fission-fusion societies such as zebra and the Asiatic wild ass (92). During a field drive, one group of animals is usually sighted and observed at a time. The observers visually observe the behaviors of the group members from distance for a certain period of time. It is possible two separate groups or more are encountered at the same time, but such occurrences are rare. Group membership in a fission-fusion society is fluid. A natural question is which group that we observe today is the same group which were previously observed. Groups in this setting are manifestations of perpetual communities. Inversely, a community is a consistent string of groups seen on different days. This is the problem of tracking communities over time. Loosely speaking, it is about how to string different groups from the same day into communities which span multiple days.

The problem of tracking communities over time is very similar to the second step in the two-step process mentioned in the previous section. Strictly speaking, the problem of tracking communities is more *general* than the Step 2 in the two-step process. Step 1 in the two-step process produces a sequence of partitions of the entire graph, while the problem of tracking communities requires as an input only a sequence of collections of disjoint groups, which are not necessarily partitions of the entire graph. That is, some communities can be unobserved at some timesteps. This means, if we use the two-step process to detect communities in a dynamic social network, then Step 1 does not need to produce a partition of the entire graph. It may selectively produce only clusters on which it has high confidence. This relieves the algorithm from trying to put every vertex in some cluster, the result of which usually is a rather artificial collection of clusters.

CHAPTER 3

RELATED WORK

In this chapter, I survey the current literature on the community identification problem and other closely related problems. First, I review the work on identifying communities in *static* graphs. Then, I review the literature on data clustering which is a long-standing problem. Many methods for identifying communities are inspired by methods for data clustering. Then, I review the work on a more recent problem, the CORRELATION CLUSTERING problem. Then, I shift the gear to consider work which explicitly takes the time aspect of networks into consideration. In particular, I gives a review of the work on tracking communities over time. Lastly, I review the work on identifying communities in dynamic social networks.

One aspect in all of the above problems, which has been completely ignored, is that *seeing patterns is a matter of magnifying scale.* There is no right or wrong way of looking at a system. You can look at a system at different *magnifying powers* and find different patterns and structures at different levels. This explains why every single method has a *magnification knob* built in, either as a hard-wired or variable parameter. This magnifying knob is what one uses to change the sizes of the communities or the number of communities. Ideally, one should be able to use the same method to obtain different results ranging from a partition with singleton clusters to a partition with one big cluster containing all vertices. The availability of a magnifying knob has been perceived as a limitation of methods (38; 91) but evidences suggest that this is inevitable in clustering problems (55; 62).

In this chapter, I will refer to the set of vertices under consideration as a *candidate set* or just a *subgraph* (induced by some certain candidate set) instead of a *community* to emphasize the fact that it may not represent an actual community yet. I will start with the sociological concept of communities, since this is where it all begins.

3.1 Sociology of Cohesive Groups

In sociology, the terms *groups* and *communities* are often used interchangeably to refer to the same concept of cohesive groups (13; 14; 45; 101). Sometimes, the term *subgroup* is used instead if the term *group* is used to refer to a sample of the population (100). According to Reeves (80), all sociological groups have the following properties in common:

Cohesiveness Cohesiveness is the force that holds a group together. The level of cohesiveness varies according to many factors such as existence of threats from the outside, strength of leadership, group identity and objectives, and relationships among group members.

Norms Group norms are the standards of behaviors which arise naturally until they have become tradition-like. Group norms are the laws of the group and reveal things that a group holds as values.

Sanctions Group sanctions are the policing force which enforces the group norms. Norms of a group change over time. A new norm maybe introduced to the group, while an old norm vanishes unless the group members maintain it.

Objectives Group objectives are the reasons why a group exists. The objectives of the group come from all of its members who continue to maintain those objectives. Objectives may change from time to time according to internal and external changes, so that the group remains competent.

Self-perpetuation A group is a living entity which has a desire to survive and continue to live. The most important element to self-perpetuation is the leader-follower relationship. The health and survival of the group depend on decisions made by the leader and how the followers evaluate those decisions as fitting the group objectives.

In summary, all groups share the same set of properties to some extent. For a group to exist and continue to survive, there must be some cohesive force which holds its members together. As a group continue to survive and thrive, it develops its own social norms and traditions which are deemed to be important by its members, and are enforced by the means of group sanctions. The objectives of the group are evaluated from time to time to reflect changes from the inside and outside. The interactions between the leader and the followers determine the group's effectiveness in achieving its objectives, its health, and its survival. These are the sociological aspects which all groups share and should be taken into consideration when formalizing the notion of groups or communities. One thing to note here is that the time aspect is part of almost all of the above properties and changes are inherent to the nature of groups. With this in mind, we proceed to review approaches taken by researchers to define and identify cohesive groups in social networks.

3.2 Community Identification in Static Networks

Having mentioned Moore's observation about the dominance of static analysis in social sciences (67) in Chapter 1, it is not so surprising to find that the majority of work on identifying communities assumes that social networks are *static*. Nevertheless, it is still an important question since there are cases in which networks can and should be assumed to be *static*. In this section, a social network is represented as an undirected graph $G = (V, E)$ in which an edge $(u, v) \in E$ represents a social relationship or interaction between the nodes u and v. Unless stated otherwise, a graph G is unweighted. There are several survey papers on this topic (16; 25; 38; 39; 41; 61; 70; 77; 83). Intuitively, communities are areas in the network which have relatively more connections within the areas than across different areas. However, no single formal definition of communities is universally accepted (38). Even then, in many cases, researchers usually start by stating some intuitions about communities, then proceed to describe a heuristic algorithm which find communities that capture the intuitions, without formally define what communities are. Such notions of communities are usually termed as being *operationally* or *algorithmically* defined, which essentially means that communities are whatever the heuristic algorithms output. In this section, I try to categorize methods for identifying communities based on the *nature* of communities that the methods capture.

3.2.1 Local Notions

In this section, communities are viewed as *independent patterns* which repeat themselves. Here, we are concerned with finding all community-like patterns without worrying about how the patterns fit together into one picture of overall community structure. To some extent, such communities can be considered as autonomous units. This was how the notion of communities was viewed in the early development of social network analysis, which was heavily influenced by graph theory (84). As a result, the early definitions of communities are mostly graph-theoretic and local. One thing that these early definitions have in common is that they are all *maximal* subgraphs with respect to some graph-theoretic properties.

The simplest, yet strictest definition of a community is a *clique*, which is a complete subgraph. Every vertex is adjacent to every other vertex in a clique. Because of its restrictive nature, many researchers have proposed "better" alternatives of cliques by considering the diameter of a (sub-) graph. Cliques are subgraphs with diameter one,[1] so other subgraphs with small diameters can potentially be community-like as well. This gives three variations of cliques: n-cliques, n-clans, and n-clubs. All are maximal subgraphs of diameter n. The subtle differences between n-cliques, n-clans, and n-clubs are in the choices of graphs and subgraphs used in computing the distances. n-cliques use the distances in the original graph. n-clubs use the distances in the subgraphs. n-clans use the distances in both the original graph and the subgraphs. These choices affect the computational complexity of the algorithms for finding them. See (100) for details.

Derényi et al. (29) generalized the notion of cliques in a slightly different way by means of clique adjacency. Two cliques of size k are considered *adjacent* if they share all but one vertex. If two cliques overlap one another significantly, then they would be put in the same community. This notion of clique adjacency is used to defined a notion of community, called *clique percolation*, which is the maximal collection of adjacent cliques of the same size k. A

[1]The diameter of a graph is the longest distance (which is defined as the length of the shortest path) between any two vertices in the graph.

collection of adjacent cliques is community-like since the vertices are highly knitted throughout, although the diameter of the subgraph can be potentially large.

Another measure of cohesiveness, which is used to define communities, is the minimum number of intra-cluster neighbors. A k-plex is a maximal subgraph whose vertices are adjacent to all but at most k other vertices in the subgraph. Formally, a k-plex is a maximal subgraph in which $|C - \{v\} - N(v)| \leq k$ for all $v \in C$. Similarly, a k-core is a maximal subgraph whose vertices are adjacent to at least k other vertices in the subgraph. Formally, a k-core is a maximal subgraph in which $|N(v) \cap C| \geq k$ for all $v \in C$. In other words, k-plex requires the minimum number of cluster members which each vertex cannot miss (being adjacent to), while k-core requires the minimum number of cluster members to which each vertex has to be adjacent. The two concepts are complimentary to one another. In fact, a k-core on ℓ vertices is an $(\ell - k - 1)$-plex. Figure 1 shows an example in which C is both a 4-plex and a 3-core.

The notions that I have presented so far only capture the cohesiveness inside a subgraph. It is also important to consider how disconnected a subgraph is from the rest of the network. An LS-set is a subgraph in which each vertex has at least half of its neighbors inside the subgraph. LS-sets are also known as *strong communities*. Instead of counting the neighbors of each vertex, we can also count the neighbors of the subgraph itself. This is known as a *weak community*, which is a subgraph which has at least half of its neighbors inside the subgraph. Another version of strong and weak communities is that of Hu et al. (52) which takes into account not only the number of neighbors in the subgraph but also those in the neighboring communities.

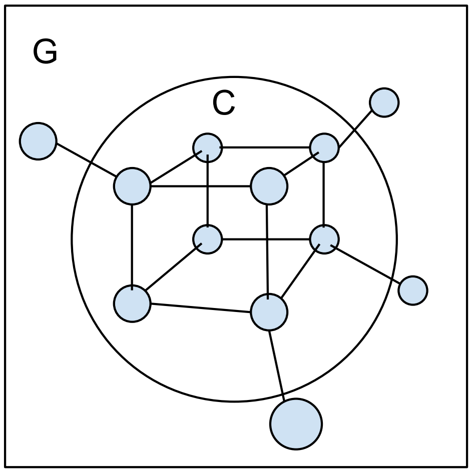


Figure 1: An example subgraph C which is both a 4-plex and a 3-core.

Another way to locally define a community is to constrain both the numbers of neighbors inside and outside the candidate set. An example is (α, β)-community proposed by He et al. (49). For $\alpha < \beta$, an (α, β)-community is a subgraph on vertex set C such that each vertex in C has at least β neighbors in C and each vertex outside C has at most α neighbors in C.

Another measure of cohesiveness is the connectivity number of a graph, which is how many vertices (or edges) needed to be removed to disconnect the graph (30). A graph is k-vertex-connected, or just k-connected, if there is a set of k vertices whose removal disconnects the graph. A graph is k-edge-connected if there is a set of k edges whose removal disconnects the graph. Not only for graph, this notion can also be used to measure the connectivity of a pair of vertices. The edge-connectivity of two vertices is the number of edges whose removal disconnects the pair. Edge-connectivity of a pair of vertices measures the fault tolerance of the network in connecting the pair. A *lambda set* is a subgraph in which any pair of verities has a larger edge-connectivity than any pair of vertices with (exactly) one end point in the subgraph.

Cohesiveness measures are also known as *fitness measures* (99). The higher the measure on a subgraph, the more community-like the subgraph is. The first is *intra-cluster density* which is the fraction of the number of edges in the subgraph,

$$\mathit{density}(C) = \frac{|E(G[C])|}{\binom{|C|}{2}}.$$

Another measure is *relative density* which is the ratio between the number of intra-cluster edges and the number of edges which touch the subgraph,

$$\mathit{relative\text{-}density}(C) = \frac{|E(G[C])|}{|\{(u,v) \in E : u \in S\}|}.$$

Then, we define a threshold of the fitness measure above which a subgraph is a community.

Another measure of how disconnected a subgraph is from the rest of the network is *conductance*, which is the ratio between the size of the cut and the volume of the set,

$$\mathit{conductance}(C) = \frac{|\{(u,v) \in E : u \in C, v \notin C\}|}{\min\left\{\mathit{volume}(C), \mathit{volume}(\overline{C})\right\}},$$

where the volume of a set C is $volume(C) = \sum_{v \in C} d_v$. The lower the conductance value, the more community-like a candidate set is. Thus, a community is defined as a candidate set whose conductance is no greater than a certain threshold.

The problem with using local definitions is that they treat communities as independent entities out of context. Using local approaches, we might be able to quickly find many interesting structures in the network, but a network is a system of interrelated parts. Local approaches lack the capability to explain how communities in a network interact with one another. This motivates kinds of definitions based on global structures of the networks, which I review next.

3.2.2 Global Notions

Compared the local notions of communities in the previous section, the notions in this section are based on the structural information about the whole graph. As mentioned previously, local definitions are usually "self-centric" in nature. Global notions look at the network link structure and all candidate sets that form a community structure as a whole. In particular, a good quality function should capture the notion of a good collection of communities. A quality function is a function which takes a partition of the vertex set of a graph and outputs a number which quantifies how likely the parts of the partition are communities of the graph. Since the notion of communities itself is not universally defined, we have to keep in mind that each quality function captures only one aspect of communities.

Let C be a clustering (a partition) of a graph and let C_v denote the cluster of node v. One quality function is *performance* which is the fraction of all vertex pairs classified correctly by the partition,

$$Performance(C) = \frac{|\{(u,v) \in E : C_u = C_v\}| + |\{(u,v) \notin E : C_u \neq C_v\}|}{\binom{n}{2}},$$

where $\sim_C$ denotes the equivalence relation associated with a partition C ($u \sim_C v$ if and only if the vertices u and v are in the same part of C).

Another quality function is the *coverage* which is the fraction of edges of the graph which fall inside the parts of the partition,

$$Coverage(C) = \frac{|\{(u,v) \in E : C_u = C_v\}|}{|E|}$$

Another way to define a global notion of communities is to use a null model. A community structure is an evidence of heterogeneity in a social network. Links are biased toward vertices which belong to the same communities. On the other hand, a random graph, such as Erdős-Rényi (34), imposes homogeneity of link structure, and thus is unlikely to have a community structure.

This leads to the definition of modularity by Newman and Girvan (72),

$$Q(C) = \sum_{u,v \in V} \left[\frac{A_{uv}}{2|E|} - \frac{d_u d_v}{4|E|^2} \right] \delta(C_u, C_v),$$

where δ is the Kronecker delta and C_u is the community ID of u. That is, $\delta(C_u, C_v) = 1$ if u and v are in the same community, otherwise $\delta(C_u, C_v) = 0$. The intuition behind the formula is that it compares the number of intra-cluster edges in the original graph with the expected number of intra-cluster edges when the graph is randomly rewired based on the degrees of the nodes. Modularity was boasted to be a way to automatically discover the *true* number of communities. However, there are known problems associated with maximizing modularity (48; 51; 61; 62) such as the tendency to favor approximately equally-sized clusters. It still remains unconvincing whether maximizing modularity can really uncover the true number of communities.

Another way to think of modularity is in the framework of the CORRELATION CLUSTERING problem (discussed in Section 5.4). Modularity, as an objective function, converts an undirected graph into a complete graph with both positive and negative weights. Thus, it is neither a similarity function nor a distance function. Clustering in a graph with both positive and negative weights is precisely the CORRELATION CLUSTERING problem.

The community detection problem is closely related to the data clustering problem. In data clustering, Jon Kleinberg (55) proved an impossibility result which states that there are no clustering algorithms (or, clustering functions) which satisfy three properties of good clustering algorithms, simultaneously. Unfortunately, it is unknown whether the theorem generalizes to the community detection problem. I will discuss about this in more details in Section 3.6.

3.2.3 Data Clustering

Data clustering, also known as cluster analysis, differs from the community detection problem in that the entities being clustered are data points from some ambient space. In other words, data clustering tries to group items based on their *attributes* while community detection tries to group items based on their *relationships.* The notions of similarity and distance are commonly used in data clustering. To use a data clustering technique to do community detection, one starts with embedding a social graph in some space so that data clustering methods can be used. For example, spectral graph clustering embeds the vertices in the eigenspace of some graph Laplacian. It then derives some similarity measures from the graph topology and finally applies data clustering methods. Some data clustering techniques, such as hierarchical methods, do not require an embedding of graph in some space as long as the similarity between every pair of data points can be computed.

3.2.3.1 Distance-based Clustering

Many community detection algorithms use transitional techniques developed for data clustering, also known as cluster analysis. In data clustering, data points belong to some vector space in which a norm or distance is defined. The number of expected clusters k is usually given as a parameter. The most popular technique is perhaps k-means in which one wants to cluster the data points such that the intra-cluster sum of squared distances is minimized,

$$\sum_{i=1}^{k} \sum_{x_j \in S_i} ||x_j - \mu_i||^2,$$

where μ_i is the mean of the points in cluster S_i. The problem is NP-hard and is usually solved using Lloyd algorithm (65) which converges to a local optimum. Other variations of k-means are k-centers, k-medians, and k-medoids. Also, Rattigan et al. (79) extended k-means to graph clustering. One advantage of these methods is that they not only partition the data points but also partition the ambient space. For example, the result of k-means is a partition of the data space into Voronoi cells. Thus, the result can be used to classify new data points by assigning each new data point to the cluster whose mean is nearest. To use these methods, we need to first embed an input graph in some metric space. Spectral clustering in Section 3.2.4 is an example of such embedding. Embedding can be natural for some graphs but artificial for others.

3.2.3.2 Hierarchical Methods

Community structures in the real world usually have nested structure. Examples of different granularities of communities include townships, counties, states, countries. Thus, it is more reasonable to model a community structure as a dendrogram instead of a partition. By looking at different levels in a dendrogram, one can *zoom in and zoom out* to look at the community structure at different granularities.

Agglomerative algorithms start from all vertices in singleton clusters (by themselves). Then, two clusters are merged based on their similarity. The merging continues until only one cluster is left. There are several ways to define the similarity of two clusters, which gives rise to different agglomerative algorithms. *Single linkage* defines the similarity between two clusters as the maximum similarity (or more commonly, minimum distance) between the vertices from the two clusters. *Complete linkage*, the opposite of single linkage, defines the cluster similarity as the minimum similarity (or maximum distance) between the vertices from the two clusters. *Average linkage* defines the cluster similarity as the average of all pairs of vertices from the two clusters. Average linkage is also known as UPGMA which stands for Unweighted Pair Group Method with Arithmetic Mean.

Divisive algorithms start with all vertices in one big cluster. Then, a cluster is spliced into two based on the members' similarity. The process continues until every vertex is in a cluster by itself. An example is CONCOR (12; 100) which stands for CONvergence of iterated CORrelations. CONCOR is based on the observation that, if one repeatedly computes the correlation matrix from a correlation matrix, the entries of the result converge to either $+1$ or -1 (except some rare circumstances). Moreover, the rows and columns of this matrix can be (simultaneously) permuted so that the entries are grouped together into four blocks such that the two blocks along the main diagonal are all 1 and the two off-diagonal blocks are all -1. The block structure can then be used to bisection the data points.

The hierarchical clustering method produces as an output a dendrogram, which formally is a *chain* of partitions $P_1 \preceq \cdots \preceq P_n$ in which each partition is *finer* than the next, denoted by $P_i \preceq P_{i+1}$. For two partitions P and Q of the same set, P is *finer* than than Q if and only if every part of P is contained in some part of Q. A chain of partitions expresses the nested structure from the finest partition to the coarsest partition. This is the point of using hierarchical clustering, to find an *a priori* nested structure. However, Fortunato and Castellino noted that, in most cases, the procedures yield dendrograms which are rather artificial (38). This raises two questions: 1) Does the method output a hierarchy which is completely different from the *true* hierarchy? 2) Does the method try to output a hierarchy even though the system is not hierarchical structured? The former question is a matter of algorithmic precision and correctness. The latter question rather raises the question of how to determine the existence of a hierarchical structure in a network.

Once we have an output from a hierarchical clustering, we usually find the level in the dendrogram which yields *the right* partition of the graph using quality functions in Section 3.2.2.

Hierarchical methods requires a similarity (or dissimilarity) function between the data points. They do not need the data points to belong to some ambient space. This sometimes makes them very slow in practice since the similarity function might be computationally hard.

3.2.4 Spectral Graph Clustering

Spectral clustering uses eigenvectors of matrices (of similarity between objects) to partition the data points. The idea is to change the basis of a matrix of similarity to the eigenspace of the matrix. Then, the eigenvectors are clustered using standard techniques such as k-means or a hierarchical method. This helps, for example, cope with the limitation of k-means that it produces a partition of space into *convex* Voronoi cells. For more information, see a tutorial on spectral clustering by Luxburg (66), a survey by Spielman and Teng (89). For spectral graph clustering, the adjacency matrix of the graph is transformed into a Laplacian matrix (73; 87). Then, the first few eigenvectors of the Laplacian are computed using, for example, Arnoldi iterative algorithm (3). These eigenvectors are then used as a basis and the k-means clustering is performed on the data points which are projected on the eigenvectors. The choice of graph Laplacians affects the results (38). An unnormalized Laplacian tends to favor clusters with inter-cluster density, while a normalized Laplacian tends to balance the high intra-cluster density and low inter-cluster density.

Spectral graph clustering has a nice interpretation when the input graph has an obvious community structure such as communities are well-separated into connected components or low-density cuts. However, it is not clear how this intuition generalizes to the more complex cases such as social networks observed in the real world.

3.2.5 Modularity Optimization

Modularity (see definition in Section 3.2.2) is a popular quality function among network researchers. There are a few algorithms which aim to optimize this function in particular. There is a betweenness-based divisive algorithm by Girvan and Newman (46). The idea is that edge-betweenness centrality can be used as an estimate of the likelihood that an edge is a bridge that connects two communities. The betweenness centrality (42) of an edge is the sum, over each vertex pair, of the fraction of the shortest paths between the vertex pair that passes through the edge. It measures how important an edge is in helping the vertices connecting to one another via shortest paths. Since communities are areas in a network with dense connections within and sparse connections in between, edges between communities tend to have relatively high betweenness centrality. The algorithm proceeds by first computing the betweenness centrality of edges. It then removes an edge with the highest betweenness centrality, one at a time, until the graph becomes disconnected for the first time. Then, the whole process is repeated on each connected component until there are no more edges.

Another algorithm is the agglomerative algorithm by Clauset, Newman, and Moore (21). The algorithm starts with all vertices in clusters by themselves. Then, it finds two clusters to merge such that the modularity value increases the most. It continues merging two clusters in this manner until all vertices are in the same cluster. The algorithm is quite fast, but it does not always produce high modularity value in practice, compared to the next algorithm.

Another algorithm is the Louvain algorithm by Blondel et al. (9). It starts with all vertices in clusters by themselves. Then, for each vertex, it tries to reassign the vertex to the cluster of its neighbor which increases the modularity value the most. If reassigning to a neighbor's cluster does not increase the modularity value, it stays with its current cluster. This process repeats until no vertices can find a better cluster to be reassigned to. The algorithm then contracts each cluster into a supervertex, keeping track of the number of multiple edges between the clusters as the edge weight. The self loops are also kept. The whole process is then repeated on this new graph until the contraction does not reduce the number of nodes. The Louvain algorithm is fast and produces good solutions in practice although neither its time complexity nor approximation guarantee is known. In this thesis, I will present a slightly modified algorithm based on the Louvain algorithm for a more general clustering problem, Correlation Clustering, which is the topic of the next section. I will also show how to cast the problem of detecting and tracking communities as a Correlation Clustering problem so as to use my algorithm and other existing approximation algorithms for Correlation Clustering to detect and track communities.

3.3 Correlation Clustering

Correlation Clustering is the problem of clustering a graph with real-valued edge weights. Positive edge weights signify the confidence levels that the two end points should be clustered together, while negative edge weights signify the confidence levels that the two end points should not be clustered together. There are two versions of Correlation Clustering problem. We can either maximize the sum of intra-cluster edge weights or minimize the sum of inter-cluster edge weights. The former is known as Maximizing Agreement and the latter is known as Minimizing Disagreement. The optimal solution of the two problems are the same, but approximation algorithms are different since one problem is maximization and the other is minimization. Maximizing Agreement is NP-hard (6) and APX-hard (19). Charikar et al. (19) used semidefinite program (SDP) relaxation and a hyperplane rounding scheme to give a 0.7664-approximation algorithm. Swamy (93), independently from Charikar et al., used the same SDP relaxation and two different hyperplane rounding schemes to give 0.75- and 0.7666-approximation algorithms. Elsner and Schudy (32) used both linear program (LP) and SDP relaxations to cluster newsgroups and observed that the SDP relaxation gave a tighter bound than the LP relaxation. Minimizing Disagreement is NP-hard and APX-hard (6) and has $O(\log n)$-approximation algorithms (19; 27; 33).

Approximation algorithms usually perform worse than heuristics on real-world data since they guarantee their performance in all cases, including rare cases. However, approximation algorithms provide not only a solution to the problem but also a bound on the optimal solution. Thus, we can think of an approximation algorithm as a way to compute a bound on the optimal solution, which can then be compared with the solutions of any other algorithms.

3.4 Tracking Communities over Time

The problem of tracking communities is motivated by a problem in behavioral ecology in studying animals which live in fission-fusion societies such as Grevy's zebra and the Asiatic wild ass (92). During a field drive, one animal group is sighted and observed, usually one at a time. The observers record the behaviors of the group members for a certain period of time. It is possible that we encounter two separate groups or more at the same time, but such occurrences are rare. Group membership in a fission-fusion society is fluid and always evolving. To understand the dynamics of a fission-fusion society, it is important to trace how groups persist and evolve over time. We can view groups as manifestations of perpetuating communities. Inversely, a community is a consistent string of groups seen on different days. The problem of tracking communities is about stringing different groups from the same day into communities which span multiple days.

The problem of tracking communities over time is very similar to Step 2 in the two-step process in Section 2.2.1. Strictly speaking, the problem of tracking communities is more *general* than Step 2 in the two-step process because Step 1 of the two-step process produces a sequence of partitions of the entire graph, while the problem of tracking communities requires as an input a sequence of collections of disjoint groups, which are not necessarily partitions of the entire graph. That is, some communities can be unobserved at some timesteps. This means, if we use the two-step process to detect communities in a dynamic social network, then Step 1 does not need to produce a partition of the entire graph. It may selectively produce only clusters on which it has high confidence. This relieves the algorithm from trying to put every vertex in some cluster, the result of which usually is a rather artificial collection of clusters.

There is a handful of work specifically on the problem of tracking communities over time. Berger-Wolf and Saia (8) proposed a framework which defines communities as independent local patterns similar to those in Section 3.2.1. There, a community (or, metagroup) is a sequence of groups which have sufficiently high similarity. The similarity between two groups is the number of common members normalized by the sizes of the two groups. Characteristics of communities are studied via community-based statistical measures such as number of all possible communities, their sizes and life spans. Also, they proposed an approach to study the survival of the communities via finding a critical set of groups whose removal leaves only short-lived communities.

Spiliopoulou et al. (90) proposed a framework, called MONIC, for tracking communities over time. The framework utilizes a similarity function of groups at different timesteps. The function takes into account the number of common members, the sizes of the groups, and the time decay between the groups. Then, two groups are strung together as being in the same community if their similarity is above a certain threshold. The framework not only strings groups into communities but also detects splitting and merging of communities by a separate sets of threshold parameters. Not only the framework has a lot of parameters to set, it also makes many assumptions about the dynamics of communities (which we are trying to infer). The resulting communities are rather *ad hoc*.

Berger-Wolf, Kempe and I proposed the first framework which rigorously formulates the problem of tracking communities as an optimization problem (95). Then, we presented several heuristic and approximation algorithms (7; 94; 95; 96). In Chapter 4, I will present an approximation algorithm for this problem, published in (94). Although the appealing aspect of this framework is the social costs model which has its roots in the social sciences view of group dynamics (75), the framework has a strong assumption that all timesteps must have the same length. In Chapter 5, I will introduce an improved framework which can handle data with timesteps of variable length. To the best of my knowledge, this is the only framework in the literature which can track communities in a sequence of partial partitions over timesteps of variable length.

3.5 Dynamic Community Detection

In this section, I review the work on community detection in dynamic networks. There are several algorithms which detect communities and simultaneously track them over time. In the area of database and data mining, Aggarwal and Yu (1) proposed an online method for detecting changes and data summarization for offline exploratory querying. Their focus is on online analytical processing (OLAP) applications rather than modeling and detecting communities. Another line of work is that of Falkowski et al. (35; 36) who, using a graph-theoretic approach, proposed a two-step methods for detecting communities and tracking their changes over time. They use Girvan-Newman algorithm (46) for both steps. Their focus is on detecting and visualization rather than modeling and inferring dynamic communities. Sun et al. (91) proposed GraphScope, a parameter-free clustering method for dynamic networks based on minimum description length principle. The method is an online heuristic algorithm which detects drastic changes in the input stream. The output is an encoded sequence of graph segments. The graph snapshots in each segment are encoded using the same grouping (clustering) of vertices. For each new arrival of graph snapshot, the method tries to incorporate the new graph snapshot into the current segment by regrouping, splitting and merging groups of vertices in the current segment so as to keep the encoding cost low. It then compares that encoding cost with the cost of starting a new segment to choose the better of the two. The graph snapshots in each segment can be considered as an i.i.d. sample and can be aggregated into a single static graph. In principle, the method does not detect dynamic communities *per se* since it does not allow gradual changes to culminate into a radical change in retrospect. It also does not try to see if there are groups which survive the drastic changes between two adjacent segments or not. Thus, it is essentially the Step 1 of the two-step process in Section 2.2.1. Palla et al. (74) studied clique percolation and evolution under a restrictive model which consider evolution only between adjacent timesteps. Chi et al. (20) proposed an evolutionary spectral clustering which regulates the temporal smoothness. The algorithm is based on k-means. Chakrabarti et al. (18) proposed a framework for clustering data points which gradually change over time based on k-means clustering and agglomerative clustering. Lin et al. (64) proposed FacetNet, a Bayesian framework with a regularization for smoothness of evolution. They gave an iterative algorithm which converges to a local optimum. Tong et al. (98) proposed Colibri, an approach to find low-rank approximations of the adjacency matrix of static and dynamic graphs. The low-rank approximations can be used to find communities, but it is still not clear how to do so or how well it would do since the claim is not supported by an argument or an experiment. The experiments there show only the performance in compression-related tasks. Asur et al. (4) proposed an event-based framework for tracking clusters over time. Xu et al. (103) proposed a hidden Markov model for a dynamic network and community evolution. They used the EM algorithm (28) to find the MAP estimation. Sarkar and Moore (82) modeled a dynamic network using using a Euclidean latent space and showed that the problem of learning the latent space from the data is tractable. Several other researchers used mixed membership stochastic block models (2; 44; 104). Following Newman's approach to formulating the network modularity (71) for static networks, Mucha et al. (68) proposed a quality function for dynamic communities based on a null model in terms of stability under Laplacian dynamics. Then, they optimize the function using the Louvain method (9) and post-process the results with Kernighan-Lin node swapping (54). Berger-Wolf and I extended our approach for tracking communities to simultaneously detect and track communities (97). There, we presented a heuristic algorithm and bounded how close the heuristic solutions were to the optimal solutions using approximation algorithms (93).

3.6 Other Related Work

In this section, I review Kleinberg's impossibility theorem for clustering and Theseus' Paradox. Kleinberg took an axiomatic approach to study clustering functions (55). He considers three desirable properties of a clustering function:

Scale Invariance The clustering function produces the same partition of data points after the distances between the data points are scaled by a positive constant.

Richness Any partition is a possible output from the clustering function (given the right input). That is, the image (or range) of the clustering function is the entire set of all possible partitions of the data points.

Consistency The clustering function is not sensitive to data *exaggeration*. For any set of data points and the corresponding partition outputted by the clustering, an *exaggeration*[1] is a combination of the following transformations: (a) decreasing the distance between data points in the same cluster, or (b) increasing the distance between data points in different clusters.

The impossibility theorem states that no clustering functions can simultaneously satisfy all of the above properties. However, the theorem applies only to the data clustering problem. It is reasonable to conjecture that a similar result holds true for the community detection problem as well. If so, this suggests that any parameter-free methods inevitably have some undesirable property. A similar problem arises when we consider the problem of tracking communities over time. The question about an object whose parts keep changing over time has been posted by ancient greeks since at least 75 A.C.E. It is known as Theseus' paradox.

According to Plutarch (76), "The ship wherein Theseus and the youth of Athens returned had thirty oars, and was preserved by the Athenians down even to the time of Demetrius Phalereus, for they took away the old planks as they decayed, putting in new and stronger timber in their place, insomuch that this ship became a standing example among the philosophers, for the logical question of things that grow; one side holding that the ship remained the same, and the other contending that it was not the same."

[1]This is known as Γ-transformation in the original paper.

The two sides differ on the perception of changes with respect to time. The former perceives the changes as happening slowly so the ship always retains its identity after each repair. The latter perceives the changes as happening quickly (time flies) so that the ship loses its identity bit by bit after each repair. In our context, the ship of Theseus is a community and the planks are the members of the community. Thus, without an *a priori* assumption about the rate of the process, any method is likely to incorrectly track communities which change over time. In other word, a method for tracking communities needs to know about how fast the underlying evolutionary process is. A method which takes the rate of change as a parameter will allow us to obtain either side of the Theseus' paradox as an answer.

With this in mind, we revisit GraphScope (91) which is a parameter-free method. Since it is parameter-free, it is restrictive in the scenario in Theseus' paradox. It is warranted to be able to find only one side of the paradox as it has no idea about the rate of changes in the data or the rate of changes that the user has in mind.

CHAPTER 4

TRACKING COMMUNITIES OVER UNIFORM TIMESTEPS

In this chapter, we consider a simple case in which all timesteps have the same length. For simplicity, we label the timesteps as $1, \ldots, m$ instead of $t_1, \ldots, t_k$. We are going to consider only the task of tracking communities. That is, the snapshots of communities are either known or already detected, and are given to us as $\mathcal{C} = \langle \mathcal{C}_1, \ldots, \mathcal{C}_m \rangle$. Recall that each $\mathcal{C}_t$ is a partition of only the *subset* of individuals whose social interactions at time t are known. In other words, it is possible that $\bigcup_{S \in \mathcal{C}_t} S \subsetneq V$. The parts of $\mathcal{C}_t$ are *community snapshots* or *groups*. I will use the two terms interchangeably.

4.1 Social Cost Model

First, we introduce a social cost model. Standard definitions of communities in social networks rely on various measures of cohesiveness (72; 100). The formulation focuses on the temporal change in group membership. In deriving an optimization formulation of community tracking, we make the following explicit assumptions about the behavior of individuals:

1. In each timestep, every group is a representative of some community. If two groups are present at the same time, there is a reason they are separate and , thus, represent distinct communities. One community should not be split into two groups or more (although temporary splitting can be accounted for by the mechanism of absence and visiting).

2. An individual is a member of exactly one community at any point in time. While the individual can change community affiliation over time, it is affiliated with only one community at any given moment. Notice that this does not preclude an individual from belonging to multiple communities over the course of the observation. It requires that the individual, in each timestep, determines "which hat to wear today."
3. An individual tends not to change its community affiliation very frequently. There is a cost of leaving the old community, missing the comfort of old friends, to join a new community, making new friends.
4. An individual is frequently present in the group representing the community with which it is affiliated. It rarely missies being with its community's group, and rarely is with other community's groups. That is, individuals within a community interact more than those in different communities.

We will use these properties to define an optimization problem, in which we assign costs to deviation from the behaviors posited above. These costs can be intuitively modeled as a graph coloring problem, albeit with a different objective function from the traditional graph coloring (30). We will use the properties to define an optimization problem, Next, we introduce a *cost graph* to help simplifying the calculation of social cost.

4.2 Cost Graph

Our graph $G_{\mathcal{C}} = (V_{\mathcal{C}}, E_{\mathcal{C}})$ has one *individual vertex* $v_{i,t}$ for each individual $i \in V$ and each timestep $t = 1, \ldots, m$. In addition, there is one *group vertex* $v_{g,t}$ for each group $g \in \mathcal{C}_t$. For each individual i and timestep $t \leq m-1$, there is an edge from $v_{i,t}$ to $v_{i,t+1}$. Finally, we have an edge between $v_{i,t}$ and $v_{g,t}$ whenever $i \in g$ at time t. Figure 2 shows an example of an interaction sequence on the left panel and the corresponding cost graph representation on the right panel, with a coloring showing the social costs in the setting according to the interpretation. In the left panel of the figure, each row corresponds to a timestep, with time going from top to bottom. Each rectangle represents an observed group and the circles within are the member individuals. The circles outside the rectangles are the unobserved individuals. Similarly, in the right panel, the squares are group vertices and the circles are individual vertices. Not all edges that incur cost are drawn for visibility. The coloring shows the costs of switching (α), absence (β_1), and visiting (β_2).

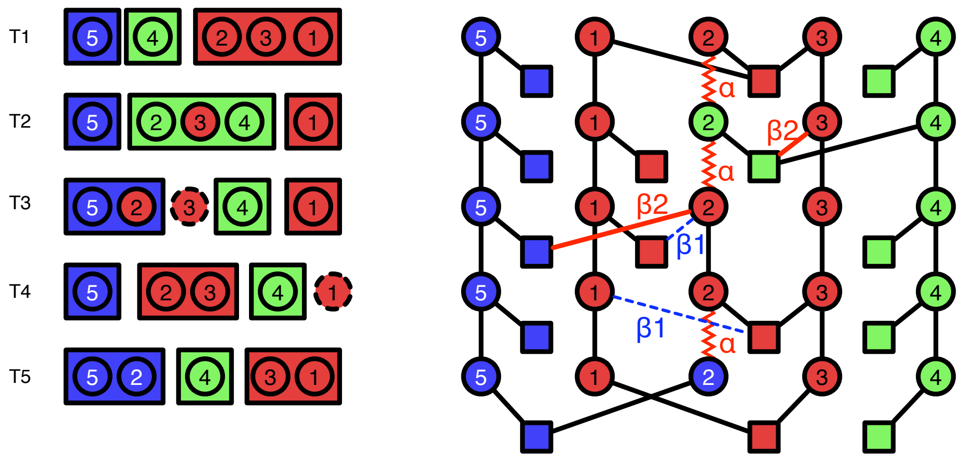


Figure 2: **Left:** An example of an interaction sequence. **Right:** The corresponding cost graph.

We define a community interpretation of a graph $G_{\mathcal{C}}$ as a vertex coloring $\chi : V_{\mathcal{C}} \to \mathbb{N}$ of $G_{\mathcal{C}}$. The color of an individual vertex $v_{i,t}$ represents the individual i's community affiliation at timestep t. Similarly, the color of a group vertex $v_{g,t}$ gives the community that g represents. Notice that this definition automatically ensures that each individual belongs to exactly one community in each timestep, and each group represents exactly one community. We call a community interpretation *valid* if and only if, for each timestep t, no two groups g and g' share the same color. This ensures Postulate 1, namely that only one group represents each community in each timestep.

To measure the quality of a community interpretation, we use costs to penalize violations of Postulates 3–4. There are three different types of costs, *switching*, *absence* and *visiting*. To allow for different relative importance of these properties, we parameterize the problem by non-negative parameters α, β_1 and β_2.

switching cost A switching cost of α is incurred whenever an individual changes its color. That is, if the individual edge does not have matching colors, $\chi(v_{i,t}) \neq \chi(v_{i,t+1})$.

absence cost An absence cost of β_1 is incurred if an individual vertex does not have an edge to the group of the same color (in the same timestep). That is, $\chi(v_{i,t}) = \chi(v_{g,t})$ but $i \notin g$.

visiting cost A visiting cost of β_2 is incurred if an individual vertex has an edge to the group of a different color from its own. That is, $i \in g$ but $\chi(v_{i,t}) \neq \chi(v_{g,t})$

If an individual is present at timestep t, but not in its group, then it incurs both absence and visiting costs. The first cost penalizes the individual for being absent from its current community, while the second cost penalizes the individual for being different from its current group.

In (95), we also have a *color cost* for each color that an individual uses. We decided to omit the color cost since it introduces, without any significant benefit, computational complexity when we want to fix the group coloring and optimally color the individual vertices (the dynamic programming algorithm is fixed-parameter tractable instead of polynomial time) . We can always compensate for the lack of color cost when breaking ties.

The optimization problem is then to find the valid community interpretation minimizing the total cost resulting from the switching costs, absence costs, and visiting costs. Once such a coloring χ has been found, we identify community c with the set of groups of coloring c. The community structure is the collection of all communities. Notice that we explicitly allow a community to change and evolve over time. once we have a community structure, we can derive from it an affiliation sequence for each individual i.

The Minimum Community Interpretation problem is, given an sequence of groups $\mathcal{C} = \langle \mathcal{C}_1, \ldots, \mathcal{C}_m \rangle$ and costs α, β_1, $\beta_2 \geq 0$, to find a minimum cost valid coloring of the corresponding cost graph.

This problem is NP-hard and APX-hard. See proof in (7; 95).

4.3 Approximation Algorithms

As we have noted, the Minimum Community Interpretation problem is NP-hard. In this section, we present two approximation algorithms: the first solves a special case and the other solves the general case of the problem. Both algorithms produce a solution within a constant factor of the optimal solution, regardless of the input size. Recall that, for a minimization problem in general, a ρ-approximation algorithm is a polynomial-time algorithm which, on all instances of the problem, produces a solution which is always at most ρ times of the optimal solution. The idea is to devise an algorithm which always prefers switching over absence or visiting. Note that, it is the ratios between the costs that affect the optimal solution, not the actual values of the costs. If the switching cost α is much higher than both β_1 and β_2 (e.g., death for betrayal) then individuals will be more likely to choose to be with one community and will never switch from it. It would rather incur absence or visit costs rather than one switch. On the other hand, if the absences and visits are much more expensive than a switch (e.g., a customer membership program) then individuals would rather switch their affiliation than incurring either an absence or visit. In this algorithm, we assume that we are in this latter case, so we would like find a community interpretation which incurs no absence or visit and minimizes the number of switches. We begin by considering a special case with the assumption that every individual is observed at all times. In other words, $\bigcup_{S \in \mathcal{C}_t} S = V$ for all t. Under this assumption, I will present an algorithm based on maximum weight bipartite matching and show that it is a ρ_1-approximation where $\rho_1 = \frac{\alpha}{\min\{\alpha, \beta_2/2\}} = \max\left\{1, \frac{2\alpha}{\beta_2}\right\}$. Later in Section 4.3.3, I will consider the general problem where some individuals might be unobserved at some point in time. I will present another algorithm which uses path cover instead of matching. I will show that it is a ρ_2-approximation where $\rho_2 = \frac{2\alpha}{\min\{\alpha, \beta_1, \beta_2/2\}} = \max\left\{2, \frac{2\alpha}{\beta_1}, \frac{4\alpha}{\beta_2}\right\}$. Note that the approximation guarantees of both algorithms depend only on the cost parameters and does not depend on the size of the input. Therefore, they are constant factor.

4.3.1 Group Graph

To design both algorithms, we use another auxiliary graph. The *group graph* of a sequence of community snapshots $\mathcal{C}$ is a directed acyclic graph $D = (V, E)$. The intuition is that the group graph represents the *flow* of individuals from one group to another. For a technical reason, we add dummy groups to represent groups that we do not see in the data at the beginning and the end of the timeline. For each (real) group g observed at time $t \geq 2$, let $g' \subseteq g$ be the individuals in g who are observed for the first time in g. If g' is not empty, then we add a dummy group g' at time 1. Similarly, for each (real) group g observed at time $t \leq T - 1$, let $g' \subseteq g$ now be the individuals in g who are observed for the last time in g. If g' is not empty, then we add a dummy group g' at time T. When all individuals are observed at all times, there are no dummy groups.

Next, we create a vertex $g \in V$ for every real or dummy group g. We create edges going out from each vertex $g \in V$ as follows. For each individual $i \in g$, we find the next group h containing i such that i does not appear in any other groups in between. If the edge (g, h) has not been created, we create it and set its label to be $\lambda(g, h) = \{i\}$. If there already is an edge (g, h), we update its label to $\lambda(g, h) \cup \{i\}$. Finally, we set the edge weight to be $w(g, h) = |\lambda(g, h)|$. Intuitively, the set $\lambda(g, h)$ contains the individuals that *flow* along the edge (g, h) and the edge weight $w(g, h)$ signifies the amount of the flow. From now on, we use the words groups and the vertices of the group graph interchangeably.

Figure 3 shows the group graph of the sequence of community snapshots from Figure 2. The edges are labeled with the individuals in $\lambda(g, h)$. There are no dummy vertices since every individual is observed at the first and the last times.

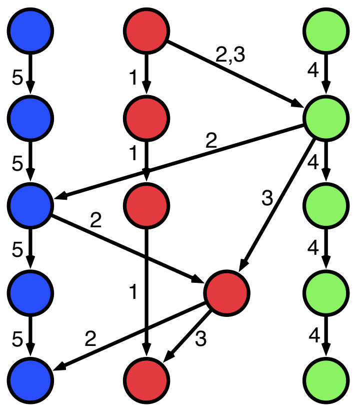


Figure 3: The group graph of the sequence of community snapshots from Figure 2.

A group graph D can be created in polynomial time in the size of the input by the following simple algorithm. The algorithm runs in $\Theta(Tk^2)$ time. This is tight since it is possible that $w(g_i, g_j) > 0$ for every $i < j$.

We note the similarity between the group graph and the meta-group graph by Berger-Wolf and Saia (8). In particular, a group graph is the transitive reduction (or Hasse diagram) of a meta-group graph.

Algorithm 1: CreateGroupGraph

```
Input: interaction sequence $\mathcal{C}$.
$g_1, \ldots, g_k \leftarrow$ the real and dummy groups in $\mathcal{C}$ in increasing order of time, breaking ties arbitrarily.
$V \leftarrow \{g_1, \ldots, g_k\}, E \leftarrow \emptyset$
for $i = 1, \ldots, k-1$ do
    $A \leftarrow g_i$
    for $j = i+1, \ldots, k$ do
        if $g_j \cap A \neq \emptyset$ then
            $E \leftarrow E \cup \{(g_i, g_j)\}$
            $w(g_i, g_j) \leftarrow |A \cap g_j|$
            $A \leftarrow A \setminus g_j$
        end
    end
end
return $D = (V, E)$
```

4.3.2 A Special Case: Complete Partitions

In this section, we assume that all individuals are observed at all timesteps. Algorithm 2 was first presented and analyzed in (94).

4.3.2.1 Algorithm Description

The heart of the above algorithm is finding a maximum weight matching on bipartite graphs, which can be done in $\Theta(n^3)$ time by the Kuhn-Munkres algorithm (also known as the Hungarian method) (56; 59; 69). Once we have matchings M_t^*'s, coloring the group vertices according to the connected components induced by the matching edges can be done in time linear in $|V|$ and $|E|$, and coloring the individual vertices can be done in time linear in n and m. Thus, the running time of the Algorithm MC is $\Theta(mn^3)$.

Algorithm 2: MatchingCommunities (MC)

```
Input: snapshots of communities $\mathcal{C}$.
$G = (V, E) \leftarrow$ undirected version of the group graph of $\mathcal{C}$ dropping edge orientations.
for time $t = 1, \ldots, m-1$ do
    $G_t \leftarrow$ induced subgraph of $G$ on $V_t \cup V_{t+1}$ where $V_t$ is the set of groups at time $t$.
    $M_t^* \leftarrow$ maximum weight matching on $G_t$.
end
$G' \leftarrow (V, \bigcup_{t=1}^{m-1} M_t^*)$        (subgraph of $G$ obtained by taking only the matching edges)
Color (the groups in) each connected component of $G'$ by a distinct color.
Color each individual at each timestep by the same color as the group in which it was
observed so that all groups are monochromatic.
```

4.3.2.2 Performance Analysis

We first give an upper bound on the cost of a coloring produced by Algorithm MC. Then, we give a lowerbound on the cost of an optimal coloring. For any set of edges M (matching or not), we write $w(M) = \sum_{e \in M} w(e)$ to denote its total weight.

Proposition 1. *Let $\mathcal{C}$ be an interaction sequence with all individuals present at all times. Let $M_1^*, \ldots, M_{m-1}^*$ be the matchings that produces a coloring χ of $\mathcal{C}$ in Algorithm* MC*. Then, the cost of χ is*

$$c(\chi) = \alpha \sum_{t=1}^{m-1} (n - w(M_t^*)).$$

Proof. We note that Algorithm MC colors the groups and individuals such that all groups are monochromatic. Thus, the resulting coloring χ does not have any absence or visit costs, and has only the switching costs. Consider each time $t \leq m-1$. The individuals who incur switching costs are those whose groups at times t and $t+1$ are not matched by M_t^*. Since there are $n - w(M_t^*)$ such individuals, the proposition holds.

□

Now, we give a lowerbound on the cost of an optimal coloring by giving a lower bound on the cost of any valid coloring of $\mathcal{C}$.

Lemma 2. *Let $\mathcal{C}$ be an interaction sequence with all individuals present at all times. Let χ be a valid coloring of $\mathcal{C}$ with cost $c(\chi)$. Let $G_1, \ldots, G_{m-1}$ be as in Algorithm* MC. *Let $\mu_1 = \min\left\{\alpha, \frac{\beta_2}{2}\right\}$ for convenience. Then, there exist matchings $M_1, \ldots, M_{m-1}$ on $G_1, \ldots, G_{m-1}$, respectively, such that*

$$c(\chi) \geq \mu_1 \sum_{t=1}^{m-1} (n - w(M_t)). \tag{4.1}$$

Proof. For any valid coloring χ, let $c_t(\chi)$ denote the cost that χ incurs between times t and $t+1$. Since there are no absence costs, this can include three possible types of costs: one for switching color between t and $t+1$ and two for visiting other communities at time t or $t+1$. We claim that, for a valid coloring χ and time $t \leq m-1$, there exists a matching M_t on G_t such that

$$c_t(\chi) \geq \mu_1 (n - w(M_t)). \tag{4.2}$$

Let M_t be the matching containing all the edges whose end points (groups) are colored the same in χ. This is well-defined since, for any color a of χ, there can be at most one group from each time colored a, since χ is a valid coloring. Thus, a vertex in V_t is matched to at most one vertex in V_{t+1} and vice versa.

Now, we consider each edge (g, h) of G_t unmatched by M_t. Since groups g and h have different colors in χ, each individual $i \in g \cap h$ must incur at least μ_1. This is so since if i switches colors, then i incurs α. Otherwise, i must visit g or h (or both) and, thus, incurs β_2 which is at least $\frac{\beta_2}{2}$. The reason for the half factor is that, for each visiting cost at time t, we might count it twice: once in c_{t-1} and once in c_t. Thus i incurs at least μ_1. Since there are $n - w(M_t)$ individuals whose groups are unmatched, inequality (Equation 4.2) holds as claimed. Since we count every cost no more than once, $c(\chi) \geq \sum_{i=1}^{m-1} c_t(\chi)$. Now, the lemma follows.

□

Now we show approximation factor of the Algorithm MC.

Theorem 3. *For convenience, let*

$$\mu_1 = \min\left\{\alpha, \frac{\beta_2}{2}\right\}, \qquad \rho_1 = \frac{\alpha}{\mu_1} = \max\left\{1, \frac{2\alpha}{\beta_2}\right\}.$$

Given an interaction sequence $\mathcal{C}$ with all individuals present at all times, Algorithm MC *produces, in polynomial time, a coloring with cost at most ρ_1 times of the optimal.*

Proof. Let χ^* be an optimal coloring of $\mathcal{C}$. Let $M = \{M_t\}$ be the set of matchings as in Lemma 2. For convenience, let $\bar{w}(M) = \sum_{t=1}^{m-1}(n - w(M_t))$ (and thus, by Lemma 2, $\mu_1 \bar{w}(M) \leq c(\chi^*)$).

Let χ be the coloring produced by Algorithm MC. Let $M^{\chi} = \{M_t^{\chi}\}$ be the set of matchings used in producing χ and $\bar{w}(M^{\chi}) = \sum_{t=1}^{m-1}(n - w(M_t^{\chi}))$. We observe that,

$$\mu_1 \cdot \bar{w}(M^{\chi}) \leq \mu_1 \cdot \bar{w}(M) \leq c(\chi^*) \leq c(\chi) = \alpha \cdot \bar{w}(M).$$

The first inequality holds since M_t^{χ} are of maximum weight, the second follows from Lemma 2, and the third follows from the optimality of χ^*. The last equality holds by Proposition 1. Since, $\dfrac{c(\chi^*)}{c(\chi)} \leq \dfrac{\alpha}{\mu_1} = \rho_1$, the theorem follows. $\square$

If $\alpha \geq \frac{\beta_2}{2}$, then $\rho_1 = 1$ and the algorithm always produces an optimal coloring. Note, that it is straightforward to convert Algorithm MC into a streaming algorithm, since we only need to store the groups at the latest time and their color while using space that is constant in n and m.

4.3.3 <u>The General Case</u>

In the previous section we made an assumption that all individuals are observed at all timesteps. In this section, we consider the general case of the problem in which some individuals might be unobserved at times. We present a ρ_2-approximation algorithm for the general case and analyze its performance guarantee. The algorithm with make uses of a polynomial-time problem Path Cover which is defined as follows.

4.3.3.1 <u>Path Cover Problem</u>

Given a directed graph $D = (V, E)$, a directed path is a sequence of distinct vertices $P = v_1, \ldots, v_k$ such that every (v_i, v_{i+1}) is an edge of D. Two directed paths P_1 and P_2 are vertex-disjoint if they share no vertices. A path cover $\mathcal{P}$ on D is a set of pairwise vertex-disjoint paths (30) in which every vertex lies on (is covered by) a path in $\mathcal{P}$. The Minimum Path Cover problem is to find a path cover with the minimum number of paths. The decision version of the problem on general graphs is NP-complete (40). However, on directed acyclic graphs (DAGs), the problem can be solved in polynomial time via a reduction to the matching problem in bipartite graphs (22).

Equivalently, we can maximize the number of edges in $\mathcal{P}$. For a directed graph $D = (V, E)$ with edge weights $w : E \to \mathbb{Z}^+$, the objective of the path cover is to maximize the total weight of the edges in the path cover, $w(\mathcal{P}) = \sum_{P \in \mathcal{P}} \sum_{e \in P} w(e)$. It is straightforward to extend the aforementioned reduction to the weighted case.

It is more convenient to use the minimization version of the weighted path cover problem, since we define our problem of identifying communities as a minimization problem. We describe the minimum weight path cover problem as follows. Let $W = \sum_{e \in E} w(e)$ and $\bar{w}(\mathcal{P}) = W - w(\mathcal{P})$. In other words, $\bar{w}(\mathcal{P})$ is the total weight of the uncovered edges. By the definition of $\bar{w}$, maximizing $w(\mathcal{P})$ is equivalent to minimizing $\bar{w}(\mathcal{P})$ at the optimal point, so in the next section, we will minimize $\bar{w}(\mathcal{P})$ instead.

4.3.3.2 Algorithm Description

Now we describe the approximation algorithm in Algorithm 3. We reserve one color ϵ not to be assigned to any group. Intuitively, individuals with color ϵ are considered to be in the "missing" community at the time. We refer to ϵ as the color of absence.

Algorithm 3: PathCoverCommunities (PCC)

```
Input: An interaction sequence $\mathcal{C}$.
$D = (V, E) \leftarrow$ CreateGroupGraph($\mathcal{C}$)
$\mathcal{P}^* \leftarrow$ Find minimum weight path cover on $D$ using Kuhn-Munkres.
Color (real groups in) each path $P \in \mathcal{P}^*$ by a distinct color.
for edges $(g, h) \in E$ do
    Color the individuals in $\lambda(g, h)$ from the time they were in $g$ until they were in $h$ by
    the same color as $g$.
end
Color the remaining vertices by $\epsilon$.
```

First, we observe that the algorithm runs in polynomial time. Furthermore, we observe that a coloring χ by Algorithm PCC is valid. If χ is not valid, then there exist two groups g, h at some timestep t to which χ assign the same color. By construction, g and h must lie on the same path P for some $P \in \mathcal{P}$. Since every edge in D is oriented from some time t to another time $t' > t$, any path in D lists its group vertices in strictly increasing order of time. Thus, g and h are at different time steps, which is a contradiction.

4.3.3.3 Performance Analysis

We first show an upper bound on the cost of the solution, a lowerbound on the optimal solution, then the approximation factor.

Proposition 4. *Let $\mathcal{P}^*$ be a minimum weight path cover on D with weight $\bar{w}(\mathcal{P}^*)$. Let χ be the coloring produced from $\mathcal{P}^*$ by Algorithm* PCC *with cost $c(\chi)$. Then,*

$$c(\chi) \leq 2\alpha \cdot \bar{w}(\mathcal{P}^*).$$

Proof. By construction, χ incurs only α costs. We consider each edge (g, h) of D uncovered by $\mathcal{P}^*$ and each individual $i \in \lambda(g, h)$:

- If g, h are consecutive in time, then i incurs a cost α.
- If g, h are not consecutive in time, then i incurs 2α, for switching to the color of absence ϵ and switching back.

Thus, each uncovered edge (g, h) incurs a cost of at most $2\alpha \cdot \bar{w}(g, h)$, resulting in the total of at most $2\alpha \cdot \bar{w}(\mathcal{P}^*)$.

$\square$

Before showing the lowerbound, we introduce some notation. Let χ be a fixed coloring. For each edge (g, h) of D, we associate with it the sum of the following costs of χ:

- α for every individual $i \in \lambda(g, h)$ that switches color between the times i was in g and h.
- β_1 for every individual $i \in \lambda(g, h)$ that is absent from group h' after the time i was in g and before i was in h.
- $\frac{\beta_2}{2}$ for every individual $i \in \lambda(g, h)$ that visits group g.
- $\frac{\beta_2}{2}$ for every individual $i \in \lambda(g, h)$ that visits group h.

Note that we omit the cost for individual $i \in \lambda(g, h)$ being absent from some group h' at the same time as from g or h. Although we could add $\frac{\beta_1}{2}$, this suffices for our purposes.

For any subgraph $D' \subseteq D$, we define $c(\chi|D')$ to be the sum of the costs of χ associated with the edges of D' as described above. As noted above, since some costs are omitted,

$$c(\chi) \geq c(\chi|D). \tag{4.3}$$

This notation is useful when we decompose group graph D into pairwise edge-disjoint subgraphs $D_1, \ldots, D_k$ such that $D = \cup_j D_j$. It is easy to see that,

$$c(\chi|D) \geq \sum_j c(\chi|D_j). \tag{4.4}$$

We will write $c(\chi|E)$ to denote $c(\chi|D)$ where E is the edge set of D.

We also use a similar notation for path cover. For any path cover $\mathcal{P}$ on group graph D, we write $\bar{w}(\mathcal{P}|D)$ to emphasize that it is the total weight of the edges of D uncovered by $\mathcal{P}$. Let the group graph D be decomposed into vertex-disjoint subgraphs $D_1, \ldots, D_k$, where $D = \cup_j D_j$. Let $C = \{e \in E : \forall j \; e \notin D_j)\}$ be the set of edges that are not in any of the parts $D_1, \ldots, D_k$. Let $w(C) = \sum_{e \in C} w(e)$ be the total weight of C. Let $\mathcal{P}_j$ be any path cover on each D_j with weight $\bar{w}(\mathcal{P}_j|D_j)$. Then, $\mathcal{P} = \cup_j \mathcal{P}_j$ is a path cover on D with weight,

$$\bar{w}(\mathcal{P}|D) = \sum_j \bar{w}(\mathcal{P}_j|D_j) + w(C). \tag{4.5}$$

Having defined the necessary notation, we give a lowerbound on the cost of any valid coloring, including the optimal solution.

Lemma 5. *Let $D = (V, E)$ be the group graph of an interaction sequence $\mathcal{C}$. Let χ be a valid coloring of $\mathcal{C}$. Then, there exists a path cover $\mathcal{P}$ on D such that*

$$c(\chi|D) \geq \mu_2 \cdot \bar{w}(\mathcal{P}|D),$$

where $\mu_2 = min\left\{\alpha, \beta_1, \frac{\beta_2}{2}\right\}$.

Proof. We show this by induction on the number of edges $|E|$. Consider the following 3 cases.

Case 1: D is not (weakly) connected. Then, D can be decomposed into connected components $D_1, \ldots, D_k$ for some $k \geq 2$. Since $|E(D_j)| < |E|$ for all j, there exists a path cover $\mathcal{P}_j$ on each D_j such that $c(\chi|D_j) \geq \mu_2 \cdot \bar{w}(\mathcal{P}_j|D_j)$ holds, by induction. By inequality (Equation 4.4) and induction,

$$c(\chi|D) \geq \sum_j c(\chi|D_j) \geq \mu_2 \sum_j \bar{w}(P_j|D_j).$$

Let $\mathcal{P} = \cup_j \mathcal{P}_j$ be the path cover of D. Since there are no edges going between D_j's, equation (Equation 4.5) amounts to $\bar{w}(\mathcal{P}|D) = \sum_j \bar{w}(\mathcal{P}_j|D_j)$ and the desired result follows,

$$c(\chi|D) \geq \mu_2 \sum_j \bar{w}(P_j|D_j) \geq \mu_2 \cdot \bar{w}(\mathcal{P}|D).$$

Case 2: D is (weakly) connected and not monochromatic. Consider the partition of V induced by the coloring χ. That is, each part in the partition is the set of vertices of each color in χ. If D is not monochromatic, then χ partitions V into parts $V_1, \ldots, V_\ell$, for some

$\ell \geq 2$, where V_j is the set of groups colored j. Let $D_j = D[V_j]$ be the induced subgraph of D corresponding to the monochromatic part j.

Let C be the set of edges that are not in any of the subgraphs D_j. We observe that, for each such edge $(g, h) \in C$, the groups g and h have different color in χ since they belong to different parts. By a similar argument as in the proof of Lemma 2, each individual $i \in \lambda(g, h)$ must incur the cost of at least $\mu_1 \geq \mu_2$. Thus, $c(\chi|C) \geq \mu_2 \cdot w(C)$.

Since $|E(D_j)| < |D|$ for all j, there exists a path cover $\mathcal{P}_j$ on each D_j such that $c(\chi|D_j) \geq \mu_2 \cdot \bar{w}(\mathcal{P}_j|D_j)$ holds, by induction. Let $\mathcal{P} = \cup_j \mathcal{P}_j$ be the path cover on D. By inequality (Equation 4.4), induction, and the above observation about $c(\chi|C)$, we obtain,

$$\begin{aligned} c(\chi|D) &\geq \sum_j c(\chi|D_j) + c(\chi|C) \\ &\geq \mu_2 \sum_j \bar{w}(P_j|D_j) + \mu_2 \cdot w(C). \end{aligned}$$

Now the desired result follows from equation (Equation 4.5),

$$c(\chi|D) \geq \mu_2 \sum_j \bar{w}(P_j|D_j) + \mu_2 \cdot w(C) = \mu_2 \cdot \bar{w}(P).$$

Case 3: D is (weakly) connected and monochromatic. Suppose for the moment that D has some edges. Let (g, h) be one of the shortest edges in D in the sense that g and h were observed closest in time. Let $h_1, \ldots, h_d$ be the out-neighbors of g in increasing order of time, and $g_1, \ldots, g_e$ be the in-neighbors of h in decreasing order of time. Note that $g = g_1$ and $h = h_1$.

If g has more than one out-neighbor, we consider each individual $i \in \cup_{j\geq 2}\lambda(g,h_j)$. If i switches color at any point between the time that group g was observed and the time that group h_j was observed, then i incurs α, which we map to (g,h_j). Otherwise, either i visits both groups g and h_j, or i is absent from group h (since g,h,h_j have the same color). In the former case, i incurs $2\beta_2$, half of which we map to (g,h_j). In the latter case, i incurs β_1, which we map to (g,h_j). In all cases, each $i \in \lambda(g,h_j)$ maps the cost of at least $\min\{\alpha,\beta_1,\beta_2\} \geq \mu_2$ to (g,h_j). If h has more than one in-neighbor, a similar argument also works for individuals $i \in \cup_{j\geq 2}\lambda(g_j,h)$.

Let $C = \{(g,h_j)\} \cup \big\{(g_{j'},h)\big\}$ be the set of edges going out from g and coming into h. The above argument gives a lower bound $c(\chi|C) \geq \mu_2 \cdot w(C \setminus \{(g,h)\}) \geq \mu_2 \cdot w(C)$. We remove the edges in C from D. Let $D' = (V, E \setminus C)$ be the resulting graph. Since $(g,h) \in C$, $|E(D')| < |E|$ and exists a path cover $\mathcal{P}'$ on D' such that $c(\chi|D') \geq \mu_2 \cdot \bar{w}(\mathcal{P}'|D')$, by induction. We join the path in $\mathcal{P}'$ ending at g with the path in $\mathcal{P}'$ starting at h. The result is a path cover $\mathcal{P}$ on D. Using inequality (Equation 4.4), induction, and the above lowerbound on $c(\chi|C)$, we obtain,

$$c(\chi|D) \geq c(\chi|D') + c(\chi|C) \geq \mu_2 \cdot \bar{w}(P'|D') + \mu_2 \cdot w(C).$$

Now the desired result follows from equation (Equation 4.5),

$$c(\chi|D) \geq \mu_2 \cdot \bar{w}(P'|D') + \mu_2 \cdot w(C) = \mu_2 \cdot \bar{w}(P).$$

It remains to show Case 3 for D without any edges. Since D is connected, D is a vertex. Let the path cover $\mathcal{P}$ be the entire graph D. Thus, $c(\chi|D) \geq \bar{w}(\mathcal{P}|D) = 0$ trivially holds.

Therefore, the lemma follows. $\square$

Now we give the performance guarantee of Algorithm PCC.

Theorem 6. *Algorithm* PCC *is a ρ_2-approximation where*

$$\rho_2 = \frac{2\alpha}{\mu_2} = \max\left\{2, \frac{2\alpha}{\beta_1}, \frac{4\alpha}{\beta_2}\right\}, \quad \mu_2 = \min\left\{\alpha, \beta_1, \frac{\beta_2}{2}\right\}.$$

Proof. The theorem follows from the lowerbound from Lemma 5, application of inequality (Equation 4.3), and the upper bound from Proposition 4. The argument is similar to the proof of Theorem 3. $\square$

4.4 Post-Processing Steps

In this section, I present two improvements for practical purposes. The first way is to recolor the individual vertices optimally while keeping the color of group vertices fixed. This will involve dynamic programming with time complexity that depends on the number of colors used by the group vertices. To further eliminate unnecessary colors, we will perform another processing step based on iterative path-cover heuristic.

4.4.1 Dynamic Programming Post-Processing

We first observe one weakness of the Algorithm PCC. It produces the same coloring regardless of the values of α, β_1, β_2. In particular, if we increase α keeping β_1 and β_2 fixed, there is a point at which, in an optimal coloring, no individual switches color. From this point on, increasing α does not change the cost of the optimal coloring. However, the cost by Algorithm PCC increases linearly in α. Nevertheless, there is a simple way to alleviate the problem. In particular, we use Algorithm PCC to color the group vertices, then use the following dynamic programming algorithm to color the individual vertices.

For each individual $x \in V$ and time $t = 1, \ldots, m$, the minimum cost of coloring x at time t with a color $c \in C$ can be written recursively as:

$$\mathrm{MinCost}(x,t,c) = \mathrm{Cost}_{\mathrm{ab+vi}}(x,t,c) + \begin{cases} \min\limits_{c' \in \mathcal{C}} [\mathrm{Cost}_{\mathrm{sw}}(c,c') + \mathrm{MinCost}(x,t-1,c')] & \text{if } t > 1, \\ 0 & \text{if } t = 1, \end{cases}$$

where $\mathrm{Cost}_{\mathrm{ab+vi}}(x,t,c)$ is the sum of the absence and visiting costs if individual x is at timestep t has color c, and $\mathrm{Cost}_{\mathrm{sw}}(c,c') = c_{\mathrm{sw}}$ if $c \neq c'$ and 0 otherwise. This recursion can be computed using dynamic programming. By keeping track of the minimizer c' of each $\mathrm{MinCost}(x,t,c)$, we can trace back the optimal coloring of each individual x starting at the color c that minimizes $\mathrm{MinCost}(x,t,c)$ at time $t = m$. Another thing we need to consider is the choice of color set C for varying c and c'. One option is to use the colors used by the groups plus one extra color, say 0. The extra color allows individuals to be colored differently from any group at any point in time. The time complexity of the dynamic programming algorithm is $\Theta(nm|C|)$. This is a stripped down version of the one in (95; 96) which involves a color cost γ. The removal of the color cost allows us to simplify the dynamic programming algorithm further and remove the exponential factor from the running time.

4.4.2 Iterative Path-Cover Heuristic

One problem arises when we apply the above dynamic programming algorithm. In particular, Algorithm PCC may produce a coloring which uses a large number of colors. This increases the complexity of the dynamic programming algorithm above. When the timeline is long, this problem is mostly caused by short paths in a cover. Recall that the vertices in the group graph are groups and they have timesteps. We say that two paths are *time-disjoint* if one path ends (in time) before the other path starts. We observe that we can recolor two time-disjoint paths with the same color for free and the resulting coloring will still be proper. This can be casted as another path cover problem as follows. We start each iteration with a path cover $\mathcal{P}$ which is initially set to be the trivial path cover in which each path contains a single vertex. We create another group graph D' where each vertex corresponds to the union of groups on each path in $\mathcal{P}$. Then, we create an edge in a similar manner as in Algorithm 1. Then, we find a maximum weight path cover P' on D'. We set $P = P'$, and iterate with the path contraction until the constructed group graph D' contains no edges with positive weights. Finally, we produce a coloring in the same way as before. Each path in the path cover is assigned a unique color which is used to color the (original) groups on the path. Since the recoloring of time-disjoint paths does not incur any new cost, the result is still a ρ_2-approximation algorithm.

4.5 Experiments

In previous section, I have showed that, theoretically, algorithm IPCC is efficient and guarantees good solutions. In this section, I will empirically evaluate the performance of algorithm IPCC on datasets. To find a path cover, I will reduce the path cover instance to an integer program. The constraint matrix of this integer program can be shown to be totally unimodular.[1] Thus, the linear program relaxation has an integral optimal solution. From there, I use a commercial package solver CPLEX to solve the LP relaxation. After the IPCC algorithm produces a coloring of the groups, we run the dynamic programming algorithm to color the individuals. We ran this on several datasets as listed in Table I. See Appendix A for the description of the datasets.

Dataset	Individuals	Timesteps	Groups
Grevy's zebra	27	44	75
Haggle-41	41	418	2131
Haggle-264	264	425	1411
Onagers	29	82	308
Plains zebra	2510	1268	7907
Reality Mining	96	1577	3958
Southern Women	18	14	14

TABLE I: Datasets and their statistics in Chapter 4.

4.5.1 Experimental Setup

On each dataset we find the optimal solution either by exhaustive search or by estimating the lowerbound on the optimal solution using the interval lowerbounding technique (see Appendix B). We compare the solution by the IPCC algorithm to this benchmark both nu-

[1] Path cover is equivalent to bipartite matching. The sufficient condition is trivial. See (22) for the necessary condition. The constraint matrix of bipartite matching is known to be totally unimodular.

merically and, where possible, structurally. For each cost setting, we compare the theoretical approximation ratio of ρ_2 to the actual ratio of the cost of the algorithm to the optimal solution (shown in parentheses in all the results tables). Note that the coloring cost of the IPCC algorithm does not change when β_2 changes since the algorithm incurs only α costs. Table II shows the performance results of PCC and IPCC algorithms.

4.5.2 Coloring Results

In this section, we show the actual community structure (coloring) on the Southern Women dataset as produced by our algorithms. Naturally, we start with the cost setting $(\alpha, \beta_1, \beta_2) = (1, 1, 1)$. Figure 4, Figure 5, and Figure 6 show the colorings produced by the PCC algorithm, the IPCC algorithm, and the exhaustive search, respectively. The details of the exhaustive search is given in (95; 96) and Appendix B. In the figures, time goes from the top to the bottom. The rectangles represent groups or community snapshots. The circles in a rectangle represent the individual members of the group. If the color of a circle is different from that of the group, then its circle has a red border indicating that the individual is paying a visiting cost. Small diamond shapes represent individuals who are not in any groups at the time. Red rectangles with white background surrounding a diamond indicate an absent cost. Lines connect an individual to itself in the next timestep. Red lines correspond to a switching cost being paid. Note, that the overall structure of dynamic communities remains very similar despite the difference in the costs of the colorings. Thus, qualitatively the algorithm infers communities that are close to the optimal.

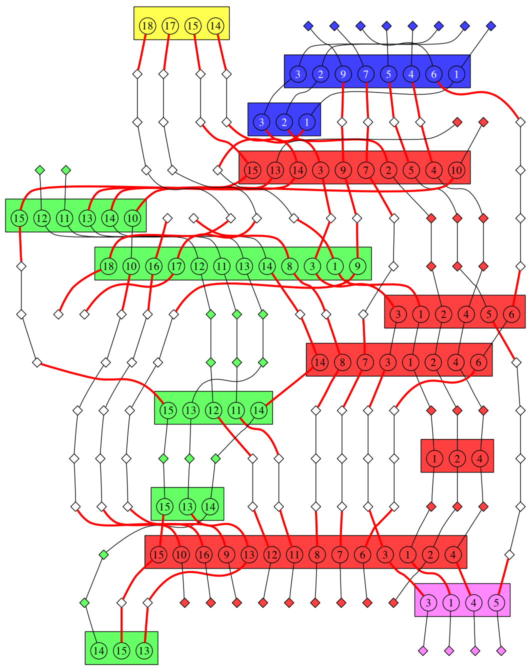


Figure 4: The dynamic communities in Southern Women of total cost 74 produced by algorithm PCC alone with cost setting $\alpha = \beta_1 = \beta_2 = 1$

4.6 Summary

In this chapter, we formulate the MINIMUM COMMUNITY INTERPRETATION problem and present two approximation algorithms: the first is for the special case when every individual is in a group at all timesteps and the other is for the general case of the problem. We analyze the performance guarantee of the algorithms. The proposed algorithms are up-to-now still the only rigorous computational solutions to the MINIMUM COMMUNITY INTERPRETATION problem in dynamic networks with provable performance guarantees. While the theoretical

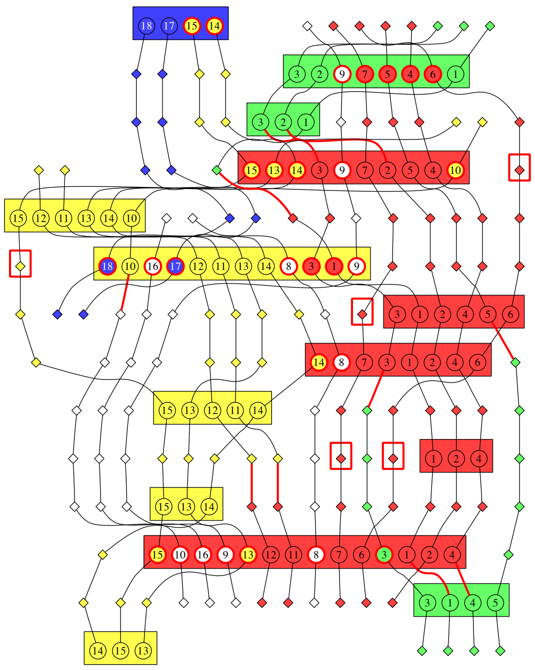


Figure 5: The dynamic communities in Southern Women of total cost 43 produced by algorithm IPCC with the dynamic programming post-processing with cost setting $\alpha = \beta_1 = \beta_2 = 1$

analysis guarantees a constant factor approximation, in practice the best implementation of our algorithm finds solutions very close to the optimal solution numerically, and even closer structurally. Moreover, both algorithms completed in less than 2 minutes on networks of several thousand individuals. Thus, our algorithms can be used in practice to infer communities in large dynamic social networks.

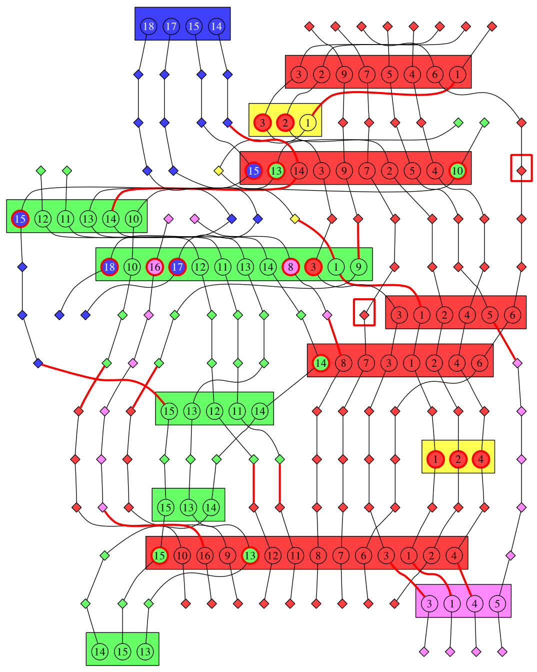


Figure 6: The optimal dynamic communities in Southern Women of total cost 36 produced by the exhaustive search with cost setting $\alpha = \beta_1 = \beta_2 = 1$

	α, β_1, β_2	Bound or optimum	ρ_2	PCC	IPCC+DP
Grevy's	1,1,3	≥ 56	4	106 (1.89)	76 (1.39)
	1,1,2	≥ 56	4	106 (1.89)	76 (1.36)
	1,1,1	≥ 51	4	106 (2.08)	69 (1.35)
	2,1,1	≥ 59	8	212 (3.59)	98 (1.66)
	3,1,1	≥ 59	12	318 (5.39)	109 (1.85)
Hg-264	1,1,3	≥ 2030	4	3347 (1.65)	2442 (1.20)
	1,1,2	≥ 2030	4	3347 (1.65)	2442 (1.20)
	1,1,1	≥ 1015	4	3347 (3.30)	2194 (2.16)
	2,1,1	≥ 1015	8	6694 (6.60)	3111 (3.06)
	3,1,1	≥ 1015	12	10041 (9.89)	3700 (3.65)
Hg-41	1,1,3	≥ 1013.0	4	1547 (1.53)	1218 (1.20)
	1,1,2	≥ 1013.0	4	1547 (1.53)	1218 (1.20)
	1,1,1	≥ 506.5	4	1547 (3.05)	1158 (2.29)
	2,1,1	≥ 506.5	8	3094 (6.11)	1700 (3.36)
	3,1,1	≥ 506.5	12	4641 (9.16)	2101 (4.15)
Onagers	1,1,3	≥ 125	4	282 (2.26)	192 (1.54)
	1,1,2	≥ 125	4	282 (2.26)	192 (1.54)
	1,1,1	≥ 125	4	282 (2.26)	175 (1.43)
	2,1,1	≥ 121	8	564 (4.66)	255 (2.11)
	3,1,1	≥ 121	12	846 (6.99)	298 (2.46)
Plains	1,1,3	≥ 44925.0	4	85630 (1.91)	45785 (1.02)
	1,1,2	≥ 44925.0	4	85630 (1.91)	45785 (1.02)
	1,1,1	≥ 22462.5	4	85630 (3.81)	45785 (2.04)
	2,1,1	≥ 22462.5	8	171260 (7.62)	55578 (2.47)
	3,1,1	≥ 22462.5	12	256890 (11.44)	58928 (2.62)
Reality	1,1,3	≥ 12498.0	4	21374 (1.71)	17066 (1.36)
	1,1,2	≥ 12489.0	4	21374 (1.71)	17066 (1.36)
	1,1,1	≥ 6244.5	4	21374 (3.42)	14943 (2.39)
	2,1,1	≥ 6244.5	8	42748 (6.85)	19792 (3.17)
	3,1,1	≥ 6244.5	12	64149 (10.27)	22147 (3.55)
SW	1,1,3	48	4	78 (1.62)	50 (1.04)
	1,1,2	48	4	78 (1.62)	50 (1.04)
	1,1,1	36	4	78 (2.17)	43 (1.19)
	2,1,1	41	8	156 (3.80)	50 (1.22)
	3,1,1	42	12	234 (5.57)	53 (1.26)

TABLE II: Performance of PCC and IPCC with the dynamic programming post-processing.

CHAPTER 5

TRACKING COMMUNITIES: THE GENERAL CASE

In the last chapter, we consider a special case of the problem of tracking communities when all timesteps have uniform length. In this chapter, we consider the general case of the problem in which timesteps have variable length. We will formulate a model using the intuition of animal sightings, but the model will be applicable to other kind of data as well. A *sighting* is a group of individuals which are observed to be interacting with each other for a duration of time. Sightings are imperfect but observable manifestations of the hidden communities. The model is conceptually a hidden Markov model. Informally, the hidden community structure determines with *which* community an individual is affiliated at a given point in time, as well as *when* an individual switches its community affiliation. The model we present below generates a community structure together with an observable sequence of sightings. Then, the problem of tracking communities becomes the reverse problem of inferring the hidden community structure from a given sequence of sightings.

5.1 Definitions and Notations

Let $\{x_1, \ldots, x_n\}$ denote a population of n individuals. We observe this population in the form of sightings $S_1, \ldots, S_k$. Each sighting S_i is a subset of the population which is observed to be socially interacting with one another at time $T_i > 0$ for a time duration $d_i > 0$. Ideally, S_i is the manifestation of some hidden community over a short period of time d_i. The assumption that all d_i's are small will be come important later on when we use Poisson processes. A sequence of sightings is the collection of all the sightings with their metadata: $\mathcal{S} = \langle (S_1, T_1, d_1), \ldots, (S_k, T_k, d_k) \rangle$. We assume that T_i's are in chronological order, that is, $T_1 \leq T_2 \leq \cdots \leq T_k$. Furthermore, we assume that an individual belongs to at most one sighting at any given time. In other words, if $S_i \cap S_j \neq \emptyset$, then $[T_i, T_i + d_i) \cap [T_j, T_j + d_j) = \emptyset$. Lastly, we assume that the observation is done in *synchronized* time steps, that is, any two consecutive sightings S_i and S_{i+1} are either in the same timestep ($T_i = T_{i+1} \wedge d_i = d_{i+1}$) or in disjoint time steps ($T_i + d_i \leq T_{i+1}$). It is easy to see that any sequence of sightings which satisfies the first two assumptions can be made to satisfy the third assumption by appropriately partitioning the sightings over time (see Figure 7 for an example).

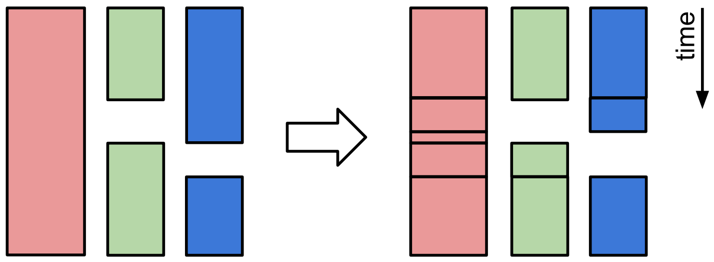


Figure 7: How to synchronize sightings.

5.2 Sighting Graph

Given a sequence of sightings $\mathcal{S}$, we create the *sighting graph* of $\mathcal{S}$ as follows. Let $t_1 < \cdots < t_m$ be the *unique* timestamps at which sightings $S_1, \ldots, S_k$ happen ($m \leq k$). Note that t_i's denote the unique timesteps and T_i denotes the timestep at which sighting S_i is observed. We create a vertex s_i for each sighting S_i and create a vertex $v_{x,t}$ for each individual $x = x_1, \ldots, x_n$ and each time $t \in \{t_1, \ldots, t_m\}$. We will leave the construction of the edges until after we explain the probabilistic model. Figure 8 shows an example of a sequence of sightings and its corresponding sighting graph when it is complete. In the left panel, the circles connected by line segments are individuals in different sightings. Rectangles are sightings whose height represent the duration of a sighting. Note that some individuals may not be sighted in some timesteps. In the right panel, the circles are individual vertices. The rectangles are sighting vertices. The solid lines between two circles connect individual vertices in adjacent timesteps. The individual and sighting vertices in each timestep form a complete bipartite subgraph. The solid lines between a circle and a rectangle connect an individual vertex to the vertex of of its sighting. A dotted line connects an individual vertex to the vertices of the sightings that the individual does not belong to.

In the same spirit as in Chapter 4, a community interpretation of a given $\mathcal{S}$ consists of two parts: a community structure and a sighting interpretation. A community structure is a coloring $\chi_C : \{v_{x,t}\} \to \mathbb{N}$ of the individual vertices in the corresponding sighting graph. Colors correspond to communities, so we will use the two terms interchangeably. We say that *an individual x switches its community affiliation between time $T_i + d_i$ and T_{i+1}* if $\chi_C(v_{x,T_i}) \neq \chi_C(v_{x,T_{i-1}})$.

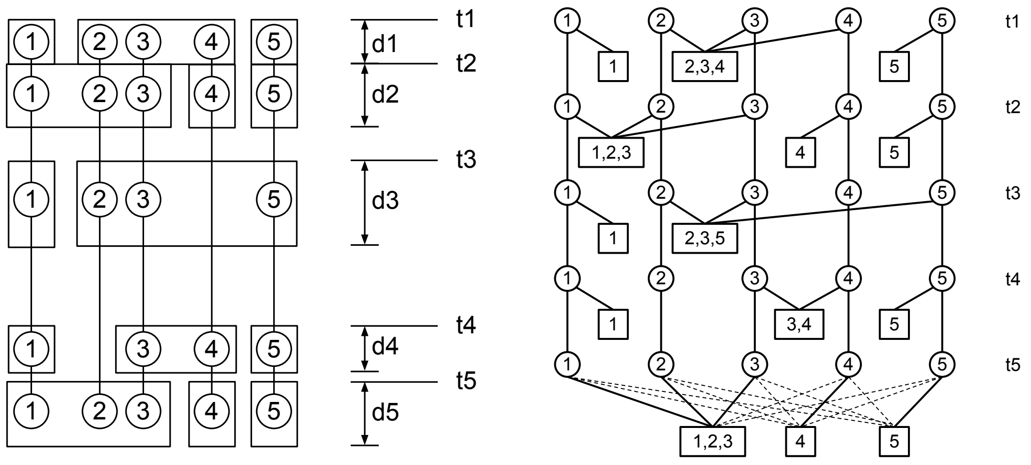


Figure 8: **Left:** an example sequence of sightings. **Right:** the corresponding sighting graph.

Note that we do not know the exact time when the switch happens. We know only the duration between which it happens. When considering a time step t, we will refer to the coloring of the individual vertices at timestep t as a *community assignment*. We also have a *sighting interpretation* which is a coloring $\chi_S : \{s_i\} \to \mathbb{N}$ of the sighting vertices in the corresponding sighting graph. We treat χ_S as a vector in $\mathbb{N}^m$ so that two different sightings with identical members but sighted at different times can have different colors. Together, we call the pair of coloring functions a *community interpretation* $\chi : \{v_{x,t}\} \cup \{s_i\} \to \mathbb{N}$ of the sighting graph or of the sequence of sighting. Intuitively, most individuals x at a timestep T_i would have the same color as its sighting $S_i \ni x$ and different sightings observed at the same timestep t would have different colors. However, the color of a member x may be different from the color of its sighting $S_i \ni x$ and may match another sighting $S_j \not\ni x$, meaning that x is just visiting with the rest of the members of the sighting but does not belong to their community. We also note that, unlike in Chapter 4, here, we allow two or more sightings at the same timestep to have the same color. This can be interpreted as a community splitting. Of course, this only happens only if it is the best explanation of the observed data. In other words, the solution is optimal. Now we are ready to present the probabilistic model which generates a community structure and an observed sequence of sightings.

5.3 TDK Model

We call our model TDK as it takes time decay into account. We are going to model how each individual switches community and becomes absent from its community as events in Poisson processes (81). In a Poisson process, the inter-arrival time between events follows an exponential distribution with the same rate as that of the Poisson process. Recall, for an exponential random variable with rate λ, the cumulative distribution function is $1 - e^{-\lambda t}$ for $t \geq 0$, and 0 elsewhere.

Let $\mathcal{U}$ be a distribution over the set of all finite-dimensional vectors to be specified later. We draw $(t_1, \ldots, t_m)$ from $\mathcal{U}$, resampling if the t_i's are not unique. At the end of the process, we relabel them so that $t_1 < \cdots < t_m$. These will be the times at which the hidden model emits observable sightings. We draw $(d_1, \ldots, d_m)$ from another distribution $\mathcal{D}$ over the set of all finite-dimensional vectors to be specified later. These will be the durations of the sightings. Our model resembles the hidden Markov model (HMM). Recall that HMM consists of three parts: the initial state, the state transition, and the emission (or the output). In our model, states correspond to community assignments and an emission is a collection of sightings which represent communities at that time.

First, we generate the hidden community structure which is essentially a sequence of assignments of individuals to communities at different times. We first pick a community assignment at time t_1 from the set of all partitions of $\{x_1, \ldots, x_n\}$ uniformly at random. Then, at any point, an individual can independently switch community with some probability. If an individual x switches, the identity of its new community depends only on its current community. Consider time $t_i = t_2, \ldots, t_{m-1}$ at which some community switching may happen. Let λ_{sw} be the rate of switching. Consider any individual x. Let the time until the next switching of x be a random variable T_{x,t_i} which follows an i.i.d. exponential distribution with rate λ_{sw}. If $T_{x,t_i} < t_{i+1} - t_i$ then x switches community between time t_i and t_{i+1}. Otherwise, x stays with its current community. Here, I assume that every interval $[t_i, t_{i+1}]$ is small enough that I can apply the orderliness assumption of Poisson processes, which states that the probability of x switching twice or more is 0. The probability of x switching is $P_{sw}(x, t_i) := 1 - e^{-\lambda_{sw}(t_{i+1}-t_i)}$. If x switches between t_i and t_{i+1}, then x chooses its new community from a probability distribution p_{x,t_i}. We will specify $p_{x,t_i}(\cdot)$ later. All we need for now is that, given that individual x switches between t_i and t_{i+1}, the function $p_{x,t_i}(c)$ gives the conditional probability that x switches to a community c. This completes the state transition of x. The transitions of all individuals together determine the community structure χ_C. Thus, the probability of a community structure χ_C is,

$$P_1 := \frac{1}{\mathcal{B}_n} \prod_{x=x_1}^{x_n} \prod_{t=t_1}^{t_{m-1}} \left[P_{sw}(x,t) \cdot p_{x,t}(\chi(v_{x,t_{i+1}}))\right]^{I_{sw}(x,t)} \cdot \left[1 - P_{sw}(x,t)\right]^{1-I_{sw}(x,t)}, \tag{5.1}$$

where $\mathcal{B}_k$ is the k^{th} Bell number[1] and $I_{sw}(x, t_i)$ is the indicator variable equal to 1 if $\chi(v_{x,t_i}) \neq \chi(v_{x,t_{i+1}})$ and 0 otherwise. Here, we define $0^0 = 1$.

Once we have determined the hidden community structure χ_C, we emit observable sightings in each timestamp t_i. The emission at time t_i depends only on the community assignment at time t_i. Consider any time $t_i = t_1, \ldots, t_m$. Let X_i be the random set variable indicating the communities to be sighted at time t_i. Let $r_{t_i}(B)$ be the probability that $X_i = B$. We will specify $r_{t_i}(\cdot)$ later. The sightings S_i are random set variables corresponding to the members of each community in X_i to be sighted with two-sided errors: some members of that community might not be in S_i (false negative) and some members of S_i might not be members of the community (false positive). Let λ_{ab} be the rate of an individual being absent from its current community. Consider some individual x. Let U_{x,t_i} be an i.i.d. exponential random variable with rate λ_{ab} indicating the time until the next absence of x. The probability that x is absent during time interval $[t_i, t_i + d_i)$ is $P_{ab}(i, t_i) := 1 - e^{-\lambda_{ab} d_i}$. If $U_{x,t_i} < d_i$, then x is absent from its current community. Here, I also assume that all d_i's are small enough such that I can use the orderliness assumption of Poisson processes. An absent individual will be sighted with a new community Y_{x,t_i} randomly chosen from a distribution q_{x,t_i} to be specified later. If the community of x was

[1]The k^{th} Bell number is the number of partitions of a k-set.

not part of X_i, then it will have to be part of a community in X_i since x has to be *sighted* as absent from some community. Thus, the sighting observed at time t_i is:

$$\begin{aligned} S_i &= \{x : \chi(v_{x,t_i}) \in X_i, U_{x,t_i} \geq d_i\} \\ &\quad \cup \{x : \chi(v_{x,t_i}) \notin X_i, U_{x,t_i} < d_i, Y_{x,t_i} \in X_i\}. \end{aligned}$$

The first part of S_i are the members of one of the communities in X_i who are not absent at the time. The second part are those who are absent from their communities to be with S_i. Once we are done with all the timesteps $t_1, \ldots, t_m$, we obtain a complete community structure χ_C. Given a community structure χ_C, the conditional probability of a sighting dataset $\mathcal{S}$ is,

$$\begin{aligned} P_2 := \prod_{x=x_1}^{x_n} \prod_{t=t_1}^{t_m} &[P_{ab}(x,t) \cdot q_{x,t}(Y_{x,t})]^{I_{ab}(x,t)} \\ &\cdot [1 - P_{ab}(x,t)]^{1-I_{ab}(x,t)} \end{aligned} \tag{5.2}$$

where $I_{ab}(x, t_i)$ is the indicator variable equal to 1 if x is absent during $[t_i, t_i + d_i)$.

Let $\theta = (\lambda_{sw}, \lambda_{ab}, \{p_{x,t}\}, \{q_{x,t}\}, \{r_t\})$ be the parameters defining all the distributions. Multiplying the probabilities in (Equation 5.1) and (Equation 5.2), we obtain the probability of a community interpretation $\chi = \chi_C \cup \chi_S$ and a sighting data set $\mathcal{S}$,

$$\Pr[\chi, \mathcal{S} | \theta] = P_1 \cdot P_2,$$

The probabilistic model is essentially complete, pending only the specification of the nonessential elements p's, q's, and r's. We call this model TDK for it takes into account the time decay. In reality, we observe only the sighting dataset $\mathcal{S}$ while the true community interpretation χ is hidden. So we would like to infer the community interpretation χ and parameters θ that maximize the likelihood of $\mathcal{S}$:

$$\ell(\chi, \theta|\mathcal{S}) = \Pr(\chi, \mathcal{S}|\theta).$$

Now we are ready to formulate the problem of inferring the community structure from the sighting data.

5.3.1 Problem Formulation

In the previous section, we describe the probabilistic model which generates a hidden community structure and an observable sequence of sightings. Now we formally state the reverse process of inferring the community structure from the sighting data. We pose it in the form of finding a maximum likelihood path for the corresponding Markov chain. Given a rate of switching λ_{sw} and a rate of absence λ_{ab}, We would like to find $\chi, p_{\mathcal{U}}, p_{\mathcal{D}}, p_{\mathcal{S}}, \{p_{x,t}\}, \{q_{x,t}\}, \{r_t\}$ which together maximize the likelihood of a given sighting dataset $\mathcal{S}$. Since we only care about the community interpretation χ, now we prove that we can easily find optimal distributions p's, q's, r's for any community interpretation χ (including the optimal one).

Theorem 1. *Given $\lambda_{sw}, \lambda_{ab}$, and χ, the following set of degenerate distributions maximizes the likelihood of $\mathcal{S}$, t_i, d_i, under the TDK model:*

$$\begin{aligned}
p_{\mathcal{U}}((t_1, \ldots, t_m)) &= 1, \\
p_{\mathcal{D}}((d_1, \ldots, d_m)) &= 1, \\
p_S(\{\chi(s_j) : t_j = t\}) &= 1 \qquad \textit{for all } t, \\
p_{x,t_i}(\chi(v_{x,t_{i+1}})) &= 1 \qquad \textit{for all } x \textit{ and } t_i \\
q_{x,t}(\chi(v_{x,t})) &= 1 \qquad \textit{for all } x \notin S_i \textit{ and all } t, \\
q_{x,t_i}(\chi(S_i)) &= 1 \qquad \textit{for all } x \in S_i \textit{ and all } t_i, \\
r_{t_i}(\chi(S_i)) &= 1 \qquad \textit{for all } t_i,
\end{aligned}$$

and 0 *elsewhere.*

Proof. Since all probabilities are at most 1, removing the LHS terms from Equation 5.1 and Equation 5.2 gives an upper bound which is equal to that when plugging in the RHS into Equation 5.1 and Equation 5.2. □

With Theorem 1, the probabilistic model is complete. Note that these over-fitting distributions p's, q's, r's can be thought of as parts of the data that we do not care to model. If we have a hypothesis related to these parts, then they should be replaced by some other distributions to reflect the hypothesis.

Theorem 1 can be used to set the distributions p's, q's, r's in finding a community interpretation χ that maximizes the likelihood of $\mathcal{S}$. We are now ready to define the problem of inferring community interpretation from a sighting dataset:

Dynamic Community Interpretation (DCI): Given a rate of switching λ_{sw}, a rate of absence λ_{ab}, and a sighting dataset $\mathcal{S} = \langle (S_1, t_1, d_1), \ldots, (S_m, t_m, d_m) \rangle$, find a community interpretation χ which maximizes the likelihood of $\mathcal{S}$ under the TDK model.

Thus, we have defined the Dynamic Community Interpretation problem as that of finding clusters of individuals over time that maximize the likelihood of sighting data under the TDK model. More explicitly, taking the log of (Equation 5.1)–(Equation 5.2), we obtain,

$$\begin{aligned}
\ell_1 &:= -\log \mathcal{B}_n + \sum_{x=x_1}^{x_n} \sum_{t=t_1}^{t_{m-1}} \left[I_{sw}(x,t) \log P_{sw}(x,t) \right. \\
&\qquad \left. + (1 - I_{sw}(x,t)) \log(1 - P_{sw}(x,t)) \right], \\
\ell_2 &:= \sum_{x=x_1}^{x_n} \sum_{t=t_1}^{t_m} \left[I_{ab}(x,t) \log P_{ab}(x,t) \right] \\
&\qquad + (1 - I_{ab}(x,t)) \log(1 - P_{ab}(x,t)) \right].
\end{aligned}$$

Thus, maximizing the likelihood is equivalent to maximizing $\ell_1 + \ell_2$. In this form, the clustering problem that we are trying to solve is known as Maximizing Agreement (6). To complete the reduction to Correlation Clustering, we use the abstraction of the sighting graph with the addition of edge weights that correspond to the terms in $\ell_1 + \ell_2$. Recall, the sighting graph has a vertex v_{x,t_i} for each individual x at time t_i and a vertex s_i for each sighting S_i. We now describe the edges of the sighting graph. We connect the vertex s_i to all the individual vertices at the same time t_i as S_i. That is, for each x and each s_i, there is an edge between s_i and v_{x,t_i}. At this point, each time $t = t_1, \ldots, t_m$ corresponds to a connected component which is a complete bipartite graph between two parts: $\{s_i : t_i = t\}$ and $\{v_{x,t} : x = x_1, \ldots, x_n\}$. Then, we connect the vertices of each individual x into a path $v_{x,t_1}, \ldots, v_{x,t_m}$. The weight of an edge is the log of the likelihood that the two endpoints of the edge belong to the same community. That is, the weight of an edge corresponds to the appropriate term in $\ell_1 + \ell_2$.

On this weighted sighting graph, the Dynamic Community Interpretation problem is nearly equivalent to the Maximizing Agreement problem. To make the two problems equivalent, we can show by some straightforward algebraic manipulation that edge weights should be set to the following log odds:

$$\begin{aligned}
w(v_{x,t_i}, v_{x,t_{i+1}}) &= \log \frac{1 - P_{sw}(x, t_i)}{P_{sw}(x, t_i)} \\
w(s_i, v_{x,t_i}) &= \log \frac{1 - P_{ab}(x, t_i)}{P_{ab}(x, t_i)} \qquad \text{if } x \in S_i \\
w(s_i, v_{x,t_i}) &= \log \frac{P_{ab}(x, t_i)}{1 - P_{ab}(x, t_i)} \qquad \text{if } x \notin S_i
\end{aligned}$$

In the right panel of Figure 8, the solid edges between two circles are weighted by $\log \frac{1-P_{sw}(x,t)}{P_{sw}(x,t)}$. The solid edges between a circle and a rectangle are weighted by $\log \frac{1-P_{ab}(x,t)}{P_{ab}(x,t)}$. The dotted edges are weighted by $\log \frac{P_{ab}(x,t)}{1-P_{ab}(x,t)}$. For visibility reason, the dotted edges are shown only for timestep t_5 but exist for every t_i.

With that, the objective function has changed from maximizing the log likelihood to maximizing the sum of intra-cluster edge weights in the sighting graph (see Figure 8), which is exactly the Maximizing Agreement problem. Thus, we can use the approximation algorithms for Correlation Clustering to solve Dynamic Community Interpretation. In the next two sections, we will present two algorithms for solving Dynamic Community Interpretation. The first is a pair of approximation algorithms for Correlation Clustering based on SDP relaxation, and the other is our UIC heuristic algorithm.

5.4 Approach via Reduction to Correlation Clustering

Now that we have the reduction from Dynamic Community Interpretation to Correlation Clustering, we use approximation algorithms for Maximizing Agreement to find a clustering for Dynamic Community Interpretation.

See Appendix C for more details.

Although the interior-point method runs in polynomial time and the two rounding schemes are very efficient, in practice, the state-of-the-art SDP solvers still take quite long to solve a decently-sized SDP to optimality. We need a faster and better algorithm for Correlation Clustering and Dynamic Community Interpretation for any realistic network sizes. In this paper, we will also solve the SDP relaxation since the optimal solution to the SDP relaxation can upper-bound the optimal solution to Correlation Clustering and Dynamic Community Interpretation. In the next section, we present a fast heuristic which produces very near-optimal solutions.

5.5 Unilateral Improvement and Contraction Algorithm

Now we introduce Unilateral Improvement and Contraction (UIC) algorithm. The algorithm is based on the Louvain algorithm based on maximizing modularity (9). The UIC algorithm starts with a trivial clustering where each vertex is in a cluster by itself. Then, it repeatedly makes a unilateral improvement until it can do no more. A unilateral improvement is the one in which a vertex chooses its new cluster from a set of candidates. The candidates are the vertex's current cluster and the neighboring clusters. It chooses a new cluster so that the resulting clustering has the highest objective value. Here, the objective function is the sum of the intra-cluster edge weights. When no vertex can find a better cluster, the algorithm contracts each cluster into a super vertex. In particular, the vertices in each cluster C_i are replaced by a super vertex v_{C_i}. All edge weights (both inter-cluster and intra-cluster[1]) are preserved by setting $w(v_{C_i}, v_{C_j}) = \sum_{u \in C_i} \sum_{v \in C_j} w(u, v)$. Then, the algorithm repeats from the top on this new graph. This process repeats until every vertex is in a cluster by itself, at which point the graph after contraction will be identical to the graph before contraction. The outline of the algorithm is shown below.

First, we show that the UIC algorithm terminates on all inputs. We observe that a vertex i always has its current cluster $\chi(i)$ as a candidate. This implies that the objective value always increases. Since the maximum sum of edge weights is finite value, the while loop terminates at some point. Then, we contract the clusters only if the size of graph will strictly decrease, thus, this contraction ends at some point.

[1]For $i = j$, the weight of each intra-cluster edge is accounted twice in the sum, preserving the objective function.

Algorithm 4: UIC algorithm for MAXIMIZING AGREEMENT

```
Input: a graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}$.
$\chi \leftarrow$ every vertex in a cluster by itself.
while objective value strictly increases do
    for each node $i \in V$ do
        Assign $i$ to the cluster in $\{\chi(j) : j \in N(i) \cup \{i\}\}$ that yields the highest objective
        value, breaking ties lexicographically.
    end
end
If $|\text{Range}(\chi)| < |V|$, then contract each cluster of size at least two into a supervertex,
aggregate edge weights accordingly, and repeat from the top.
```

We implemented the UIC algorithm in Python. This is sufficiently efficient as it takes only a few seconds in all the cases that we consider. Since the ordering of the vertices might affect the quality of the solutions, we report our vertex ordering here:

$$\langle s_1, \ldots, s_m \rangle || \langle v_{x_1,t_1}, \ldots, v_{x_1,t_k} \rangle || \cdots || \langle v_{x_n,t_1}, \ldots, v_{x_n,t_k} \rangle.$$

In the next section, we will show that the UIC algorithm produces near-optimal solutions.

5.6 Experiments

In this section, we present our experiments and results. We use two datasetes Grevy's zebra and onager (see Sections A.2.1 and A.2.2 in the appendices for more details).

5.6.1 Data Pre-Processing

We use two datasets of animal sightings to showcase the TDK model. The sizes of the datasets are small enough that the SDP relaxation can be solved in timely fashion so that we can try a wide range of $\lambda_{sw}, \lambda_{ab}$ values.

We preprocessed the datasets as follows. The field records do not contain the durations of sightings, thus, we estimate them from the average delay between two consecutive sightings within six hours of one another. The average delay between sightings is 57.90 minutes in Grevy's zebra and 46.39 minutes in onager. Therefore, we think one hour is a good estimate of the maximum duration of sightings. Having this in mind, we estimate the starting time and ending time of each sighting by extending from the time point at which it was recorded 30 minutes before and 30 later, as long as it does not overlap with another sighting with some common members. In other words, if S_i and S_j overlap ($t_i \leq t_j$), we use the middle point $(t_i + t_j)/2$ as the ending time of S_i and the starting time of S_j.

5.6.2 Experimental Setup

In this section, we compare the quality of the clusterings produced by Swamy's algorithms and the UIC algorithm. We show the results on both Grevy's zebra and onager datasets under different rate values $\lambda_{sw}, \lambda_{ab} \in \{0.1, 0.2, \ldots, 1.0\}$ in the unit of events per day.

To measure the quality of a solution, we compute the *optimality gap.* For an instance I of a problem, the optimality gap of an algorithm $\mathcal{A}$ is $\gamma(\mathcal{A}, I) = \frac{\mathcal{A}(I)}{\text{OPT}(I)}$ where $\mathcal{A}(I)$ is the solution value by $\mathcal{A}$ and $\text{OPT}(I)$ is the optimal value. Since computing $\text{OPT}(I)$ is NP-hard in general, we usually use a bound on $\text{OPT}(I)$ to bound $\gamma(\mathcal{A}, I)$ instead. The relationship between optimality gap and the *approximation ratio* is that the latter is the optimality gap in the worst case of the algorithm. For a maximization problem, the approximation ratio of $\mathcal{A}$ is $\rho(\mathcal{A}) = \inf_{I \in \Pi} \gamma(\mathcal{A}, I)$ which is always at most 1.

Even though we reduce Dynamic Community Interpretation to Correlation Clustering, their objective values are different. Let LL denote the log-likelihood which is the objective value of Dynamic Community Interpretation. Let LO denote the sum of log odds which is the objective value of Correlation Clustering. For LO, we know that Swamy's 2-hyperplane rounding has approximation ratio of 0.75 and, when choosing the best between 2-hyperplane rounding and 6-center rounding, the approximation ratio is 0.7666 (93). Although we can compute LL from LO via LL = LO − offset, the approximation ratio of any algorithm for Correlation Clustering does not translate in a straightforward way to an approximation ratio for Dynamic Community Interpretation. Therefore, we will have to look at the optimality gaps associated with both LL and LO.

For the experiment, we ran each randomized rounding algorithm for 1000 trials. We noticed that the 4-hyperplane rounding performed better than the 2-hyperplane rounding in all the cases we considered. Thus, we show only the results of the 4-hyperplane rounding (SDP+4HP) and the 6-center rounding (SDP+6CT). Moreover, we also tried to improve the solutions further by post-processing with dynamic programming (95). The idea is that we keep the coloring of the sightings fixed and optimally recolor the individual vertices. The post-processing is guaranteed not to worsen the solution, so we post-processed both of our baseline algorithms and denote the results by appending +DP appropriately.

5.6.3 Results

First, we look at the results on the Grevy's dataset. Figure 9 shows the optimality ratios of LO. Even before the post-processing with the dynamic programming algorithm, UIC outperforms both SDP+4HP and SDP+6CT in all cases. After the post-processing, SDP+4HP+DP outperforms UIC in the 21 cases and UIC+DP in 10 cases. SDP+6CT+DP is still outperformed by UIC in all the cases. In cases in which UIC is outperformed, the optimality gaps of UIC are already at least 0.99. We omit SDP+6CT since it performs relatively poorly. It is clear from the figure that the post-processing improves UIC only slightly. The most important observation is that UIC is the only algorithm here which produces a solution with optimality gap at least 0.99 in all the cases (SDP+4HP+DP misses the cut by only 10 cases). Figure 10 shows the optimality ratios for LL. Note that, since DYNAMIC COMMUNITY INTERPRETATION technically[1] is a minimization problem, the optimality gap is at least 1. The smaller the gap, the better. The results are similar to the optimality ratios for LO. For completeness, we compare the actual LL values and the bounds from the SDP relaxation in Figure 11.

Now we look at the results on the onager dataset. Figure 12 shows the optimality ratios for LO. It is clear from the figure that UIC outperforms both baseline algorithms even after the post-processing. Again, the optimality gap of UIC is at least 0.99 in all the cases. We omit both SDP+4HP and SDP+6CT since they perform relatively poorly. The optimality ratio of UIC is at least 0.99 in all cases. For completeness, we compare the optimality ratios for LL

[1]Log-likelihood is always negative, so the absolute value of the optimal value is smaller than the absolute value of a solution value produced by an algorithm.

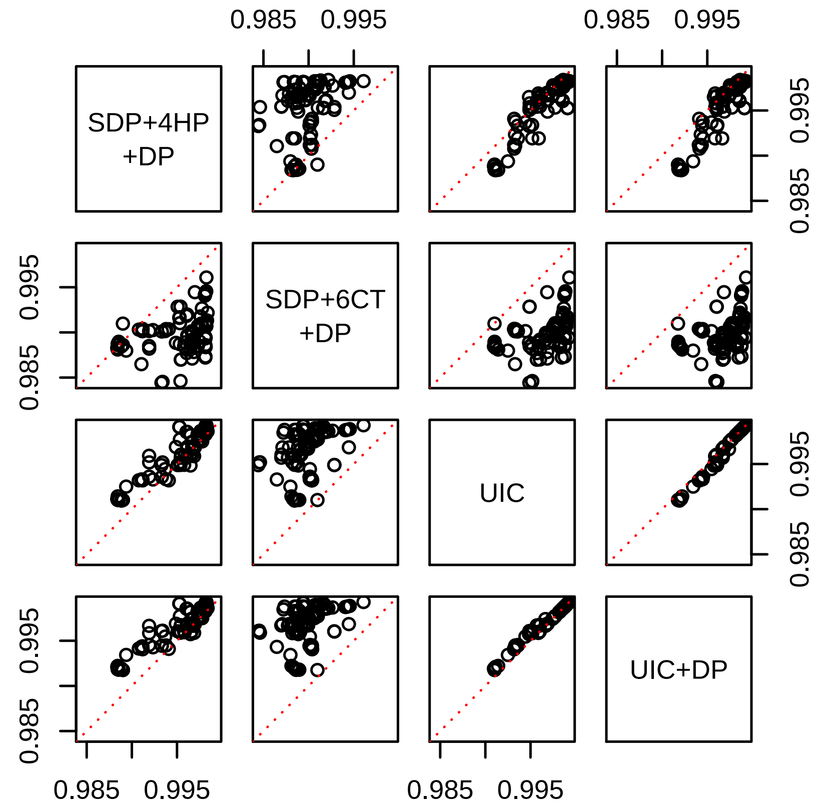


Figure 9: Optimality gaps LO/OPT$_f$ on Grevy's zebra dataset.

in Figure 13 and compare the actual LL values with the bounds from the SDP relaxation in Figure 14.

5.7 Summary

We present the TDK model which is a probabilistic model for community and observation data. Then, we formulate Dynamic Community Interpretation as the problem of finding

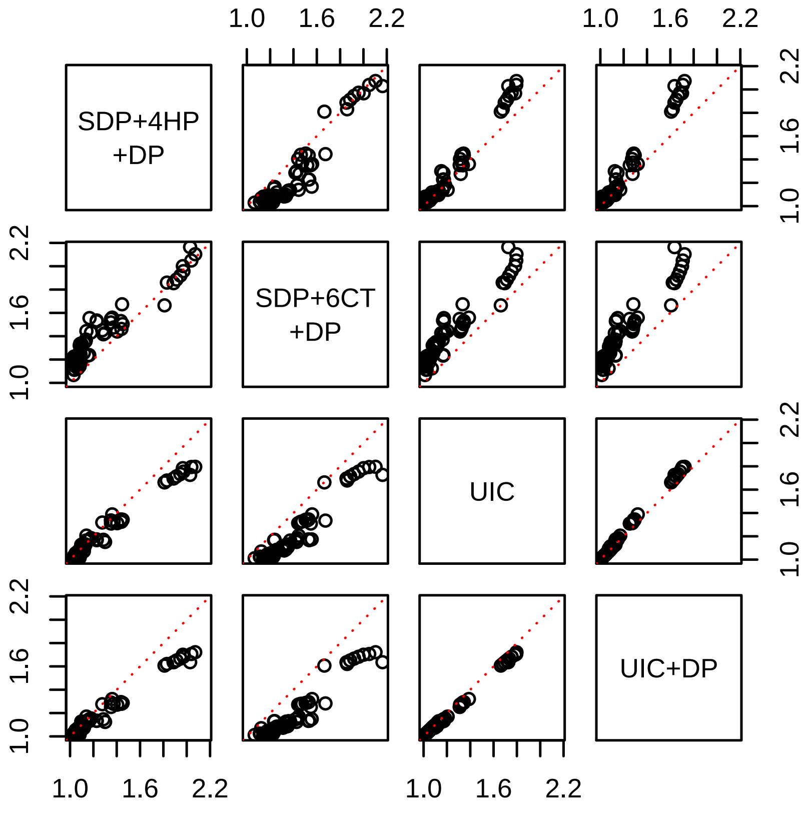


Figure 10: Optimality gaps LL/(OPT_f − offset) on Grevy's zebra dataset.

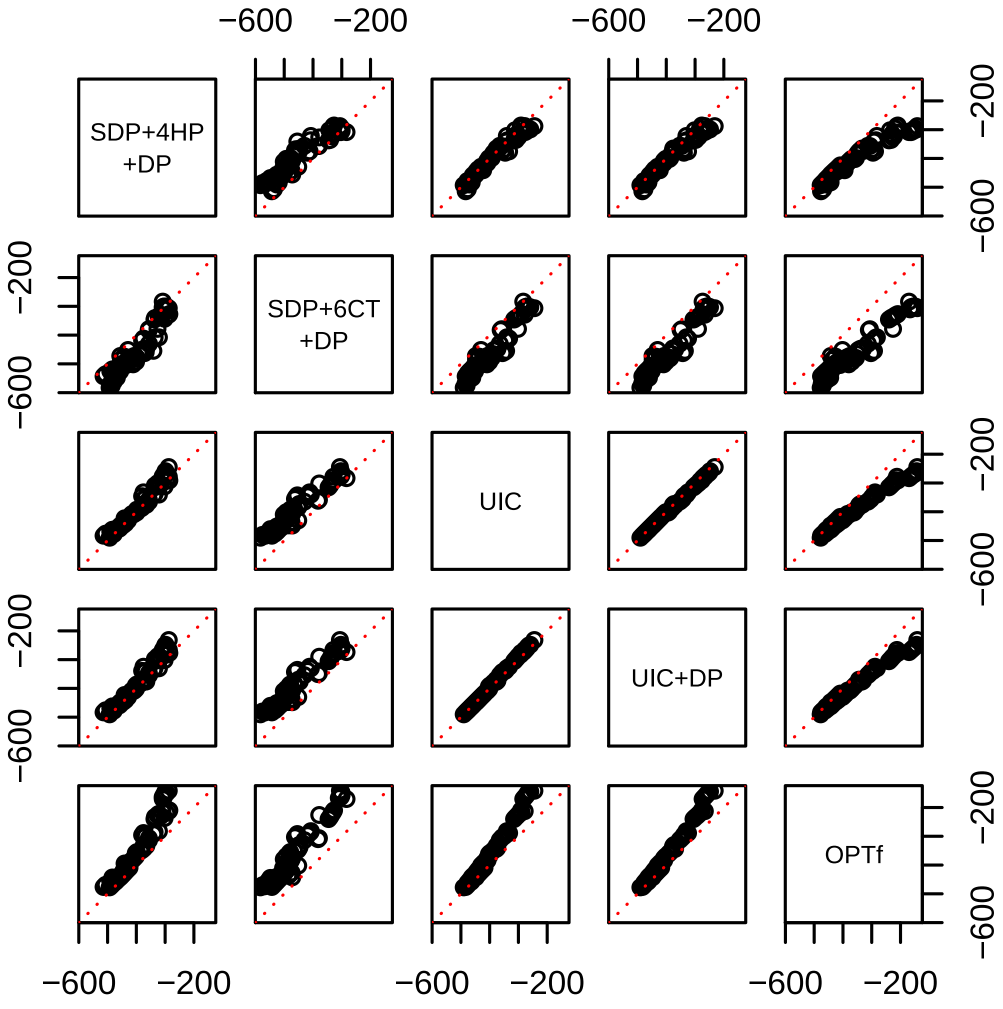


Figure 11: LL on Grevy's zebra dataset. OPT$_f$ is the optimal value to the SDP relaxation which is an upper bound on the optimal LL.

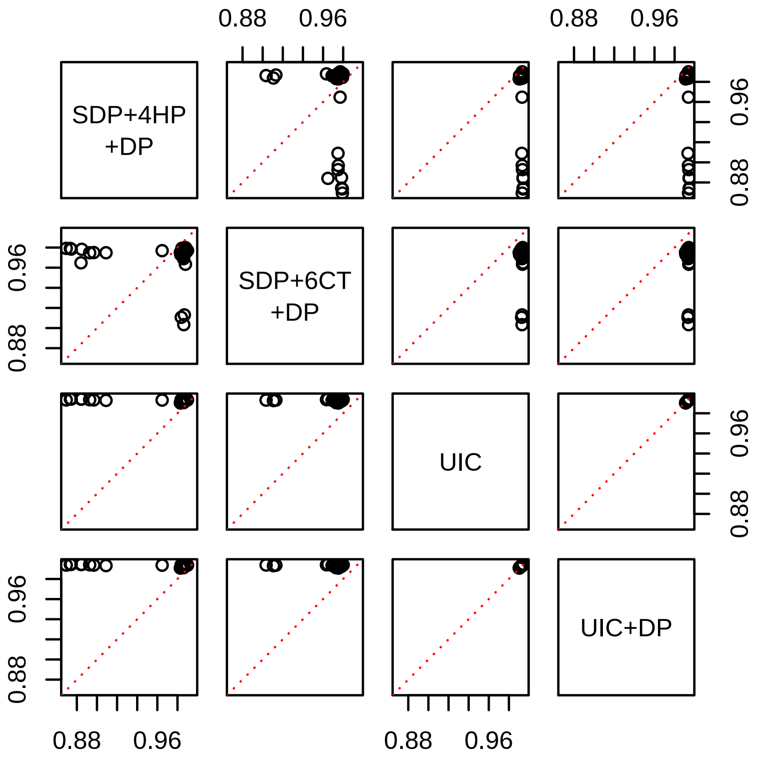


Figure 12: Optimality gaps LO/OPT$_f$ on Onager dataset.

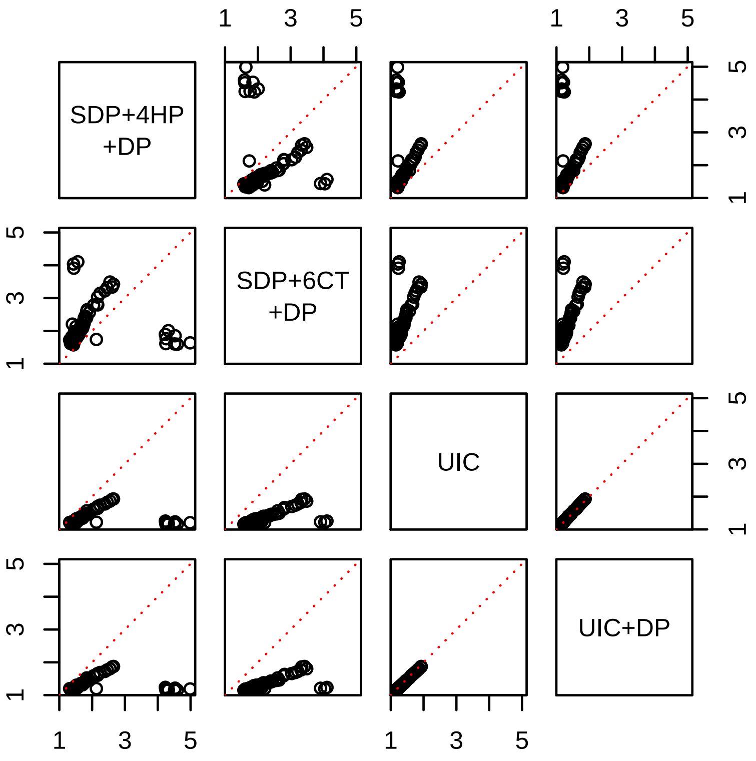


Figure 13: Optimality gaps LL/(OPT$_f$ − offset) on Onager dataset.

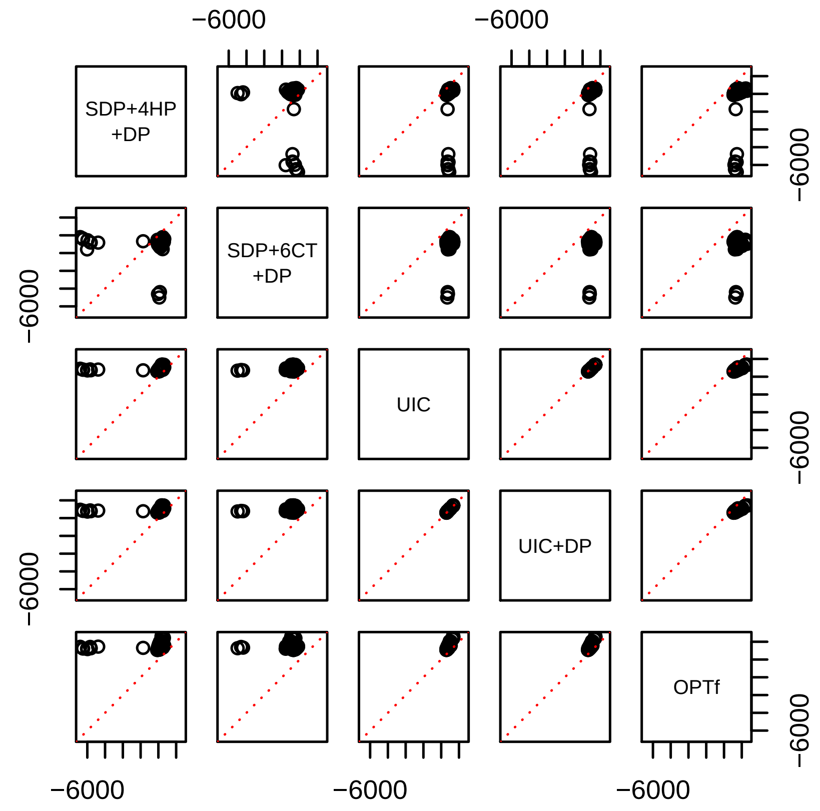


Figure 14: LL on onager dataset. OPT$_f$ is the optimal value to the SDP relaxation which is an upper bound on the optimal LL value.

a community structure in dynamic networks which maximizes the likelihood of the observed data under the TDK model. We show how to solve the problem by reducing DYNAMIC COMMUNITY INTERPRETATION to CORRELATION CLUSTERING problem so as to use the available approximation algorithms for CORRELATION CLUSTERING to solve it. The approximation algorithms provides a good upper bound on the optimal value but it is slow in practice. As a solution, we propose UIC algorithm, a fast heuristic for solving CORRELATION CLUSTERING and DYNAMIC COMMUNITY INTERPRETATION and use the upper bound given by the approximation algorithms to measure the quality of the solutions produced by the UIC algorithm. The experimental results are surprising, that the solutions produced by UIC algorithm are 0.99 times of the optimal solutions to CORRELATION CLUSTERING and 1.93 times of the optimal solutions to DYNAMIC COMMUNITY INTERPRETATION. Thus, the UIC algorithm finds near-optimal solutions to DYNAMIC COMMUNITY INTERPRETATION (for which it was designed) but can also be used as a general algorithm for CORRELATION CLUSTERING, which opens a line of theoretical inquiry, from time complexity to approximability.

CHAPTER 6

DETECTING COMMUNITIES IN DYNAMIC NETWORKS

In this chapter, I consider the problem of detecting communities in a network which changes over time and simultaneously tracking changes of community structure. This work first appeared in (97).

6.1 Notations and Definitions

Recall from Chapter 2 that a *network* on a vertex set V is a sequence of *snapshot* graphs $\mathcal{G} = \langle G_1, \ldots, G_m \rangle$ over m discrete *timesteps* $t_1, \ldots, t_m$. In this chapter, we assume that timesteps are uniform ($t_{i+1} - t_i = t_{j+1} - t_j$ for all i, j). The *snapshot* of the network at time t_i is a graph $G_i = (V_i, E_i)$ on some subset $V_i \subseteq V$ of all vertices in the network. The vertices in V_i are *known* to be interacting (or not interacting) at timestep t_i and the edges in E_i represent the interactions at that timestep. For the vertices not in V_i, we do not know anything about their interactions at timestep t_i, or the lack thereof. A community structure is a coloring function $\chi : V \times \{t_1, \ldots, t_m\} \to \mathbb{N}$. Each color represents a unique community. The interpretation is, at a timestep t, a vertex v is in the community represented by the color $\chi(v, t)$. When restricted to a particular timestep t_i, we refer to χ as the community membership at time t_i. The problem of detecting and tracking communities in a network $\mathcal{G}$ can be posted as finding a community structure χ which *best* explains the interactions in $\mathcal{G}$. Now, let us formally define what *best* means in this context.

We assume the following behavioral model:

Gradual changes: Individuals change community affiliation not very often.

Reliable true positive: Members of the same community mostly interact with each other.

Negligible false positive: Members of different communities rarely interact with each other.

We translate these three assumptions into social costs:

Switching cost: each individual u incurs C_{sw} when changing community affiliation:

$$\chi(u, t_i) \neq \chi(u, t_{i+1}).$$

False negative cost: two individuals u and v incur C_{fn} when they belong to the same community but do not interact:

$$\chi(u, t_i) = \chi(v, t_i) \text{ and } (u, v) \notin E_i.$$

False positive cost: two individuals u and v incur C_{fp} when they belong to different communities but do interact:

$$\chi(u, t_i) \neq \chi(v, t_i) \text{ and } (u, v) \in E_i.$$

The Network Community Interpretation problem is to find a coloring χ that minimizes the total cost of switching, false negative, and false positive:

$$\begin{aligned} c(\chi) = C_{\text{sw}} \sum_{i=1}^{m-1} \mid \{u \in V : \chi(u,t_i) \neq \chi(u,t_{i+1})\} \mid \\ + C_{\text{fn}} \sum_{i=1}^{m} \mid \{(u,v) \notin E_i : \chi(u,t_i) = \chi(v,t_i)\} \mid \\ + C_{\text{fp}} \sum_{i=1}^{m} \mid \{(u,v) \in E_i : \chi(u,t_i) \neq \chi(v,t_i)\} \mid . \end{aligned}$$

It is easy to see that the decision version of Network Community Interpretation is NP-complete. This can be shown by a straightforward reduction from the unweighted Minimizing Disagreement problem (see Appendix C). Given a complete graph $G = (V, E)$ in which each edge is labeled either as $+$ or $-$, we create an instance of Network Community Interpretation on one-timestep network $\mathcal{G} = \langle (V, E^+) \rangle$ where E^+ is the set of the edges with the $+$ label. To complete the construction, we set the cost parameters $(C_{\text{sw}}, C_{\text{fn}}, C_{\text{fp}}) = (0, 1, 1)$. If $\mathcal{G}$ has a coloring of cost at most b then G has a clustering with at most b disagreements.

Next, we show that there is a probabilistic generative model the likelihood of which is maximized by the Network Community Interpretation problem.

6.2 Probabilistic Model

We have formulated the Network Community Interpretation. We now present a generative probabilistic model whose likelihood is maximized by the Network Community Interpretation problem. The first three parameters are:

- k the number of communities,

- n the number of individuals, and
- m the the number of timesteps.

The probabilistic model generates a community interpretation χ and a dynamic network $\mathcal{G} = \langle G_1, \ldots, G_T \rangle$. Each snapshot is a graph $G_i = (V, E_i)$ on the same vertex set V of n vertices. We introduce three more parameters:

- p_{sw} the probability of an individual switching community affiliation,
- p_{fn} the probability of a false non-edge (missing intra-community edge), and
- p_{fp} the probability of a false edge (extra inter-community edge).

First, we generate the hidden community structure χ. We begin by generating the initial community membership at the first timestep t_1. From there on, the community membership at timestep t_i is generated based only on the community membership at previous timestep t_{i_1}, independently from everything else. At the first timestep t_1, each individual independently picks one of the k choices as its initial community uniformly at random. Then, given the community membership at timestep t_{i-1}, we generate the community membership at timestep t_i as follows. Each individual independently decides to switch community with probability p_{sw}. Otherwise, the individual stays with its current community. An individual who decided to switch picks its new community uniformly at random from the $k - 1$ choices.

Then, we generate the an observable dynamic network. The network snapshot $G_i = (V, E_i)$ depends only on the community membership at time t_i and is independent of everything else. Thus, we generate each G_i separately. For each pair of vertices (u, v) from the same community, we join them by an edge with probability $1 - p_{\text{fn}}$. Otherwise (u, v) is a *missing* intra-cluster edge. Similarly, for each pair of vertices (u, v) from different communities, they are joined by an edge with probability p_{fp}. Such (u, v) is an *extra* intra-cluster edge.

With respect to χ and $\mathcal{G}$, recall that a false negative is a non-edge between two community members, and a false positive is an edge that goes across two communities. Let N_{sw} be the number of switches in χ, N_{fn} be the number of false negatives, and N_{fp} be the number of false positives. Let M_{fp} be the total number of edges in $\mathcal{G}$, and M_{fn} be the total number of non-edges in $\mathcal{G}$. Note that $M_{\text{fp}} = \sum_{t=1}^{T} |E_t|$ and $M_{\text{fp}} + M_{\text{fn}} = T\binom{N}{2}$. Under this model with parameter $\lambda = (k, N, T, p_{\text{sw}}, p_{\text{fn}}, p_{\text{fp}})$, the probability of a given $\mathcal{G}, \chi$ is,

$$\begin{aligned}
\Pr[\mathcal{G}, \chi | \lambda] &= \frac{1}{k^N} \left(\frac{p_{\text{sw}}}{(k-1)(1-p_{\text{sw}})} \right)^{N_{\text{sw}}} (1-p_{\text{sw}})^{N(T-1)} \\
&\quad \times \left(\frac{p_{\text{fn}}}{1-p_{\text{fn}}} \right)^{N_{\text{fn}}} (1-p_{\text{fn}})^{M_{\text{fn}}} \\
&\quad \times \left(\frac{p_{\text{fp}}}{1-p_{\text{fp}}} \right)^{N_{\text{fp}}} (1-p_{\text{fp}})^{M_{\text{fp}}}
\end{aligned}$$

By taking natural logarithm and simplifying, we obtain,

$$\begin{aligned}
\ln \Pr[\mathcal{G}, \chi | \lambda] &= C + N_{\text{sw}} \ln \left(\frac{p_{\text{sw}}}{(k-1)(1-p_{\text{sw}})} \right) \\
&\quad + N_{\text{fn}} \ln \left(\frac{p_{\text{fn}}}{1-p_{\text{fn}}} \right) + N_{\text{fp}} \ln \left(\frac{p_{\text{fp}}}{1-p_{\text{fp}}} \right).
\end{aligned}$$

where C is the sum of the terms not depending on χ. We observe that the log-probability has terms similar to the coloring cost $c(\chi)$ in Equation Equation 6.1. Thus, maximizing the log-probability is the same as minimizing the coloring cost by setting,

$$\begin{aligned} C_{\text{sw}} &= -\ln \frac{p_{\text{sw}}}{(k-1)(1-p_{\text{sw}})}, \\ C_{\text{fn}} &= -\ln \frac{p_{\text{fn}}}{1-p_{\text{fn}}}, \qquad C_{\text{fp}} = -\ln \frac{p_{\text{fp}}}{1-p_{\text{fp}}}. \end{aligned}$$

In practice, we try to infer χ from an observed dynamic network $\mathcal{G}$. For a given χ, $\ln \Pr[\mathcal{G}, \chi|\lambda]$ is the likelihood of the parameter λ. Thus, an optimal coloring χ of a dynamic network $\mathcal{G}$, which minimizes $c(\chi)$, also maximizes the likelihood of the parameter λ.

Solving the above equation system gives model parameters in terms of cost setting.

$$p_{\text{sw}} = \frac{k-1}{e^{C_{\text{sw}}}+k-1}, \quad p_{\text{fn}} = \frac{1}{e^{C_{\text{fn}}}+1}, \quad p_{\text{fp}} = \frac{1}{e^{C_{\text{fp}}}+1}.$$

One issue is that multiplying C_{sw}, C_{fn},C_{fp} by a constant changes the base of the logarithm. So we do not actually have a one-to-one correspondence between cost setting and model parameter setting.

Also, it can be shown that:

- $p_{\text{sw}} < \frac{1}{2}$ if and only if $C_{\text{sw}} > \ln(k-1)$,
- $p_{\text{fn}} < \frac{1}{2}$ if and only if $C_{\text{fn}} > 0$,
- $p_{\text{fp}} < \frac{1}{2}$ if and only if $C_{\text{fp}} > 0$.

6.2.1 Maximum Likelihood

When we infer a community interpretation from an observation, we do not know the parameter λ. One way to estimate this is to use the maximum likelihood principle. Recall that for a multinomial distribution, the maximum likelihood estimators are the proportions of the events that occur. Naturally, one would assume a hidden Markov model and apply the EM algorithm. However, computing the expectation of numbers of switches, absences, and visits is not easy since it involves a complicated summing over all community interpretations. So we use N_{sw}, N_{fn}, N_{fp} of the community interpretation χ that maximizes $\Pr[\chi|\lambda]$ instead, and estimate p_{sw}, p_{fn}, p_{fp} as,

$$\widehat{p_{\mathrm{sw}}} = \frac{N_{\mathrm{sw}}}{N(T-1)}, \quad \widehat{p_{\mathrm{fn}}} = \frac{N_{\mathrm{fn}}}{M_{\mathrm{fn}}}, \quad \widehat{p_{\mathrm{fp}}} = \frac{N_{\mathrm{fp}}}{M_{\mathrm{fp}}}.$$

So the approach is similar to the EM algorithm (28). We start with some p_{sw}, p_{fn}, p_{fp}, say equal to $\frac{1}{4}$. We compute the corresponding costs C_{sw}, C_{fn}, C_{fp}, then find a community interpretation χ that minimizes the cost which also maximizes the probability $\Pr[\chi|\lambda]$. Then, we recompute p_{sw}, p_{fn}, p_{fp} from χ and repeat until converge.

6.3 Method

Since we have reduced the problem to the MINIMIZING DISAGREEMENT in the CORRELATION CLUSTERING problem, I use an approach based on Swamy's approximation algorithms (see Section C.2 in the appendix) to solve the problem. Instead of rounding the SDP optimal solution with random k-hyperplane rounding or k-center rounding, I use single-linkage clustering to round the optimal solution to the SDP to a solution to the strict quadratic program. In particular, the first block of the semidefinite matrix is the Gram's matrix and can be transformed into Euclidean distances, which are used in single-linkage clustering.

6.4 Experimental Results

We now present the results of our algorithm on several synthetic and real datasets. First, we generate synthetic datasets with known community structures to test the accuracy of our method and to compare it to alternative approaches. We then apply the algorithms to real dynamic networks. Although our algorithm take three parameters C_{sw},C_{fn},C_{fp}, it is the ratios between them that affect the optimal community structure since scaling the parameters by any positive constant does not change the optimal solution.

6.4.1 Synthetic Network Generator

Due to space limitation, we briefly describe our synthetic data generator. As a hidden Markov model, it first randomly assigns each individual at time $t = 1$ to one of the k communities. For each timestep $t \geq 1$, it generates an observed network G_t based on the probability p_{fn} of false negatives and the probability p_{fp} of false positives. Then, it reassigns community affiliation for the next timestep $t + 1$ based on the probability p_{sw} of switching.

6.4.2 Results

The first question is: If we know the true community structure, how close to the truth is the community structure produced by the SDP method? To answer this, we generate dynamic networks using different p_{sw}, p_{fn}, p_{fp}, and run the SDP method using different C_{sw}, C_{fn}, C_{fp}. Since the results are similar, show only one representative case $p_{\text{sw}} = 0.3$. In Figure 15, the left column shows the distance (Rand index (78)) to the ground truth. In the top row, the SDP method can find the true structure (distance=0), given the right parameter setting. In the other rows, it finds solutions close to the truth, but not identical. This is because of the high perturbation. However, the relationship between ratios $C_{\text{fn}}/C_{\text{fp}}$ and $p_{\text{fn}}/p_{\text{fp}}$ is clear. We observe how the probability ratio $p_{\text{fn}}/p_{\text{fp}}$ (which varies from the top to the bottom rows) affects the cost ratio $C_{\text{fn}}/C_{\text{fp}}$ at which the SDP method achieves the minimum distance to the ground truth. This conforms with the intuition that when $C_{\text{fn}} < C_{\text{fp}}$, the algorithm trades false positives for false negatives (since $p_{\text{fn}} > p_{\text{fn}}$). The right column of Figure 15 shows the upper bound on the approximation ratio of the SDP algorithm to the optimal solution (as discussed in Section 6.3). Note that in some cases, this upper bound is exactly 1, indicating that the SDP method has found an optimal solution. However, the distance to the ground truth (in the corresponding plots in the left column) is greater than 0, indicating that the optimal structures found under those parameter settings do not match the ground truth.

6.4.3 Comparison With Other Methods

We compared our SDP algorithm with other competitive methods for detecting dynamic communities using the two-step approach. First, we ran methods for detecting static communities on each snapshot graph to find a group structure. We use Girvan-Newman algorithm (GN) and Clauset-Newman-Moore algorithm (CNM), in particular, the implementations from SNAP (63). Once we discovered the group structure in each timestep, we ran our branch-and-bound algorithm (95) to find an optimal community interpretation. We vary the parameter settings within a certain range. Although this performs an exhaustive search, our test datasets

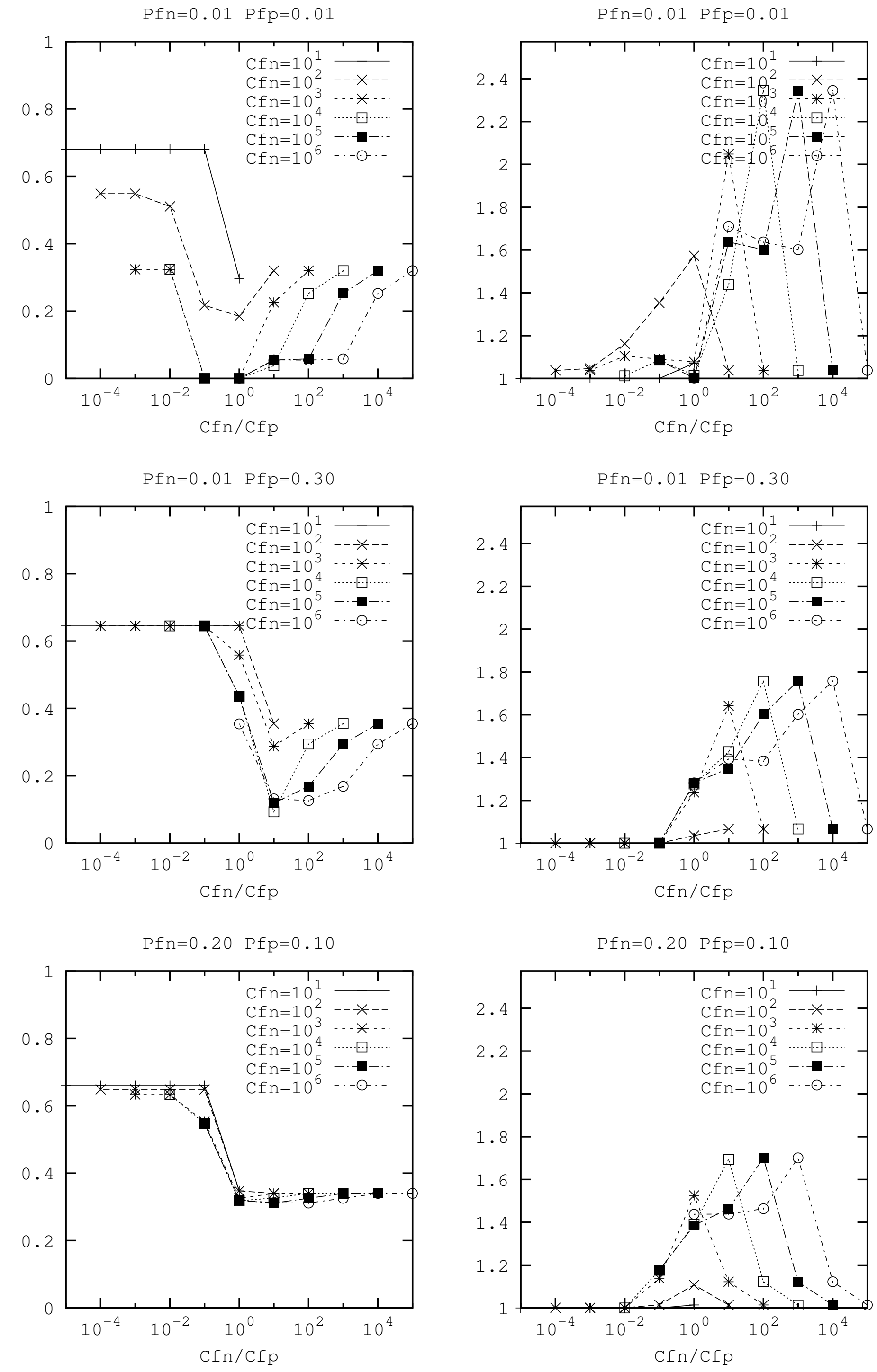


Figure 15: Rand index distance to ground truth (left column) and the upper bound on the approximation ratio (right column) of the SDP algorithm for $p_{\mathrm{sw}} = 0.30, N = 15, T = 10, k = 3$.

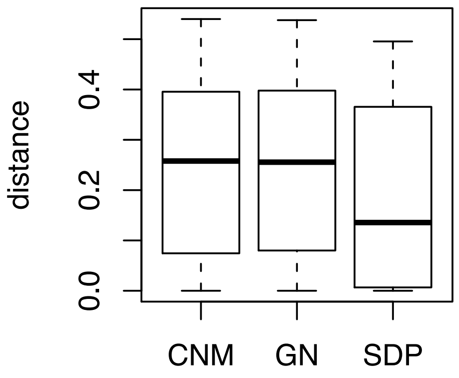


Figure 16: The distance to the ground truth of three algorithms with the best parameter settings, over 10 trials of synthetic data sets with $N = 10, T = 10, k = 2$.

are sufficiently small and the branch-and-bound algorithm can find the optimal solution within one minute in most cases with only few cases that take longer.

We generated 10 dynamic networks for each generator parameter values $p_{\text{sw}}, p_{\text{fn}}$, p_{fp}. For each trial, we ran each method over different parameter settings and take the one that is closest to the ground truth. Figure 16 shows this result. Overall, SDP method outperforms the two other methods.

6.4.4 Experiments on Real Datasets

We have done experiments on two Bluetooth traces from the Haggle project. The results on both traces are similar so we present only one of them (86). We consider only the Bluetooth devices in the project and ignore the other devices since we do not have their complete traces. For each one hour interval, we create an undirected edge (u, v) at the corresponding timestep if u has seen v in this interval. This result in a dynamic network with one has 9 nodes, 467 edges, 84 timesteps. The number of timesteps is still too large for the SDP method to analyze in its entirety, so we split the timeline into four smaller chunks, each is ≈ 20 timesteps. Figure 17 shows the network at the first timestep with nodes color-coded by community affiliation detected by the SDP method. Black solid lines are true positives. Red solid lines are false positives. Red dotted lines are false negatives. As can be seen in the figure, the higher the ratio $C_{\text{fn}}/C_{\text{fp}}$ is, the fewer the false negatives and the more the false positives there are. The fewer the false negatives decreases there are, the smaller and denser the clusters become.

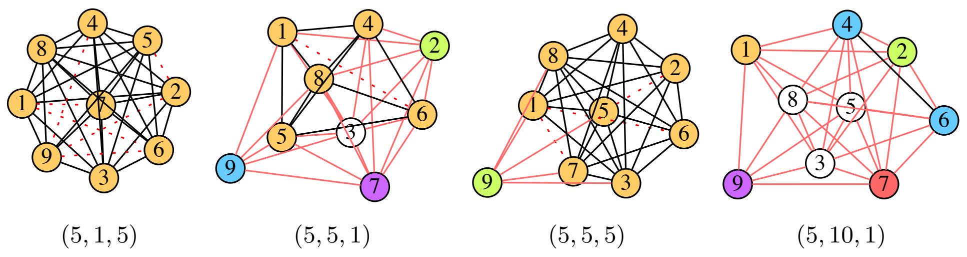


Figure 17: Community structures in the Haggle at the first timestep detected under various $(C_{\text{sw}}, C_{\text{fn}}, C_{\text{fp}})$.

Next we show the community structure in the first few timesteps detected with $(C_{\text{sw}}, C_{\text{fn}}, C_{\text{fp}}) = (5, 5, 5)$, which seems to be a good setting. Figure 18 shows that there is cluster $\{1, 2, 4\text{–}8\}$ which is quite stable over timesteps 4–6.

Next, we vary $C_{\text{sw}} \in \{1, 5\}$ and fix $C_{\text{fn}} = 1$ and $C_{\text{fp}} = 5$. Figure 19 illustrates that that, when switching is cheap ($C_{\text{sw}} = 1$ v.s. $C_{\text{sw}} = 5$), clusters are more stable over time with few

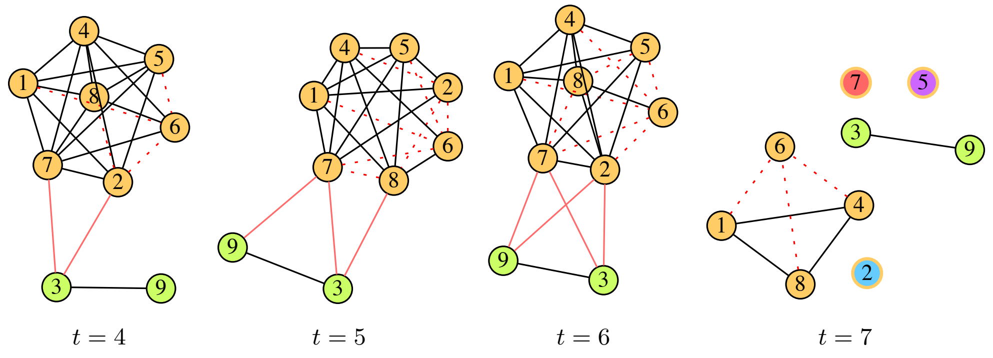


Figure 18: Community structures in the Haggle dataset with parameters $(C_{\text{sw}}, C_{\text{fn}}, C_{\text{fp}}) = (5, 5, 5)$.

nodes changing affiliation. As can be seen in the first column at $t = 33$, the big cluster still exists even when most of its connections are gone. The expensive C_{sw} cost slows down the rate of affiliation changes.

To quantitatively illustrate how our SDP method behaves under different parameter settings, we compute the intra-cluster and inter-cluster densities within each timestep to show how densely connected and well-separated the clusters are. Figures Figure 20 and Figure 21 show the time series of intra-cluster density and inter-cluster density, respectively. The density is computed as the percentage of edges inside the clusters (or between clusters).

6.4.5 Haggle3 Dataset

We run the SDP method on Haggle3 using parameter settings as used in the previous section. Figure 22 shows the community structures at timesteps 5–8 in Haggle 3. We set $C_{\text{sw}} = 5$ and $C_{\text{fn}} = 5$. We can observe that as the ratio $C_{\text{fn}}/C_{\text{fp}}$ decreases, the inter-cluster edges are penalized more and the intra-cluster non-edges are penalized less. As a result, the clusters become larger as they need to include more edges. The big clusters start to form at timestep 6 and disband at timestep 8. A closer look at timesteps 6 and 7 and parameter settings $C_{\text{fn}} = 1$ and 5 tells us more about the dynamics of this cluster. At time 6, the nodes $\{1, 3, 8, 23, 34, 41\}$ are considered a part of the big clusters in both $C_{\text{fp}} = 1$ and $C_{\text{fp}} = 5$. However, at time 7, these nodes leave the big cluster when setting $C_{\text{fp}} = 1$. The setting $C_{\text{fp}} = 5$ still considers these nodes a part of the big cluster. There can be two possible interpretations of node affiliation.

A similar observation can be made about nodes $\{8, 9, 10, 14, 22, 24, 35, 37, 38\}$. These nodes are a part of the big cluster at time 5 when $C_{\text{fp}} = 10$, but not when $C_{\text{fp}} = 5$. Their affiliation with the big cluster is not strong enough to be considered part of the big cluster in the latter parameter setting.

Another interesting observation is in Figure 23. In both parameter settings and most timesteps, there is one big cluster with a few nodes weakly connected to it. Timesteps 25 and 30 are the points where the community structures detected can be ambiguous. With parameter setting $(C_{\text{sw}}, C_{\text{fn}}, C_{\text{fp}}) = (5, 1, 10)$, the SDP detects that the big cluster is still a big cluster. However, with parameter setting $(C_{\text{sw}}, C_{\text{fn}}, C_{\text{fp}}) = (5, 5, 10)$, the previously big cluster is detected as being separated into $2 - -3$ smaller clusters. These clusters are denser and contain only few intra-cluster non-adjacent members. Moreover, these clusters are well connected through several inter-cluster edges. This is a result of the C_{fn} cost becoming more expensive unilaterally.

6.5 Summary

We extend our previous work on finding communities in dynamic social networks (7) to arbitrary networks, getting rid of the assumption that the individuals in each timestep are partitioned into groups. To the best of our knowledge, our method is the first that is based on social theory. We formulate the Network Community Interpretation problem and devise an approximation algorithm via SDP relaxation and a heuristic rounding scheme. The algorithm produces a community interpretation with an approximation guarantee. We show empirically that the method produces solutions which are near optimal and close to the ground truth of the synthetic data. Our method outperforms the two-step approach. We also show how the SDP method can be used to find dynamic communities in a real dataset. A natural question one may have is: How good is the theoretical approximation guarantee? Another future direction is the available SDP solvers are not scalable and speeding up SDP solvers is an active area of research. Thus, speeding up our algorithm will need a new angle.

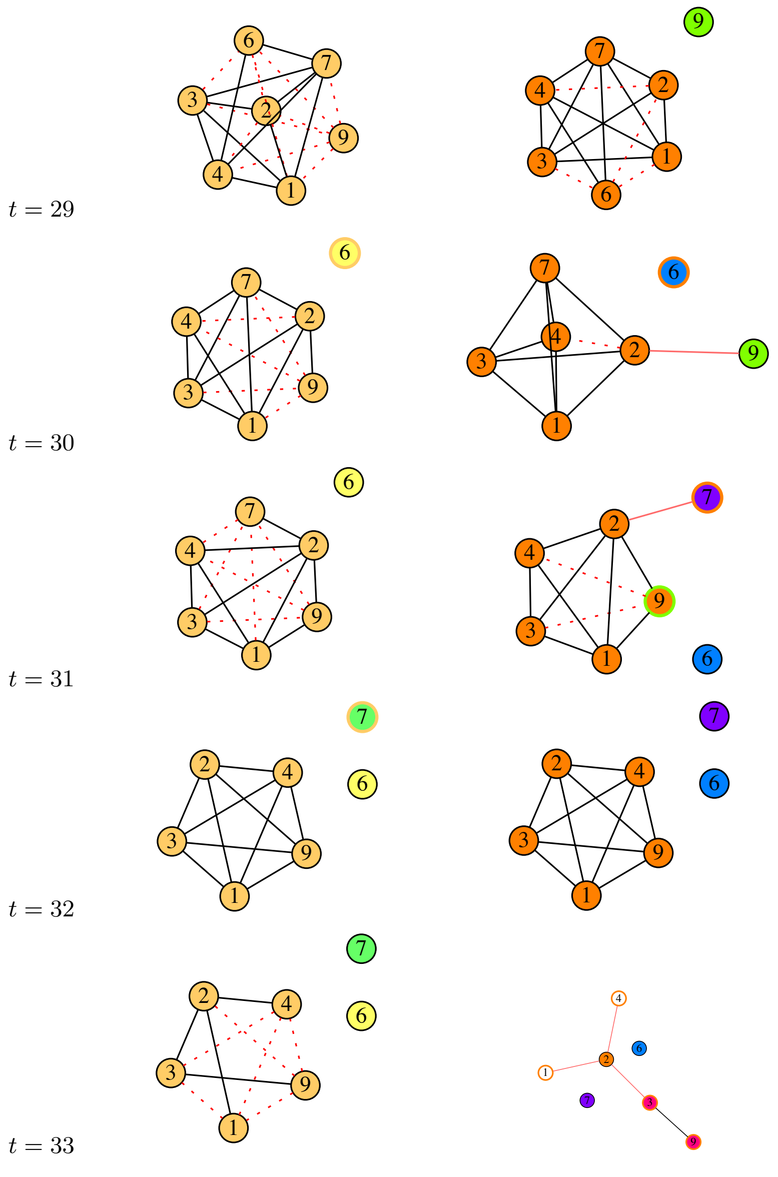
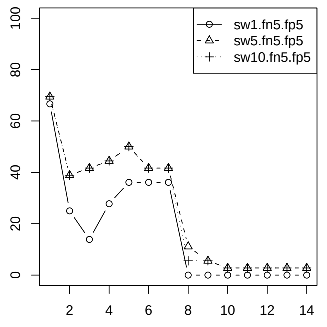
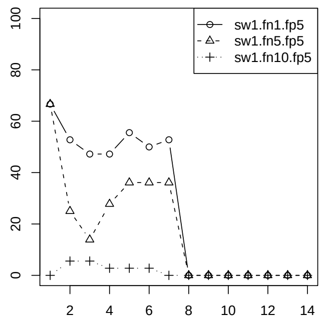


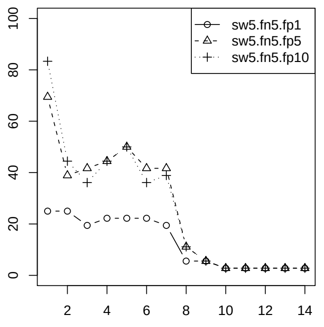
Figure 19: Community structures in Haggle dataset at timesteps 29–33. The first column is detected with $(C_{\text{sw}}, C_{\text{fn}}, C_{\text{fp}}) = (5, 1, 5)$ and the second column is detected with $(C_{\text{sw}}, C_{\text{fn}}, C_{\text{fp}}) = (1, 1, 5)$.



$C_{\text{sw}} \in \{1, 5, 10\}$, $C_{\text{fn}} = 5$, $C_{\text{fp}} = 5$

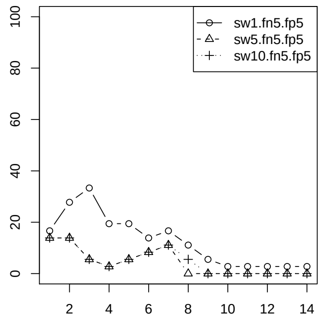


$C_{\text{sw}} = 1$, $C_{\text{fn}} \in \{1, 5, 10\}$, $C_{\text{fp}} = 5$

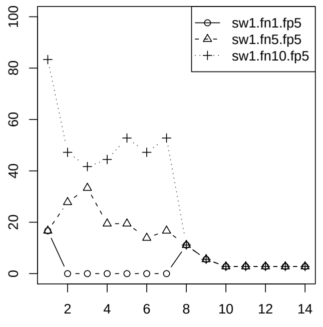


$C_{\text{sw}} = 5$, $C_{\text{fn}} = 5$, $C_{\text{fp}} \in \{1, 5, 10\}$

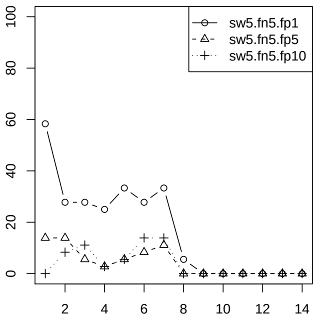
Figure 20: Intra-cluster density at timesteps 1–14 in the Haggle dataset.



$C_{\text{sw}} \in \{1, 5, 10\}$, $C_{\text{fn}} = 5$, $C_{\text{fp}} = 5$



$C_{\text{fn}} = 1$, $C_{\text{fn}} \in \{1, 5, 10\}$, $C_{\text{fp}} = 5$



$C_{\text{sw}} = 5$, $C_{\text{fn}} = 5$, $C_{\text{fp}} \in \{1, 5, 10\}$

Figure 21: Inter-cluster density over timesteps 1–14 in the Haggle dataset.

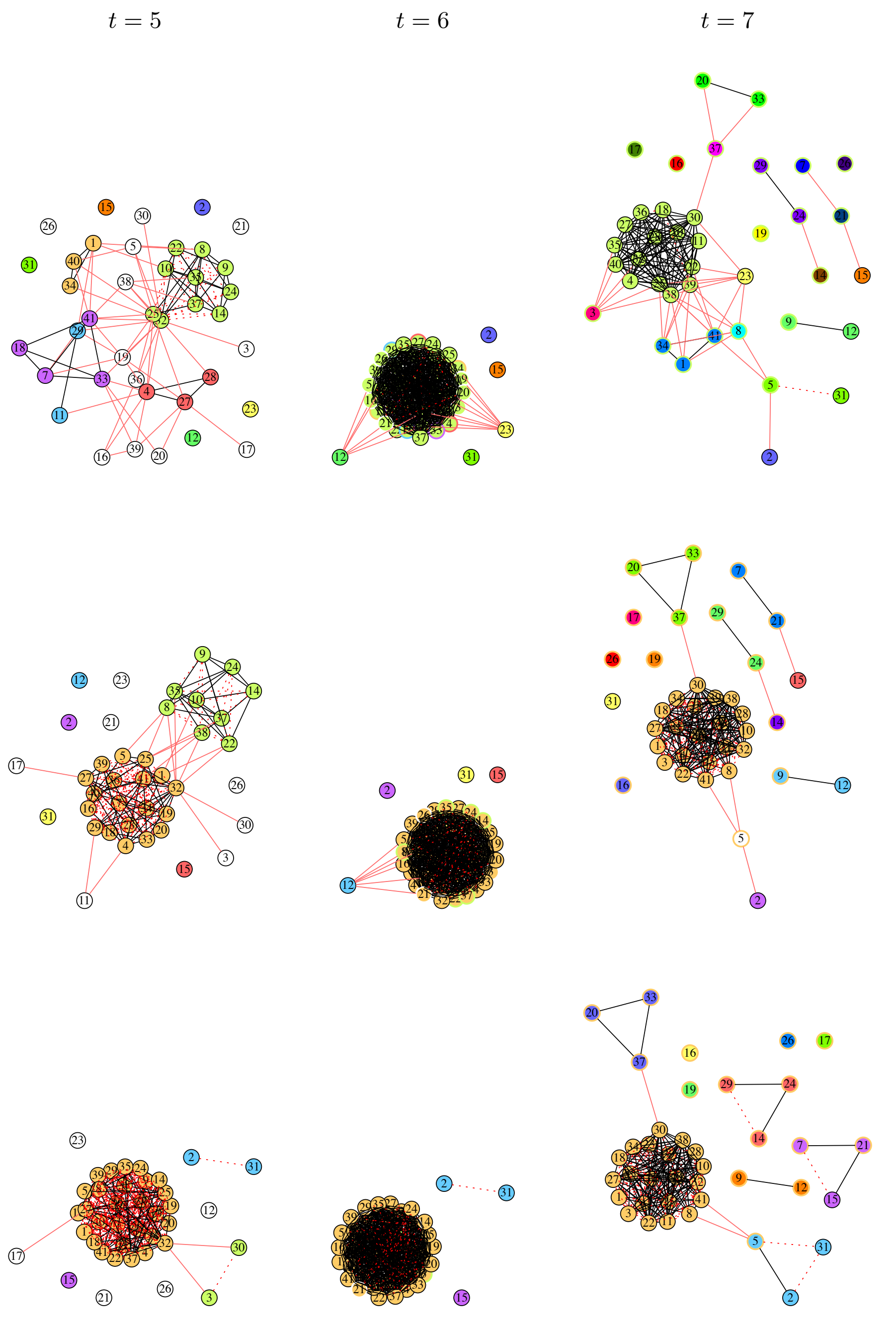


Figure 22: Haggle3. The timesteps are 5–7. Parameters are set as follows. In all three rows, $C_{\text{sw}} = 5$, $C_{\text{fn}} = 1$. From the top to the bottom row, $C_{\text{fp}} = 1, 5, 10$. As C_{fp} increases, the cluster becomes larger as the inter-cluster edges are penalized more and intra-cluster non-edges are penalized less.

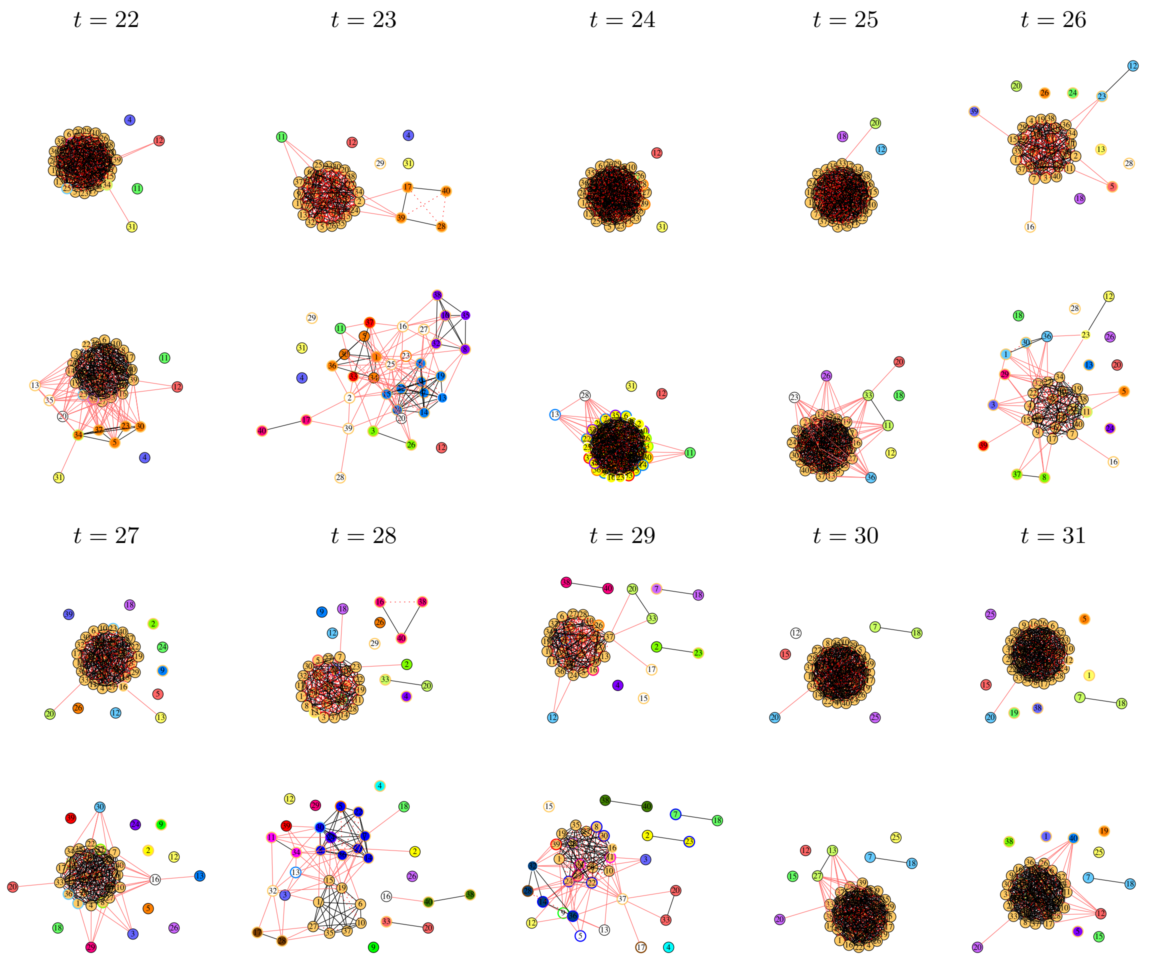


Figure 23: Haggle3, timesteps 22–31. The first and third rows are detected with parameters $C_{\text{sw}} = 5, C_{\text{fn}} = 1, C_{\text{fp}} = 10$. The second and forth rows are detected with parameters $C_{\text{sw}} = 5, C_{\text{fn}} = 5, C_{\text{fp}} = 10$. There are noteworthy differences between these two parameter settings at timesteps 25 and 30. The former gives one big, sparser cluster while the latter gives several smaller, denser clusters.

CHAPTER 7

CONCLUSION

In this chapter, I summarize the state of my work and propose future directions. I have presented a survey on the existing literature on community detection in both static networks and dynamic networks. There is not much work which explicitly considers the temporal aspect of social networks, especially for the community detection problem. Most work still tries perfecting models or algorithms, assuming that the networks and their community structures as essentially static. I emphasize the importance of time and refer the observation of Moore (67) that social analysis should consider time as both a boundary condition and a measure of persistence and change. No samples are perfectly i.i.d., thus it is important to model the data at hand as generated from a latent structure which changes over time. In my setting, it is important to detect communities in social networks and allow the inferred community structure to evolve.

7.1 Summary of Contributions

This thesis yields solutions to two versions of the general problem of detecting and tracking communities over time. The first is the problem of tracking communities over time, assuming that communities in each snapshot are already detected. The other is the problem of simultaneously detecting and tracking communities over time. In the following, I summarize my contributions for each problem separately.

7.1.1 Tracking Communities

When the communities in each snapshot are already detected, the problem is to string the communities across time, answering the question: Which community becomes which community? I have presented two frameworks for tracking communities over discrete timesteps.

The first framework assumes that timesteps are of uniform length, thus simplifying the math involved. The framework is grounded on social costs which are well-motivated by both sociology and behavioral ecology. I have presented approximation algorithms based on bipartite matching and path cover which guarantee solutions to be within constant factors of the optimal solutions. I also presented a post-processing step via a dynamic programming to optimally color the individuals further.

The other framework allows timesteps to have variable length. I have presented a framework which has a probabilistic interpretation and formulated an optimization whose solution is a maximum likelihood estimate of the probabilistic model. Then, I proposed the UIC algorithm, a fast heuristic algorithm which was originally designed to maximize only the modularity. I showed that this algorithm can be used to solve the CORRELATION CLUSTERING problem, and to solve the problem of tracking communities over time. Then, I showed the approximation performance of the UIC algorithm by comparing its solutions with the bounds on the optimal solutions. The results showed that the UIC algorithm performs surprisingly well in all cases that I have considered.

Both are the only work on the subject which can compute the approximation ratios of the solutions and allow some low-confidence clusters to be omitted from the input.

7.1.2 Detecting and Tracking Communities

For the setting in which we have a dynamic social network, I have presented a framework which has a probabilistic interpretation and formulated an optimization problem whose solution is a maximum likelihood estimate of the probabilistic model. I showed that we can used approximation algorithms for Correlation Clustering problem to solve this optimization problem and also bound the optimal solutions. This is the only work on this problem which can compute the approximation factors of the solutions.

7.2 Future Directions

There are many directions which one may take from here. Again, I discussed about future directions for each problem separately in the first two subsections that follow. The rest of the section are about miscellaneous future directions.

7.2.1 Tracking Communities

The question about the time complexity of the Louvain algorithm, as well as the UIC algorithm, is still an open one. Although each iteration of the Louvain algorithm has time complexity of $O(m)$ where m is the number of edges, the time complexity of the entire algorithm is still unknown. The algorithm terminates, but nobody knows how many iterations it takes in the worst case. Although the original authors observed that the time complexity is linear in their experiments on synthetic data (9), the time complexity $O(m)$ listed in the latest survey paper (38) is rather an erroneous one since, otherwise, it would imply that the algorithm always takes a constant number of iterations in the worst case, a rather plausible contradiction.

I also have noted the difference between minimization problems and maximization problems in terms of their approximation ratios. Although I used Swamy's approximation algorithms for Correlation Clustering (which are for a maximization problem) to compute bounds for the problem of tracking communities (a minimization problem, technically), the constant factors of Swamy's algorithms do not translate into constant factors in my setting since the optimization sense changes. It is still an open question how to bound the approximation guarantee of both Swamy's algorithms and the UIC algorithm in my setting.

Another interesting question is, can we solve the dual of the SDP relaxation, heuristically? Is there an algorithm similar to the UIC algorithm which can heuristically solve the dual SDP? The pair of the UIC algorithm and its "dual" algorithm will be a more efficient and more scalable alternative than the UIC algorithm and the interior-point method for SDP.

7.2.2 Detecting and Tracking Communities

The UIC algorithm is applicable to a wide variety of problems, especially those which involve maximum likelihood estimates or maximum a posterior probability estimates. Both MLE and MAP problems can be reduced to the Correlation Clustering problem in which the UIC algorithm is applicable, and so is the SDP bounding technique. For the problem of detecting and tracking communities, my previous algorithm is based on solving an SDP relaxation using the interior-point method and a rounding via hierarchical clustering. I expect that the UIC algorithm will outperform my previous algorithm by an order of magnitude.

As I mentioned that the UIC algorithm is applicable to any MLE and MAP problems, it would be interesting to try other probabilistic models which simultaneously generate dynamic networks and communities structures such as those by Lin et al. (64) and Xu et al. (103). It would be interesting to compare the performance of the UIC algorithm and other algorithms such as those based on the EM algorithm.

7.2.3 Miscellaneous

In Chapter 4, I formulated the DYNAMIC COMMUNITY INTERPRETATION problem with a constraint that a coloring has to be *proper*. Recall that, a proper coloring does not assign the same color to two or more groups at each timestep. This constraint originated rather only for technical convenience to justify the use of discrete objects such as matching and path cover to devise approximation algorithms. We can always relax this constraint to allow two or more groups at the same timestep to have the same color. This will allow the framework to take into account splitting of a community and merging of communities in a rigorous manner. Fortunately, the UIC algorithm can still find a solution to this relaxed problem.

An ability to track communities over time in the way that I formulated allows us to do more accurate job at detecting communities in each snapshot. We are not constrained to find a partition of the entire snapshot anymore. We can identify only the communities with high confidence level and omit the ones with low confidence level. The algorithms for tracking communities are still able to track them over time. This not only eliminates artificial communities from the snapshot but also improve the efficiency of the whole process.

APPENDICES

Appendix A

DATASETS

A.1 Social Networks

A.1.1 Southern Women

Southern Women dataset. Collected by Davis et al. (26). Meta-analysis by Freeman (41). Affiliation network is shown in Table III. Table IV shows the assignments of individuals into three communities A, B, and C by the 21 static methods by Freeman.

A.2 Animal Sightings

A.2.1 Grevy's zebra

The Grevy's zebra (*Equus grevyi*) dataset (92) was collected by observing a population of Grevy's zebra in Laikipia, Kenya, from June to August in 2002. Predetermined loops were driven typically once a day by ecologists making visual scans of the herds. Individual zebras were identified by the pattern of stripes (60). An interaction represents social association, as determined by spatial proximity and the domain knowledge of ecologists.

A.2.2 Onager

The onager (*Equus hemionus khur*) dataset (92) was collected by observing a population of onagers (Asiatic wild asses) in the Little Rann of Kutch desert in Gujarat, India, from January to May in 2003. Individual onagers were identified by markings on their body and, similar to zebras, the data represent visual scans of the population by ecologists, typically once a

Appendix A (Continued)

Names of Participants	Social events in *Old City Herald*													
	6/27	3/2	4/12	9/26	2/25	5/19	3/15	9/16	4/8	6/10	2/23	4/7	11/21	8/3
1. Mrs. Evelyn Jefferson	x	x	x	x	x	x		x	x					
2. Miss Laura Mandeville	x	x	x		x	x	x	x						
3. Miss Theresa Anderson		x	x	x	x	x	x	x	x					
4. Miss Brenda Rogers	x		x	x	x	x	x	x						
5. Miss Charlotte McDowd			x	x	x		x							
6. Miss Frances Anderson			x		x	x		x						
7. Miss Eleanor Nye					x	x	x	x						
8. Miss Pearl Oglethorpe						x		x	x					
9. Miss Ruth DeSand					x		x	x	x					
10. Miss Verne Sanderson							x	x	x			x		
11. Miss Myra Liddell								x	x	x		x		
12. Miss Katherine Rogers								x	x	x		x	x	x
13. Mrs. Sylvia Avondale							x	x	x	x		x	x	x
14. Mrs. Nora Fayette						x	x		x	x	x	x	x	x
15. Mrs. Helen Lloyd							x	x		x	x	x		
16. Mrs. Dorothy Murchison								x	x					
17. Mrs. Olivia Carleton									x		x			
18. Mrs. Flora Price									x		x			

TABLE III: Reproduction of the Southern Women dataset

Method	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1. DGG41	A	A	A	A	A	A	A	A	AB	B	B	B	B	B	B	B	B	B
2. HOM50	A	A	A	A	A	A	A	AB			B	B	B	B	B		B	B
3. P&c72	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B
4. BGR74	A	A	A	A	A	A	A		A	B	B	B	B	BC	BC		C	C
5. BBA75	A	A	A	A	A	A	A	B	A	B	B	B	B	B	B	B	B	B
6. BCH78	A	A	A	A	A	A				B	B	B	B	B	B			
7. DOR79	A	A	A	A	A	A	A		A	B	B	B	B	B	B			
8. BCH91	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B
9. FRE92	A	A	A	A	A	A	A		A	B	B	B	B	B	B	B		
10. E&B93	A	A	A	A	A	A	A		A	B	B	B	B	B	B			
11. FR193	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B
12. FR293	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B	B	B
13. FW193	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	AB	B	B
14. FW293	A	A	A	A	A	A	A		A	B	B	B	B	B	B		B	B
15. BE197	A	A	A	A	A	A	A		A	B	B	B	B	B	B			
16. BE297	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B
17. BE397	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B
18. S&F99	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B		B	B
19. ROB00	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B
20. OSB00	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	B	B
21. NEW01	A	A	A	A	A	A	A	B	A	B	B	B	B	B	B	B	B	B

TABLE IV: Freeman's 21 static community assignments in Southern Women.

day. An interaction represents an association as determined by physical proximity and domain knowledge.

TABLE V: Datasets and their statistics in Appendix A

Statistics	Grevy's	Onager
Number of sightings	130	408
Number of timestamps	130	405
Number of individuals	17	29
Avg. sighting size	3.91	1.94
Avg. sightings per individual	29.88	27.31

A.2.3 Plains Zebra

The plains zebra (*Equus burchelli*) dataset was collected by observing the population in Kenya from 2003 to 2008 (37; 53) in a manner similar to the Grevy's zebra dataset.

A.3 Bluetooth Proximity Networks

In general, the range of Bluetooth is 5–30 meters (10).

A.3.1 Reality Mining

We use three networks derived from Bluetooth proximity log. The data were collected by the MIT Media lab (31). They consist of communication, proximity, location, and activity information from 100 subjects at MIT over the course of the 2004–2006 academic years.

A.3.2 Haggle

The Haggle dataset consists of social interactions among attendees of the 2005 IEEE Infocom conference who were carrying a Bluetooth-enabled device which record handshakes with other Bluetooth devices during the conference (85). Two separate datasets represent interactions at this event. One consists only of 41 participants and the other consists of 264 nodes, containing the 41 participants and the other divides seen by the first 41 devices. The participants were tracked over the full 4 days of the conference at 10 minute intervals.

Appendix B

INTERVAL LOWERBOUND

This is the same exhaustive search as in (95; 96). A new contribution is the use of divide-and-conquer together with dynamic programming to improve the lowerbound in the branch-and-bound exhaustive search. The idea is we want to partition the dataset along the time axis into smaller datasets such that they are small enough that the exhaustive search terminates quickly. In particular, we want to find a partition of the time interval $[1, m]$ into disjoint intervals $[1, t_1 - 1], [t_1, t_2 - 1], [t_2, t_3 - 1], \ldots, [t_k, m]$ so that the exhaustive search on each interval terminates within a certain small time limit, say one second. In particular, we will attempt the exhaustive search with time limit on all $I \subseteq [1, m]$. Let $c(I)$ be the minimum coloring cost on an interval I computed by the exhaustive search, setting $c(I) = \infty$ if the exhaustive search does not terminates in time. We will use $c(I)$ to construct a lowerbound on $c([1, m])$.

Let us fix any optimal coloring χ with the minimum cost $c([1, m])$. Let $c_\chi(t, t')$ denote the coloring cost of χ when restricting the dataset to the interval $[t, t']$. First, we observe that, for each interval I with $c(I) < \infty$, we have

$$c(I) \leq c_\chi(I) \tag{B.1}$$

since χ gives a feasible solution to the problem in which $c(I)$ is optimal. With this in mind, we want to find a partition $\mathcal{I}$ of $[1, m]$ into disjoint intervals which maximizes $c(\mathcal{I})$ since,

$$c(\mathcal{I}) = \sum_{I \in \mathcal{I}} c(I) \leq \sum_{I \in \mathcal{I}} c_\chi(I) \leq c_\chi([1, m]) = c([1, m]).$$

The first inequality follows from Equation B.1. The second inequality holds since the quantity on the LHS accounts for a subset of the quantity on the RHS. Let $\ell(t, t')$ denote a lowerbound on $c_\chi([t, t'])$. We can express $\ell(t, t')$ recursively as,

$$\ell(t, t') = \begin{cases} c([t, t']) & \text{if } c([t, t']) < \infty, \\ \max_{i=1,\dots,t'-t} \{\ell(t, t+i-1) + \ell(t+i, t')\} & \text{otherwise.} \end{cases}$$

To make sure that the recursion is well-defined, we make sure that the time limit on the exhaustive search is long enough that it terminates on all intervals $[t, t+1]$. (We cannot rely on $c([t, t])$ since it is zero.) This recursion can be computed using dynamic programming by filling the table in increasing order of interval lengths $t' - t$. We can use $\ell([1, m])$ as a lowerbound to compute the approximation guarantee of the solution by any algorithm. Furthermore, we can use $\ell([t, t'])$ in branch-and-bound (95; 96).

Appendix C

CORRELATION CLUSTERING

This is an introduction to Swamy's algorithm for Correlation Clustering (93), followed by the details of my implementation.

C.1 General Approach for Dividing an Approximation Algorithm Using SDP

Semidefinite programming is a popular tool for devising approximation algorithms especially for graph problems. Introduced by Goemans and Williamson (47), the technique is to first formulate the problem as a strict quadratic program, which is then relaxed to a vector program. Such vector program relaxation has an equivalent formulation as a semidefinite program (SDP), which is solvable in polynomial time by the interior-point method (50). Once we obtain an optimal solution to the SDP, we perform the Choleskey factorization on it to obtain an optimal solution to the vector program relaxation, which is, in turn, *rounded* by a randomized algorithm to get a solution to the strict quadratic program, which gives a solution to the original problem. The approximation guarantee is analytically obtained by comparing the expectation of the rounded solution to the optimal solution to the SDP.

C.2 SDP Relaxation for Correlation Clustering

Swamy gave two approximations algorithms, both are based on the standard approach for designing an approximation algorithm using semidefinite programming relaxation. The only difference between the two is the rounding schemes. First, the Maximizing Agreement problem is formulated as a strict quadratic program (QP), which is then relaxed to a vector program (VP). The VP relaxation has an equivalent formulation as a semidefinite program (SDP) which can be solved to any arbitrary precision in polynomial time using the interior-point method (50). Once an optimal solution to the SDP is obtained, Cholesky factorization is used to obtain an optimal solution to the VP. Then, the optimal solution to the VP is rounded to obtain a solution to the QP. The rounded solution is also a solution to Maximizing Agreement and Dynamic Community Interpretation. The two rounding schemes that give rise to two slightly different algorithms are k-hyperplane rounding and k-center rounding. We refer the reader to (93) for more details on algorithms and the rounding schemes.

Now we describe Swamy's SDP relaxation for Maximizing Agreement. All matrices in this section are n-by-n real symmetric matrices unless noted otherwise. For a matrix X, let x_{ij} denote its entry at row i and column j. We write $X \geq 0$ if all entries x_{ij}'s are nonnegative. We write $X \succeq 0$ if X is positive semidefinite, that is, $x^T X x \geq 0$ for all $x \in \mathbb{R}^n$. For two matrices X and Y, we write $X \succeq Y$ if $X - Y$ is positive semidefinite. The matrix inner product is $X \bullet Y = \operatorname{tr} XY = \sum_i \sum_j x_{ij} y_{ij}$. Consider a weighted undirected graph $G = (V, E)$. The edge weights are represented as a matrix C where c_{ij} is the weight of $(i, j) \in E$. Let $\{e_1, \ldots, e_n\}$ be the standard basis for $\mathbb{R}^n$, that is, e_i is the unit vector whose i^{th} element is 1. Swamy's SDP relaxation for Maximizing Agreement on G is:

(Primal SDP)

$$\begin{array}{lrl} \text{maximize:} & C \bullet X & \\ \text{subject to:} & e_i e_i^T \bullet X = 1 & 1 \leq i \leq n \\ & (e_i e_j^T + e_j e_i^T) \bullet X \geq 0 & \{i,j\} \in E \\ & X \succeq 0 & \end{array}$$

(Dual SDP)

$$\begin{array}{lrl} \text{minimize:} & I \bullet Z & \\ \text{subject to:} & (e_i e_j^T + e_j e_i^T) \bullet Z \leq 0 & \{i,j\} \in E \\ & Z \succeq C & \end{array}$$

The interior-point method (50) can be used to solve an SDP in time polynomial in the size of the problem and the desired precision of the solution. Once we have an optimal solution to the SDP, we use the following rounding schemes to obtain a solution to the original problem of Maximizing Agreement.

We solve the SDP relaxation using CSDP (11) an efficient implementation of a primal-dual algorithm for SDP written in C. CSDP solves an SDP only with equality constraints, so we use a standard technique to transform an inequality constraint to an equality constraint. We replace each inequality constraint $A_i \bullet X \geq 0, i = 1, \ldots, m$, with the following equality constraint where y_i's are slack variables.

$$A_i' \bullet X' := \begin{bmatrix} A_i & \\ & -e_i^T e_i \end{bmatrix} \bullet \begin{bmatrix} X & & & \\ & y_1 & & \\ & & \ddots & \\ & & & y_m \end{bmatrix} = 0.$$

Here, $\{e_1, \ldots, e_m\}$ is the standard basis of $\mathbb{R}^m$. The constraint $X' \succeq 0$ ensures that the slack variables y_i are nonnegative, implying the replaced inequality constraints. Since CSDP appropriately handles block matrices with a combination of dense, sparse, and diagonal blocks, this does not introduce any extra complexity. The primal SDP becomes.

$$\begin{array}{lll} \text{maximize:} & \begin{bmatrix} C & \\ & 0 \end{bmatrix} \bullet X' & \\ \text{subject to:} & \begin{bmatrix} e_i e_i^T & \\ & 0 \end{bmatrix} \bullet X' = 1 & 1 \leq i \leq n \\ & \begin{bmatrix} e_i e_j^T + e_j e_i^T & \\ & -e_k e_k^T \end{bmatrix} \bullet X' = 0 & e_k = (i,j) \in E \\ & X' \succeq 0 & \end{array}$$

We compiled CSDP using Intel C++ Composer XE 2013 with Math Kernel Library 11.0. We ran the solver on a machine with four 8-core Intel Xeon 2GHz CPUs and 64GB RAM. The average solving time is 1.45 minutes on Grevy's zebra and 138 minutes on onager. The big difference is because each iteration of the interior-point method involves $O(m^3)$ factorization of a dense m-by-m Schur complement matrix where m is the number of constraints. In our case, m is equal to $|V| + |E|$.

C.3 Rounding Schemes

After we obtain an optimal solution X to the primal SDP, we compute the Cholesky factorization $Y^T Y = X$ where Y is an upper triangular matrix. This can be done in $O(n^3)$ and efficient implementations are available from LAPACK packages. Let $y_{\bullet j}$ denote the j^{th} column of Y. Since the diagonal entries of X are constrained to be one, the vectors $y_{\bullet j}$'s are points on the surface of the unit $(n-1)$-hypersphere. A rounding scheme, in general, partitions the space and the $y_{\bullet j}$'s into parts. Then, the $y_{\bullet j}$'s in each part are assigned to one cluster. For an SDP relaxation with unitary constraints on the diagonal entries like ours, there are two popular rounding schemes: hyperplane rounding and center rounding.

C.3.1 Hyperplane Rounding

This rounding scheme was first introduced by Goemans and Williamson for Max k-Cut problem. We generate k random unit vectors $v_1, \ldots, v_k \in \mathbb{R}^n$ as follows. We first draw each component independently from the standard normal distribution. The probability that each $v_i = 0$ is zero. In practice, we reject and resample until $v_i \neq 0$. Then, we normalize each vector into a unit vector. Each vector v_i serves as the normal vector of a hyperplane which altogether divide the space into at most 2^k parts (some parts may not contain any point $y_{\bullet j}$). Recall that the points $y_{\bullet j}$'s correspond to the vertices in the sighting graph. We then assign all the points in each part to the same cluster.

C.3.2 Center Rounding

This rounding scheme was first introduced by Karger, Motwani and Sudan for Graph Coloring problem. We start by generating k random vectors $v_1, \ldots, v_k \in \mathbb{R}^n$ by drawing each coordinate independently from the standard normal distribution. Here, we do not normalize the vectors as we did previously. They will serve as centers of clusters. Then, we assign each $y_{\bullet j}$ to the nearest v_j, breaking ties lexicographically.

Swamy shows that 2-hyperplane rounding produces a solution which is at least 0.75 of the optimal value in expectation. By randomly choosing between 2-hyperplane rounding and 6-center rounding, the solution is at least 0.7666 of optimal value in expectation. In later section, we will look at how these two rounding schemes perform in practice.

Technically, k-hyperplane (k-center) rounding produces at most 2^k clusters (k clusters). However, these clusters are not necessarily connected. To be fair, we have to count each connected component in each part as a cluster. This will allow extreme cases to happen, for example, the one in which all nodes are in the same cluster or the one in which all nodes are in their own clusters by themselves.

CITED LITERATURE

1. Aggarwal, C. C. and Yu., P. S.: Online analysis of community evolution in data streams. In Proceedings of the SIAM International Data Mining Conference (SDM'05), 2005.

2. Airoldi, E. M., Blei, D. M., Fienberg, S. E., and Xing, E. P.: Mixed membership stochastic blockmodels. Journal of Machine Learning Research, 9:1981–2014, June 2008.

3. Arnoldi, W. E.: The principle of minimized iterations in the solution of the matrix eigenvalue problem. Quarterly of Applied Mathematics, 9(1):17–29, 1951.

4. Asur, S., Parthasarathy, S., and Ucar, D.: An event-based framework for characterizing the evolutionary behavior of interaction graphs. ACM Transactions on Knowledge Discovery from Data (TKDD), 3(4):16:1–16:36, December 2009.

5. Backstrom, L., Huttenlocher, D., Kleinberg, J., and Lan, X.: Group formation in large social networks: membership, growth, and evolution. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '06, pages 44–54. ACM, 2006.

6. Bansal, N., Blum, A., and Chawla, S.: Correlation clustering. Machine Learning, 56(1):89–113, 2004.

7. Berger-Wolf, T., Tantipathananandh, C., and Kempe, D.: Dynamic community identification. In Link Mining: Models, Algorithms, and Applications, eds, P. S. Yu, J. Han, and C. Faloutsos, pages 307–336. Springer New York, 2010.

8. Berger-Wolf, T. and Saia, J.: A framework for analysis of dynamic social networks. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '06, pages 523–528. ACM, 2006.

9. Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment, 2008(10):P10008, 2008.

10. Bluetrace: Bluetooth range: 100m, 1km, or 10km? http://www.bluair.pl/bluetooth-range, March 2013.

11. Borchers, B. and Young, J.: Implementation of a primal–dual method for sdp on a shared memory parallel architecture. Computational Optimization and Applications, 37:355–369, 2007. 10.1007/s10589-007-9030-3.

12. Breiger, R. L., Boorman, S. A., and Arabie, P.: An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling. Journal of Mathematical Psychology, 12(3):328–383, 8 1975.

13. Brown, H.: People, groups, and society. Open University Press, 1985.

14. Brown, R.: Group Processes: Dynamics Within and Between Groups. Wiley, 2000.

15. Bruhn, J.: The Sociology of Community Connections. Springer Science+Business Media B.V., 2011.

16. Caldarelli, G. and Veespignani, A.: Large Scale Structure and Dynamics of Compex Networks: From Information Technology to Finance and Natural Science. Complex Systems and Interdisciplinary Science Series. World Scientific Publishing Company, Incorporated, 2007.

17. Caro, T.: Cheetahs of the Serengeti Plains: Group Living in an Asocial Species. Wildlife Behavior and Ecology series. University of Chicago Press, 1994.

18. Chakrabarti, D., Kumar, R., and Tomkins, A.: Evolutionary clustering. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '06, pages 554–560, New York, NY, USA, 2006. ACM.

19. Charikar, M., Guruswami, V., and Wirth, A.: Clustering with qualitative information. Journal of Computer and System Sciences, 71(3):360–383, 2005.

20. Chi, Y., Song, X., Zhou, D., Hino, K., and Tseng, B. L.: Evolutionary spectral clustering by incorporating temporal smoothness. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '07, pages 153–162, New York, NY, USA, 2007. ACM.

21. Clauset, A., Newman, M. E. J., and Moore, C.: Finding community structure in very large networks. Physical Review E, 70:066111, Dec 2004.

22. Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C.: Introduction to algorithms. Cambridge, MA, USA, MIT Press, 2001.

23. Croft, D. P., Arrowsmith, B. J., Bielby, J., Skinner, K., White, E., Couzin, I. D., Magurran, A. E., Ramnarine, I., and Krause, J.: Mechanisms underlying shoal composition in the trinidadian guppy, poecilia reticulata. Oikos, 100(3):429–438, 2003.

24. Croft, D. P., James, R., and Krause, J.: Exploring animal social networks. Princeton University Press, 2010.

25. Danon, L., Díaz-Guilera, A., Duch, J., and Arenas, A.: Comparing community structure identification. Journal of Statistical Mechanics: Theory and Experiment, 2005(09):P09008, 2005.

26. Davis, A., Gardner, B. B., and Gardner, M. R.: Deep South. Chicago, IL, The University of Chicago Press, 1941.

27. Demaine, E. and Immorlica, N.: Correlation clustering with partial information. In Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques, eds, S. Arora, K. Jansen, J. Rolim, and A. Sahai, volume 2764 of Lecture Notes in Computer Science, pages 71–80. Springer Berlin / Heidelberg, 2003. 10.1007/978-3-540-45198-3_1.

28. Dempster, A. P., Laird, N. M., and Rubin, D. B.: Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society. Series B (Methodological), 39(1):pp. 1–38, 1977.

29. Derényi, I., Palla, G., and Vicsek, T.: Clique percolation in random networks. Physical Review Letters, 94:160202, Apr 2005.

30. Diestel, R.: Graph Theory, 4th Edition, volume 173 of Graduate texts in mathematics. Springer, 2012.

31. Eagle, N. and Pentland, A.: Reality mining: Sensing complex social systems. Journal of Personal and Ubiquitous Computing, 2006.

32. Elsner, M. and Schudy, W.: Bounding and comparing methods for correlation clustering beyond ilp. In Proceedings of the Workshop on Integer Linear Programming for Natural Langauge Processing, ILP '09, pages 19–27, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

33. Emanuel, D. and Fiat, A.: Correlation clustering – minimizing disagreements on arbitrary weighted graphs. In Algorithms - ESA 2003, eds, G. Battista and U. Zwick, volume 2832 of Lecture Notes in Computer Science, pages 208–220. Springer Berlin Heidelberg, 2003.

34. Erdos, P. and Renyi, A.: On random graphs. Publicationes Mathematicae, 6:290–297, 1959.

35. Falkowski, T.: Community Analysis in Dynamic Social Networks. Dissertation, University Magdeburg, 2009.

36. Falkowski, T., Bartelheimer, J., and Spiliopoulou, M.: Mining and visualizing the evolution of subgroups in social networks. Web Intelligence, 2006.

37. Fischhoff, I. R., Sundaresan, S. R., Cordingley, J., and Rubenstein, D. I.: Habitat use and movements of plains zebra (*equus burchelli*) in response to predation danger from lions. Submitted, 2007.

38. Fortunato, S.: Community detection in graphs. Physics Reports, 486(3–5):75–174, 2 2010.

39. Fortunato, S. and Castellano, C.: Community structure in graphs. In Encyclopedia of Complexity and Systems Science, ed. R. A. Meyers, pages 1141–1163. Springer, 2009.

40. Franzblau, D. S. and Raychaudhuri, A.: Optimal hamiltonian completions and path covers for trees, and a reduction to maximum flow. The ANZIAM Journal, 44:193–204, September 2002.

41. Freeman, L.: Finding social groups: A meta-analysis of the southern women data. In Dynamic Social Network Modeling and Analysis, eds, R. Breiger, K. Carley, and P. Pattison. Washington, D.C., The National Academies Press, 2003.

42. Freeman, L. C.: A set of measures of centrality based on betweenness. Sociometry, pages 35–41, 1977.

43. Freeman, L. C.: The sociological concept of "group": An empirical test of two models. American Journal of Sociology, 98(1):152–166, 1992.

44. Fu, W., Song, L., and Xing, E. P.: Dynamic mixed membership blockmodel for evolving networks. In Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, pages 329–336, New York, NY, USA, 2009. ACM.

45. Gallaher, A. and Padfield, H.: The Dying community. Advanced seminar series. University of New Mexico Press, 1980.

46. Girvan, M. and Newman, M. E. J.: Community structure in social and biological networks. Proceedings of the National Academy of Sciences, 99(12):7821–7826, 2002.

47. Goemans, M. X. and Williamson, D. P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM (JACM), 42(6):1115–1145, November 1995.

48. Good, B. H., de Montjoye, Y.-A., and Clauset, A.: Performance of modularity maximization in practical contexts. Physical Review E, 81:046106, Apr 2010.

49. He, J., Hopcroft, J., Liang, H., Suwajanakorn, S., and Wang, L.: Detecting the structure of social networks using (,)-communities. In Algorithms and Models for the Web Graph, eds, A. Frieze, P. Horn, and P. Prałat, volume 6732 of Lecture Notes in Computer Science, pages 26–37. Springer Berlin Heidelberg, 2011.

50. Helmberg, C., Rendl, F., Vanderbei, R., and Wolkowicz, H.: An interior-point method for semidefinite programming. SIAM Journal on Optimization, 6(2):342–361, 1996.

51. Holmström, E., Bock, N., and Brännlund, J.: Modularity density of network community divisions. Physica D: Nonlinear Phenomena, 238(14):1161–1167, 7 2009.

52. Hu, Y., Chen, H., Zhang, P., Li, M., Di, Z., and Fan, Y.: Comparative definition of community and corresponding identifying algorithm. Physical Review E, 78:026121, Aug 2008.

53. Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L. S., and Rubenstein, D.: Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. ACM SIGOPS Operating Systems Review, 36(5):96–107, October 2002.

54. Kernighan, B. and Lin, S.: An eficient heuristic procedure for partitioning graphs. Bell system technical journal, 1970.

55. Kleinberg, J.: An impossibility theorem for clustering. Advances in neural information processing systems, pages 463–470, 2003.

56. Kleinberg, J. and Tardos, E.: Algorithm Design. Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc., 2005.

57. Krause, J. and Ruxton, G.: Living in Groups. Oxford Series in Ecology and Evolution. OUP Oxford, 2002.

58. Krebs, J. and Davies, N.: Behavioural Ecology: An Evolutionary Approach. Wiley, 1997.

59. Kuhn, H. W. and Yaw, B.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly, pages 83–97, 1955.

60. Lahiri, M., Tantipathananandh, C., Warungu, R., Rubenstein, D. I., and Berger-Wolf, T. Y.: Biometric animal databases from field photographs: Identification of individual zebra in the wild. In Proceedings of the 1st ACM International Conference on Multimedia Retrieval, page 6. ACM, 2011.

61. Lancichinetti, A. and Fortunato, S.: Community detection algorithms: a comparative analysis. Physical Review E, 80(5):056117, 2009.

62. Lancichinetti, A. and Fortunato, S.: Limits of modularity maximization in community detection. Physical Review E, 84:066122, Dec 2011.

63. Leskovec, J.: Stanford network analysis platform (SNAP), Oct 2010. http://snap.stanford.edu/.

64. Lin, Y.-R., Chi, Y., Zhu, S., Sundaram, H., and Tseng, B. L.: Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In Proceedings of the 17th international conference on World Wide Web (WWW '08), 2008.

65. Lloyd, S.: Least squares quantization in PCM. IEEE Transactions on Information Theory, 28(2):129–137, September 2006.

66. Luxburg, U.: A tutorial on spectral clustering. Statistics and Computing, 17(4):395–416, 2007.

67. Moore, W.: Man, time, and society. Wiley, 1963.

68. Mucha, P. J., Richardson, T., Macon, K., Porter, M. A., and Onnela, J.-P.: Community structure in time-dependent, multiscale, and multiplex networks. Science, 328(5980):876–878, 2010.

69. Munkres, J.: Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics, 5(1):pp. 32–38, 1957.

70. Newman, M. E. J.: Detecting community structure in networks. The European Physical Journal B - Condensed Matter and Complex Systems, 38(2):321–330, 2004.

71. Newman, M. E. J.: Modularity and community structure in networks. Proceedings of the National Academy of Sciences, 103(23):8577–8582, 2006.

72. Newman, M. E. J. and Girvan, M.: Finding and evaluating community structure in networks. Physical Review E, 69, 2004.

73. Ng, A. Y., Jordan, M. I., Weiss, Y., et al.: On spectral clustering: Analysis and an algorithm. Advances in neural information processing systems, 2:849–856, 2002.

74. Palla, G., Barabási, A.-L., and Vicsek, T.: Quantifying social group evolution. Nature, 446(7136):664–667, 2007.

75. Pearson, M. and West, P.: Drifting smoke rings: Social network analysis and markov processes in a longitudinal study of friendship groups and risk-taking. Connections, 25(2), 2003.

76. Plutarch: Theseus. 75 A.C.E.

77. Porter, M., Onnela, J., and Mucha, P.: Communities in networks. Notices of the AMS, 56(9):1082–1097, 2009.

78. Rand, W. M.: Objective criteria for the evaluation of clustering methods. Journal of the American Statistical Association, 66(336):pp. 846–850, 1971.

79. Rattigan, M. J., Maier, M., and Jensen, D.: Graph clustering with network structure indices. In Proceedings of the 24th international conference on Machine learning (ICML'07), pages 783–790, New York, NY, USA, 2007. ACM.

80. Reeves, E.: The Dynamics of Group Behavior. American Management Assoc., 1970.

81. Ross, S.: Introduction to Probability Models. Elsevier Science, 2006.

82. Sarkar, P. and Moore, A.: Dynamic social network analysis using latent space models. ACM SIGKDD Explorations Newsletter, 7(2), December 2005.

83. Schaeffer, S. E.: Graph clustering. Computer Science Review, 1(1):27 – 64, 2007.

84. Scott, J.: Social Network Analysis: A Handbook. SAGE Publications, 2000.

85. Scott, J., Gass, R., Crowcroft, J., Hui, P., Diot, C., and Chaintreau, A.: CRAWDAD trace cambridge/haggle/imote/infocom (v. 2006-01-31). Downloaded from http://crawdad.cs.dartmouth.edu/cambridge/haggle/imote/infocom, January 2006.

86. Scott, J., Gass, R., Crowcroft, J., Hui, P., Diot, C., and Chaintreau, A.: CRAWDAD trace cambridge/haggle/imote/intel (v. 2006-01-31). Downloaded from http://crawdad.cs.dartmouth.edu/cambridge/haggle/imote/intel, January 2006.

87. Shi, J. and Malik, J.: Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(8):888–905, August 2000.

88. Shultz, S.: Bondedness and sociality. Behaviour, 147:775–803(28), 2010.

89. Spielman, D. A. and Teng, S.-H.: Spectral partitioning works: Planar graphs and finite element meshes. Linear Algebra and its Applications, 421(2 - 3):284 – 305, 2007.

90. Spiliopoulou, M., Ntoutsi, I., Theodoridis, Y., and Schult, R.: Monic: modeling and monitoring cluster transitions. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '06, pages 706–711, New York, NY, USA, 2006. ACM.

91. Sun, J., Faloutsos, C., Papadimitriou, S., and Yu, P.: Graphscope: parameter-free mining of large time-evolving graphs. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '07, pages 687–696. ACM, 2007.

92. Sundaresan, S., Fischhoff, I., Dushoff, J., and Rubenstein, D.: Network metrics reveal differences in social organization between two fission–fusion species, grevy's zebra and onager. Oecologia, 151:140–149, 2007.

93. Swamy, C.: Correlation clustering: Maximizing agreements via semidefinite programming. In Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '04, pages 519–520, 2004.

94. Tantipathananandh, C. and Berger-Wolf, T.: Constant-factor approximation algorithms for identifying dynamic communities. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '09, pages 827–836. ACM, 2009.

95. Tantipathananandh, C., Berger-Wolf, T., and Kempe, D.: A framework for community identification in dynamic social networks. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '07, pages 717–726. ACM, 2007.

96. Tantipathananandh, C.: Community Identification in Dynamic Social Networks Using Generalized Coloring. Master's thesis, University of Illinois at Chicago, 2006.

97. Tantipathananandh, C. and Berger-Wolf, T. Y.: Finding communities in dynamic social networks. In Proceedings of the 11th IEEE International Conference on Data Mining, eds, D. J. Cook, J. Pei, W. Wang, O. R. Zaïane, and X. Wu, ICDM '11, pages 1236–1241. IEEE, 2011.

98. Tong, H., Papadimitriou, S., Sun, J., Yu, P. S., and Faloutsos, C.: Colibri: fast mining of large static and dynamic graphs. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '08, 2008.

99. Šíma, J. and Schaeffer, S. E.: On the np-completeness of some graph cluster measures. In Proceedings of the 32nd conference on Current Trends in Theory and Practice of Computer Science, SOFSEM'06, pages 530–537, Berlin, Heidelberg, 2006. Springer-Verlag.

100. Wasserman, S. and Faust, K.: Social Network Analysis. Cambridge University Press, 1994.

101. Wellman, B.: The network community: An introduction to networks in the global village. Networks in the Global Village, 1999.

102. Wellman, B. and Gulia, M.: Net surfers don't ride alone: Virtual communities as communities. Networks in the global village, pages 331–366, 1999.

103. Xu, T., Zhang, Z., Yu, P. S., and Long, B.: Generative models for evolutionary clustering. ACM Transactions on Knowledge Discovery from Data (TKDD), 6(2):7:1–7:27, July 2012.

104. Yang, T., Chi, Y., Zhu, S., Gong, Y., and Jin, R.: A bayesian approach toward finding communities and their evolutions in dynamic social networks. In Proceedings of the Ninth SIAM International Conference on Data Mining, SDM '09, February 2009.

VITA

NAME:	Chayant Tantipathananandh
EDUCATION:	Ph.D., Computer Science, University of Illinois at Chicago, Chicago, Illinois, 2013.
	M.S., Computer Science, University of Illinois at Chicago, Chicago, Illinois, 2007.
	B.Eng. (honor), Computer Engineering, Chiang Mai University, Chiang Mai, Thailand, 2002.
ACADEMIC EXPERIENCE:	Research Assistant, Computational Population Biology Lab, Department of Computer Science, University of Illinois at Chicago, 2006–2013.
	Teaching Assistant, Department of Computer Science, University of Illinois at Chicago: • Computer Algorithm I, 2008 and 2011. • Computer Architecture I, 2011. • Languages and Automata, 2012.
PROFESSIONAL EXPERIENCE:	Software Engineering Intern, Google, Mountain View, California, 2012.
	System Engineer, C.S.I. Group, Bangkok, Thailand, 2002–2004.
	Software Engineering Intern, Electricity Generating Authority of Thailand (EGAT), Nonthaburi, Thailand, 2001.
PROFESSIONAL MEMBERSHIP:	Institute of Electrical and Electronics Engineers (IEEE)
	Special Interest Group on Knowledge Discovery and Data Mining (ACM SIGKDD)
HONORS:	Phi Kappa Phi, University of Illinois at Chicago, 2007.
	Outstanding Engineering Student Award, Crown Prince Foundation, Thailand, 2001

Outstanding Engineering Student Awards, Chiang Mai University, 1999, 2000, 2001, 2002.

PUBLICATIONS: M. Schumer, R. Birger, C. Tantipathananandh, J. Aurisano, M. Maggioni, P. Mwangi Infestation by a Common Parasite is Correlated with Ant Symbiont Identity in a Plant-Ant Mutualism. Biotropica, 2013.

C. Tantipathananandh, T. Berger-Wolf. Finding Communities in Dynamic Social Networks. Proceedings of the 11th IEEE International Conference on Data Mining, 2011.

T. Berger-Wolf, I. R. Fischhoff, D. I. Rubenstein, S. R. Sundaresan, C. Tantipathananandh. Dynamic Analysis of Social Networks of Equids. Applications of Social Network Anslysis, 2010.

T. Berger-Wolf, M. Lahiri, C. Tantipathananandh, D. Kempe. Finding Structure in Dynamic Networks. The 1st Workshop on Information in Networks, 2009.

K. Reda, C. Tantipathananandh, T. Berger-Wolf, J. Leigh, A. Johnson. Poster: SocioScape a Tool for Interactive Exploration of Spatio-Temporal Group Dynamics in Social Networks. InfoVis, 2009.

C. Tantipathananandh, T. Berger-Wolf. Constant-Factor Approximation Algorithm for Identifying Dynamic Communities. Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2009.

C. Tantipathananandh, T. Berger-Wolf, D. Kempe. A Framework For Identifying Communities in Dynamic Social Networks. Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2007.

Habiba, C. Tantipathananandh, T. Berger-Wolf. Betweenness Centrality Measure in Dynamic Networks. DIMACS Technical Report, 2007.

C. Tantipathananandh. Community Identification in Dynamic Social Networks Using Generalized Coloring. Master's thesis, University of Illinois at Chicago, 2007.