COMMUNITY EVOLUTION IN TEMPORAL NETWORKS

By

James Thompson

A Dissertation Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: COMPUTER SCIENCE

Examining Committee:

_____

Malik Magdon-Ismail, Dissertation Adviser

_____

Mark Goldberg, Member

_____

William Wallace, Member

_____

Boleslaw Szymanski, Member

Rensselaer Polytechnic Institute
Troy, New York

April 2015
(For Graduation May 2015)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The purpose of this research is to study the structure of social networks with an added temporal element. Specifically, we examine dynamic community behavior within social networks. We base our experiments on a simple theoretical foundation which allows us to efficiently identify dynamic community evolutions. Based on this framework, we empirically study evolutions in large social networks and structural features of evolutions across all networks. Results show that structural properties remain similar across multiple social networks and it is possible to correlate the lifespan of a community to specific features of its early evolution.

We also develop a framework for generating social networks with structures similar to those found in real world systems. Using this framework, we examine the behavior of evolution detection algorithms in full networks and more isolated situations. Finally we examine the robustness of our developed community evolution tracking framework in noisy systems.

CHAPTER 1
Introduction

Extremely complex systems can arise from the interactions between a group of entities. These systems could involve people interacting via friendships and conversations, words that occur in the same documents, or proteins that bind with one another. Understanding the structure and characteristics of these systems provides insight into the underlying mechanics of the system. Unfortunately, these systems are frequently extremely large and complex. Advances in technology have been especially responsible for both an increase in the size of some interaction systems and an increase in the ability to accurately and efficiently represent and document those systems.

The study of interaction systems has classically focused on representing them as a single, static network. There have been many approaches to understanding the structure of these networks, one of which involves detecting a mesoscopic, community structure. A community is a group of entities or vertices that share some characteristic or interest and can therefore be considered a single entity in some respect. The community structure of the network simply provides a large reduction in the number of entities to consider in order to understand the network, from the number of vertices to the number of communities. For example, it is often easier to understand a political voting system when considering the voting patterns of political parties rather than looking at each individual voter's actions separately. The members of a single party typically have similar goals and beliefs, and hence end up voting in a similar way.

Community detection algorithms try to recreate coherent groups of entities based off of their interactions. To continue the political example, a community detection algorithm would consider all of the votes cast by each member of Congress and try to reconstruct the political parties. These algorithms are necessary because helpful community structures are generally unknown for a vast majority of real world networks. For instance, many complex biological [Brohee and Van Helden 2006] and

economic [Nanda et al. 2010] systems that can be modelled as interaction networks are not yet fully understood and have been studied with such algorithms.

Time plays a crucial role in many interaction networks. Congresspeople will vote differently on different bills, proteins in biological processes will interact with different proteins as the process, and stock prices are constantly changing. Modelling these systems as a single, static network may miss important information. As a simple example, consider three people, Alice, Bob and Eve, who decide to meet for lunch. Bob tells Eve to meet at 12:00, but Alice later tells Bob to meet at 1:00. The information cannot reach Eve unless one of Alice or Bob talks to her again. However, if the two interactions were modelled in a static network, then Alice is shown talking to Bob, who is talking to Eve, so there is a (false) line of communication from Alice to Eve.

With the network changing over time, it is imperative to understand both the structure of the network at any given point in time, and the dynamics of how the network structure is changing. One way is to use a generalization of static community detection approaches.

1.1 Problem Formulation

We start by assuming that the social network we are interested in has been processed into a series of static networks $\{N_1, N_2, \ldots, N_t\}$, commonly called snapshots. Furthermore, we assume that community structures have been detected in each static network $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_t\}$. Each community is a manifestation of an interest group or functional unit at a single point of time in the temporal network. Being able to track a single group or functional unit over time would help researches understand the dynamics of the network as well as possibly predict future behavior of either the network or the community. Our goal is to find sets of communities, $\{C_1, C_2, \ldots, C_k\}$, that are the different manifestations of the same interest group or functional unit at different times.

1.2 Our Contributions

In this report, we first describe a model for representing temporal networks and define a two step framework for tracking the changing community structure in temporal networks. Specifically, we track each individual community as it changes throughout the life of the network, calling it a community evolution. In the first step, we detect continuously active community evolutions using a simple, axiomatic framework. The second step merges these continuously active evolutions together in order to detect community evolutions that may only be periodically active.

We then use this framework to detect evolutions in real world networks and use structural properties of the early stages of evolutions to predict how long it will survive. Being able to successfully predict the survival time of evolutions may allow algorithms to be developed that control the dynamics of the network in future iterations. For example, if we can say that dense communities of proteins tend to survive longer in a biological process, a process that to forces proteins to densely interact with one another may succeed in making them last longer.

We also develop a framework for generating temporal networks which mimic many of the characteristics of real world networks, including having an embedded community evolution structure. The ability to quickly generate networks with an embedded evolutionary structure allows us to test and evaluate community evolution detection algorithms quickly and efficiently. We use our framework to evaluate four evolution detection algorithms on detecting single evolutionary events in isolation, and complete evolutions in a multi step network. Finally, we test the robustness of our framework on instances of temporal networks where some entities have unknown identities.

CHAPTER 2
Related Works

2.1 Static Social Networks

Classically, social network analysis focuses on a single, static graph $G = \{V, E, W\}$ where the vertices V represent the entities of the social network and the edges E represent the interactions between entities. In addition, there is a weight function $W : e \in E \rightarrow R$, which maps each edge to a real number representing the importance or strength of the interaction. Since much of the analysis and algorithms used on static networks is easily applied or extended to the temporal framework, we discuss some of the most popular metrics and methods used in static networks. Throughout the rest of this report, we use the terms 'network' and 'graph' interchangeably, unless explicitly stated otherwise.

2.1.1 Network Characteristics

A number of structural characteristics have been found to be common in social networks. These characteristics distinguish social networks from random structures and provide quantitative metrics for comparing two networks. We quickly discuss four of the main distinguishing properties here. A more complete treatment of the structural differences between random and real world networks can be found in various reviews [McGlohon et al. 2011; Newman 2003].

2.1.1.1 Degree Distribution

Social networks exhibit degree distributions that approximate power law as opposed to the exponential distributions found in many random network models

Portions of this chapter previously appeared as: Mark Goldberg, Malik Magdon-Ismail, Srinivas Nambirajan, and James Thompson. 2011. Tracking and predicting evolution of social communities. In *2011 IEEE Third International Conference on Social Computing (SocialCom)*. IEEE, Boston, MA, USA, 780-783.

Portions of this chapter previously appeared as: Mark Goldberg, Malik Magdon-Ismail, and James Thompson. 2012. Identifying long lived social communities using structural properties. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*. IEEE, Istanbul, Turkey, 647-653.

[Barabási and Albert 1999; Clauset et al. 2009]. In a power law distribution, the probability that a vertex v is connected to k other vertices is about $k^{-\gamma}$. The appearance of power laws is classically modelled using a 'preferential attachment' mechanism [Barabási and Albert 1999] (see Section 2.1.4), but the underlying reasons for the appearance of power laws is still being studied [Muchnik et al. 2013].

Many other properties of social networks have been shown to follow power laws as well, including the clustering coefficient of vertices [Tsourakakis 2008] and the eigenvalues of the characteristic matrix [Mihail and Papadimitriou 2002]. In addition, slight deviations from the power law have also been observed in social networks. In some systems, there seems to be a soft maximum threshold on vertex degree, causing the distribution to be more accurately modelled with an exponential cut-off [Newman 2001]. Many other systems display concavity in the log-log plot of vertex degree frequencies, which has been shown to be closely modelled with a discrete Gaussian exponential [Bi et al. 2001].

2.1.1.2 'Small-World' Property

The small-world property of social networks is more commonly known as the 'six degrees of separation' phenomenon, made famous by Milgram's well-known study [Milgram 1967]. A network is said to have the small-world property if, starting from any vertex, any other vertex can be reached by traversing only a few edges in the network. More specifically, if we define the distance between a pair of vertices in a network as the length of the shortest path between them, the diameter of the net- work is defined as the maximum distance over all pairs of vertices in the network. The small-world property states that the diameter of a social network is very small. The model of Watts and Strogatz [Watts and Strogatz 1998] shows that this behavior arises due to a small number of 'short cuts' in a network that might otherwise have a large diameter.

Because the diameter is a maximum, it can be susceptible to a single outlier. As a result, an 'effective' diameter is usually considered. The effective diameter is the minimum distance required to connect some fraction (usually 90%) of the vertices. Calculating the exact effective diameter of a large network can be extremely time

consuming, however, as it takes time at least in $O(N^2)$ due to having to calculate shortest paths between all pairs of vertices. The effective diameter is therefore usually estimated using sampling [Kang et al. 2008; Palmer et al. 2002].

2.1.1.3 Clustering Coefficient

In social networks, it has been observed that if two vertices share a neighbor, they are more likely to be connected to each other than vertices that do not share a neighbor [Holland and Leinhardt 1972; Watts and Strogatz 1998]. The clustering coefficient was designed to quantify the degree to which this phenomenon occurs in a network. The clustering coefficient of a node v is defined as the fraction of pairs of neighbors of v that share an edge [Watts and Strogatz 1998]. Formally, if there are t edges between neighbors of v and d_v is the degree of v, the clustering coefficient of v can be written as:

$$\mu_v = \frac{2t}{(d_v - 1)d_v}$$

There are a couple of different ways to characterize the global clustering of a network G. One way is to simply average the clustering coefficients of each vertex, so that

$$\mu_G = \frac{1}{|V|} \sum_{v \in V} \mu_v$$

The values calculated using this definition can be difficult to interpret due to vertices with low degrees. Consider a vertex v with degree d_v and only a single pair of neighbors that share an edge. The clustering coefficient is then $\frac{2}{(d_v-1)d_v}$, which evaluates to 1 for a vertex of degree 2, but quickly drops to only $\frac{1}{3}$ for a vertex with a degree of 3. The highly volatile clustering coefficients of the low degree vertices can therefore skew the global clustering coefficient one way or the other.

There is an alternate formulation for the global clustering which is easier to interpret and has cleaner mathematical properties [Newman and Park 2003]. Let ω denote the number of 'wedges' in the network, defined as triples of vertices $\{u, v, w\}$ where $(v, u) \in E$ and $(v, w) \in E$. If Δ is the number of 'closed wedges', or triples where $(u, w) \in E$ as well, the clustering coefficient is simply $\frac{\Delta}{\omega}$, or the fraction of all

wedges that are closed.

A number of extensions of the clustering coefficient to handle weighted and directed networks have been proposed [Saramäki et al. 2007; Opsahl and Panzarasa 2009; Fagiolo 2007].

2.1.1.4 Community Structure

Entities in social networks that share a common interest, function or role often tend to form densely connected groups called communities. It is even possible that a hierarchical structure develops in a network where smaller communities are present within larger ones. Communities have been observed in a wide variety of networks, including the graph representation of the world wide web [Flake et al. 2002], social networks, and protein interaction networks [Girvan and Newman 2002]. The community structure provides insight into organizational patterns and can help with understanding the underlying processes of the network. The problem of determining the community structure of a network has received a lot of attention in the past couple of decades and still remains an active area of research.

2.1.2 Community Detection

While most network properties, such as clustering coefficient and diameter, are directly calculable, community structures are more ambiguous. To begin with, there is no specific, widely accepted definition of what a community is. Generally, a community is understood as a set of vertices that are more densely connected with each other than with other vertices in the network. Even with a specific community definition, however, detecting all the communities in a network is frequently an NP-hard task, and requires the use of an approximation scheme. As a result, detected community structures are often highly sensitive to noise. Even so, community detection has been successfully applied to a wide variety of fields including computer networking [Hui et al. 2011], biology [Brohee and Van Helden 2006], and economics [Nanda et al. 2010], with new algorithms constantly being developed. In this section we review some of the most popular approaches to community detection. More in depth reviews can be found in reviews of the field [Xie et al. 2013b; Fortunato 2010].

2.1.2.1 Local Search

A popular approach to defining a community is to use a heuristic function that takes an arbitrary set of vertices and calculates a community 'score'. If the function returns a high score, the input set of vertices is considered to be a well defined community. A community detection algorithm then searches for sets of vertices in a network with the highest scores. Many different functions have been investigated. For example, algorithms have been developed with statistical functions [Lancichinetti et al. 2011], functions based on random walks [Macropol et al. 2009], and density and conductance functions [Goldberg et al. 2010; Lancichinetti et al. 2009]. All of these functions only consider the local structure surrounding a set of vertices when calculating the score, which allows for a quick computation time.

Detecting communities by calculating the heuristic on every possible set of vertices from the network is computationally prohibitive. Instead, local search algorithms typically detect communities one at a time, starting from a given set of vertices, commonly called a 'seed'. Local search algorithms then proceed in a two step process. The first step adds neighbors of the seed to the seed if doing so would increase the community score of the seed. Similarly, the second step removes members of the seed if the removal would increase the seeds score. The process continues until a stopping criterion - usually a maximum community size or convergence of the quality function - is reached, at which point the final set of vertices is considered a community. In order to find the full community structure of a network, the process is repeated multiple times, starting from seeds taken from across the network structure.

All local search algorithms must consider two main issues. First, the set of seeds chosen for the algorithm greatly influences the calculation time and accuracy. Too many seeds will result in a long run time while not enough - or poorly chosen - seeds will result in inaccuracy. More subtly, each algorithm needs to determine the order in which new vertices are added to the seed and members are removed. Simply adding the vertex that gives the largest increase in quality can cause certain areas of the network to go unexplored.

2.1.2.2 Spectral Algorithms

Any algorithm that considers the eigenvalues or eigenvectors of a representative matrix of a network in order to find a community structure is referred to as a spectral algorithm. Spectral algorithms typically deal with a characteristic matrix of the network called the Laplacian. Let $\mathbf{A}$ be the adjacency matrix where $\mathbf{A}_{ij} = \mathbf{A}_{ji}$ is the weight of the edge from vertex i to vertex j. In addition, let $\mathbf{D}$ be the diagonal matrix with $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. The unnormalized Laplacian of a network can then be written as $\mathbf{L} = \mathbf{DA}$. There are multiple ways to normalize the matrix, including $\mathbf{L}_{norm} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$ or $\mathbf{L}_{norm} = \mathbf{D}^{-1}\mathbf{L}$, but the basic approach to community detection remains the same regardless of which version of the matrix is considered.

Once the Laplacian for a network is constructed, spectral algorithms calculate the k eigenvectors corresponding to the k smallest eigenvalues of the Laplacian. The rows of the nk matrix constructed by combining the eigenvectors are then considered as data points in $\mathbb{R}_k$ and are clustered using more traditional methods such as fitting Gaussian mixture models [Magdon-Ismail and Purnell 2011]. It may not be immediately clear why the spectral method of community detection even works, but analyzing the eigenvectors of a graph Laplacian had been equated to approaches such as minimizing cuts in a network [Von Luxburg 2007] or maximizing modularity [White and Smyth 2005].

The main downside to spectral clustering is that the number of communities present in the network, which is almost always an unknown value for real world networks, must be passed as an argument. The algorithm could be run multiple times, detecting a different number of communities each time, but this approach relies on a heuristic method to determine which run has the best results and can require significantly more computation time. Still, the spectral approach remains popular since approximations can result in very fast algorithms and the method can utilize existing linear algebra software.

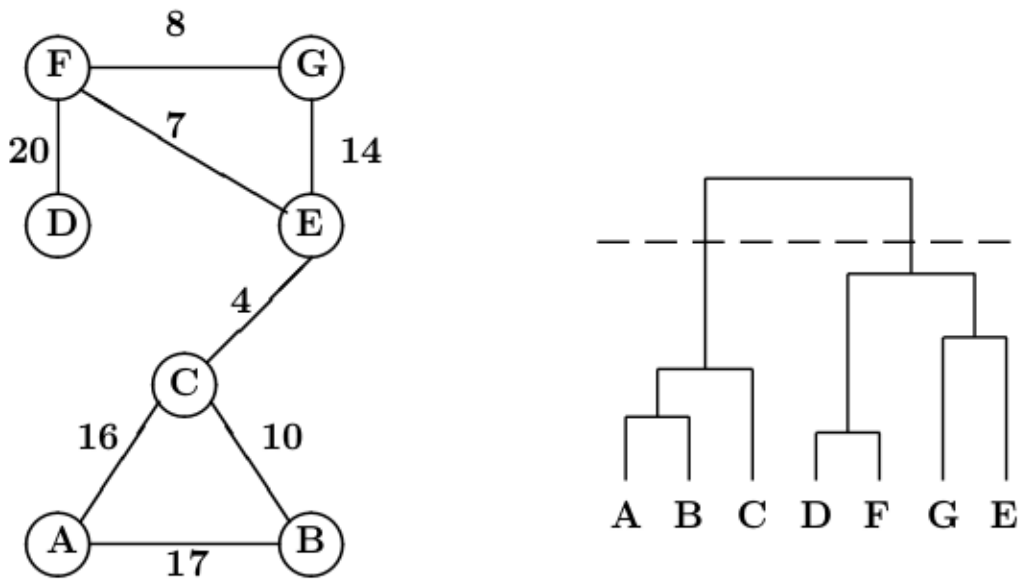


Figure 2.1: A sample weighted network and the accompanying dendrogram for single-link hierarchical clustering based off of edge weights. The horizontal dotted line represents the cut in the dendrogram associated with breaking the network into two communities - $\{A, B, C\}$ and $\{D, E, F, G\}$

2.1.2.3 Hierarchical

Hierarchical community detection algorithms attempt to capture the structure of a network at different scopes. One common example used to motivate the approach is that students at a university may be grouped into classes at one level, department at a higher level, and even by institution. Hierarchical algorithms fall in one of two categories - agglomerative and divisive. Agglomerative algorithms start with a large number of small communities and iteratively merge similar communities together until only a single community consisting of all vertices in a network remains. Divisive algorithms proceed in the opposite direction, starting from a single community consisting of all vertices and iteratively splitting it into smaller communities until the communities become small enough.

The main difference between hierarchical algorithms is the criteria used to merge or split communities. Some algorithms merge small communities simply based

on the weights of edges between communities [Shen et al. 2009; Karypis et al. 1999] or use a more global method such as modularity [Newman 2003], while some use a variation of centralities [Gregory 2008] to split a network into communities. The process of these algorithms can be visualized using a dendrogram, like the one shown in Figure 2.1. A single, flat community structure can be retrieved by considering the structure at some horizontal point in the dengrogram. A heuristic function, such as modularity, is typically used to determine where the optimal point of the dendrogram lies.

2.1.2.4 Other Algorithms

One of the first algorithms to allow for overlapping communities was clique percolation [Adamcsek et al. 2006; Derényi et al. 2005]. A clique in a network is a set of vertices that are all connected with one another. A k-clique is a clique of k vertices. In the method, two k-cliques are considered adjacent if they share $k-1$ vertices. For a given k, a community is defined as a maximal set of k-cliques that can reach each other by moving through adjacent k-cliques. This can be an overly strict definition in many applications, and finding all k-cliques in a network can become computationally prohibitive.

Label propagation has also been used for community detection [Gregory 2010; Xie et al. 2011]. In these algorithms, each vertex starts as its own community, being associated with a unique community label. The algorithms then proceed iteratively, with each vertex randomly sending community labels its neighbors. Each vertex also tracks the labels that it has received from its neighbors and, after a certain number of iterations, is assigned to communities whose labels have been received enough times.

2.1.3 Evaluation

2.1.3.1 Extrinsic Metrics

When evaluating a community structure, extrinsic metrics require a knowledge of the actual communities in a network, commonly referred to as the 'true' community structure or the 'ground truth'. A detected community structure is considered good if it comes close to reproducing the ground truth. An accurate, useful and

Table 2.1: Possible metrics for the similarity between two communities, A and B, in set matching.

$\frac{\lvert A \cap B \rvert}{\lvert A \cup B \rvert}$	Jaccard Index
$\frac{2\lvert A \cap B \rvert}{\lvert A \rvert + \lvert B \rvert}$	Sorenson Coefficient
$\lvert A \cap \bar{B} \rvert + \lvert \bar{A} \cap B \rvert$	Edit Distance (no swapping)
$H(A\|B)$	Entropy

efficient definition of 'close', however, is difficult to attain and many different definitions have been proposed. In this section, we describe some of the most popular metrics used for evaluating the quality of an overlapping community structure.

Set Matching

An intuitive and popular approach to evaluating community structure is to find a one-to-one matching from the detected communities to the ground truth community structure [Girvan and Newman 2002]. If a pair in the matching is strong enough, the true community is considered to be recovered by the detected structure. Detected community structures that recover more ground truth communities can then be considered more accurate. We only consider one-to-one matchings in this report, but similar approaches can be used with one-to-many matchings.

To construct a matching and determine the closeness between any two communities, we use a similarity function, $S(\cdot, \cdot)$, which takes, as input, two communities, A and B, and produces a numeric value. The function should be symmetric ($S(A, B) = S(B, A)$) and represent either a similarity or distance between the two communities. There are many ways to define such a function (see Table 2.1). The choice of function will affect the actual results of a comparison, but does not affect the procedure. Also, if $S(\cdot, \cdot)$ is asymmetric, similar metrics can be defined as long as the asymmetry is taken into account. Let $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ be the true community structure of a network and $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ be a community structure detected by an algorithm. A matching is constructed by first calculating the similarity value between every true community and every detected community,

$S(T_i, C_j)\forall_{i,j}$. This requires computing kn values, which can become computationally prohibitive if done explicitly, depending on the choice of $S(\cdot, \cdot)$. In practice, many of the pairs may have a zero value (especially for similarity measures), and the efficiency can usually be improved by indexing communities in order to skip such pairs. Once all values are known, any bipartite matching algorithm can then be used to construct a one-to-one matching from communities in $\mathbb{T}$ to communities in $\mathbb{C}$ that either maximizes similarity or minimizes distance. If one set of communities is larger than the other, there will be unmatched communities. We assume that these communities are artificially mapped with an empty community. Given such a matching, M, let $M(T_i)$ represent the community that is matched to T_i (and similarly for a community C_j).

If $S(\cdot, \cdot)$ is a similarity measure and γ represents a user defined threshold, the communities in $\mathbb{T}$ and $\mathbb{C}$ can then be grouped into three categories: true positives, which are T_i where $S(T_i, M(T_i)) >= \gamma$, false positives, which are C_j where $S(C_j, M(C_j)) < \gamma$, and false negatives, which are T_i where $S(T_i, M(T_i)) < \gamma$. The corresponding definitions for a distance measure simply reverse the inequalities. Each of these groups, taken independently, gives a very narrow view of the community structure quality. For instance, if we just considered true positives, a structure that included every possible community would be optimal. Two slightly more useful metrics that combine these values are precision (true positives / (true positives + false positives)) and recall (true positives / (true positives + false negatives)). Both have a restricted range of values ([0, 1]), but taken separately, these two metrics still have drawbacks - detecting a single true community out of 100 returns the maximum precision. Going one step further, the F_1 score is defined as

$$F_1 = 2\left(\frac{precision \times recall}{precision + recall}\right)$$

and also ranges from 0 to 1 with higher scores more desirable. This metric is symmetric, and can therefore be used as a choice for $S(\cdot, \cdot)$ where vertices are considered in place of communities. The score can also be tuned to weight the relative importance of precision or recall, but the metric then becomes asymmetric.

More complex measures can also be constructed from matchings, such as nor-

malized mutual information (NMI). Developed by borrowing concepts from information theory, the measure tries to quantify the amount of information that is lost from the true community structure when only considering the detected community structure. It was originally proposed for evaluating partitions in [Meilă 2007], the metric was extended to consider overlapping structures in [Lancichinetti et al. 2009]. The metric was further developed in [McDaid et al. 2011] to fix unintuitive behavior when the community structure sizes became disparate. This version of NMI is defined as (see Appendix A for an in depth description):

$$NMI(X,Y) = \frac{\frac{1}{2}(H(X) - H(X|Y) + H(Y) - H(Y|X))}{max(H(X), H(Y))}$$

Pairwise Evaluation

In [Rand 1971], the authors propose a method for comparing partitions, where each vertex is assigned to a single community, that considers pairs of vertices. If a pair of vertices share a community (or are in different communities) in both the true partition and in the detected partition, then the pair is considered to be correctly recovered. The metric, called the Rand Index, is simply the ratio of correctly recovered pairs to the total number of vertex pairs, $\binom{N}{2}$. It was shown that the Rand Index suffers from a bias due to the fact that some fraction of vertex pairs will be correctly recovered even in random community structures. A normalized version of the measure, called the Adjusted Rand Index, was proposed in [Hubert and Arabie 1985] to remove this bias.

The Adjusted Rand Index is unsuitable for overlapping community structures, however, where pairs can share multiple communities. To account for this, an extension called the Omega Index was proposed in [Collins and Dent 1988]. For a pair of vertices to be correctly recovered in this case, they must share the same number of community memberships in the detected structure as they do in the true structure.

While recent studies have used the Omega Index as a way to evaluate overlapping community structure (see [Murray et al. 2012], for instance), the definition of a correctly recovered pair can be extremely harsh. In the case where two vertices share 6 true communities, but are detected as sharing only 5, the pair is not considered to be correctly recovered. It would be treated with the same importance as a

pair of vertices that truly share a single community, but are detected as sharing 6. A measure similar to the Omega Index, named Bcubed, that takes this into account was proposed in [Amigó et al. 2009].

The Bcubed measure is defined in terms of the 'multiplicity precision' and 'multiplicity recall' of a community structure. Let $\mathbb{T}(u)$ (similarly $\mathbb{C}(u)$) be the set of communities that vertex u belongs to in $\mathbb{T}$ (resp. $\mathbb{C}$). Then, the multiplicity precision and recall of a pair of vertices u and v are defined as:

$$Pre(u,v) = \frac{Min(|\mathbb{T}(u) \cap \mathbb{T}(v)|, |\mathbb{C}(u) \cap \mathbb{C}(v)|)}{|\mathbb{C}(u) \cap \mathbb{C}(v)|}$$

$$Rec(u,v) = \frac{Min(|\mathbb{T}(u) \cap \mathbb{T}(v)|, |\mathbb{C}(u) \cap \mathbb{C}(v)|)}{|\mathbb{T}(u) \cap \mathbb{T}(v)|}$$

It is important to note that the precision value is only defined when the two vertices share at least one detected community and the recall value is only defined when the vertices share a true community. To get the final Bcubed precision and recall values, the defined pairwise precision and recall values are simply averaged:

$$Bcubed_{Pre} = Avg_u \left[Avg_{v,|C(u)C(v)|=0}(Pre(u,v)) \right]$$

$$Bcubed_{Rec} = Avg_u \left[Avg_{v,|T(u)T(v)|=0}(Rec(u,v)) \right]$$

2.1.3.2 Intrinsic Metrics

Unfortunately, the ground truth communities are frequently unknown in real world networks. Intrinsic metrics focus on the structural properties of the given network in order to determine the quality of a community. Many different community quality functions have been proposed, such as the density or conductance, and the quality of a full community structure is usually an average, maximum, or equivalent statistic of all community quality values. Personal preferences and structural properties of the network being studied usually dictate which quality measure should be applied.

Modularity

One of most popular and widely-used measures is called modularity, originally developed to analyze graph partitions [Newman and Girvan 2004]. The quality of a graph partition has classically been corresponded with the cut size of the partition, defined as the number of edges that connect vertices from different sets of the partition. Since the goal is to split the graph into strongly cohesive components, a smaller number of such edges signifies a better partitioning. Conversely, a larger portion of the edges of the network should reside within the partition sets.

A community is considered well defined with respect to the modularity measure if the number of edges connecting vertices inside the community is significantly larger than what would be expected in a random 'null model' graph. The actual values and usefulness of the measure depend on what type of graph is chosen as the null model. The most popular choice for a null model has become the configuration model, which randomly rewires the edges of a network while keeping the vertex degrees constant. Random graph models are discussed more in Section 2.1.4.

Suppose we have determined or been given a community structure on a graph with the adjacency matrix $\mathbf{A}$. Assume the network is unweighted and undirected in this case, so $\mathbf{A}_{ij}$ is defined to be 1 if there is an edge between vertices i and j in the network and 0 otherwise. Also, let $\delta(i, j)$ be defined as equal to 1 if vertices i and j are assigned to the same community and 0 otherwise. The probability that two vertices are connected in a random network using the configuration model is $\frac{k_i k_j}{2m}$ (in the limit of large m) where m is the total number of edges in the network and k_i is the degree of vertex i. Modularity is then defined as

$$Q = \frac{1}{2m} \sum_{ij} \left(\mathbf{A}_{ij} \frac{k_i k_j}{2m} \right) \delta_{ij}$$

Modularity has been criticized with regards to issues such as interpretation [Guimera et al. 2004; Reichardt and Bornholdt 2006; Good et al. 2010] and resolution limits [Fortunato and Barthélemy 2007; Lancichinetti and Fortunato 2011]. Nevertheless, it has been used successfully in a number of algorithms and applications [Lancichinetti et al. 2009; Newman 2006; Clauset et al. 2004], and has also been extended to consider directed and weighted networks with overlapping communities

[Lázár et al. 2010; Nicosia et al. 2009].

2.1.4 Network Generation Models

Real world networks can be difficult to acquire, both from the practical standpoint of having to collect all the data and the moral issues such as privacy concerns. Random network generators offer an alternative. A good generator can help with a number of tasks, including testing algorithms and simulating processes in different network structures, network compression, and anonymization.

Any discussion of random networks begins with Erdos-Renyi networks [Erdős and Rényi 1960], where, to generate a network with N vertices, each of the $\binom{N}{2}$ possible edges is generated with the same probability. Unfortunately, the characteristics of networks generated this way do not display the distinctive characteristics of real world social networks. For instance, the degree distribution of an Erdos-Renyi network generally follows a Poisson distribution as opposed to the power law distribution commonly found in real world networks. As a result, numerous models have since been developed with the explicit goal of modelling specific characteristics of social networks.

The model of [Watts and Strogatz 1998], which we refer to as the WS model, was developed to capture the small-world properties of social networks. In the model, the vertices are first organized in a one-dimensional lattice, and each vertex is connected the k vertices immediately to either side of it. Next, each edge is rewired, with probability p, to a random edge not already in the network. It was shown that with a small p, the clustering coefficient of the network remains relatively high while the diameter of the network is reduced considerably. However, the degree distribution of these graphs did not follow a power law.

The preferential attachment model (PA) [Barabási and Albert 1999] has become one of the most well-known models for generating networks with a power law degree distribution. The original PA model starts with a small number of vertices and then simulates network growth by iteratively grows the network by adding a single vertex v connected to m existing vertices at each step. An existing vertex is connected to the new vertex with a probability proportional to the existing ver-

texs degree, resulting in a power law degree distribution. Numerous extensions to this model have been developed in order to capture different network characteristics such as clustering coefficient or a slightly different degree distribution [Albert and Barabási 2000; Pennock et al. 2002].

While the PA model was developed to explicitly generate networks with a power law degree distribution, the configuration model [Molloy and Reed 1995] was developed to generate networks with any given degree sequence. Given a specific degree sequence $\{d_1, d_2, d_3, \ldots, d_n\}$, the configuration model constructs disconnected 'stubs' for each vertex where d_i stubs are generated for vertex i. Then, two disconnected stubs are chosen at random to be connected until no disconnected stubs remain. Given a degree sequence sampled from a power law, the resulting network will have a power law degree distribution. The original formulation of the model allowed for self-edges and multiple edges, although the probability of these edges being generated tends towards zero as the number of vertices considered grows large and they can simply be ignored in most cases. Alternatively, graph construction strategies that incorporate 'forbidden' edges [Kim et al. 2009] can be adopted. In addition, the model has been extended to include a 'triangle membership' distribution in order to more directly control the clustering coefficient of the network [Newman 2009].

More recently, the Kronecker model has gained popularity [Leskovec et al. 2010, 2005; Seshadhri et al. 2013]. This model uses the Kronecker matrix product in order to produce networks that are self-similar at different scopes. The general Kronecker matrix product is defined as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \ldots & a_{1k}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \ldots & a_{2k}\mathbf{B} \\ \vdots & \vdots & \ldots & \vdots \\ a_{n1}\mathbf{B} & a_{n2}\mathbf{B} & \ldots & a_{nk}\mathbf{B} \end{bmatrix}$$

where $\mathbf{A}$ is an nk matrix and $\mathbf{B}$ is of any size.

The n^{th} Kronecker power of a matrix $\mathbf{A}$ is then $\mathbf{A} \otimes \mathbf{A} \ldots \otimes \mathbf{A}$. The Kronecker model can be seen as generating a probabilistic adjacency matrix by taking the n^{th}

Kronecker power of a small (usually 2 x 2) seed matrix with entries between 0 and 1. This produces a 2^n x 2^n matrix, $\mathbf{A}^n$, with entries between 0 and 1, with the value of n determined by the desired size of the network. To construct a realization of the network, an edge is placed between vertices u and v with probability $A^n_{u,v}$. The two main drawbacks of explicitly calculating the n^{th} Kronecker power of even a small matrix are the time required for the calculation and the fact that the network must have a number of vertices that is a power of 2. Both of these drawbacks can be averted using an equivalent, more direct approach described in [Leskovec et al. 2010].

None of the previously discussed network generation models have explicit community structure in the generated networks. One of the first models that incorporated community structure is that of Girvan and Newman [Girvan and Newman 2002], where each graph has four groups of 32 vertices each. Similar to the Erdos-Renyi model, each possible edge in the network is generated with some probability. The difference in this model is that an edge between vertices in the same community is constructed with probability p_{in} while an edge between vertices in different communities is constructed with probability p_{out} (with $p_{in} > p_{out}$). However, networks generated with this model are still missing many of the distinctive characteristics of real world networks as each vertex has the same expected degree in this model, and each community is the same size.

A more recent model was proposed in [Lancichinetti et al. 2008], which we refer to as the LFR model or benchmark. The LFR model uses the configuration model described above. In this model, each vertex is explicitly assigned to some number of communities and given a specific degree. In addition, a global parameter determines what fraction of each vertexs edges is connected to vertices that share a community. Then, using the configuration model approach and the parameters given, a sub-network is constructed for each community. Finally, the configuration model is used a second time to construct edges that connected vertices between communities. While the clustering coefficient is difficult to control in this model, the generated networks portray a power law degree distribution, have a small diameter, and have a known community structure. In addition, the model has been extended

to weighted and directed networks [Lancichinetti and Fortunato 2009], and it has become one of the most widely used benchmarks for clustering algorithms.

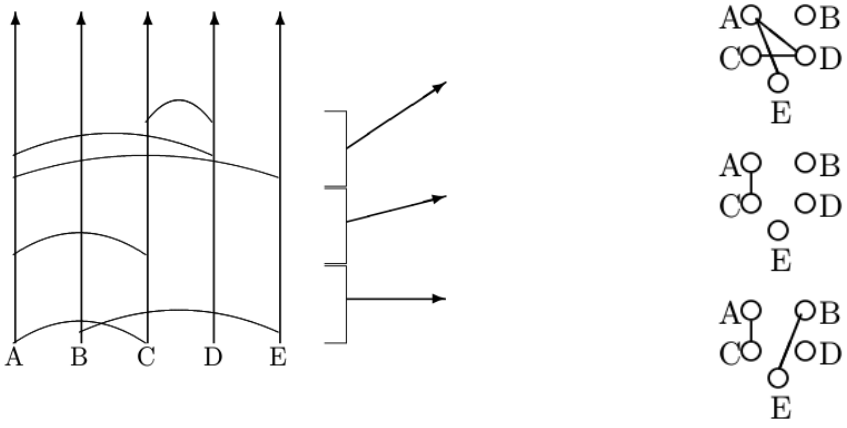
2.2 Temporal Networks

Many times, the connections or interactions between entities in a social network change frequently over time. For example, researchers will co-author papers with different colleagues and actors will co-star with different actors in different movies. Representing such a system with a single, static network can lead to misleading conclusions. Entities in the graph that appear to be closely related may, in fact, be very distant with respect to time.

These systems can be more accurately modelled by temporal networks. In a static network, an interaction can be represented by a triple $< u, v, w >$, where u and v are the interacting entities, and w is a valuation on the strength of the connection. In temporal networks, we consider an interaction as a tuple $< u, v, t, \delta t >$, where u and v are still entities, t is the time that the interaction began, and δt is the duration of the interaction. This representation captures more information about the system, but it is difficult to visualize both the temporal and structural properties of this representation. Figure 2.2 shows a few popular approaches. We use the time-aggregated graph construction and visualization in this report. Specifically, we assume all interactions take place within some time interval $[0 \dots T]$.

This time range is then uniformly partitioned into N time windows. The i^{th} time window is then $\tau_i = [(i-1)\frac{T}{N}, i\frac{T}{N})$. In addition, a graph corresponding to each time window, $G(\tau_i)$ or just G_i, is constructed by only considering interactions $< u, v, t_k, \delta t_k >$ where the time interval $[t_k, t_k + \delta t_k]$ overlaps with time window τ_i . We refer to the sequence of networks $\{G(\tau_1), G(\tau_2), \dots, G(\tau_N)\}$ as a temporal network.

The number of time windows used greatly influences the dynamics of the system [Caceres and Berger-Wolf 2013]. Choosing the 'right' number of time windows for any given system is difficult. First, a formal definition for the 'right' number of time windows has to be constructed. Then an efficient method of finding that interval needs to be developed. Current approaches analyze the compression ratios



(a) Time-line plot

(b) Time series

Figure 2.2: Two possible visualizations of a temporal network. In (a), each vertical arrow is a single entity throughout time, and an arc represents an interaction between entities. This layout emphasizes the temporal characteristics of the network. In (b), the time line of interactions has been broken into time windows, and a graph has been constructed for each window, emphasizing the structural features of the network.

and variance of network characteristic across time windows in order to determine time slice duration [Sulo et al. 2010; Sun et al. 2007]. We use manually determined time windows in this report.

2.2.1 Metrics

All of the static network metrics discussed in Section 2.1.1 can be used to analyze each of the networks constructed within a single time window separately. However, the metrics that consider the full temporal network can provide insight into the time dependent dynamics of the network. Due to the only recent rise in popularity of temporal network analysis, the characteristic features of temporal networks are not as well-established as those of static networks. In this section, we describe some of the more prominent features of temporal networks to date and the metrics that have been developed to quantify them.

(a) Exponential **(b) Power Law**

Figure 2.3: Comparison between interaction streams with (a) exponential wait times and (b) power law wait times. The power law wait times result in bunches of interactions mixed with longer periods of down time.

2.2.1.1 Small World Property

Paths in static networks are based on the idea of adjacency of vertices. Two vertices are adjacent if they simply share an edge. In temporal networks, adjacency is time dependent, as vertices are only adjacent for the duration of an edge. Similar to static, directed graphs, this means paths in temporal graphs are not reversible in general. For example, consider the time series in Figure 2.2. The exists a temporal path from B to A, first traversing the edge from B to E in the first time window, waiting at vertex E during the second time window, and then traversing the edge from E to A in the last. There are no such temporal paths from A to B, however. Time respecting paths have two separate intuitive lengths. The structural length is the number of edges traversed along the path, the same as in static networks. In addition, there is a temporal length, which is the difference in time between the end of the paths last edge and the beginning of the paths first edge. The example path from B to A in Figure 2.2 would therefore have a structural length of two, and a temporal length of three. The temporal distance between two vertices is defined as the minimum temporal length of any path between the vertices. This distance can be used in exactly the same manner as in static networks. The diameter of a temporal network, for instance, is taken as the maximum temporal distance between any two vertices in the network, and has been shown to be comparable to the diameter of temporal networks [Tang et al. 2010].

2.2.1.2 Persistence

While edges typically do not persist across all time windows, it has been observed that if an edge between two vertices exists in a time window k, there is a non-negligible probability that it also exists in time window $k+1$ [Tang et al. 2010]. This characteristic was quantified by a measure called temporal-correlation coefficient in [Nicosia et al. 2013]. For a single vertex i, the persistence of edges across two time windows, t_k and t_{k+1} , is quantified by looking at the overlap between the neighborhoods of i in the two time windows. Specifically, the topological overlap is defined as:

$$\varphi_i(t_k, t_{k+1}) = \frac{\sum_j a_{ij}(t_k)a_{ij}(t_{k+1})}{\sqrt{(\sum_j a_{ij}(t_k))(\sum_j a_{ij}(t_{k+1}))}}$$

The average temporal overlap is simply the average of this value across all adjacent pairs of time windows:

$$\varphi_i = \frac{1}{N-1}\sum_{k=1}^{N-1} \varphi_i(t_k, t_{k+1})$$

The temporal-correlation coefficient, which we denote as φ, is simply the average φ_i over all vertices. Values for φ in real networks have been shown to be significantly higher than values in temporal networks where the time windows have been randomly permuted [Tang et al. 2010].

2.2.1.3 Burstiness

Burstiness looks at the pattern of interactions between entities in a social network. Human activities were originally thought to be homogeneous with respect to time, like in Fig 2.3a. It has been empirically shown, however, that human activities are overwhelmingly inhomogeneous, portraying a 'bursty' behavior [Paxson and Floyd 1995; Dewes et al. 2003] such as that in Fig 2.3b. The time between interactions, which we will refer to as wait times, have been found to approximate power law distributions while the lengths of consecutive wait times have been found to be highly correlated [Min and Goh 2013].

We utilize the two measures were developed in [Min and Goh 2013]. Consider

a set of N interactions between two entities. There are $N-1$ wait times between consecutive interactions with a duration w_i . The burstiness measure quantifies the difference of the distribution of waiting times from a Poisson distribution using the coefficient of variation. If μ is the average waiting time and σ is the standard deviation of the waiting times, the burstiness parameter is defined as :

$$B = \frac{\sigma - \mu}{\sigma + \mu}$$

A similar parameter is constructed to quantify the correlation between pairs of consecutive wait times $< w_i, w_{i+1} >$. This measure, called the memory parameter is defined as :

$$M = \frac{1}{N-1} \sum_{i=1}^{N-1} \frac{(w_i - \mu_1)(w_{i+1} - \mu_2)}{\sigma_1 \sigma_2}$$

where μ_1 and σ_1 are, respectively, the mean and standard deviation of the w_i values and μ_2 and σ_2 are the same for the w_{i+1} values.

Both measures have values that range between -1 and 1. Higher values correspond to a more bursty or correlated system. Negative values indicate a much more regular system.

2.2.2 Temporal Communities

Community detection in temporal networks is usually focused on how community structure changes over time, or community 'evolutions'. Typically, for each time window τ_k, a community structure $\mathcal{C}_k = \{C_{k1}, C_{k2}, \ldots, C_{kn_k}\}$ is constructed using any of the community detection algorithms for static networks (see Section 2.1.2). Some methods augment the static community detection process at time window k with information from the community structures detected in previous time windows. Usually, a 'temporal cost' is introduced into the detection process, penalizing an algorithm for constructing community structures that differ too greatly from previous structures [Lin et al. 2008; Chakrabarti et al. 2006].

Once communities are detected within each time window, a second process is used to associate similar communities across time windows. Groups of associ-

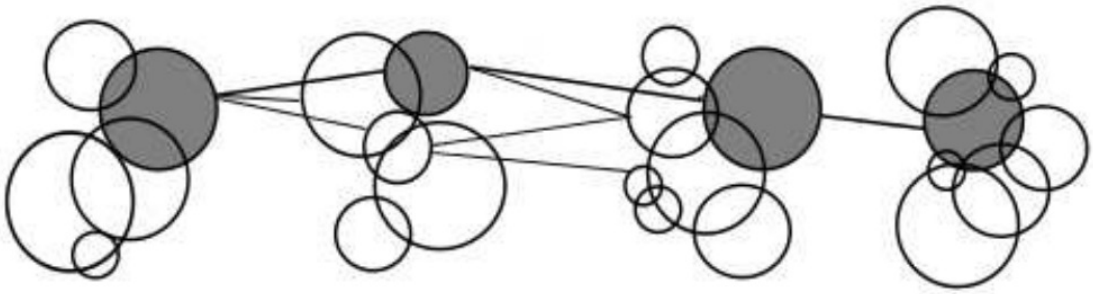


Figure 2.4: Visualization of a community evolution. Each column is a time window and each circle represents a community detected in that window. The edges between communities represent transitions that have been determined to be valid evolution steps between communities in different time windows. The shaded communities show a maximal length evolution of a single community.

ated communities are taken to be community evolutions. Many different techniques have been used to quantify the similarity between communities, including using the Jaccard Index [Bródka et al. 2013; Goldberg et al. 2012; Greene et al. 2010; Takaffoli et al. 2011], properties of random walks [Lin et al. 2008], and graph coloring [Tantipathananandh et al. 2007]. In addition, the time windows do not have to be consecutive, in general.

2.2.3 Temporal Models

Just like in static networks, random temporal network models are useful for everything from testing algorithms on a wide range of different structures to providing a mechanistic explanation of different phenomena.

2.2.3.1 Randomized Null Models

All the different metrics developed for temporal networks attempt to quantitatively capture some structural property. Taken independently, however, a value for any metric is extremely difficult to interpret. For example, a high temporal clustering coefficient value would be meaningless if a random sequence of random temporal networks also displayed a high temporal-clustering coefficient. These metrics are most useful when compared to a baseline value calculated from random

networks.

Baseline metric values are typically calculated using a class of random networks that provide a 'neutral' representation of the structure the metric quantifies. In static networks, the configuration model is able to construct networks that keep the same degree distribution without implicit edge correlations, and was used to derive the modularity measure. Therefore, if the value of a metric of a specific network, such as the clustering coefficient, differs significantly from the value of the metric applied to networks constructed using the configuration model with the same degree distribution, the structural characteristics of the network can be considered meaningful.

Random models similar to the configuration model have been used in temporal networks to remove correlations between the timing of edges. These networks keep the static structure of the network constant and permute the time stamps associated with the edges. One approach permutes the ordering of time slices while another randomly switches the time stamps between pairs of edges [Holme and Saramäki 2013]. Both approaches remove temporal correlation between edges, providing a neutral structure to construct baseline values for metrics such as burstiness or persistence.

It is also possible that the time stamps themselves follow a specific pattern of interest. For instance, human circadian rhythms may cause interactions and communications to concentrate around certain hours of the day. In this case, a random model may pick new time stamps for each edge in order to investigate the impact of overall timing structure on network dynamics.

2.2.3.2 Queue Models

Another class of network models attempts to construct networks with a specific property or characteristic. In static networks, the WS model focused on networks with a small diameter while preferential attachment models focused on power law degree distribution. With temporal networks, the characteristic of interest is the burstiness of interactions.

In [Barabási 2005], a simple model based on priority queues was proposed to

explain bursty behavior. In this model, every entity in the network is modelled with a fixed-size queue of tasks. Each task is also given a priority between 0 and 1, drawn from a uniform distribution. The model then proceeds iteratively, removing the task with the highest priority (representing the task being performed) and replacing it in the queue with a new task with a random priority. The resulting waiting times of tasks, measured by the number of iterations the task is in the queue, was shown to approximate a power law with an exponent of about 1.

The original model only considers activity of single entities, not interactions between entities. An extension of the model that incorporates interaction tasks, where two entities need to execute the same task at the same time, was made in [Oliveira and Vazquez 2009]. The interaction task is added to both entities' queues with random priorities and is not removed until it becomes the task with the highest priority in each queue. Extensions to the model, including interaction tasks that can be initiated by a single entity and interactions requiring more than two people, were investigated in [Min and Goh 2013]. Many of the same properties where found in all models, with waiting times between interactions approximating a power law with an exponent between 1 and 2.

2.2.3.3 Random streams

A third class of network models attempts to construct networks with structures that mimic all of the characteristics of real world networks. One of the simplest approaches constructs a static network for each of the time slices. Any method of network construction for static networks can be used for each time slice. Unfortunately, this method does not construct networks with temporal correlations in general, and the networks will not display burstiness or persistence. Extra constraints in the static network construction or an extra post-processing step is required. A more general approach assigns a generation probability to every possible edge in every time slice [Caceres and Berger-Wolf 2013]. If all probabilities are equal, a conceptual extension of the Erdos-Renyi model for static networks is recovered.

CHAPTER 3
Community Evolution Framework

In this section, we develop an algorithmic framework for analyzing evolutions in temporal networks. We call this the LOS-Temporal or LOST framework, as it stems from work done on the static algorithm LOS [Goldberg et al. 2010]. We assume that a community structure has been detected for the network in each window of a temporal network, resulting in a sequence of community structures, $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_N\}$. An evolution is defined as a set of communities, with no more than one community from each window, representing the different manifestations of a single community across time. In order to construct meaningful evolutions, we first consider continuous evolutions, where the communities are taken from consecutive time windows. We then merge similar evolutions together in order to detect intermittent evolutionary structures.

3.1 Uninterrupted Evolutions

We define a chain of communities as a set of communities from consecutive time windows, $\{X_0, X_1, \ldots, X_l\}$, that contains a single community from each time window. Detecting continuous evolutions in temporal networks can then be formulated as finding meaningful or interesting chains. To determine if a chain is meaningful, we assume that there exists a function $F(\cdot)$, which takes a chain of communities as input, and outputs a strength value between 0 and 1. A higher strength value should indicate a more meaningful chain. An evolution is therefore a chain with a strength value above some threshold θ.

Even with the function $F(\cdot)$, detecting such chains is non-trivial. A brute

Portions of this chapter previously appeared as: Mark Goldberg, Malik Magdon-Ismail, Srinivas Nambirajan, and James Thompson. 2011. Tracking and predicting evolution of social communities. In *2011 IEEE Third International Conference on Social Computing (SocialCom)*. IEEE, Boston, MA, USA, 780-783.

Portions of this chapter previously appeared as: Mark Goldberg, Malik Magdon-Ismail, and James Thompson. 2012. Identifying long lived social communities using structural properties. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*. IEEE, Istanbul, Turkey, 647-653.

force search over all possible chains in a temporal network will quickly become computationally prohibitive as it is possible that there are an exponential number of chains. However, efficient detection algorithms can be utilized as long as $F(\cdot)$ satisfies only a few properties.

3.1.1 Axioms for Community Evolution

We propose the three basic properties that a function $F(\cdot)$ should have in order for it to be considered a valid measure of the strength of a chain of communities:

Property 1. *Identity*

A chain constructed by repeating a single community X has maximum strength:

$$F(X, X, ..., X) = 1$$

Property 2. *Monotonicity*

A chains strength is no larger than the strength of any subchain:

$$\forall_{0 \leq i \leq j \leq l} \; F(X_0, X_1, \ldots, X_l) \leq F(X_i, X_{i+1}, \ldots, X_j)$$

Property 3. *Extension*

Consider two evolutions that have a community in common, $\mathcal{X} = \{X_0, X_1, \ldots, X_l\}$ and $\mathcal{Y} = \{Y_0, Y_1, \ldots, Y_k\}$ where $X_i = Y_j$ for some i and j. The chain constructed by extending the prefix of $\mathcal{X}$ up to X_i with the suffix of $\mathcal{Y}$ from Y_{j+1} is also an evolution:

$$\begin{matrix} F(X) \geq \theta \\ F(Y) \geq \theta \end{matrix} \Rightarrow F(X_0, X_1, \ldots, X_i, Y_{j+1}, \ldots, Y_k) \geq \theta$$

3.1.2 Constructing Evolutions

These properties are intuitive and allow an evolution X to be defined simply as a chain $\{X_0, X_1, \ldots, X_l\}$ where the strength of each consecutive pair of communities is above some threshold ($\forall_{0 \leq i < k} F(X_i, X_{i+1}) \geq \theta$). Furthermore, it allows evolutions

to be characterized by $F(\cdot)$ functions that are only defined for pairs of communities instead of chains of arbitrary length.

Theorem 1. *If F satisfies the identity, monotonicity and extension axioms, then, for any chain $\{X_0, \ldots, X_l\}$, $F(X_0, \ldots, X_l) = min_{i=0,\ldots,l-1} F(X_i, X_{i+1})$.*

Proof. Let $\lambda = min_i\ F(X_i, X_{i+1})$. By monotonicity, $F(X_0, \ldots, X_k) \leq \lambda$ since for some pair (a special case of a subchain) $F(X_i, X_{i+1}) = \lambda$. Suppose that $F(X_0, \ldots, X_k) < \lambda$. Then, for threshold λ, $X_0, \ldots, X_k$ is not valid, but every consecutive pair in $X_0, \ldots, X_k$ is valid. By the extension axiom, since (X_0, X_1) is valid and (X_1, X_2) is valid, (X_0, X_1, X_2) must be valid. An easy induction shows that $X_0, \ldots, X_k$ must be valid, a contradiction. Thus, $F(X_0, \ldots, X_k) = \lambda$. □

A direct consequence of this formulation is the basis for an efficient evolution detection algorithm. If evolution $\mathcal{X}$ ends at time window k, it can be extended to an evolution that ends at time window $k+1$ by adding any community $Y \in C_{k+1}$ where $F(X_k, Y) \geq \theta$. Therefore, in order to detect evolutions, the values of $F(\cdot)$ only need to be calculated for all pairs of communities in consecutive time windows. Evolutions can then be constructed one step at a time, in one pass of the data (see Algorithm 1).

Naively calculating $F(\cdot)$ for every possible pair of consecutive communities can still be computationally prohibitive. This can usually be avoided using a simple indexing method to avoid wasting time on pairs of communities with a strength of zero. Consider using the Jaccard Index to compare communities. If two communities do not share a member, their similarity will be zero. Naively comparing the memberships of pairs of communities with no similar members is therefore computationally wasteful.

To make the computation more efficient, the indexing approach first iterates through all the vertices of the temporal network and examines the communities it is a member of. Each pair of communities in consecutive time windows that the vertex is a member of is kept in a list. After all the vertices have been iterated over, only the $F(\cdot)$ values for pairs of communities in the list are calculated (see Algorithm 1). In theory, this may not reduce the number of necessary calculations of $F(\cdot)$, but

normally provides a significant speed up in practice.

Data: Community structures for l time windows, $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_l\}$, comparison function $F(\cdot, \cdot)$, threshold λ

Result: All valid evolution chains of one step

```
/* Calculate all of the single-step evolutions using F() and
   the indexing method to increase performance                */
```

for $i = l - 1; i \geq 0; --i$ **do**

```
    /* Index memberships in the following time window         */
```

for $j = 0; j < |\mathcal{C}_{i+1}|; ++j$ **do**

for $v \in \mathcal{C}_{i+1,j}$ **do**

$membership[v] \leftarrow membership[v] \cap j$

end

end

```
    /* Compare communities across time windows                */
```

for $j = 0; k < |\mathcal{C}_i|; ++j$ **do**

for $v \in \mathcal{C}_{i,j}$ **do**

$to_compare \leftarrow to_compare \cap membership[v]$

end

for $k \in to_compare$ **do**

if $F(\mathcal{C}_{i,j}, \mathcal{C}_{i+1,k})$ **then**

$M[i][j] \leftarrow M[i][j] \cap \mathcal{C}_{i+1,k}$

end

end

end

end

```
/* An evolution starting at the j^th community of time window i
   can be recovered by following single step evolutions using
   M, starting at M[i][j]                                     */
```

Algorithm 1: Constructing evolutions using $F(\cdot)$ oracle to compare strength of connections between pairs of communities.

3.2 Combining Evolutions

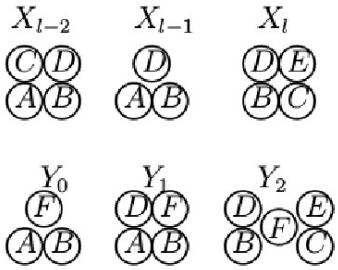
It is possible that a community evolution is not manifested in consecutive time windows, but is intermittently active. This can be due to any of a number of factors. If the members of the community only interact periodically, the choice of time window duration may be short enough that a full time window falls during a period of inactivity. In addition, community detection algorithms may miss some communities in each time window. Whatever the reason, if the evolution construction approach described in the previous section is used, the evolution will be detected as multiple disjoint evolutions.

To detect intermittent evolutions, we combine continuous evolutions in a similar manner used previously to connect similar communities. Consider two continuous evolutions, $\mathcal{X} = \{X_0, X_1, \ldots, X_l\}$ and $\mathcal{Y} = \{Y_0, Y_1, \ldots, Y_k\}$, where X_l is in time window l, and Y_0 is in a later time window $t > l$. To compare the two evolutions, we assume there exists a function $R(\cdot)$ that calculates a feature vector for a given evolution. We then define the similarity between evolutions as the cosine similarity between the two feature vectors:

$$L(X, Y) = \frac{R(X) \cdot R(Y)}{||R(X)|| * ||R(Y)||}$$

If the similarity between two evolutions is higher than some threshold, we merge the two evolutions together into a single evolution $\{X_0, X_1, \ldots, X_l, Y_0, Y_1, \ldots, Y_k\}$.

In the analysis in this paper, we consider a feature vector based on community memberships of $\mathcal{X}$ and $\mathcal{Y}$. Specifically, we only consider the last p communities of $\mathcal{X}$ the earlier evolution and the first p communities of $\mathcal{Y}$. If p is larger than the length of either $\mathcal{X}$ or $\mathcal{Y}$, we just consider whatever communities are present in that evolution. To make certain that the feature vectors are the same length, we calculate an entry for each vertex in the union of all considered communities. The entry for vertex v is set to be simply the number of communities in the evolution which v is a member of. Figure 3.1 shows an example calculation of the similarity between evolutions where p is 3.



Node Participation						
Node	X_{l-2}	X_{l-1}	X_l	Y_0	Y_1	Y_2
A	1	1	0	1	1	0
B	1	1	1	1	1	1
C	1	0	1	0	0	1
D	1	1	1	0	1	1
E	0	0	1	0	0	1
F	0	0	0	1	1	1

Participation Vectors

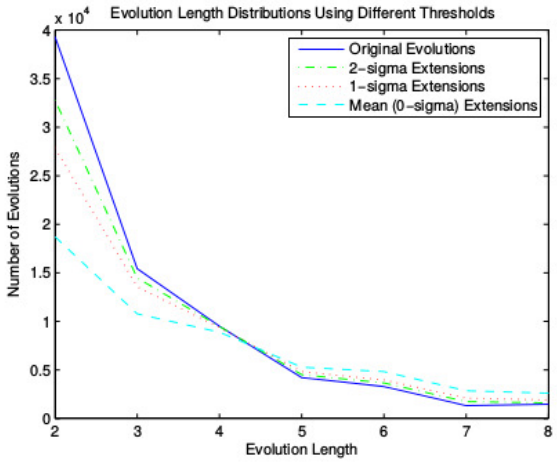
$$R(\mathcal{X}) = \begin{bmatrix} 2 & 3 & 2 & 3 & 1 & 0 \end{bmatrix}$$
$$R(\mathcal{Y}) = \begin{bmatrix} 2 & 3 & 1 & 2 & 1 & 3 \end{bmatrix}$$

$$L(\mathcal{X},\mathcal{Y}) = \frac{(2*2)+(3*3)+(2*1)+(3*2)+(1*1)+(0*3)}{(2+3+2+3+1+0)*(2+3+1+2+1+3)} = \frac{1}{6}$$

Figure 3.1: Calculating the M-value between two evolutions. The nodes in the last three communities of evolution $\mathcal{X}$ and the first three evolutions of $\mathcal{Y}$ are shown at the top, followed by the specific calculation.

3.3 Empirical Experiments

We use the evolution detection framework described in the previous section to empirically study two real world networks: the DBLP co-authorship database and LiveJournal (BLOG). Both data sets represent systems where groups of people collaborate on a single object. In the DBLP data set, authors collaborate on researching and writing an academic paper while in the BLOG data set, users collaborate in a conversation via comment threads. The publish date of the papers in DBLP and the comment time of comments in BLOG are used as time stamps for interactions. A time window of one year was used in DBLP (there are 19 total time windows), while one week was used in the BLOG data set (66 total).



	Number of Evolutions			
Len	Orig	$+ 2\sigma$	$+ \sigma$	Avg M
2	39335	32795	27900	18731
3	15429	14468	13565	10779
4	9509	9483	9375	8901
5	4212	4529	4830	5293
6	3288	3670	3961	4837
7	1372	1747	2096	2861
8	1440	1613	1872	2568
9	509	728	957	1512
10	618	797	1001	1432
11	223	383	510	868
12	292	376	470	826
13	81	176	262	514
14	125	187	292	420
15	36	83	112	252
16	74	100	137	254
17	8	30	72	174
18	40	57	64	98
19	9	25	33	63
20	23	37	40	45

Figure 3.2: Distributions of evolution lengths for original evolutions and evolutions found using different merging thresholds. A large number of merges occurs between smaller evolutionos, causing the distributions to even out as the threshold is lowered. The figure displays the most drastic changes in evolution length distributions across thresholds and the table shows corresponding values.

The same algorithms and thresholds were used in both data sets. Specifically we use the LOS algorithm [Goldberg et al. 2010] to detect static communities within each time window. To detect continuous evolutions, we use the Jaccard Index for the $F(\cdot)$ function:

$$F(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

Values of $F(\cdot)$ above 0.2 are considered valid for evolutions. To construct a threshold for evolution similarities, we consider the set of all similarity values between evolutions. We then define the threshold as the average of this set plus two standard deviations in order to be confident that the evolutions merges are meaningful.

Figure 3.2 shows the distribution of evolution lengths for both continuous evolutions and merged evolutions with different thresholds for the merge process in both data sets. The distribution is heavy-tailed, with the majority of evolutions having a length less than four, but some evolutions growing as long as 20 time windows.

Since there are no known 'true' communities in either data set, we validate the accuracy or legitimacy of the detected evolutions manually by examining the abstracts of papers in the same evolution for DBLP and similarly looking at blog topic for the BLOG data set. To construct a visualization of an evolution, we use word clouds to view the most frequently used terms in the paper abstracts or blogs. More frequently used words are drawn bigger in the word cloud. A valid community will have main ideas or topics that are easily recovered from word clouds while valid evolution will have consistency or logical development of ideas and topics throughout the chain of communities.

Figure 3.3 shows an example set of word clouds for an evolution in DBLP. Each column of words was originally detected as a continuous evolution and the two evolutions were combined during the merging algorithm. In the beginning of the first evolution, the authors main focus was on using inductive inference to solve problems. The focus quickly becomes learning, a more general topic that includes inductive inference. Alongside learning comes the interest in using or studying languages. The end of the first evolution and the beginning of the second share the main common topic of validation, a logical progression from developing techniques for learning. Finally, the continuity of the second evolution is made clear when the focus on learning returns strongly, with the other keywords appearing frequently as well.

3.4 Evolution Prediction

We consider the first four communities of an evolution, $\{X_0, X_1, X_2, X_3\}$, and try to predict the length of the evolution. The same analysis could use any number of beginning communities of an evolution. Using four communities provides a long enough chain to give confidence that the evolution is not random, and also allows

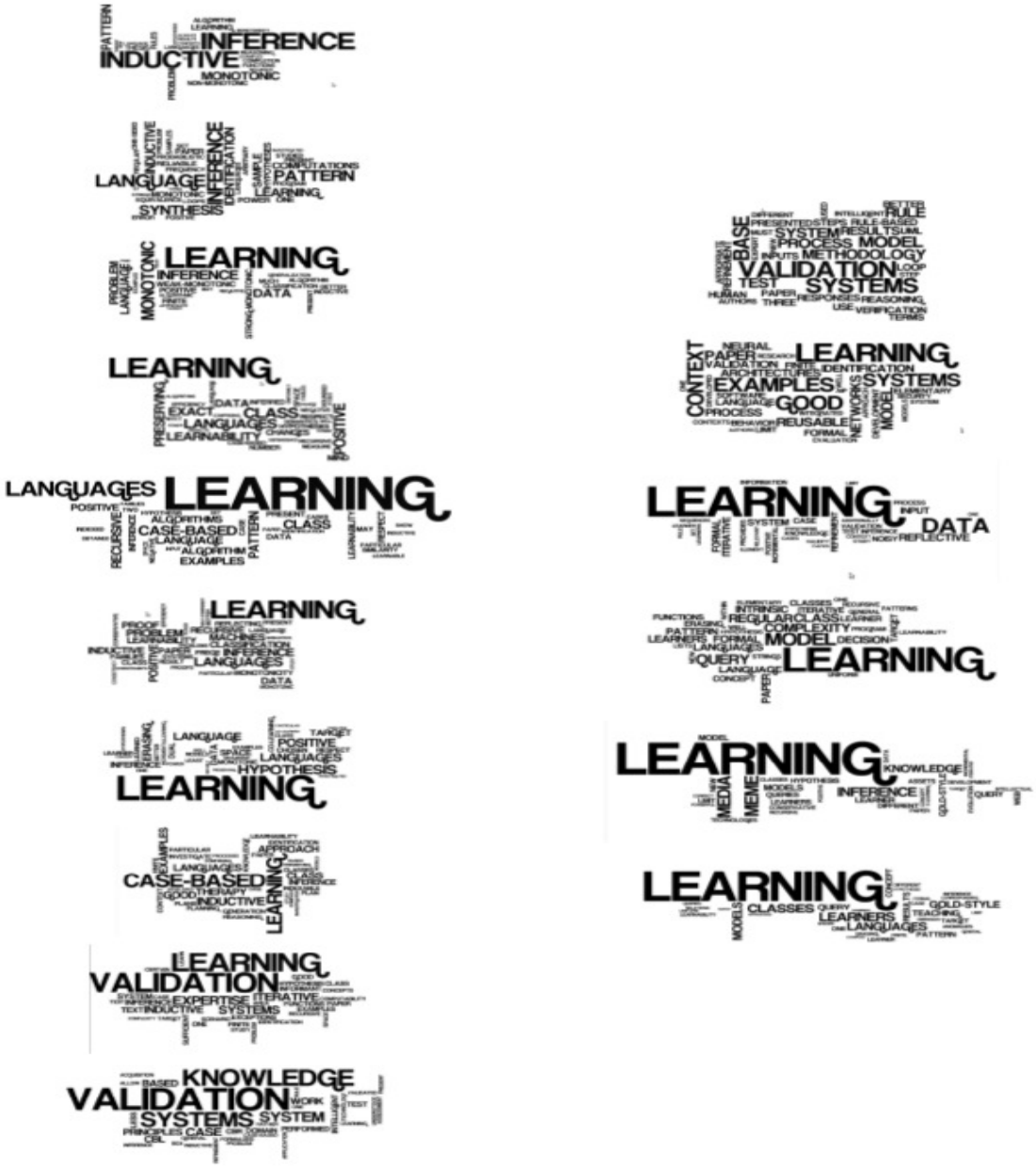


Figure 3.3: Word clouds of paper abstracts in communities of two merged evolutionos in the DBLP network. Word sizes are determined by frequency in abstracts. Each column represents an original evolution.

Table 3.1: Most predictive characteristics of evolution length in the original and merged evolutions. The Wgt entry denotes whether the features based on the characteristic generally had a positive (+) or negative (-) correlation with evolution length.

Parameter	Score	Wgt
Size (s_i)	29	
Growth		+
Average		-
Density (d_i)	26	+
Intersection (r_i)	19	+
Core (q_i)	9	+

for the examination of most of the detected evolutions.

Let $s_i = |X_i|$ and $d_i = D(X_i)$ be the size and conductance of a community, respectively. The conductance of a community X_i is defined as:

$$D(X_i) = \frac{W_{in}}{W_{in} + W_{out}}$$

where W_{in} is the sum of the weights of edges connecting two members of the community and W_{out} is the sum of the weights of edges connected to at least one member of the community. We let $r_i = |X_i \cap X_{i+1}|$ be the size of the pairwise intersections, define the core sizes $q_i = |X_i \cap X_{i+1} \cap X_{i+2}|$ as the intersection sizes of the r_i , and define the hypercore size $h_i = |X_i \cap X_{i+1} \cap X_{i+2} \cap X_{i+3}|$ as the size of the intersection of the cores. In addition, let $c_{r_i} = D(|X_i \cap X_{i+1}|)$ be the conductances of the nodes in the intersection of two communities in respect to the union of the graphs from each time step. Similarly, let $c_{q_i} = D(|X_i \cap X_{i+1} \cap X_{i+2}|)$ be the core conductances and $c_{h_i} = D(|\cap_{k=i}^{i+3} X_k|)$ be the hyper-core conductance. Using these parameters, we derive 79 features to characterize the early stages of an evolution. For community sizes, we use the average size, average change in size between consecutive communities, and normalizations of these values using the minimum sized community and the first community of the evolution. Equivalent features are constructed for all of the other properties.

We use these properties to construct features to use in a simple linear regression

framework with leave-one-out cross validation in order to identify the properties most indicative of evolution longevity. Table 3.1 shows the results of the linear regression framework. To construct a parameters score, we consider the fifteen features determined to be the most predictive of evolution length. The i_{th} most predictive feature is given a score of $15 - i$. The final score of a property is the sum of the scores of the features derived from it. The most predictive feature was the size of the communities of the evolution. If the evolution experiences growth in the first four communities, it is likely to last longer. If the communities are small, however, the evolution will likely be short lived. This reflects the intuition that a strong, meaningful community should be able to attract members as it starts to establish itself.

CHAPTER 4
Synthetic Evolutions

Validation of community detection algorithms has been a difficult task in any framework. Gathering data from real world networks can be time consuming and prone to errors or noise, even with the recent advances in technology. As a result, there are only a few static networks with a known community structure. There are even fewer temporal networks with a known evolution structure, thanks to increases in complexity and ambiguity. In this chapter, we develop a framework for generating synthetic temporal networks with properties that mimic those of real world networks and use it to compare and evaluate the performance of evolution detection algorithms.

4.1 Temporal Network Model

We construct a temporal network with M time windows in three steps. First, we generate vertex and community structures for each time window, $\{V_1, V_2, \ldots, V_M\}$ and $\{C_1, C_2, \ldots, C_M\}$, respectively. Next, evolutionary events are embedded into transitions between consecutive time windows. The evolutionary events that occur between time window i and window $i+1$ do not influence those between windows $i+1$ and $i+2$.

Finally, we also define a function $H(V, C)$, which takes a vertex structure and a community structure as input and generates a network $G = (V, E)$. The specific characteristics of each static network are determined by the definition for H. The vertex and community structures associated with each time window are then used as inputs to the function, $H(V_i, C_i)$, which generates the network for that time window, $G_i = (V_i, E_i)$. Figure 4.1 describes the overall process.

4.1.1 Seeding the network

Being the first network in the series, the vertex and community structures of G_1 do not have to take restrictions related to embedding evolutionary events in

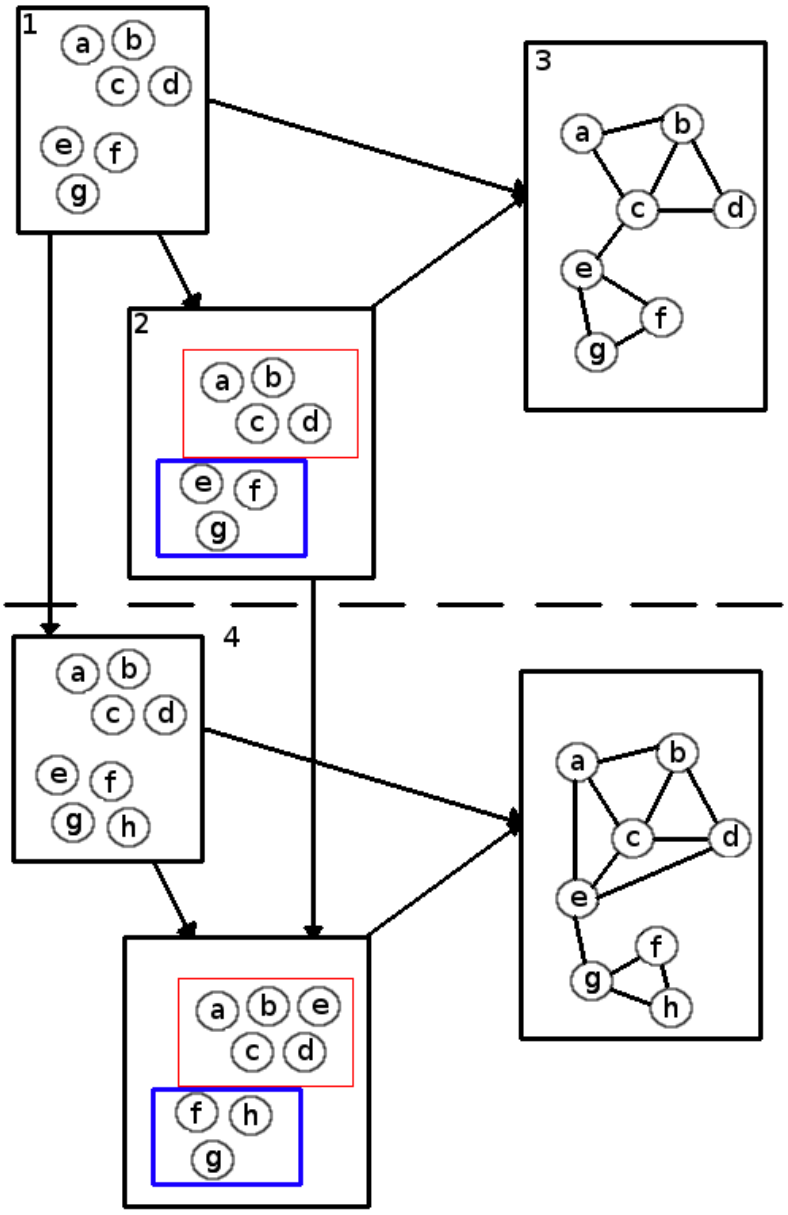


Figure 4.1: Constructing a synthetic network. (1) Random vertices are generated. (2) Random communities are generated by randomly selecting vertices. (3) Using the vertex and community structures, a network structure is generated. (4) The vertex and community structures are perturbed to create the next time window's vertex and community structures.

the network into account. We assume that number of vertices in the first network is explicitly specified. In order to generate a power law degree distribution, we assign each vertex a 'lag' value from a predefined power law distribution. The lag value of a vertex approximates the minimum wait time between interactions between the vertex and any other, single vertex in units of time windows. In other words, edges with low lag values can interact with others more frequently, resulting in more edges with larger weights being connected to the vertex. In the DBLP co-authorship network, the low lag vertices represent the authors that publish either frequently or with a large number of colleagues.

Once the number of vertices and their lag values have been determined, we generate a random community structure. The number of communities is either directly specified or given as an average community membership for vertices. The sizes of the generated communities are generated from a specified power law distribution. Although the community memberships have no temporal restrictions to follow, uniformly choosing random vertices to fill each community can cause issues. Since vertices with high lag values generally end up with a small degree, if they belong to a large number of communities, the communities either need to be highly redundant and overlapping or the vertex needs to be weakly connected to members in any single community. Conversely, a vertex with a low lag value in a single community will end up with a large number of edges or interactions with vertices it does not share a community membership with. In order to help prevent high lag vertices from becoming members of many communities, vertices with lower lag values are given a proportionally higher probability of being chosen for the initial community memberships.

4.1.2 Embedding Evolutions

We break transitioning between time windows into two main categories, perturbing the vertices of the network, and perturbing the community structure. In both cases, we use a simple, generally accepted events empirically found in real world networks [Takaffoli et al. 2011; Bródka et al. 2013] to imitate real world dynamics.

4.1.2.1 Vertex Events

Real world interaction networks have been empirically shown to grow in size between time windows, especially in their first few time windows. Sometimes, this is due to a lack of data gathering. The rise of digital technologies in recent decades has made gathering and tracking data easy, and the growth of networks is usually attributed to increasing interest, as when starting a new social networking application.

To mimic this phenomenon, a random number of nodes is added to the network at each time window. The number of added vertices is equivalent to a percentage of the current size of the network. A minimum and maximum growth rate is specified, and a random value is uniformly sample between these boundaries for the growth at each interval. New vertices are also given a lag value, sampled from the same power law distribution used in the first network.

We also consider each individual vertex's choice to either stay in the communities they are a part of or leave between time windows. We specify a probability that a vertex will be removed from its current communities. Each community membership is considered independently, and we again keep the probability constant for all memberships for simplicity. To keep the community sizes from drastically decreasing at each interval and represent the possibility of vertices choosing to join communities between time windows, we immediately replace vertices removed from a community with randomly selected vertices (based on lag values). There is a possibility that a removed vertex is chosen to replace itself, but it is extremely small and does not influence network dynamics.

4.1.2.2 Community Events

Many evolution detection approaches utilize a community-level evolution event framework to analyze evolutionary structures [Takaffoli et al. 2011; Bródka et al. 2013]. Our framework uses six of the commonly used events to perturb the community structure between time windows:

Merge

Two communities from the same time window, $C_{i,1}$ and $C_{i,2}$, are said to merge if they both evolve into the same community in the next time window C_{i+1}.

It is possible for groups of more than two communities to merge together between time windows, but we only consider merges between pairs of communities in our framework. To find pairs of communities to merge, we randomly select a subset of all communities to participate in merges. Each community is given the same probability of being included in this set. We then randomly match the communities in pairs and merge their memberships. If the number of communities chosen was odd, we simply ignore one of the chosen communities.

Split

The opposite of merging, a community in some time window C_i splits into communities $C_{i+1,1}$ and $C_{i+1,2}$ in the next time window if both $C_{i+1,1}$ and $C_{i+1,2}$ retain a high number of members from C_i.

For a community to be considered for splitting in our framework, it first must not have been chosen for the merging process. To avoid splitting smaller communities, there is also a specified size threshold that any community must be larger than in order to be split. A specified percentage of these communities is then chosen to be split. To split the community, a subset of community members are chosen to 'split off' and form another community. A random number of the members 'splitting off' are chosen to remain in the original community as well, resulting in these vertices gaining a community membership. It is possible for no members to be copied in this manner, resulting in a partition of the original community. We force the new community to chose at least three members to start with and ensure the original community retains at least three members, which is the minimum size we consider for communities.

Grow / Shrink

This event reflects no big changes in the community from one step to the next. Simply, a community C_i grows (or shrinks) into community C_{i+1} if the two communities are not part of a merge or split event and their memberships are

extremely similar.

In our framework, every community grows or shrinks at each interval and can then also partake in a merge or split event. The growth (shrinking) process is governed by two values. First, each community has the same probability of growing. If it does not grow, it will shrink. Second, the amount of growth (or shrinking) is uniformly sampled from an interval between zero and a specified upper limit, inclusively. Therefore, a community may 'grow' by 0 vertices, which is typically referred to as a 'continue' or 'survive' event.

Form

Communities do not have to be present from the very beginning of a temporal network. New communities of scientific research are constantly being formed, for example. A community C_i is said to have formed if it is not part of any event with any community in a preceding time window. Newly formed communities provide the starting point for new evolutions.

We embed a random number of brand new communities at each time window in our framework. Similar to how growth works, the number of communities added is uniformly sampled from a specified range (inclusive). It is therefore possible to specify a temporal network that may not add any new evolutions between time windows. The new community sizes are sampled from the same power law used when constructing the original network, and member vertices are chosen randomly based on lag values in order to keep consistent with the network that was already established.

Dissolve

Once started, many evolutions do not last the duration of the temporal network and end at a 'dissolve' event. A community C_i dissolves if it is not part of any events with a community in any later time window.

Each community is given a probability of being removed at each interval. For simplicity, we use a single probability for each community.

4.1.3 Generating Edge Structures

The edge structure of a network is constructed in two rounds. The first round iterates through every pair of vertices that share at least one community membership. We place an edge between each pair with a randomly generated edge weight. There is a relatively high probability that the generated edge weight is zero, in which case we ignore the edge. The second round randomly adds edges between vertices that do not share a community membership. A specified mixing parameter $0 < \mu < 1$ designates the fraction of edges of the network that should be found within communities. Therefore, if $|V_{in}|$ is the number of edges constructed between vertices that share a community in the first iteration, the number of edges generated in the second iteration is $\frac{|V_{in}|}{\mu} - |V_{in}|$.

In general networks, edge weights can represent similarity measures, physical distances, capacities, or just be somewhat arbitrary numbers. We are interested in generating interaction networks, where the edge weight between vertices u and v in a certain time window represents the number of times the two entities interacted within the time span that defines the window. As discussed in Section 2.2.1.3, it has been empirically shown that the time between interactions in real world networks generally exhibits bursty behavior. This is why we model the lag times of vertices with a power law.

The power law distribution takes three parameters: ρ_{min}, the minimum possible value of the distribution, ρ_{max}, the maximum possible value of the distribution, and λ, the distribution exponent, which controls the probability of higher values being generated. We take the ρ_{max} and λ variables as user defined, and vary ρ_{min} based off of the assigned vertex lag values. If, without loss of generality, the lag value of vertex u is lower than that of vertex v, we set the minimum waiting time for interactions between u and v as:

$$\rho_{min} = \varsigma(u) + \zeta * (\varsigma(v) - \varsigma(u)))$$

which is independent of the time window being considered and where ζ is a parameter between zero and one, and $\varsigma(v)$ is the lag value of vertex v. Typical distributions for edge weights between different pairs of vertices can be seen in Figure 4.2.

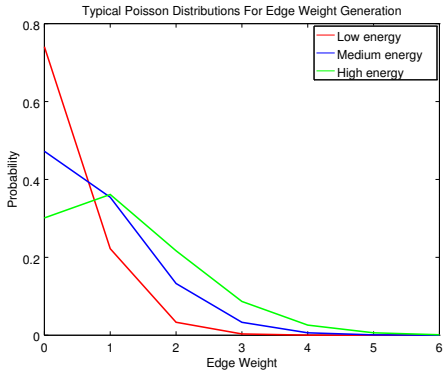


Figure 4.2: Probability distributions for edge weights between different pairs of vertices. Vertices with higher energies have higher probabilities of larger edge weights. Edge weights of zero are not included in networks.

The ζ parameter should be between zero and one, and mainly controls the behavior of edge generation between a low lag vertex and a high lag vertex. No matter what value of ζ is used, two high lag vertices will have a relatively low probability of being connected and two low lag vertices will have a high probability of being connected. Higher values of ζ result in higher lag vertices connecting with low lag vertices more frequently. The network will then become denser and the degree distribution will flatten out. Conversely, if high lag vertices do not have a high probability of connecting to a low lag vertex (a low value of ζ), they will not connect to any vertices, and the degree distribution will not include any hub vertices.

Using this approach, we have been able to develop networks that mimic a number of known characteristics of real world interaction networks (see Section 4.3).

4.2 Evaluating Evolution Detection Algorithms

A synthetic network model allows us to generate temporal networks with known community and evolution structures, which we can then use to analyze the performance of evolution detection algorithms.

4.2.1 Algorithms

We evaluate four evolution detection methods using our framework for generating networks. To make this report self-contained, we provide a short description of each algorithm here. More details can be found in the respective publications.

4.2.1.1 Community matching

The community matching approach is the same approach taken in the LOST framework described in Chapter 3 and the algorithm from [Greene et al. 2010], which we will call the GDC algorithm. In both approaches, it is assumed that communities have been detected for each static network, and a function $F(\cdot, \cdot)$ is used to compare communities across time windows. If the function value is above a threshold, the two communities are possible matches as a valid step of evolutions, and the possible match that maximizes $F(\cdot, \cdot)$ is considered at an evolution transition. The only difference between the two approaches is that, in GDC, instead of requiring communities to be from consecutive time windows, a community from time window i can be matched with any community from time window $j > i$.

In this evaluation, we test the LOST framework using two different metrics to compare communities: the Jaccard index used previously and a similar metric we call MinNorm:

$$MinNorm(A, B) = \frac{A \cap B}{min(|A|, |B|)}$$

4.2.1.2 Clique percolation method

In the clique percolation method (CPM) [Palla et al. 2005], communities are defined to be groups of vertices that can all be reached from one another by traversing cliques that differ by a single vertex. For our experiments, we use cliques of size three. Evolution detection using CPM follows the same framework as the community matching techniques. First, communities for all time windows, $\{C_1, C_2, \ldots, C_m\}$, are constructed. Then, to match communities from window i to communities from window $i + 1$, the algorithm is used to find a community structure $C_{i,i+1}$ in the union of the networks in both windows. Due to the properties of clique percolation, every community in C_i or C_{i+1} is a subset of exactly one community in $C_{i,i+1}$. Given a community in $C_{i,i+1}$, the communities in C_i that are a subset of $C_{i,i+1}$ are matched

to the communities of C_{i+1} that are a subset of $C_{i,i+1}$ in decreasing order of Jaccard similarity. If a community in C_i goes unmatched, it dissolves. Similarly, if a community in C_{i+1} goes unmatched, it starts a new evolution.

4.2.1.3 CommDy

CommDy [Tantipathananandh et al. 2007; Tantipathananandh and Berger-Wolf 2009] works a little differently from community matching algorithms, generating communities and evolutions simultaneously. First, instead of working from a typical network structure, the algorithm assumes that there were groups of entities observed together at each time window. This algorithm assumes that groups observed at each time window are pairwise independent, and each entity only participates in a single group at each time window. We use the ground truth communities at each time window as substitutes for these groups even though they overlap.

A separate network is then constructed with vertices representing each group at each time window, with $g_{i,k}$ being the i^{th} group in window k. For each entity that is present in at least one time window, there are also vertices which represent that entity's presence in every time window. We use $v_{j,k}$ to represent the vertex for the j^{th} entity's presence in time window k. Each vertex $v_{j,k}$ is connected to the same entity's vertex in the next time window, $v_{j,k+1}$, and to the group it participated in during the current time window, $g_{i,k}$.

A community structure is then equated to a vertex coloring on this network. At each time interval, the vertices with the same color are part of the same community. Three costs are defined on a coloring of this network to provide a quality measure. The i-cost is a fixed cost incurred if an entity changes communities (colors) between time windows. A g-cost is assessed to the coloring if a vertex is either connected to a group of a different color or is not connected to the group of the same color. Finally, a c-cost is assessed to a vertex for each community (color) it is a part of after its first. For our experiments, we assign an i-cost of 1 and g- and c-costs of 2.

Finding the minimum cost coloring was found to be NP-complete. As a result, fast heuristics and approximations are used in the algorithm to find a suitable network coloring.

4.2.1.4 LabelRankT

LabelRankT [Xie et al. 2013a] is a label propagation algorithm, with roots in the static community detection algorithm LabelRank [Xie and Szymanski 2013]. LabelRank starts by associating a vector of values with each vertex. If N vertices are in the network, the vectors are all of size N. Each vertex is also associated with its own, unique label. The i^{th} value in vertex v's vector represents the frequency with which v has seen the label associated with vertex i. In the beginning the vectors are set to have values all equal to zero, except for a single value of one, which is associated with the vertex's own label. The algorithm then proceeds iteratively, which each vertex receiving the label vector of all of its neighbors at each iteration. The label vector for a vertex at iteration j is calculated as the average of the label vectors of its neighbors at iteration $j-1$. The algorithm then proceeds until convergence, at which point vertices are assigned to communities associated with labels which have a value higher than a given threshold in the vertex's label vector. Label inflation and trimming procedures are used to increase the algorithm's efficiency.

To find communities in temporal networks, LabelRankT uses LabelRank to detect communities in each of the static time windows. However, for time windows other than the first, LabelRankT begins with label vectors detected in the previous time window, and uses the approach of LabelRank to simply update them. Because of this, the algorithm only needs to consider vertices whose neighborhoods change between time windows, and the resulting community structures generally do not experience extreme changes.

LabelRankT does not provide an explicit way of connecting communities in consecutive time windows to construct evolutions. We run the community matching approach using the Jaccard index on the communities detected by LabelRankT to construct evolutions when necessary. For our experiments we use the default parameters of LabelRankT described in the documentation.

4.2.2 Single Event Tests

We test each algorithm's ability to detect four of the six common evolutionary events mentioned in 4.1.2.2. We omit tests for the birth and death events, as the

detection of these events relies almost exclusively on the underlying static algorithm used to generate the community structure at each time window and we are mainly interested in examining the approaches to grouping the detected communities into evolutions. For each event, we generate networks consisting of two time windows with a single instance of the event embedded within them. For instance, in the test networks for the growth event, there is a single community in each network. The community in the first network consists of ten vertices. The community in the second network consists of anywhere from ten to 15 vertices, and is a superset of the original community. For testing the shrink event, the second community falls somewhere between five and ten vertices. The merge event test networks have two communities with ten members each in the first time window and a single community whose membership is the union of the first two communities in the second window. Test networks for the split event contain a single community of twenty members in the first window and two communities of random sizes between three and 20 in the second network. Thirty test networks were generated for each event.

For an algorithm to successfully detect an event, all community transitions must be detected. A successful run on a split test must result with the community in the first window transitioning into both communities in the second window. Many of the evolution detection algorithms that use a community matching approach, as strictly defined, only match a community in window i, C_i, with a single community in window $i + 1$, C_{i+1}, in order to simplify the results. For these algorithms, we consider results that would occur if multiple matchings were allowed. For example, when considering the community matching approach utilizing the LOST framework with the Jaccard index, we match communities C_i to all communities C_{i+1} that result in a Jaccard index value above a given threshold.

In the cases where the algorithm also detects the communities at each time window, we take community to be successfully detected if the Jaccard index between the true community membership and the detected membership is above 0.85. Results for the tests can be found in Table 4.1.

The CPM algorithm does extremely well on these tests, detecting nearly all instances of all events. The algorithm has a very strict definition of a community,

Table 4.1: Rate of successful recovery of single, isolated evolutionary events for each evolution detection algorithm. Values are the fraction of the thirty test networks that each algorithm completely recovered.

Algorithm	Grow	Shrink	Merge	Split
CPM	0.97	0.97	1	0.97
Jaccard	0.4	0.27	0	0
Min	1	1	1	1
CommDy	1	1	1	0.2
LabelRankT	0.2	0.1	0	0

and in the few cases where the algorithm misses an event, a true community's structure is sparse enough that it is detected as two separate communities.

Using the Jaccard index to detect community transitions in the LOST framework gives poor results. Consider the grow and shrink events. The community in the first time window has a size of 10. If the community in the second time window either gains or loses more than one vertex, the Jaccard index between the two communities drops below the threshold. Similar issues hold for the split and merge events. If the threshold is lowered to 0.5, using the Jaccard index would identify all grow, shrink, and merge events. It still has trouble with split events, however, as there is a chance that the community will split unevenly. If the community of size 20 breaks into a community of size 4 and one of size 16, the smaller community shares a Jaccard index of 0.2 with the original community.

Using the $MinNorm(\cdot, \cdot)$ measure avoids these issues. Even if a community were to massively shrink between time windows, all of the vertices in the smaller community would be shared between both. This measure is not suitable for general scenarios. Communities do not generally experience large size discrepancies between time windows, and detecting such situations as events would be undesirable. The measure is extremely effective in small, controlled test environments like this, however.

The CommDy algorithm has trouble finding a few extreme shrink events. Since every vertex needs to belong to some community at each time window, the vertices that change communities during the split event can either form a new community,

incurring the i-cost and a c-cost, or simply remain as part of the same community. The g-cost will be incurred either way as the vertices that leave will not be connected to any community node. As a result, the coloring generated by the CommDy algorithm keeps all of the vertices in the same community, forcing the Jaccard index between the true second community and the detected second community to dip below 0.85 for larger shrink events.

LabelRankT has trouble detecting all events due to the small size of the networks in relation to the communities and the possibility of large changes from one time window to the next. The small network size results in many fringe vertices with a large fraction of their edges connected to the community, even though they are not members. As a result, there are no competing labels to whichever label ends up representing the community, and the fringe vertices are considered part of the community. In other words, there are simply no other communities for the algorithm to assign these vertices to. Again, if the community matching threshold were reduced to 0.5 instead of 0.85, the algorithm detects 97% of grow events and 83% of shrink events.

LabelRankT has slightly different difficulty detecting split and merge events. In the split event, all of the vertices in each of the two communities in the second time window are seeded with the same label. There are still no other labels that might become injected into either community, so all the vertices keep virtually the same labelling. As a result, the split goes undetected and the community is detected as simply continuing or surviving.

4.3 Full Evolutions

Detecting single, isolated evolutionary events is a much easier task than detecting complete, multi-step evolutions. In some sense, detecting full length evolutions can be seen as iteratively detecting single step evolutionary events between consecutive time windows and piecing the steps together. There are extra complications with this, however, even if the ground truth community structures are known. If the criteria for a community transition to be a valid evolutionary event is too relaxed, there are a potentially extremely large number of chains of communities that are

considered valid evolutions. Even if the true evolutions were detected, the large number of extra evolutions would hide and overwhelm any useful information that could be gathered. On the other hand, restrictive criteria for valid evolutions may not detect all true evolutions.

To evaluate the performance of each evolution detection algorithm in detecting full length evolutions, we generated ten synthetic temporal networks with ten time windows each and randomly embedded evolutions using our model. The complete set of parameters used are given in Table 4.2. The generated networks simulate many of the characteristic of real world networks. Figure 4.3 shows a comparison of the characteristics between the generated networks, the networks from Chapter 3, and two new networks. The IMDB network consists of actors connected if they starred in the same movie together with time windows of one year. The NRON network is the now well-known network of Enron emails collected by the Federal Energy Regulatory Commission during its investigation of the company, with a time window length of one week.

In order to evaluate the evolution detection algorithms, we construct a simple similarity measure. Consider two evolutions, $\mathcal{E} = \{E_i, E_{i+1}, \ldots, E_j\}$ and $\mathcal{F} = \{F_k, F_{k+1}, \ldots, F_l\}$. For each time window w between $min(i, j)$ and $max(k, l)$, we calculate the Jaccard index between E_w and F_w. If any such community doesn't exist, it is considered an empty community. We then take the evolution similarity to be the average Jaccard index across all time windows. If this average is above a certain threshold λ, we consider the two evolutions a match.

Many of the algorithms have difficulty detecting full length evolutions. CommDy will not run on the generated networks, citing the need for too many colors to evaluate the temporal network. The CPM method had difficulty finding any of the ground truth evolutions even with λ set to 0.5. Due to a relatively large amount of overlap between some communities, clique percolation tend to discover communities that are actually unions of multiple true communities. As a result, the Jaccard similarity measure between detected and ground truth communities becomes very small. LabelRankT was able to detect a few evolutions with small, smooth evolutionary events. Many events were missed due to issues similar to those discussed in

Table 4.2: Parameters used to generate syntehtic temporal networks for testing each algorithm's ability to detect multi-step evolutions.

Param	Description	Value
v	Number of vertices in the network for the first time window	100
v_{exp} v_{min} v_{max}	Exponent, minimum values and maximum values for the power law sampled for vertex lag values	1.75 0.2 1.2
$v_{birth_{min}}$ $v_{birth_{max}}$	Minimum and maximum growth percentage for number of vertices between consecutive time windows	0.1 0.3
c_{exp} c_{min} c_{max}	Exponent, minimum values and maximum values for the power law sampled for community size values	2.75 3 15
c_{birth} c_{death}	Percentage of active communities added and removed between consecutive time windows.	0.05 0.05
$c_{growth_{prob}}$	Probability that a community will grow in size from one time window to the next. If it doesn't grow, it shrinks.	0.55
$c_{growth_{max}}$	Maximum increase or decrease in size for a community between consecutive time windows	0.25
$c_{merge_{prob}}$	The probability of a community C being chosen for a merge event is $\frac{1}{\lvert C\rvert^{c_{merge_{prob}}}}$	1
$c_{split_{prob}}$	If a community C is not picked for a merge event, it has a $\lvert C\rvert * c_{split_{prob}}$ probability of being split.	0.01
$c_{split_{size}}$	Minimum size a community must be in order to be considered for a split event	6
mp	Mixing parameter. Percentage of all edges found between vertices sharing at least one community membership	0.85
c_{avg}	Average community membership for a vertex.	1.2
t	Time windows to generate	10

the single event test.

The LOST method was able to reconstruct the most evolutions. With λ set to 0.75, 63% of the true evolutions were recovered while 68% of the detected evolutions matched a true evolution. Dropping λ to 0.5 raises these values to 73% and 72% respectively. Evolutions incorporating small communities were the most troublesome for the LOST framework. Events on small communities can cause low Jaccard values between consecutive communities in an evolution, which causes the framework to miss them.

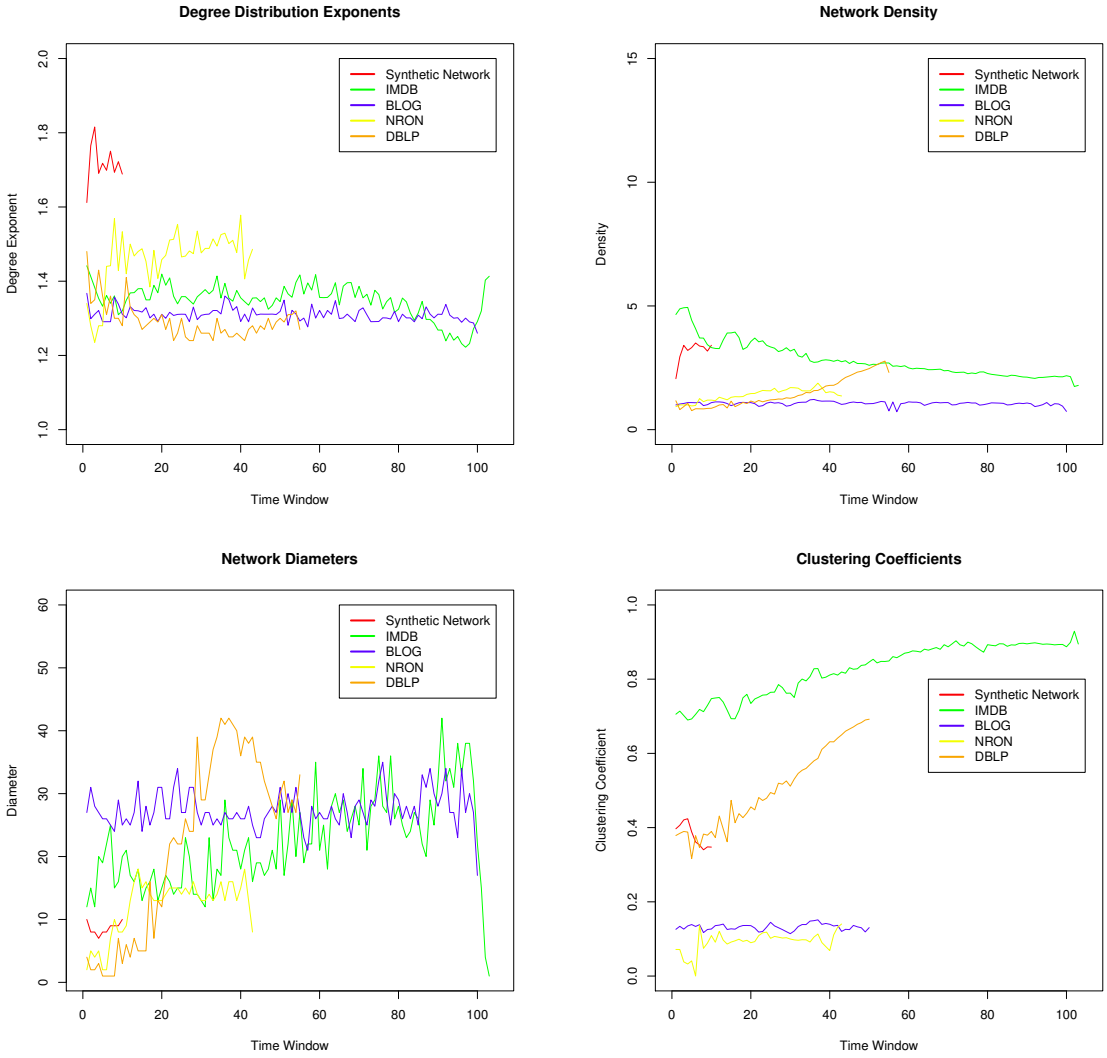


Figure 4.3: Comparison of network characteristics found in real world and synthetic networks. Values for synthetic networks represent an average of results from 10 networks.

4.4 Anonymous

In most networks, every separate entity has its own, known label. However, this is not always the case. Anonymous or unknown vertices can arise either from data gathering limits or errors when constructing the network, or they be part of the system itself. A website that allows users to remain anonymous if they want to, for instance, will have community structures with unknown vertex identities. To test the robustness of the LOST framework, we consider the case where the identities of

some vertices in the network are unknown.

To generate community structures with anonymous vertices, we start from the randomly generated community structures from the previous section. We then take the union of all vertex identifiers from all time windows, and choose a certain percentage of those to remove from the community structure. All instances of the chosen vertices are relabelled as unknown.

The unknown vertices raise issues when evaluating a Jaccard similarity between two communities. There are two different ways we handle the unknown vertices. The first way, which we will call the naive approach, is to ignore them, where two unknown vertices are assumed to be different entities. The second way is to assume that two unknowns are the same identity with some probability p. We consider two different ways of setting p. One way, which we call the naive probability, assumes that the total number of unknown vertices is known. Say there are 10 unknown entities in total, and we are comparing community A, which has 3 unknown members, with community B, which has 4. If we randomly select unknown vertices to include in community B and consider the unknowns of community A to be fixed, we can formulate the expected number of matches as the expected number of successful Bernoulli trials out of four, with the probability of success being 0.3. This results in an expected number of 1.2 matches. The other way is to manually set a static probability p. Using the same A and B communities, the static probability approach considers the unknown vertices of community A one at a time. For each unknown in community B, the vertex from A is given a p probability of matching. If the matching is successful, both unknown vertices are immediately removed from consideration and the calculation continues with the next unknown vertex in A. This method results in a much higher number of matched unknown vertices than using the naive probability.

To test each approach, we construct networks with 10%, 20%, ..., 90% of vertices anonymized. For each value, we construct ten temporal networks and average accuracy results calculated in the same manner as the previous section. Two evolutions are considered to be a match if more than 75% of the communities (paired by time window) have a Jaccard similarity above 0.75. The results can be seen in

Figure 4.4.

As might be expected, the accuracy of all approaches diminishes linearly with respect to the number of vertices anonymized. The less information known, the harder it should be to recover the true evolutions. However, adding the mechanisms to match possible similar anonymized vertices seem to decrease the accuracy of the LOST method. The benefit of possibly detecting true matches of anonymized vertices is outweighed by the danger of matching two vertices that are truly different entities. This is seen more dramatically when the threshold for considering an evolution matched is dropped from 0.75 to 0.25. The random matching of anonymous vertices now holds more influence, as increasing a Jaccard similarity above 0.25 is easier than increasing it above 0.75. Communities start to be matched based solely on anonymized vertices, which results in random matches, random evolutions, and a lower accuracy. Evolutions that are accurately recovered typically share enough known vertices to break the threshold without having to match anonymized vertices. Using a higher threshold diminishes the detrimental effects of matching anonymized vertices, but does not introduce any benefit.

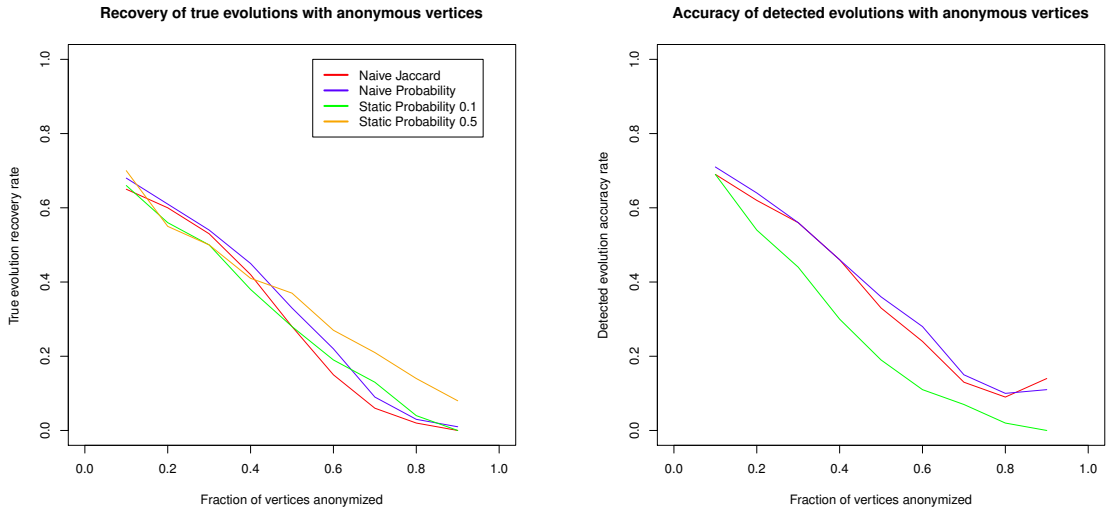


Figure 4.4: Accuracy of using different variations of the Jaccard similarity in the LOST evolution detection algorithm when a fraction of the vertex identities have been anonymized and the evolution detecting similarity threshold is 0.75. Incorporating mechanisms to try to match possibly similar anonymized vertices does not improve detection, and may actually impede it.

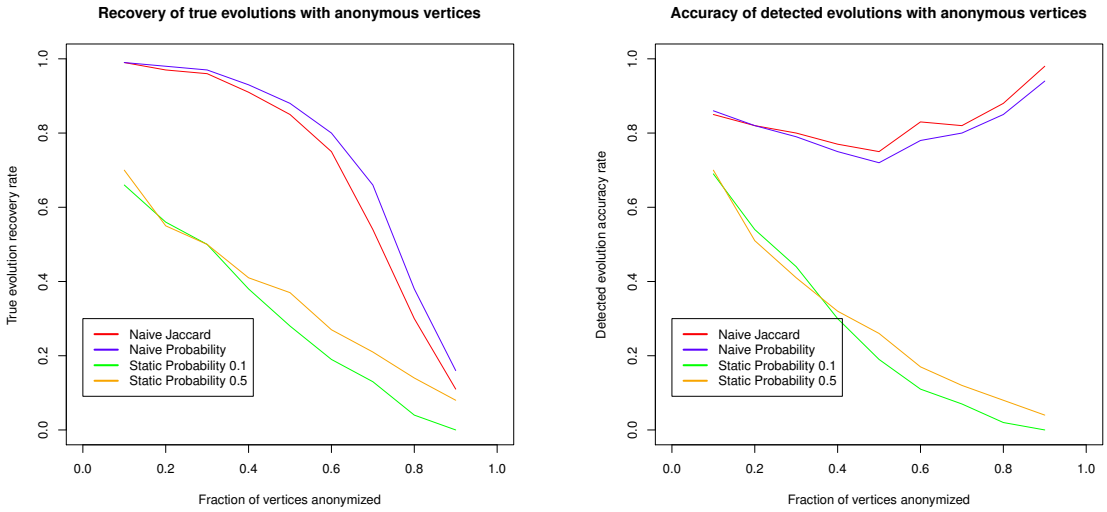


Figure 4.5: Accuracy of using different variations of the Jaccard similarity in the LOST evolution detection algorithm when a fraction of the vertex identities have been anonymized and the evolution detecting similarity threshold is 0.25. Higher probabilities of anonymous entities matching results in random matching of communities, leading to randomly constructed evolutions. It is better to depend on what little information is known when detecting evolutions in anonymized frameworks.

CHAPTER 5
Conclusion

In this work, we have proposed a framework for detecting community evolutions in temporal social networks and a model for generating synthetic networks with characteristics similar to those of real world networks. Detecting communities and evolutions in temporal social networks if from from being a solved problem, however. Algorithms can consistently be improved to run faster or be more accurate. Within our evolution detection framework, we use the Jaccard index to compare communities. We have demonstrated some of the drawbacks of the Jaccard index in Chapter 4. More sophisticated comparison methods may increase the accuracy of our framework. Also, in the second step of our framework, we only merge two evolutions if one starts in the time window directly following the window where the other ends. Merging evolutions that are removed by more than one time window might detect more intermittently active evolutions. Finally, we use a simple, linear regression framework to predict evolution length based off of simple features. More sophisticated features or a more sophisticated learning framework might improve prediction accuracy.

There are a number of possible extensions to our network generation model as well. Many mechanisms of our network generation model can be modified, and new mechanisms added, in order to produce networks that more accurately reflect real world systems. For instance, we could consider community merges of more than two communities, vertices that are only intermittently active in the network, and probabilities of merges, splits, and community memberships that are based on network structure. There is also room for optimization of the speed of the algorithm.

When considering networks with anonymous users, we show that simple mechanisms for detecting possible matches between anonymous users only provides marginal improvement, if any. More sophisticated approaches may perform better than the simpler approaches studied in this report. For example, an expectation-maximization algorithm may help with the issue of a single anonymous vertex in one time win-

dow being potentially matched with a large number of anonymous vertices in the following window. A possible matching for anonymous users would result in a community matching across consecutive time windows that can then be used to refine the matching for anonymous users.

Finally, we only consider one framework for temporal networks. Extensions may be developed for our approaches in order to handle frameworks with directed networks or streams of interactions.

CHAPTER 6
BIBLIOGRAPHY

Balázs Adamcsek, Gergely Palla, Illés J Farkas, Imre Derényi, and Tamás Vicsek. 2006. CFinder: locating cliques and overlapping modules in biological networks. *Bioinformatics* 22, 8 (January 2006), 1021–1023.

Réka Albert and Albert-László Barabási. 2000. Topology of evolving networks: local events and universality. *Phys. Rev. Lett.* 85, 24 (July 2000), 5234.

Enrique Amigó, Julio Gonzalo, Javier Artiles, and Felisa Verdejo. 2009. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Inform. Retrieval* 12, 4 (August 2009), 461–486.

Albert-László Barabási. 2005. The origin of bursts and heavy tails in human dynamics. *Nature* 435, 7039 (June 2005), 207–211.

Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *Science* 286, 5439 (October 1999), 509–512.

Zhiqiang Bi, Christos Faloutsos, and Flip Korn. 2001. The DGX distribution for mining massive, skewed data. In *Proceedings of the 7th. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, San Francisco, CA, USA, 17–26.

Piotr Bródka, Stanisław Saganowski, and Przemysław Kazienko. 2013. GED: the method for group evolution discovery in social networks. *Soc. Netw. Anal. Min.* 3, 1 (March 2013), 1–14.

Sylvain Brohee and Jacques Van Helden. 2006. Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics* 7, 1 (January 2006), 488.

Rajmonda Sulo Caceres and Tanya Berger-Wolf. 2013. Temporal Scale of Dynamic Networks. In *Temporal Networks*. Springer-Verlag Berlin Heidelberg, New York, NY, USA, 65–94.

Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. 2006. Evolutionary clustering. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Philadelphia, PA, USA, 554–560.

Aaron Clauset, Mark EJ Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Phys. Rev. E* 70, 6 (December 2004), 066111.

Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. 2009. Power-law distributions in empirical data. *SIAM Rev.* 51, 4 (October 2009), 661–703.

Linda M Collins and Clyde W Dent. 1988. Omega: A general formulation of the rand index of cluster recovery suitable for non-disjoint solutions. *Multivar. Behav. Res.* 23, 2 (April 1988), 231–242.

Imre Derényi, Gergely Palla, and Tamás Vicsek. 2005. Clique percolation in random networks. *Phys. Rev. Lett.* 94, 16 (April 2005), 160202.

Christian Dewes, Arne Wichmann, and Anja Feldmann. 2003. An analysis of Internet chat systems. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*. ACM, Miami Beach, FL, USA, 51–64.

Paul Erdős and A Rényi. 1960. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci.* 5 (1960), 17–61.

Giorgio Fagiolo. 2007. Clustering in complex directed networks. *Phys. Rev. E* 76, 2 (August 2007), 026107.

Gary William Flake, Steve Lawrence, C Lee Giles, and Frans M Coetzee. 2002. Self-organization and identification of web communities. *Computer* 35, 3 (March 2002), 66–70.

Santo Fortunato. 2010. Community detection in graphs. *Phys. Rep.* 486, 3 (February 2010), 75–174.

Santo Fortunato and Marc Barthélemy. 2007. Resolution limit in community detection. *P. Natl. Acad. Sci. USA* 104, 1 (September 2007), 36–41.

Michelle Girvan and Mark EJ Newman. 2002. Community structure in social and biological networks. *P. Natl. Acad. Sci. USA* 99, 12 (June 2002), 7821–7826.

Mark Goldberg, Malik Magdon-Ismail, Srinivas Nambirajan, and James Thompson. 2011. Tracking and predicting evolution of social communities. In *2011 IEEE 3rd. International Conference on Social Computing (SocialCom)*. IEEE, Boston, MA, USA, 780–783.

Mark Goldberg, Malik Magdon-Ismail, and James Thompson. 2012. Identifying long lived social communities using structural properties. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, Istanbul, Turkey, 647–653.

M. K. Goldberg, S. Kelley, M. Magdon-Ismail, K. Mertsalov, and W. Wallace. 2010. Finding Overlapping Communities in Social Networks. In *Proceedings of the 2nd Conference on Social Computation (SocialCom)*. Minneapolis, MN, USA, 104–113.

Benjamin H Good, Yves-Alexandre de Montjoye, and Aaron Clauset. 2010. Performance of modularity maximization in practical contexts. *Phys. Rev. E* 81, 4 (April 2010), 046106.

Derek Greene, Donal Doyle, and Padraig Cunningham. 2010. Tracking the evolution of communities in dynamic social networks. In *Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, Odense, Denmark, 176–183.

Steve Gregory. 2008. A fast algorithm to find overlapping communities in networks. In *Machine Learning and Knowledge Discovery in Databases*. Springer-Verlag Berlin Heidelberg, New York, NY, USA, 408–423.

Steve Gregory. 2010. Finding overlapping communities in networks by label propagation. *New J. Phys.* 12, 10 (October 2010), 103018.

Roger Guimera, Marta Sales-Pardo, and Luís A Nunes Amaral. 2004. Modularity from fluctuations in random graphs and complex networks. *Phys. Rev. E* 70, 2 (August 2004), 025101.

Paul W Holland and Samuel Leinhardt. 1972. Holland and Leinhardt reply: some evidence on the transitivity of positive interpersonal sentiment. *Amer. J. Sociology* 77, 6 (May 1972), 1205–1209.

Petter Holme and Jari Saramäki. 2013. Temporal networks as a modeling framework. In *Temporal Networks*. Springer-Verlag Berlin Heidelberg, New York, NY, USA, 1–14.

Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *J. Classif.* 2, 1 (January 1985), 193–218.

Pan Hui, Jon Crowcroft, and Eiko Yoneki. 2011. Bubble rap: Social-based forwarding in delay-tolerant networks. *IEEE T. Mob. Comput.* 10, 11 (November 2011), 1576–1589.

U Kang, Charalampos Tsourakakis, Ana Paula Appel, Christos Faloutsos, and Jure Leskovec. 2008. *HADI: Fast diameter estimation and mining in massive graphs with Hadoop*. Carnegie Mellon University, School of Computer Science, Machine Learning Department, Pittsburgh, PA, USA.

George Karypis, Eui-Hong Han, and Vipin Kumar. 1999. Chameleon: Hierarchical clustering using dynamic modeling. *Computer* 32, 8 (August 1999), 68–75.

Hyunju Kim, Zoltán Toroczkai, Péter L Erdős, István Miklós, and László A Székely. 2009. Degree-based graph construction. *J.Phys. A-Math. Theor.* 42, 39 (October 2009), 392001.

Andrea Lancichinetti and Santo Fortunato. 2009. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Phys. Rev. E* 80, 1 (July 2009), 016118.

Andrea Lancichinetti and Santo Fortunato. 2011. Limits of modularity maximization in community detection. *Phys. Rev. E* 84, 6 (December 2011), 066122.

Andrea Lancichinetti, Santo Fortunato, and Jnos Kertsz. 2009. Detecting the overlapping and hierarchical community structure in complex networks. *New J. Phys.* 11, 3 (March 2009), 033015.

Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. 2008. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E* 78, 4 (October 2008), 046110.

Andrea Lancichinetti, Filippo Radicchi, José J Ramasco, and Santo Fortunato. 2011. Finding statistically significant communities in networks. *PloS One* 6, 4 (April 2011), e18961.

Anna Lázár, Dániel Ábel, and Tamás Vicsek. 2010. Modularity measure of networks with overlapping communities. *E.P.L.* 90, 1 (April 2010), 18001.

Jurij Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. 2005. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *Proceedings of the 9th. European Conference on Priciples and Practice of Knowledge Discovery in Databases (PKDD)*. Springer, Porto, Portugal, 133–145.

Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. 2010. Kronecker graphs: An approach to modeling networks. *J. Mach. Learn. Res.* 11 (January 2010), 985–1042.

Yu-Ru Lin, Yun Chi, Shenghuo Zhu, Hari Sundaram, and Belle L Tseng. 2008. Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In *Proceedings of the 17th International Conference on World Wide Web*. ACM, Beijing, China, 685–694.

Kathy Macropol, Tolga Can, and Ambuj K Singh. 2009. RRW: repeated random walks on genome-scale protein networks for local cluster discovery. *BMC Bioinformatics* 10, 1 (January 2009), 283.

Malik Magdon-Ismail and Jonathan Purnell. 2011. SSDE-cluster: fast overlapping clustering of networks using sampled spectral distance embedding and GMMs. In *2011 IEEE Third Inernational Conference on Social Computing (SocialCom)*. IEEE, Boston, MA, USA, 756–759.

A. F. McDaid, D. Greene, and N. Hurley. 2011. Normalized Mutual Information to evaluate overlapping community finding algorithms. *ArXiv e-prints* (October 2011).

Mary McGlohon, Leman Akoglu, and Christos Faloutsos. 2011. Statistical properties of social networks. In *Social Network Data Analytics*. Springer US, New York, NY, USA, 17–42.

Marina Meilă. 2007. Comparing clusterings - an information based distance. *J. Multivariate Anal.* 98, 5 (2007), 873–895.

Milena Mihail and Christos Papadimitriou. 2002. On the eigenvalue power law. In *Lect. Notes Comput. Sc.* Springer, Cambridge, MA, 254–262.

Stanley Milgram. 1967. The small world problem. *Psychol. Today* 2, 1 (May 1967), 60–67.

Byungjoon Min and K-I Goh. 2013. Burstiness: Measures, Models, and Dynamic Consequences. In *Temporal Networks*. Springer-Verlag Berlin Heidelberg, New York, NY, USA, 41–64.

Michael Molloy and Bruce Reed. 1995. A critical point for random graphs with a given degree sequence. *Random Struct. Algor.* 6, 2-3 (March-May 1995), 161–180.

Lev Muchnik, Sen Pei, Lucas C Parra, Saulo DS Reis, José S Andrade Jr, Shlomo Havlin, and Hernán A Makse. 2013. Origins of power-law degree distribution in the heterogeneity of human activity in social networks. *Scientific Reports* 3 (May 2013).

Gabriel Murray, Giuseppe Carenini, and Raymond Ng. 2012. Using the omega index for evaluating abstractive community detection. In *Proceedings of Workshop on Evaluation Metrics and System Comparison for Automatic Summarization*. Association for Computational Linguistics, Montreal, Quebec, Canada, 10–18.

SR Nanda, Biswajit Mahanty, and MK Tiwari. 2010. Clustering Indian stock market data for portfolio management. *Expert Syst. Appl.* 37, 12 (December 2010), 8793–8798.

Mark EJ Newman. 2001. The structure of scientific collaboration networks. *P. Natl. Acad. Sci. USA* 98, 2 (January 2001), 404–409.

Mark EJ Newman. 2003. The structure and function of complex networks. *SIAM Rev.* 45, 2 (April 2003), 167–256.

Mark EJ Newman. 2006. Modularity and community structure in networks. *P. Natl. Acad. Sci. USA* 103, 23 (June 2006), 8577–8582.

Mark EJ Newman. 2009. Random graphs with clustering. *Phys. Rev. Lett.* 103, 5 (July 2009), 058701.

Mark EJ Newman and Michelle Girvan. 2004. Finding and evaluating community structure in networks. *Phys. Rev. E* 69, 2 (February 2004), 026113.

Mark EJ Newman and Juyong Park. 2003. Why social networks are different from other types of networks. *Phys. Rev. E* 68, 3 (September 2003), 036122.

Vincenzo Nicosia, Giuseppe Mangioni, Vincenza Carchiolo, and Michele Malgeri. 2009. Extending the definition of modularity to directed graphs with overlapping communities. *J. Stat. Mech-Theory Exp.* 2009, 03 (March 2009), P03024.

Vincenzo Nicosia, John Tang, Cecilia Mascolo, Mirco Musolesi, Giovanni Russo, and Vito Latora. 2013. Graph metrics for temporal networks. In *Temporal Networks*. Springer-Verlag Berlin Heidelberg, New York, NY, USA, 15–40.

JG Oliveira and A Vazquez. 2009. Impact of interactions on human dynamics. *Phys. A* 388, 2 (January 2009), 187–192.

Tore Opsahl and Pietro Panzarasa. 2009. Clustering in weighted networks. *Soc. Networks* 31, 2 (May 2009), 155–163.

Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435, 7043 (June 2005), 814–818.

Christopher R Palmer, Phillip B Gibbons, and Christos Faloutsos. 2002. ANF: A fast and scalable tool for data mining in massive graphs. In *Proceedings of the 8th. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Edmonton, Alberta, Canada, 81–90.

Vern Paxson and Sally Floyd. 1995. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Trans. on Net.* 3, 3 (June 1995), 226–244.

David M Pennock, Gary W Flake, Steve Lawrence, Eric J Glover, and C Lee Giles. 2002. Winners don't take all: Characterizing the competition for links on the web. *P. Natl. Acad. Sci. USA* 99, 8 (April 2002), 5207–5211.

William M Rand. 1971. Objective criteria for the evaluation of clustering methods. *J. Amer. Statist. Assoc.* 66, 336 (1971), 846–850.

Jörg Reichardt and Stefan Bornholdt. 2006. When are networks truly modular? *Phys. D* 224, 1 (December 2006), 20–26.

Jari Saramäki, Mikko Kivelä, Jukka-Pekka Onnela, Kimmo Kaski, and Janos Kertesz. 2007. Generalizations of the clustering coefficient to weighted complex networks. *Phys. Rev. E* 75, 2 (February 2007), 027105.

C Seshadhri, Ali Pinar, and Tamara G Kolda. 2013. An in-depth analysis of stochastic Kronecker graphs. *J. ACM* 60, 2 (April 2013), 13.

Huawei Shen, Xueqi Cheng, Kai Cai, and Mao-Bin Hu. 2009. Detect overlapping and hierarchical community structure in networks. *Phys. A* 388, 8 (April 2009), 1706–1712.

Rajmonda Sulo, Tanya Berger-Wolf, and Robert Grossman. 2010. Meaningful selection of temporal resolution for dynamic networks. In *Proceedings of the 8th. Workshop on Mining and Learning with Graphs*. ACM, Washington, D.C., USA, 127–136.

Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S Yu. 2007. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, San Jose, CA, USA, 687–696.

Mansoureh Takaffoli, Farzad Sangi, Justin Fagnan, and Osmar R Zäıane. 2011. Community evolution mining in dynamic social networks. *Procedia Soc. Behav. Sci.* 22 (July 2011), 49–58.

J. Tang, S. Scellato, M. Musolesi, C. Mascolo, and V. Latora. 2010. Small-world behavior in time-varying graphs. *ArXiv e-prints* 81, 5 (May 2010), 055101.

Chayant Tantipathananandh and Tanya Berger-Wolf. 2009. Constant-factor approximation algorithms for identifying dynamic communities. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Paris, France, 827–836.

Chayant Tantipathananandh, Tanya Berger-Wolf, and David Kempe. 2007. A framework for community identification in dynamic social networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, San Jose, CA, USA, 717–726.

Charalampos E Tsourakakis. 2008. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *8th. IEEE International Conference on Data Mining*. IEEE, Pisa, Italy, 608–617.

Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Stat. Comput.* 17, 4 (December 2007), 395–416.

Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of small-world networks. *Nature* 393, 6684 (June 1998), 440–442.

Scott White and Padhraic Smyth. 2005. A spectral clustering approach To finding communities in graphs. In *International Conference on Data Mining*, Vol. 5. SIAM, Newport Beach, CA, 76–84.

Jierui Xie, Mingming Chen, and Boleslaw K Szymanski. 2013a. LabelrankT: Incremental community detection in dynamic networks via label propagation. In *Proceedings of the Workshop on Dynamic Networks Management and Mining*. ACM, New York, NY, USA, 25–32.

Jierui Xie, Stephen Kelley, and Boleslaw K Szymanski. 2013b. Overlapping community detection in networks: The state-of-the-art and comparative study. *Comput. Surveys* 45, 4 (August 2013), 43.

Jierui Xie and Boleslaw K Szymanski. 2013. Labelrank: A stabilized label propagation algorithm for community detection in networks. In *2nd IEEE Network Science Workshop*. IEEE, West Point, NY, USA, 138–143.

Jierui Xie, Boleslaw K Szymanski, and Xiaoming Liu. 2011. Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In *11th International Conference on Data Mining Workshops*. IEEE, Vancouver, Canada, 344–349.

APPENDIX A

To define the conditional entropy between two communities, $H(A|B)$, the vertices are divided into four categories:

$a = |\bar{A} \cap \bar{B}| \quad b = |\bar{A} \cap B| \quad c = |A \cap \bar{B}| \quad d = |A \cap B|$

The conditional entropy between two communities is then defined as:

$$H(A|B) = \begin{cases} h(a,n) + h(b,n) + & \text{if } h(a,n) + h(d,n) \geq \\ h(c,n) + h(d,n) - & h(b,n) + h(c,n) \\ h(b+d,n) - h(a+c,n) & \\ & \\ h(c+d,n) + h(a+b,n) & \text{otherwise} \end{cases}$$

where $h(x,y) = x \times log_2\left(\frac{x}{y}\right)$ and n is the number of vertices in the network.