The Impact of Different Edge Weightings on Community Detection in Social Networks

Sarah Teichman

April 4, 2018

Submitted to the
Department of Mathematics and Statistics
of Amherst College
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Arts with Honors

Faculty Advisor: Professor Amy Wagaman

Acknowledgements

I would like to thank several people for their impact on my statistics education and this thesis. First, I would like to thank my thesis advisor Professor Amy Wagaman. Her patience, knowledge of the subject, and encouragement have been invaluable in this process. I would also like to thank Professor Nick Horton for all that he has taught me over the last several years, as a professor, advisor, and mentor. His support and encouragement have propelled me forward in my study of statistics. I have had the privilege of learning from and working with many faculty and students in the Mathematics and Statistics department at Amherst, and I appreciate all of the interactions that I've had with them that have pushed me to think, learn, and communicate in new ways. I would like to thank team Scatterplot/Box and Whiskers, Tim Lee and Jonathan Che, for all of the laughter that we have shared both in and out of statistics classes over the past four years. I would also like to thank my friends, especially the five women who I am so lucky to live with this year, who have helped me celebrate accomplishments and overcome challenges during my time at Amherst. Finally, I would like to thank my parents for their support of my education, and my sister Emily for the inspiration that her hard work on two simultaneous theses has given me for this thesis.

Table of Contents

List of Tables

List of Figures

Abstract

Community detection is a subfield within social networks that attempts to uncover the natural divisions in networks, in order to better understand network structure and analyze network data. This thesis begins with an overview of social networks, and continues with an explanation of the goals of community detection and a discussion of several common methods. We explain a recent method that works with overlapping and hierarchical networks, and discuss potential effects of various edge weightings to represent different types of relationships between nodes. Within the simulation, we test this new method on networks created using different graph models and edges weighted in different ways. We study the distributions of edge weights and results of community detection to support claims made in two papers about this new method and different edge weighting schemes. Finally, we apply these tools to Twitter data. The simulation and work with Twitter data show the importance of having methods that can work with networks with complex structures, and suggest that different edge weightings produce different community detection results. However, more work could be done to create simulations that more accurately represent real networks and to collect larger networks of Twitter data.

Introduction

Social networks model the interactions between various individuals within a society, such as members of a club, academics in a certain field, or teams within a conference. The study of social networks has existed since the 1930's, but has undergone many changes in recent years. Initially the field was populated by social scientists, but as it grew it began to attract researchers in business and public health, as well as other fields. There are several major challenges in the study of social networks, including difficulties with sampling, gathering data from real networks, and biased responses. However, the Internet has created many spaces in which social interactions arise online, providing researchers with larger social networks with lower barriers for data collection (Kolaczyk, 2009).

0.1 Basic Terminology

A social network is any kind of group in society that can be represented as a graph. A graph, G = (V,E) is a structure with a set of vertices (or nodes), V, and a set of edges, E. A graph has |V| = N_v nodes and |E| = N_e edges. Nodes represent individual observations within a population (in the case of a social network, these would be the social actors). Nodes can be assigned attributes that provide additional information about individuals. Edges are connections between nodes, and represent relationships between the two social actors. Edges are represented as pairs {u,v} of vertices, u, v $\in$ V. Edges can be directed if they represent one-sided relationships, like a mentor-mentee connection, or undirected if they represent mutual relationships, like friendships. They can also be weighted by some quantity that describes the relationship.

Networks can be represented in several different ways. One representation is an adjacency list, which is a N_v size array, in which the i^{th} entry contains a list of all of the i^{th} node's neighbors (nodes connected by an edge to the i^{th} node). Another representation is an edge list, which contains two columns and N_e rows, where each row represents an edge between two nodes. A third representation is an adjacency matrix. This is a N_v by N_v sized matrix, A, in which the A[i,j] entry is 0 if there is not an edge between the two nodes and 1 if there is (Kolaczyk, 2014). That is,

$$A[i,j] = \begin{cases} 1 & (i,j) \in E \\ 0 & otherwise. \end{cases}$$

One way that networks can be characterized is in terms of summary statistics for the nodes and edges. For example, for an undirected graph, a node's degree, k, is given by the number of edges that it has with other nodes. The total number of edges in an undirected graph is given by the sum of the degree of each node divided by two, to avoid double counting each edge: $N_e = \frac{1}{2}\sum_{i=1}^{N_v} k_i$. The average degree of a graph, $\bar{k}$ can be calculated: $\bar{k} = \frac{1}{N_v}\sum_{i=1}^{N_v} k_i = \frac{2N_e}{N_v} = \sum_{k=0}^{\infty} kp_k$, where $p_k = \frac{N_k}{N_v}$.

Networks can also be characterized by distances between nodes. Since networks often lack a measure of physical distance, this is measured by the path length between two nodes. A path is a list of edges that connect one node to another. The length of a path refers to the number of edges (or the number of nodes minus one along the path). The distance between nodes is the shortest path between them. There can be multiple shortest paths if they have the same length. Shortest paths can be found using several algorithms, including breadth first search. The diameter of a graph is defined as the maximum shortest path between any pair of nodes.

Another way to characterize individual nodes or a graph overall is through connectedness. In an undirected network, two nodes are connected if a path exists between them. A network is connected if every node is connected to every other node (Barabasi, 2016). If a network is unconnected, the diameter is equal to ∞, and many summary statistics cannot be computed. In these cases, the largest connected component of the graph is often studied instead of the entire graph.

A network can also be described through its cohesion. This measures the degree to which groups of nodes are clustered together. Cohesion metrics often try to answer the following question: if nodes A and B both have edges to node C, how likely is it that an edge exists between nodes A and B? There are different ways to define and measure cohesion, but a common metric is transitivity, which compares the number of triangles to connected triples in a graph. A triangle is defined as three nodes with three edges connecting them to each other, and a connected triple is defined as three nodes connected by two edges. The difference can be seen in Figure 1.

The transitivity of the graph ($cl_T(G)$) is calculated using the ratio of the number of triangles (τ_Δ) to the number of connected triples (τ_3): $cl_T(G) = \frac{3\tau_\Delta(G)}{\tau_3(G)}$. This version of transitivity, applied to an entire graph, is often referred to as global transitivity.

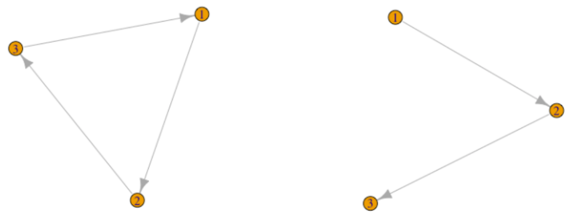

Figure 1: Triangle Vs. Connected Triple

Transitivity can also be defined locally, for a specific vertex, by comparing the number of triangles to the number of connected triples that the vertex is part of, assuming that it is part of at least one connected triple. The transitivity of vertex v can be calculated as: $cl(v) = \frac{\tau_{\Delta}(c)}{\tau_3(c)}$. Transitivity is often referred to as a clustering coefficient. There are also ways to measure cohesion specific to directed graphs (Kolaczyk, 2014).

Social networks tend to have higher transitivity than would be expected in random graphs. This can be understood by considering the types of relationships studied in social networks. For example, undirected edges between nodes often represent friendships between people. If two people have a friend in common, it is more likely that they will be friends than two people without a friend in common. Similarly, in citation networks, two academics are far more likely to collaborate with each other if they have a mutual collaborator than two academics without that relationship (Newman, 2010).

0.2 Network Models

One simple class of models for networks are random graph models. These models specify a collection of graphs and a uniform probability over them. In some cases they are as simple as giving equal probability to all graphs with a specified number of nodes and edges (Kolaczyk, 2014). This means that when we are using this model in a simulation, we randomly select one graph from the collection of graphs with the same specified features. However, real networks often have characteristics that cannot be captured simply by fixing the total number of nodes and edges. Many real networks have degree distributions that follow a power law, in which there are many nodes with low degrees and few nodes with much larger degrees. Edges are not distributed equally in networks but often occur in high concentrations within groups of nodes. Because of this, there are several more complicated classes of models that try to imitate properties of real networks. (Fortunato & Hric, 2016). We will discuss two simple models, one that only depends on the number of nodes and edges, and another that tries to capture two properties of social networks.

0.2.1 Erdős-Rényi Model

The random graph model suggested by Erdős and Rényi gives an equal probability to all graphs that have a specified N_v and N_e. This model suggests that for each N_v and N_e, there is a collection $\mathcal{G}_{N_v,N_e}$ of all graphs with those size parameters, and each graph $G \in \mathcal{G}_{N_v,N_e}$ has a probability of occurring equal to the reciprocal of the number of ways the edges could be assigned to the vertices. Therefore, $\mathbb{P}(G) = \binom{N}{N_e}^{-1}$ where $N = \binom{N_v}{2}$. This model can be generalized by specifying N_v and some other characteristic of the graph, such as a degree sequence, and then assigning equal probabilities to all graphs with that size and characteristic. The random graph model with a specified N_v and N_e usually has short path lengths between any two vertices, and low cohesion (Kolaczyk, 2014). An example of an Erdős-Rényi graph is shown in Figure 2.

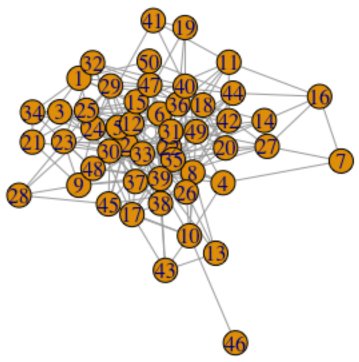

Figure 2: Erdős-Rényi Graph with 50 nodes and 200 edges

0.2.2 Watts-Strogatz Model

Random graph models can imitate properties specific to certain types of networks. Watts and Strogatz introduced a model known as the rewired ring model that tries to engineer networks with the high levels of clustering but low path lengths between most vertices, like those seen in many social networks. This model imagines vertices positioned in a ring, and attaches each vertex to a number x of its nearest neighbors, where the value of x can be adjusted. Then, a certain percentage p of the edges are randomly rewired (Watts & Stogatz, 1998). This first property creates strong clustering, because a specific node and its adjacent neighbors have connections with a similar subset of other nodes. However, a graph constructed in this way would have long path lengths between nodes on opposite sides of the ring. The second property eliminates most of these long path lengths by creating several shortcuts across the graph. These two properties make the rewiring model a "small-world" model. An example of a Watts-Strogatz graph is shown in Figure 3.

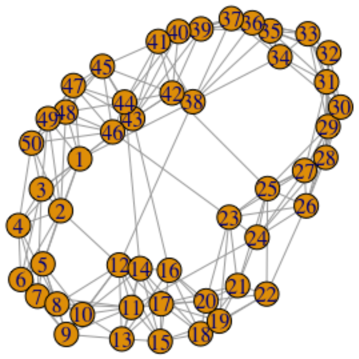

Figure 3: Watts-Strogatz Graph with 50 nodes and 200 edges

0.3 Twitter as a Social Network

Twitter can be considered as a social network, in which nodes represent people with different accounts and edges represent follower/followee connections. It is a directed social network, because one user who is followed by another user might not follow them back. Each node in the network has several characteristics that can be collected from their tweets: when and how often they tweet, the topics they focus on, and the other users that they interact with via retweets and mentions. Twitter is a massive network, which makes the process of sampling an interesting question. Since a simple random sample would not preserve network structure (it would create a set of nodes with few edges between them), one common approach is to use snowball sampling. This approach randomly adds a set number of initial users to the sample to start, and then includes the users with which they share edges. It continues to widen the network in this way, stopping when a certain number of users is reached. An advantage of this sampling method is that it preserves local edge structure. However, a disadvantage is that the selection of most nodes is not random. It is therefore a convenience sample, and not an accurate representation of the entire network (Heckathorn & Cameron, 2017).

0.4 Layout of Future Chapters

This thesis explores community detection in social networks, and how the definition of a community affects community detection results. The first chapter discusses how a community is defined and reviews common methods and difficulties in the field. It ends with a description of a recent paper that studies community detection within a Twitter network. The second chapter explains a simulation that we did to try to understand and verify the results found in the paper. The third chapter describes the exploration that we did with real data from a Twitter network.

Chapter 1

Community Detection

In a network, a community is a group of nodes that have more connections to each other than to nodes outside of the group. Within social networks, examples of communities include family groups, towns, or specific Facebook groups. Identifying communities within networks can provide information about the structure and organization of a network on a large scale (Newman, 2010). This can be useful for many academic and commercial reasons. For example, companies could target advertisements at communities of customers with similar preferences. Researchers could estimate characteristics about individuals from a difficult-to-sample population with information about community membership. People studying a certain network could also examine the structure of communities in that network and connections between the communities to obtain a broader understanding of the network (Fortunato, 2009).

1.1 What is a Community?

A community can be defined more specifically as a "locally dense connected subgraph on a network" (Barabasi, 2016). This definition can have several different technical interpretations, with varying levels of strictness. The maximum clique technique defines communities as complete subgraphs (cliques), in which each node has an edge to every other node in a community. This description is too strict to identify communities in real networks. A more relaxed property is the strong community property, where every node in the community has more links within the community than with the rest of the graph. That is, for a community C and node i, $k_i^{int} > k_i^{ext}(C)$ for each $i \in C$. An even more relaxed property is the weak community property, in which the total internal degree of a community is greater than the total external degree of a community, i.e. $\sum_{i \in C} k_i^{int}(C) > \sum_{i \in C} k_i^{ext}(C)$. In this case, most nodes in a community would abide by the strong community property but not all would have to (Barabasi, 2016).

These definitions of communities take a local view of the network, characterizing one node's community membership based on its relations with its neighbors and other nearby nodes. However, communities can also be defined in relation to the entire network. One popular way to identify communities is to compare the clustering in a real network to the clustering that might exist in a random graph with a similar structure, referred to as a null model. Using this type of definition, a subgraph might be considered a community if it has many more internal edges than expected in a subgraph of the same size in the null model (Fortunato & Hric, 2016).

1.1.1 Special Types of Communities

Although most traditional community detection algorithms identify partitions of nodes, in which each node is assigned to exactly one community, in many contexts it makes sense that an individual could have several community memberships. In a social network, for example, a person could belong to several groups, including friends, family, coworkers, or teams. This suggests that community detection methods should provide a cover of a network instead of a partition. A cover would create clusters of nodes within the network, but allow nodes to be part of any number of clusters instead of just one.

Real-world networks do not only have overlapping community structure, they also exhibit several layers of communities. These layers are nested: large communities are made up of smaller communities, which in turn contain even smaller communities. For example, a national honor society may form a large community, but also could be understood in terms of smaller regional chapters.

1.2 Community Detection Methods

A network can be divided into groups by two different types of algorithms: graph partitioning and community detection. Graph partitioning is a problem that comes from computer science and mathematics, and typically is used to divide a network into smaller pieces on which to do numerical calculations when the number and size of groups is known or fixed. It seeks to divide a network into non-overlapping clusters where the number of edges between clusters is minimized. Contrastingly, community detection algorithms do not specify the number and size of clusters, and seek to discover these quantities based on the network itself. They are able to detect clusters of different sizes within a single network. The problem within community detection is to find the natural divisions within a network by identifying clusters that minimize the number of edges between them (Newman, 2010).

1.2.1 Common Community Detection Methods

There are a wide variety of community detection methods that have been developed within several different fields, especially in the last twenty years (Fortunato, 2009). One common class of methods involves hierarchical clustering. This method begins with either all of the nodes in a single cluster, or every node in an individual cluster. It then divides or merges clusters gradually until the structure reaches the other extreme. This creates a hierarchical tree of partitions (known as a dendrogram), in which some measure of optimality is used to choose a single partition (Lancichinetti, Fortunato, & Kertesz, 2009). One of the most common hierarchical clustering methods was introduced by Girvan and Newman, in a paper that incited much of the recent work in community detection. This method uses a measure of edge betweenness centrality, which counts the number of shortest paths between two vertices that run through a specific edge. Girvan and Newman hypothesize that if a network contains tightly clustered communities that are linked together by a small number of edges, the edges linking the clusters together are the ones that should be removed in order to detect communities. This method is divisive, which means that it starts with all of the nodes in a single cluster. The method works by calculating the betweenness for all edges in the network, removing the edge with the highest betweenness, and then recalculating the betweenness for the network with that edge removed. This continues until all the edges are removed (and each node is therefore in its own cluster) (Girvan & Newman, 2002).

Another common class of community detection methods use modularity. Modularity measures the differences between the fraction of edges that run between vertices of the same community and the fraction of those edges that would be expected if the vertices were positioned at random, without regard to community membership. The modularity for a network is $Q = \frac{1}{2m}\sum_{(i,j)}(A_{ij} - \frac{k_i k_j}{2m})\delta(c_i, c_j)$, where m is the number of edges in the graph, (i, j) are all the pairs of vertices connected by an edge, k_i and k_j are the degrees of vertices i and j, and $\delta(c_i, c_j)$ is the Kronecker delta, which evaluates to 1 if i and j are in the same community, and 0 otherwise (Newman, 2010). High values of modularity suggest good clustering, which motivates modularity based community detection methods. The most basic modularity algorithm is Newman's greedy method. Although modularity was used within the Girvan-Newman method to test whether the given partitions were meaningful, in his 2003 paper, Newman suggests simply optimizing modularity over all potential divisions. This is a hard problem, so it is simplified with a greedy approach. The method is an agglomerative hierarchical method, where all of the vertices start in separate communities, and in each step, pairs of communities are joined together that cause the greatest increase or lowest decrease in the overall modularity. This builds a dendrogram, and the level with the maximal modularity is chosen as the optimal division (Newman, 2003). Several other modularity methods exist that build on this method.

A traditional way to detect overlapping communities is with clique percolation. This method finds cliques (completely connected subgraphs) of a specified size k. Cliques are adjacent if they share $k-1$ nodes. Adjacent k-cliques can be combined to create a k-clique chain. Then, a k-clique community is constructed as the union of a k-clique and all of the other k-cliques that are connected to it by a k-clique chain. However, this method is often too strict and computationally unfeasible, and does not identify different hierarchical levels of communities (Fortunato & Hric, 2016).

1.2.2 Order Statistics Local Optimization Method (OSLOM)

Order Statistics Local Optimization Method (OSLOM) is a community detection method developed by Lanichinetti, Radicchi, Ramasco, and Fortunato (Lancichinetti, Radicchi, Ramasco, & Fortunato, 2011). The goal of these researchers was to create a method for networks with weighted and directed edges that would be able to detect overlapping and hierarchical communities. OSLOM works to optimize local communities by finding statistically significant clusters compared to a null model. This will be explained further below, with our exposition following that of the paper (Lancichinetti et al., 2011).

OSLOM can be initially given a network or a community partition produced by another method. It uses local optimization to determine individual clusters, which are defined as groups of nodes that could potentially form a community, and checks whether any neighboring nodes should be added or nodes within the cluster should be removed. This is an iterative process that continues until the clusters are stable. It then checks whether clusters with many overlapping nodes should be joined together into a single cluster. Since this method is stochastic, this cover is compared with many other covers obtained in the same way to create a final cover. Once this process is finished, the algorithm processes the next hierarchical layer in a similar way. It does this by considering the clusters as super-vertices, with super-edges between them weighted by the number of edges connecting the two clusters, and doing the same analysis for the super-vertices and super-edges. This creates a cover of the network with potentially overlapping clusters at each hierarchical level of the network.

The local optimization in OSLOM works by estimating the statistical significance of a specific cluster, which the authors call single cluster analysis. This significance is determined by the probability of finding the cluster in a random null model. In this case, the null model retains the degree distribution of the given network, but the edges are randomly rewired under that constraint. The first half of the single cluster analysis deals with all nodes in the graph not within a cluster C that share an edge with a vertex in C, which are considered to be neighbors of C. For each vertex i in the neighbors of C, a probability defined as the rank of that vertex is computed. This is done in three steps.

In step 1, the probability that i would have as many edges as it does into C under the null graph is calculated. For notation, the total degree of the subgraph C is m_c, k_i is the degree of i, and all other vertices have a total degree of M. The edges included within m_c and k_i can be split into m_c^{in} for edges between two vertices in C and m_c^{out} for edges between one vertex in C and one out of it, and k_i^{in} for edges between i and a vertex in C and k_i^{out} for edges between i and a vertex outside of C. The degree of the entire graph with the cluster C and vertex i removed, $G\backslash[C \cup \{i\}]$, is M^*. Then, the probability that i has k_i^{in} edges into C under the null model is proportional to:

$$p(k_i^{in}|i, C, G) \propto \frac{\frac{1}{2}^{-k_i^{in}}}{k_i^{out}!k_i^{in}!(m_c^{out} - k_i^{in})!(M^*/2)!}.$$

When multiplied by the correct normalization factor, the sum of $p(k_i^{in}|i, C, G)$ over all potential values of k_i^{in} equals one.

In step 2, these probabilities are then used to calculate a cumulative probability that i has a number of edges to vertices within C greater than or equal to k_i^{in}:

$$r(k_i^{in}) = \sum_{j=k_i^{in}}^{k_i} p(j|i, C, G).$$

In step 3, each vertex i is assigned a value r_i that is randomly drawn from $[r(k_i^{in} + 1), r(k_i^{in})]$. This introduces an element of stochasticity to the process and makes it easier to compare vertices with different degrees.

Once the rank values, r_i, for each vertex i are computed, the smallest r_i is chosen. The cumulative distribution function for this first order statistic can easily be defined under the assumption that each r value is a uniform random variable between zero and one, as:

$$\Omega_1(r) = P(r_1 < r) = 1 - (1 - r)^{N - n_C}.$$

Using this distribution function, the probability of observing r_i or less can be calculated. Probabilities related to the other order statistics (r_2, etc.) can also be calculated in a similar way, from the following distribution functions:

$$\Omega_q(r) = P(r_q < x) = \sum_{i=q}^{N-n_C} \binom{N - n_C}{i} x^i (1 - x)^{N - n_C - i}.$$

Once these probabilities are calculated, the focus turns to the most extreme order statistic: $c_m = min_q\{\Omega_q(r_q)\}$. The cumulative distribution function for this random quantity depends only on $N - n_C$, and can be defined as $P(c_m \leq x) = \phi(x, N - n_C)$.

The vertices are considered in the order of their values of r_i, from smallest to largest, by comparing the value of $\phi(\Omega_q(r_q), N - n_C)$ to a hyperparameter P that determines when a certain order statistic is considered significantly more extreme from what would be expected in the null model. For the first vertex, $\phi(\Omega_1(r), N - n_C)$ is evaluated. If $\phi(\Omega_1(r), N - n_C) < P$, it is added to C. If not, $\phi(\Omega_q(r), N - n_C)$ is compared to P for the next vertices. If $\phi(\Omega_q(r), N - n_C) < P$ for any vertex, then the vertices that were considered from one to q are added to C. If $\phi(\Omega_q(r), N - n_C) > P$ for all vertices, then no vertices are added to C. In either case, the resulting union of C with any vertices i added is notated as C'.

The second part of the single cluster analysis deals with all of the vertices j within C'. For each j, r_j is calculated with respect to $C' \backslash \{j\}$. The vertices j in C' are then considered in order of those with the greatest values of r_j to the smallest. The evaluation of the vertices is the same as for vertex i outside of C, and if the vertex j with the highest value of r_j is found to be significant (its value of ϕ is less than P), then it is left in C' and the analysis is completed. Otherwise, the process continues for vertices j with lower values of r_j, and if a vertex to found to be insignificant, it is removed along with all previous vertices. If all vertices are significant, then C' remains the same. Once this process is finished, the single cluster analysis is completed.

This process is stochastic because of the way the value of r_i is randomly chosen from the calculated interval, and therefore each analysis of the cluster C could produce a different subgraph C'. Cluster membership is determined by calculating the participation frequency, f_i, for each vertex i, where f_i is defined as the ratio between the number of iterations that assign i to C' to the total number of iterations that produce a non-empty C'. Vertices with values of $f_i > 0.5$ are included in C', which only exists if it is non-empty for more than half of the iterations.

This process of evaluating the significance of clusters is also used to create the entire network. A random vertex i is sampled from G (the graph of the entire network), and the cluster $C = \{i\}$ is created. Then, the q most significant vertices are added to C, where q is sampled from a user-specified distribution. The single cluster analysis is done on this cluster, which adds or removes vertices. This is done for several random vertices in different parts of the network, which produces a set of potentially overlapping clusters.

Two clusters C_1 and C_2 are said to be similar if $\frac{|C_1 \cup C_2|}{min(|C_1|,|C_2|)} > 0.5$, where $|C_1|$ is the number of vertices in the set C_1, and $|C_1 \cup C_2|$ is the number of vertices that are in either C_1 or C_2. If the two clusters are similar, there is a process to decide whether to keep the two clusters, or to replace them with their union, $C_3 = C_1 \cup C_2$. To determine this, the method considers C_1' and C_2'. C_1' is obtained by taking the subgraph G_3, that only contains vertices contained within C_3, and running the single cluster analysis on C_1 relative to G_3. C_2' is obtained in the same way. Then, for a chosen value of a parameter P_2, if $|C_1' \cup C_2'| > P_2|C_3|$, then C_1 and C_2 are left unchanged, and otherwise, they are discarded in favor of C_3. Running this process on each group of similar clusters results in a group of clusters without any significant structures inside of the clusters.

This process produces a cover of the network. It is repeated many times, and done for each hierarchical level. Within each hierarchical level, clusters that are non-empty in more than half of the iterations are included in the cover for that level.

1.3 Detecting Communities Using Different Edge Weightings

In most traditional methods, community detection uses the structure of nodes and edges to create a partition of a network. However, communities in social networks overlap, and people can be members of many communities related to different parts of their lives. For example, a student may be part of a professional community in their field of study, a community of students who come from the same city, and another community of students who play on the same sports team. These communities may be created around different characteristics, and function in different ways.

In their paper “Followers Are Not Enough: A Multifaceted Approach to Community Detection in Online Social Networks,” Darmon et. al argued that in order to uncover different types of overlapping communities, the methods to detect communities can be adapted by weighting edges in different ways. Using the Twitter API, they collected information about accounts, follower-followee relationships, and timing and content of tweets over a continuous time period. They used the additional information to create three different types of edge weighting schemes besides the traditional structural scheme (in which unweighted edges represent follower/followee relationships). Using the timing of tweets, they weighted edges based on similar or different activity profiles of users to create communities of users with similar Twitter behaviors. They analyzed hashtags to weight edges based on content similarity to create communities of users focused on similar topics. Finally, they weighted edges based on retweets and mentions to create communities of users who frequently interacted with each other on Twitter. They ran OSLOM on each of the four sets of edge weights, and found that it produced different covers of the same network (Darmon, Omodei, & Garland, 2015).

Chapter 2

Simulations

2.1 Motivation

The motivation for these simulations was to explore the effects of edge weighting on community detection in a simple network, and see how these results compare to the results found in "Followers Are Not Enough: A Multifaceted Approach to Community Detection in Online Social Networks" (Darmon et al., 2015). It was also a testing ground to understand the edge weighting computations and gain familiarity with the OSLOM software for later work with real Twitter data.

2.2 Setup

2.2.1 Network Generation

Both the simulation and work with real data was implemented using the `igraph` package in R (Csardi & Nepusz, 2006). These simulations are run using both the Erdős-Rényi and Watts-Strogatz models to generate graphs that represent networks. The setup will be explained using the Watts-Strogatz model, but is comparable for the Erdős-Rényi model. We compare two different graphs, one with a clear community structure and one without. In the first simulation, each graph has 100 nodes, and about 1000 directed edges. The graph with community structure is generated by creating four different Watts-Strogatz graphs, each with 25 nodes. Then 50 edges are randomly placed throughout the 100 nodes, connecting the four separate graphs to make one with four clear separate clusters. The other graph is generated by creating one Watts-Strogatz graph with 100 nodes. The second and third simulations are similar, but have different numbers of nodes and edges.

Four topics are then propagated through each network. We were originally going to use these topics for the hashtag topic weighting scheme, but later created a larger list of topics to propagate. These four topics are instead used to model the interactions. Four source nodes are randomly sampled from the graph, and a breadth first search is run for each topic to assign each node a distance from that topic's source. Each topic is assigned a probability of propagation between 0 and 1. The topics are assigned in order from one to four, and if a node does not already have an assignment, it gets the topic in question with the topic's probability to the power of the distance of the node from the topic's source. This creates topic assignments that are clustered according to the graph's structure. This is done for each of the two graphs. The structural edge weighting simply uses the nodes and directed, unweighted edges.

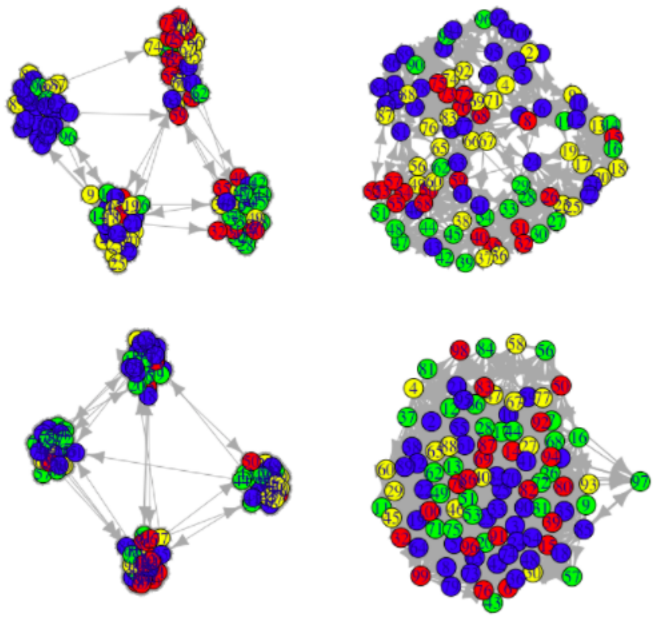

Figure 2.1: Watts-Strogatz Graphs (top row) and Erdős-Rényi Graphs (bottom row)

In this figure, the first two graphs are generated using the Watts-Strogatz graph and last two are generated with the Erdős-Rényi graph. We use interaction edge weighting and topic edge weighting in our simulation, adapted from the methodology used in the Darmon et al. paper, but do not attempt to use activity edge weighting because that has a much higher computation cost (Darmon et al., 2015).

2.2.2 Interaction Edge Weighting

Interactions are generated between nodes that share edges, with probabilities that are higher for nodes that share the same topic. For each node, a specific value is drawn from a Poisson distribution with $\lambda = 10$ to model the number of tweets that mention or retweet another user originating from that user. Then, using a multinomial model, the tweets are assigned to nodes that share an edge with the source node, and the nodes with the same topic as the source node have probabilities of receiving the tweets that are twice that of nodes with different topics. This creates a certain number of interactions for each edge. These interactions are normalized by dividing the number of tweets from node one to node two by the total number of tweets that node two received. This corresponds with the calculations for interaction edges in the Darmon et al. paper (Darmon et al., 2015). They define separate interaction weights for retweets and mentions, each of which is calculated as the number of interactions going from node u to node f, divided by the number of interactions received by node f. We calculate the simulated interaction scores in this way, which have values between 0 and 1, and use them as edge weights. There are roughly 25 possible edge weights in each run through the simulation, although this depends on how exactly the tweets are distributed between neighbors. The nodes and interaction edge weights between them are then processed by OSLOM to create a interaction based community cover.

2.2.3 Topic Edge Weighting

Twenty topics (or tags as they are called in some places in the R scripts) are propagated through the network in a similar way to the original four topics. Twenty source nodes are randomly sampled from the graph, and a breadth first search is done to calculate the distance from each node to the source node of each topic. Then, a probability between 0.25 and 0.75 is sampled from a random uniform distribution for each topic. Each node receives each topic with a probability of the topic's sampled probability to the power of the node's distance from the topic's source. That calculated probability is compared to a random variable between 0 and 1, and if the probability is larger, the node is assigned the topic. Unlike in the original propagation of four topics, each node could get any number of the twenty topics. Once each node has some set of topics, the topic edge weightings are calculated similarly to the weightings in the Darmon et al. (2015) paper. They specify a vector of hashtags for each user, $\vec{h}(u)$, where $h_i(u) = \theta_i(u)log(\frac{N}{n_i})$ and $\theta_i(u)$ is the frequency of the topic i in the user u's tweets, N is the total number of users, and n_i is the number of users who use topic i in their tweets. This weights the frequency of a hashtag in a certain user's tweets by the number of people in the dataset who used that hashtag. Then, the similarity between two users is determined by the cosine similarity between the two vectors: $w_{u \to f} = \frac{\vec{h}(u) \cdot \vec{f}(u)}{||\vec{h}(u)||||\vec{h}(f)||}$. The simulation has a binary variable for whether or not a node is associated with a certain topic, instead of a frequency. However, the h vectors and cosine similarities were calculated in the same way as in the paper (although θ_i was either 0 or 1, instead of a frequency). These similarity measures are used as the topic edge weights, and processed by OSLOM to produce a topic based community cover.

2.2.4 Switched Edge Weightings

Because of the method in which we propagate interactions and topics through the graphs, both of these node characteristics are greatly affected by the underlying graph structure. Although these relationships exist in real networks, we wanted to try to distinguish the effect of the different edge weighting schemes from the effect of the underlying graph structure. We tried to diminish the relationship of the graph structure with the topics and interactions by taking the topics (both the four that are used to propagate interactions and twenty that are used to calculate the topic weightings) that are associated with the 100 nodes in one graph, and using them to calculate edge weightings for the edges present in the other graph. For example, when we discuss the switched edge topic weighting for the four community graph, we mean that we use the topics propagated on the one community graph based on the one community structure to calculate topic weights for the edges that exist in the four community graph. For both the one community graph and four community graph, we have five edge weightings: the structural weighting, interaction weighting, topic weighting, switched interaction weighting, and switched topic weighting. These ten different weightings are done for graphs generated from both the Erdős-Rényi and Watts-Strogatz models. Overall, this sums to twenty sets of nodes and edge weights for each run through the simulation.

2.2.5 Different Settings

Going forward, we will refer to the four potential setups with labels. Setting **a** is the graph with one community and setting **b** is the graph with one community, but edge weights calculated from the four community graph topics. Setting **c** is the graph with four communities and setting **d** is the graph with four communities, but edge weights calculated from the one community graph topics.

2.2.6 OSLOM Software

In conjunction with their paper introducing the OSLOM method of detecting overlapping hierarchical communities in networks, Lancichinetti, Radicchi, Ramasco, and Fortunato released a freely available software to run OSLOM on network data (Lancichinetti et al., 2011). We run simulations through the OSLOM software by saving the nodes and weighted edges of a network to a file, and using a command in the console to run the software on the specific network data. Although this process creates several files, the most important outputs are text files for each hierarchical level of communities found by the method that indicate which nodes belong to each community.

2.2.7 Three simulations

Overall we run three different simulations, changing the underlying graph structure each time. The first simulation uses four graphs with 100 nodes and about 1000 edges (two generated from the Watts-Strogatz model and two generated from the Erdős-Rényi model). However, this is about a fifth of the possible edges, which is relatively high, so the second simulation uses smaller graphs, with 100 nodes and about 600 edges. The goal of the third simulation is to test the claim made in the OSLOM paper about working with overlapping and hierarchical networks. For this simulation, the graphs with four communities are modified so that for each graph, five nodes in two of the four clusters are deleted, and all of the edges going in and out of these nodes are connected to nodes in one of the other two clusters. This creates graphs where eighty nodes belong to one community each, and ten nodes belong to two communities each. The differences between the four community graphs for the second and third simulations can be seen below.

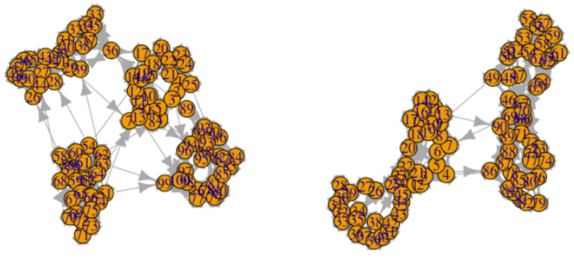

Figure 2.2: Four Community Graph from the Watts-Strogatz model, Simulation 2 (left) vs Simulation 3 (right)

2.3 Results

2.3.1 Network Statistics

We measured various network statistics (number of nodes, number of edges, diameter, average path length, and global transitivity) in each simulation for the two graph structures for both the Watts-Strogatz and Erdős-Rényi models. In the table, 'ws' refers to Watts-Strogatz and 'er' refers to Erdős-Rényi, while 'one' refers to the graph created from one original graph, and 'four' refers to the graph created from four original graphs combined together.

Table 2.1: Network Statistics for Four Networks with ~1000 edges

Network Statistic	ws one	ws four	er one	er four
Number of Nodes	100	100	100	100
Number of Edges	1000	1025	1000	1025
Diameter	4.94	7.60	4.00	7.46
Average Path Length	2.72	3.50	2.23	3.51
Global Transitivity	0.50	0.52	0.20	0.62
Local Transitivity	0.12	0.13	0.18	0.40

Table 2.2: Network Statistics for Four Networks with ~600 edges

Network Statistic	ws one	ws four	er one	er four
Number of Nodes	100	100	100	100
Number of Edges	600	625	600	625
Diameter	7.02	9.74	5.40	9.16
Average Path Length	3.68	4.22	2.74	4.02
Global Transitivity	0.44	0.41	0.12	0.41
Local Transitivity	0.10	0.10	0.11	0.32

Table 2.3: Network Statistics for Four Overlapping Networks with ~600 edges

Network Statistic	ws one	ws four	er one	er four
Number of Nodes	90	90	90	90
Number of Edges	540	609.26	540	609.44
Diameter	6.78	9.35	5.25	7.48
Average Path Length	3.56	4.01	2.68	3.42
Global Transitivity	0.44	0.42	0.13	0.38
Local Transitivity	0.10	0.11	0.12	0.33

As mentioned above, these simulations can be differentiated by the number of nodes and edges in each. However, certain trends hold for all three of the simulations. The diameter and average path lengths are larger for the graph with four communities than the graphs with one community. The diameter and average path lengths are also larger for the Watts-Strogatz graphs than for the Erdős-Rényi graphs. This makes sense because the ring structure of the Watts-Strogatz graphs creates fewer edges between non-adjacent nodes than in the Erdős-Rényi graphs, therefore causing higher path lengths.

The transitivity values also follow similar patterns across the three simulations. The global transitivity is similar for both Watts-Strogatz graphs in each simulation, while it is much smaller for the Erdős-Rényi graph with one community than the Erdős-Rényi graph with four communities. The global transitivity for the Erdős-Rényi graph with four communities is larger than the global transitivities for the Watts-Strogatz graphs in simulation one, similar to the global transitivities for the Watts-Strogatz graphs in simulation two, and smaller than the global transitivities for the Watts-Strogatz graphs in simulation three. This change in the global transitivity for the Erdős-Rényi graph with four communities across the three simulations can be explained by the changes in the number of nodes and edges.

Because the Erdős-Rényi model generates a random graph, the expected global transitivity can be estimated by $\frac{c}{n}$, where c is the number of neighbors of each node and n is the number of nodes in the network. In the first simulation, where there are 100 nodes and 1000 edges, the mean degree is 20, so the expected global transitivity can be estimated by 0.2 (which is the observed value). However, for a random graph with 25 nodes and 250 edges (one of the building blocks of the four community graph), the mean degree is still 20, but the global transitivity jumps to 0.8. Since the four community graph has links between the clusters, the observed value is lower, at 0.6. In the second simulation, the lower number of edges leads to a lower mean degree and lower expected global transitivity than in the first simulation for both the one community and four communities graphs. In the third simulation, the expected (and observed) value of the global transitivity for the one community graph is slightly larger than in the second simulation, because of the smaller number of nodes. However, the larger number of edges between the clusters in the four community graph leads to the lowest observed global transitivity among the three simulations.

2.3.2 Community Detection Results

Table 2.4: OSLOM Results

Edge Weighting	structural	interaction	topic	switched interaction	switched topic
s1 ws one	several	several	several	several	several
s2 ws one	several	several	several	several	several
s3 ws one	several	several	several	several	several
s1 ws four	perfect	perfect	perfect	perfect	almost
s2 ws four	perfect	almost	poor	fair	almost
s3 ws four	fair	fair	poor	fair	poor
s1 er one	none	none	none	none	none
s2 er one	none	none	none	n/a	n/a
s3 er one	none	none	none	n/a	n/a
s1 er four	perfect	perfect	perfect	perfect	perfect
s2 er four	perfect	perfect	almost	n/a	n/a
s3 er four	perfect	almost	fair	n/a	n/a

This table qualitatively describes the results from OSLOM for the three simulations. The abbreviation ‘ws’ corresponds to Watts-Strogatz graphs and ‘er’ to Erdős-Rényi graphs, and ‘one’ means graphs with one large community while ‘four’ means graphs with four communities. For the graphs with one community, the two possibilities are either ‘several’, which means that OSLOM produced a cover with several communities within a few hierarchical levels, or ‘none’, which means that OSLOM found no communities. For the graphs with four communities, the possibilities are 'perfect', 'almost', 'fair', and 'poor'. 'Perfect' means that within one hierarchical level, OSLOM produced the four communities that were expected (for simulations one and two, this means grouping nodes 1-25, 26-50, 51-75, and 76-100, for simulation three, this means grouping nodes 1-25, 21-45, 46-70, 66-90). 'Almost' means that a few nodes may have been misgrouped or not included, 'fair' means that a significant number of nodes were not correctly identified, and 'poor' means that there were groups significantly smaller or larger than expected, and no hierarchical levels were close to identifying the four communities. The switched edge weightings for the Erdős-Rényi graphs in simulations two and three are labeled 'n/a' because OSLOM did not converge to a solution for these edgelists.

Watts-Strogatz

For the Watts-Strogatz graphs, OSLOM produced results influenced by our imposed one or four community structure and the small world structure. OSLOM found communities within the one community graph for each simulation and edge weighting scheme. This is because the structure of the Watts-Strogatz graph causes a higher clustering than expected in a random graph. The communities found are mostly sequential lists of nodes.

There was greater variety in the results for the four community graphs. For the first simulation, where there was a higher number of edges, the four communities were found perfectly by most of the weighting schemes, and almost entirely by the switched topic weighting. In the second simulation, when there was a lower number of edges (and therefore the communities were less tightly clustered), only the structural weighting identified the original four communities. The interaction and switched topic weightings almost found the four clusters, while the topic and switched interaction weightings found less accurate community covers. The third simulation provided less accurate results, with all of the edge weighting schemes finding covers that could be described as fair or poor representations of the original communities. This makes sense because overlapping communities are more difficult for all community detection algorithms to uncover, and the structure of the Watts-Strogatz graph could distract from this overlapping community structure.

Erdős-Rényi

The simpler structure of the Erdős-Rényi graphs produced results based on the imposed one or four community structure. OSLOM found no communities within the one community graph for each simulation and edge weighting scheme, supporting the statement by Lancichinetti et al. that OSLOM is capable of identifying a lack of community structure in situations where no communities exist (Lancichinetti et al., 2011).

OSLOM identified the four communities more accurately in the Erdős-Rényi graphs than in the Watts-Strogatz graphs. In the first simulation, each edge weighting scheme uncovered the four communities perfectly. In the second simulation, the structural and interaction weightings perfectly identified the four communities, while the topic weighting misplaced a few nodes. In the third simulation, with overlapping communities, the structural weighting perfectly identified the four overlapping communities, while the interaction weighting missed a few nodes, and the topic weighting found less accurate community covers. This shows that OSLOM can correctly identify overlapping communities, which was one of the major motivations behind its creation (Lancichinetti et al., 2011).

Comparison of Community Detection Algorithms for Overlapping Communities

We further explored OSLOM's ability to identify overlapping communities by visualizing the covers that it created for the overlapping four community Erdős-Rényi graph using the structural weighting, and comparing them with the results from other community detection algorithms. Using the structural weighting for this graph, OSLOM detected the four overlapping communities in the first hierarchical level, and combined each pair of overlapping communities into larger communities in the second hierarchical level. These covers can be seen in the graphs below, where nodes within single clusters are assigned colors based on their cluster, and nodes within multiple clusters are colored black.

These results are better than those for any of the community detection methods that are implemented within `igraph`. We first tried the walktrap algorithm implemented in `igraph`, because it is the only one that works with directed graphs. This approach is based on the idea that random walks on a graph often become trapped within clusters. A random walk is a process on a graph, where at each time set, the walker moves from the vertex it is currently on to a randomly chosen neighbor (Pons & Latapy, 2006).

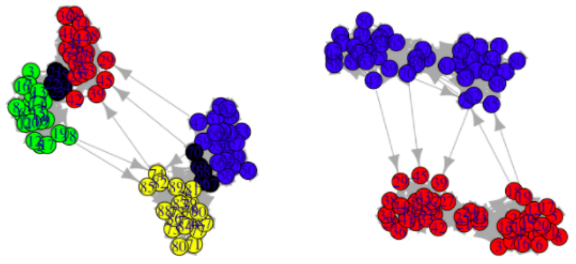

Figure 2.3: Simulation 3 Erdős-Rényi Graph, First (left) and Second (right) Hierarchical Levels

This method detected four communities, but did not put any node into more than one community, so there are are two communities with twenty-five nodes each, and two with twenty nodes each. This can be seen below.

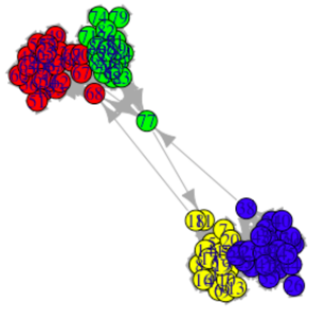

Figure 2.4: Simulation 3 Erdős-Rényi Graph, Walktrap Algorithm

We also tried several community detection algorithms that cannot work with directed graphs, and instead use the underlying undirected graphs. These include the edge betweenness and leading eigenvector methods. Edge betweenness uses the Girvan-Newman divisive hierarchical method, and leading eigenvector uses a complex type of modularity optimization. These methods detected three and four communities respectively, and neither found overlapping communities. This comparison, along with a more careful test of other community detection methods implemented in `igraph`, shows that none of them work well with overlapping communities. Currently, none of the methods created to deal with overlapping communities, such as clique percolation, are implemented as functions within `igraph`.

2.3.3 Edge Distributions

After looking at the individual community membership, we looked at the distributions of the edge weights for both graph structures using the Watts-Strogatz and Erdős-Rényi models. Comparing edge weight densities provides a more global view of the edge weighting schemes. We present and discuss distributions from simulation two because all three simulations gave similar results for the edge weight distributions. These histograms show the aggregated edge weight densities over ten simulations.

Watts-Strogatz Edge Distributions

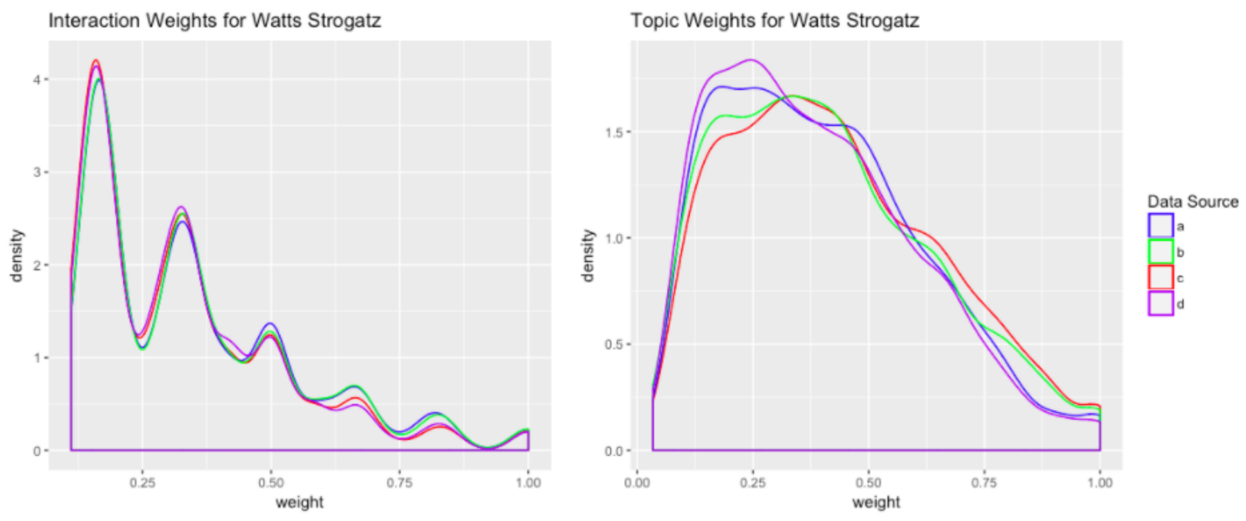

Figure 2.5: Interaction and Topic Weights for Watts-Strogatz Graphs

We start with the edge weight distributions for the Watts-Strogatz model, seen in Figure 2.5. For the structural weighting, all of the edge weights have magnitudes of 1. The interaction and topic weights both range between 0 and 1. The density plot for the interaction weights shows a highly right skewed distribution with clear peaks, suggesting that although weights could take on several values, they are most likely to be about 0.15, followed by 0.30, 0.50, 0.65, and other values with lesser probabilities. The densities are very similar for the four different settings. However, there is a small amount of variation between the four densities, although there is a clear pairing between settings **a** and **b**, and settings **c** and **d**. The density plot for the topic weights shows more variation between the different settings. The distributions are continuous and unimodal, and only slightly right skewed. Although each distribution is relatively similar, the peaks match most closely between settings **a** and **d**, and settings **b** and **c**.

Erdős-Rényi Edge Distributions

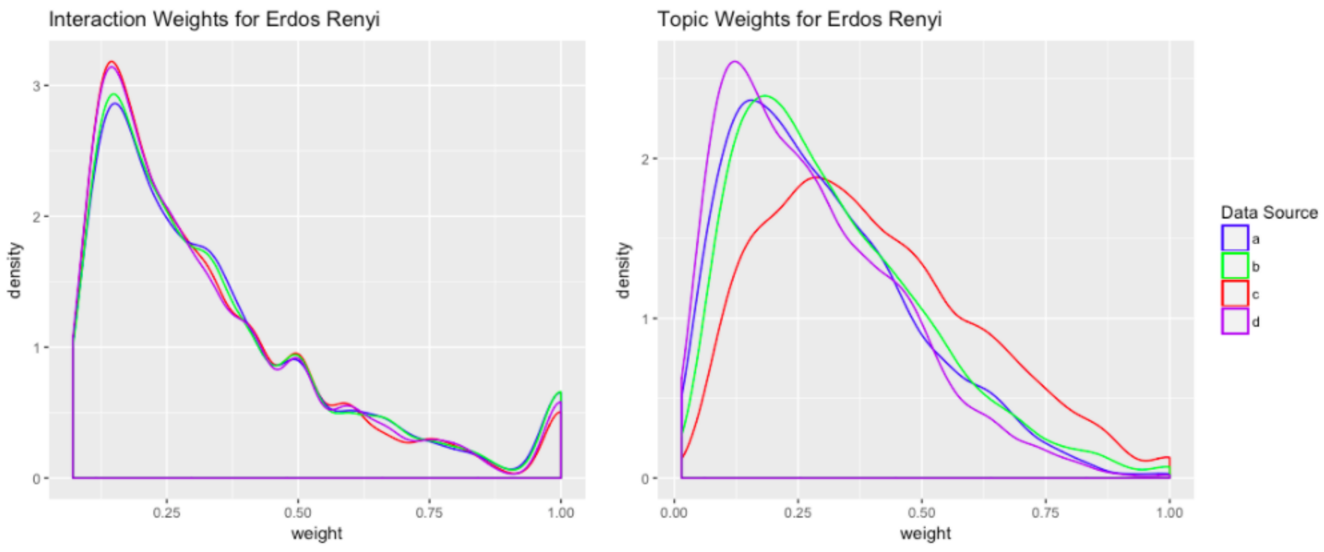

Figure 2.6: Interaction and Topic Weights for Erdős-Rényi Graphs

The graphs generated from the Erdős-Rényi model have slightly different edge weight distributions than the graphs from the Watts-Strogatz model, as seen in Figure 2.6. The interaction and topic weights both range between 0 and 1. In the interaction weight density plot, the distribution is also very right skewed and there is a clear peak around 0.15, although this peak is slightly higher for the graphs with four communities (settings **c** and **d**) than for the graphs with one community (settings **a** and **b**). The rest of the distribution appears more continuous than the distribution for the Watts-Strogatz graphs, and there are slight bumps instead of noticeable peaks. The topic weight density plot has three distinct unimodal right skewed distributions: the most skewed one for setting **d**, an intermediate one for settings **a** and **b**, and the least skewed one for setting **c**.

2.4 Discussion

Although this simulation mainly provided experience with OSLOM and a better understanding of the edge weighting schemes, it gave some insight into how changing the edge weights changed the ability of OSLOM to identify the original network structure. OSLOM generally detected the four communities for each graph that originally had that four community structure. It detected communities of sequential nodes in the graphs that had no community structure (where all the nodes were grouped in one community) when the graphs were generated by the Watts-Strogatz model, and detected no communities in the one community Erdős-Rényi graphs. The fact that the community covers based on the structural weightings were the most accurate in detecting the four community structure suggests that either the interaction and topic edge weighting schemes are adding additional noise to the graph, making it harder to detect the underlying structure, or that they are detecting different types of communities. It is not possible to tell which of these is happening with a simulation this simple, but this is worth studying with real data or a more complex simulation.

The edge weight distributions for the different generative models and the interaction and topic weights provide insight about the effects of switching topics and of the small-world principle in the Watts-Strogatz model. Comparing both the weight distributions between the graphs generated with the Watts-Strogatz and Erdős-Rényi-models, the distributions of the different settings are more similar within the Watts-Strogatz graphs. This suggests that imposing the different number of communities and switching topics had less of an effect on edge weights when the graph structure was also influenced by the small-world principle.

It is also interesting to compare which settings have similar distributions. It is difficult to see with the interaction weights for the Watts-Strogatz graphs, since the four distributions are very similar, but in the interaction weights for the Erdős-Rényi graphs there is a clear connection between the two combinations with one community (settings **a** and **b**), and the two combinations with four communities (settings **c** and **d**). This suggests that switching the topics (generating interactions for the four community graph based on the initial topics from the one community graph, or vice versa) had little effect on the edge weights. The opposite pattern was seen for the topic weights. For both models, the distributions for the graphs where the topics were switched mirrored the distribution for graph with the same topics. This means that setting **a** matched setting **d**, and setting **b** matched setting **c**. This suggests that for the topic weights, the topics matter more than the underlying structure. It makes sense that the topic weights would be more affected than the interaction weights by switching the topics, because the topic weights are directly affected by the topics, while the interaction weights are only indirectly affected by them.

This simulation did not capture many aspects of the Twitter context that we were modeling. The way that we modeled interactions had a much higher percentage of non-zero edges than would make sense in the context of a Twitter network. The topics were also too simple to represent the large set of potential hashtags, which have frequencies in the original Twitter network instead of the binary indicator used in the simulation.

Another problem with the simulation is that it relied on the structure of the graph to propagate the other information (interactions and topics) throughout the network. Both versions of the topics were propagated with probabilities to the power of distances from the node to the source node, which made nodes close to each other more likely to have similar topics. This makes sense in the context of Twitter, because people who follow or are followed by each other likely have some similarity, whether they regularly interact or use discuss similar topics. However, in this simple simulation, the effect of the graph's structure may have muted other effects. We tried to control for this by using the edge weightings for the interaction and topic weightings that switched the structural information for the two graphs. We also tried to control for it by using both the Erdős-Rényi and Watts-Strogatz models to create the social network graphs. The Watts-Strogatz model is considered to be a better approximation for social networks, however the effect of the small world structure it generates was too strong for this simulation. The Erdős-Rényi model did not fit the Twitter context as well, but was better able to isolate the effects of the different edge weightings.

Chapter 3

Real Network Data

3.1 SNAP Data

We obtained data from the Stanford Network Analysis Project (SNAP) (Leskovec & Krevl, 2014). We specifically used a dataset of social circles from Twitter, which was collected by McAuley and Leskovec for their 2012 paper "Learning to Discover Social Circles in Ego Networks" (McAuley & Leskovec, 2012). The Twitter data was obtained from public sources. The dataset consists of 983 separate networks, that range in size from 10 to 5000 nodes. These networks were each collected around a central "ego" user, and although this user was not included in the network, all of the users that the ego user follows were added to the network. These followees of the ego user are the nodes in the network, and the follower relationships form the edges between them. The dataset also includes the set of hashtags and mentions used by each user during a two-week period of tweets.

3.1.1 Getting Data Into R

For each of the 983 ego networks in the data there are five files, four which are used in this analysis. The files are named by the number of the ego node, which is referred to as the 'nodeId.' The file 'nodeId.edges,' contains directed edges between the nodes in the network. The file 'nodeId.featnames' contains a list of all of the features (hashtags and mentions) that were used by any node within the network, indexed by numbers. Within 'nodeId.egofeat,' there are binary indicators for each feature in the order with which they are indexed in the 'nodeId.featnames' file. The file 'nodeId.feat' includes the binary indicators for each feature for each node, with the nodeId preceding its list of indicators.

These files were uploaded into R using the `readtext` and `stringr` packages, and converted from text files into data frames. Using the `igraph` package, the edge list data frames were turned into igraph objects. The network visualizations were done with the `pajek` software (Batagelj & Mrvar, 1999).

3.1.2 Circles

The three circles that we chose are some of the larger networks that the dataset included, with about 220 nodes each. The first circle has the most edges, around 8000, while the other two have about 5000. The other network statistics are relatively similar for each circle.

Table 3.1: Network Statistics for Three Circles

Network Statistic	circle 100318079	circle 102903198	circle 105150583
Number of Nodes	221	226	210
Number of Edges	8354	4925	4795
Diameter	6	7	8
Average Path Length	2.19	2.36	2.52
Global Transitivity	0.57	0.40	0.50
Local Transitivity	0.33	0.25	0.29

3.1.3 Community Detection Results

Order Statistics Local Optimization Method

OSLOM detected two hierarchical levels using each circle's structural weighting, and one hierarchical level using the topic weightings. The topic weight were calculated in the same way as in the simulation, using the features in the 'featnames' files as topics. There were more communities found in the first hierarchical level of the structural weighting than in the second hierarchical level of the structural weighting and in the topic weighting for each circle. It is interesting to note that OSLOM detected overlapping communities within each hierarchical level. This means that it placed at least one node in more than one community in each cover. These nodes can be seen in the figures as the nodes that are placed in the space between clusters.

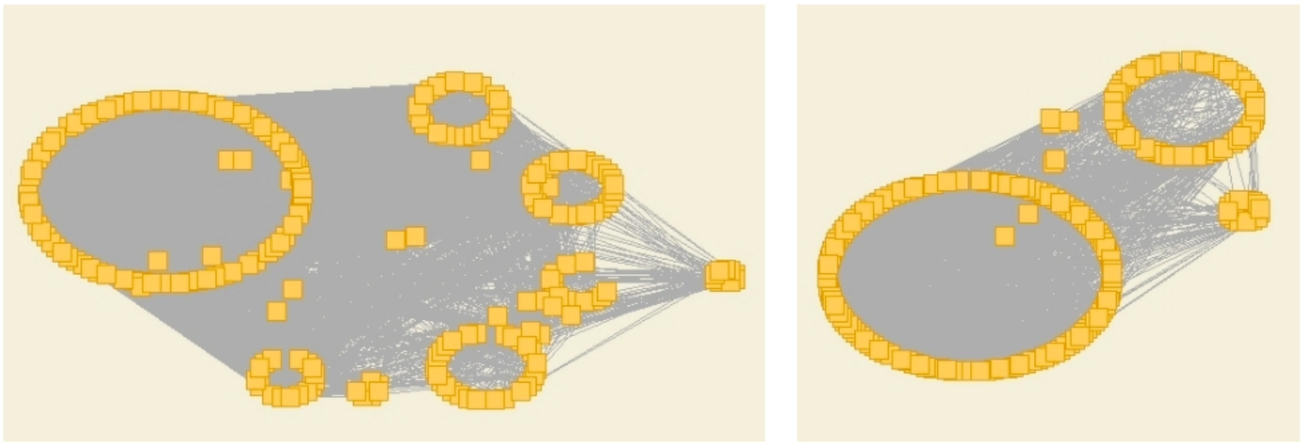

Figure 3.1: Communities in OSLOM First Hierarchical Level of Circle 100318079 for Structural (left) and Topic (right) Weightings

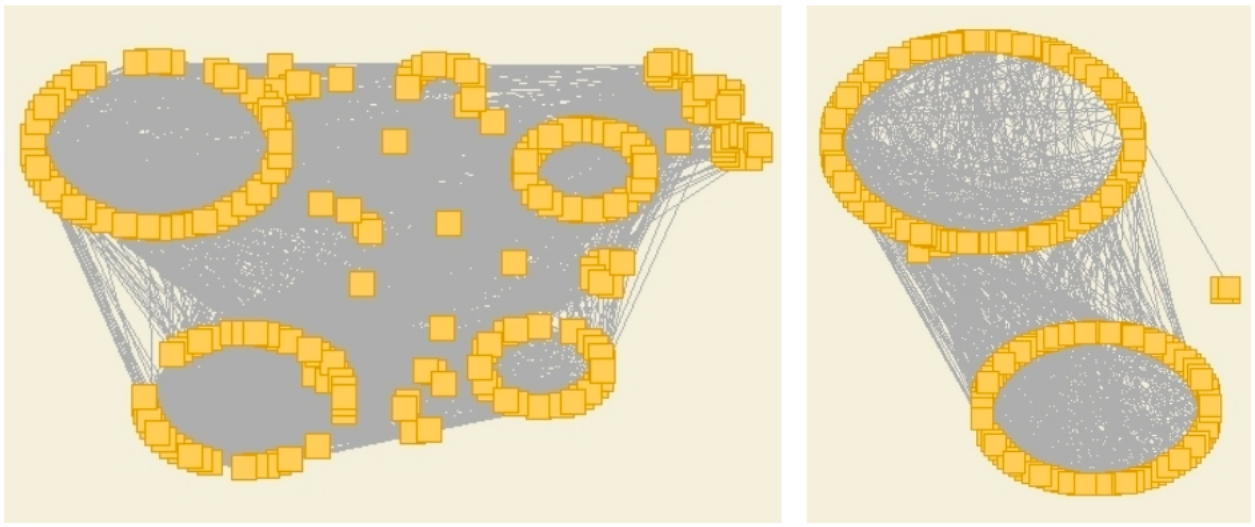

Figure 3.2: Communities in OSLOM First Hierarchical Level of Circle 102903198 for Structural (left) and Topic (right) Weightings

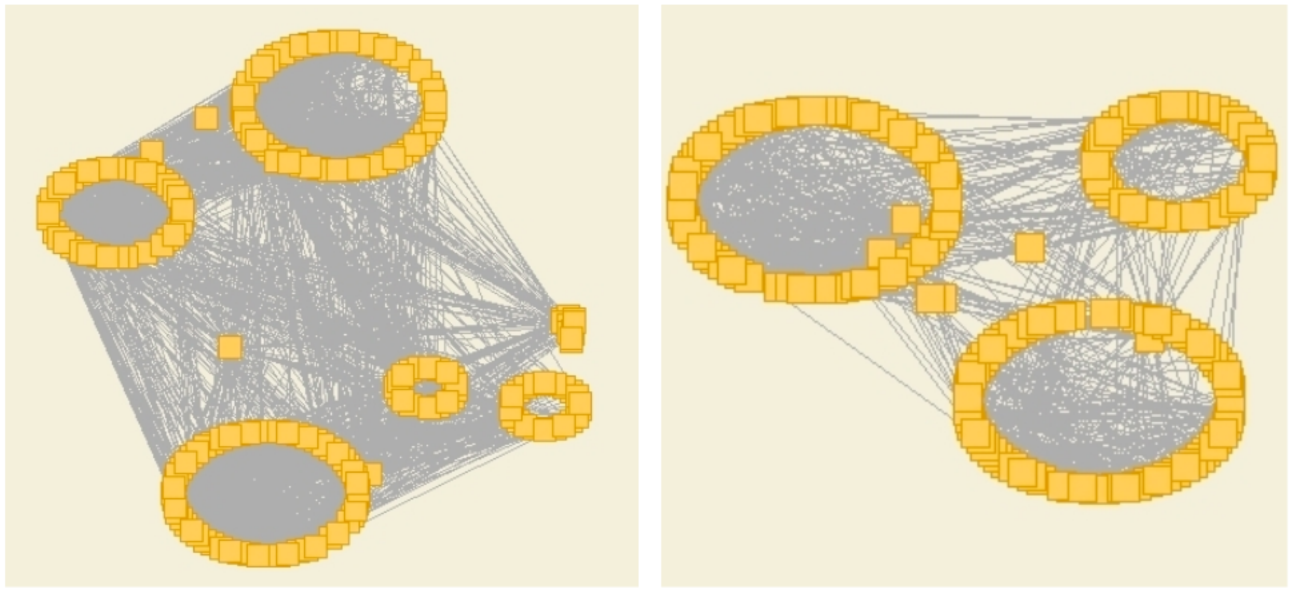

Figure 3.3: Communities in OSLOM First Hierarchical Level of Circle 105150583 for Structural (left) and Topic (right) Weightings

Other Community Detection Algorithms

We also ran three other community detection algorithms implemented in `igraph` for each of the three circles. The walktrap method is implemented for directed graphs, while the edge betweenness, and leading eigenvector methods are only implemented for undirected graphs (which were obtained by forcing the directed graphs to be undirected).

It is difficult to compare the community covers or partitions created by different methods in this setting, because there is no ground truth of what communities should exist. However, several things can be seen in these results. Each of the methods find a smaller number of communities that contain many nodes except for the edge betweenness method, which puts most of the nodes in individual communities, and then creates only a few clusters of nodes. Based on numbers of communities, the walktrap method seems to identify similar types of communities to the first hierarchical level of OSLOM for the structural weighting, while the leading eigenvector method seems to identify similar types of communities to the second hierarchical level of OSLOM for the structural weighting and to the OSLOM results for the topic weighting.

Table 3.2: Numbers of Communities Found for Different Methods

Community Detection Method	100318079	102903198	102903198
OSLOM structural level 1	8	9	8
OSLOM structural level 2	2	4	5
OSLOM topic level 1	3	3	3
Walktrap	9	11	9
Edge Betweennness	74	107	63
Leading Eigenvector	3	2	6

3.1.4 Topic Edge Weight Distributions

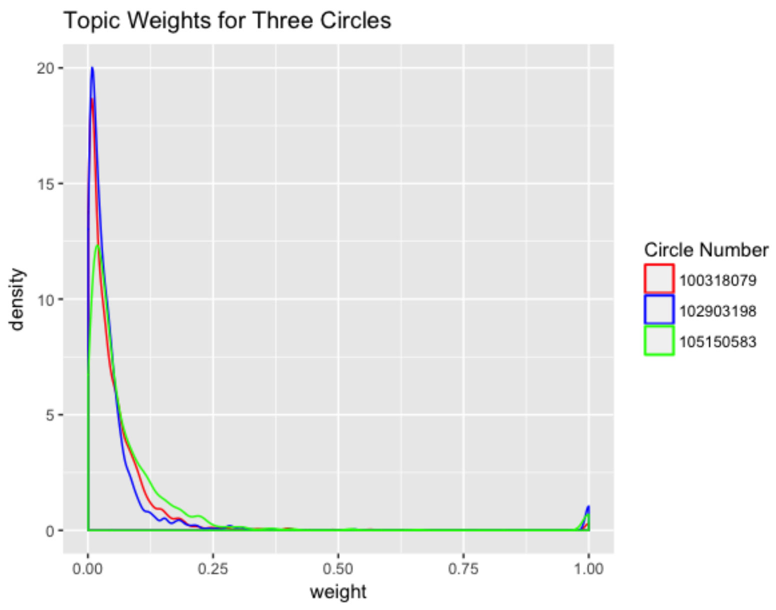

Figure 3.4: Edge Weight Distribution for Three Circles)

The topic edge weights for each of the three circles have similar distributions, as seen in Figure 3.4. The distribution for each is unimodal, with a peak close to 0.00, few weights between 0.25 and 0.95, and a small peak at 1.00.

3.1.5 Discussion

For each of the three circles, the structural weighting and topic weighting clearly produced different covers of the network. In each of the three circles, OSLOM found more communities with fewer numbers of nodes for the structural weighting than for the topic weighting. This confirms the hypothesis that weighting the edges affects the communities that are found. However, since the topic information is very sparse, it is possible that instead of adding more information, the topic weighting is simply creating edges with lower weights or removing edges entirely from the original structural network, and then trying to detect communities with less information.

The sparseness of the information can especially be seen in the topic edge weight distribution. Most of the edges had weights very close to zero, which suggests that they had very few hashtags or mentions in common. However, this data only includes a two-week period of tweets.

The data available was a major limitation to this part of the study. Within their paper on this subject, Darmon, Omodei, and Garland used the Twitter API and a great deal of data processing to collect a sample network of 15,000 users from Twitter that included the underlying structural network and tweets for each user involved over a nine week period (Darmon et al., 2015). It quickly became obvious that a similar type of data collection would be out of the scope of this study. However, we were not able to find Twitter data that included structural information for a large network and the contents of tweets. The SNAP data was useful because it includes several small networks and topic information attached to these networks, but it is not the ideal type of data to test the conclusions about different edge weighting schemes producing different network covers.

3.2 Russian Trolls Data

An interesting extension of this study could be to investigate the network of users from the tweets of Russian trolls that were recently uncovered with connections to the 2016 presidential election. The Twitter accounts were suspended, but NBC published a file with information about users and tweets. If information about the underlying structure of the network were available, it would be interesting to compare the communities found using the structural weighting with the communities found using a topic or interaction weighting scene that takes into account hashtags and mentions. This could provide insight into how the different accounts were working in groups, and how they interacted with non-troll accounts.

Conclusion

Community detection is an important problem in the study of social networks. Identifying communities can provide information about the macroscopic structure of a network and help solve problems that use network data. The field has a strong foundation, with methods that have been used and improved for decades, as well as innovative new methods that can be used for even more complex networks. In this thesis, we first provided an overview of social networks and then used that foundation to describe community detection. We discussed several of the most popular methods of community detection, and then described a newer approach, called OSLOM, that aims to work for networks with overlapping and hierarchical structure. We also introduced the hypothesis stated in Darmon, Omodei, and Garland's paper that different edge weightings based on node and edge attributes will result in different communities detected (Darmon et al., 2015). Next, we ran several simulations to test this hypothesis and the several of the claims made about OSLOM's performance for different types of networks. Finally, we used OSLOM and a topic-based edge weighting to explore a small amount of real data from Twitter, and discussed other applications of community detection to Twitter data.

The simulation work was the main part of this study, and although it gave some insight into differences between edge weightings and the potential of OSLOM, it had several limitations. Although the underlying structure was easy to simulate in a variety of different ways, there was no clear way to propagate the kind of information that would exist in real Twitter data. There were also limitations on the amount of information that could be propagated without causing too much confusion. In this sense, the simulation did not accurately reflect the amount of information that is available to be used for community detection in real data. In the future, it could be useful to simulate both the underlying structure of the graph and node and edge properties using parameters calculated from real network data.

However, it was difficult to collect or find real Twitter data in the form of a network sample, with both the underlying structure and information about tweets. The SNAP data was a useful place to test OSLOM and the different edge weighting schemes on a real network, but it was too small and sparse to let us test most of the conclusions put forth in Darmon, Omodei, and Garland's paper. Future work in this direction could work towards collecting a large sample from the Twitter API that would contain enough tweet information to be used for all of the edge weighting schemes proposed in that paper.

The emphasis in this study on OSLOM and different edge weightings for Twitter data suggests that one important direction for the field of community detection is work with network data that has a complex structure and a great deal of additional information. Rapid innovations in technology have generated a massive amount of data, a great deal of which can be studied in a network format. However, this data will inevitably have overlapping and hierarchical structure, and contain information beyond structure that can help with the problem of community detection. Considering how complex most network data is, it is surprising that the popular network R package `igraph` only has nine built-in community detection methods. Out of these, only three can deal with directed graphs, and none of them can detect overlapping communities. We would recommend that developers of new community detection algorithms work to implement them within commonly used packages, in order to increase access to methods that work for complex network data.

Appendix A

Simulation Appendix

Below is the code used for different parts of the simulation. The code is spread across several files in Github, but the order that it is included in this appendix is: ‘creating_graphs.R’, ‘overlapping_graphs_er_sm.R’, ‘full_simulation.Rmd’,‘network_stats.Rmd’, ‘plotOSLOM.Rmd’, and ‘OtherCDAlgs.Rmd’. These files are stored in the ‘NetworkSimulation’ folder. It is useful to note that the four community graphs are referred to in the code as ‘gws’ when they were generated by both the Watts-Strogatz or Erdős-Rényi models, and the one community graphs are reffered to as ‘gbig’ when they were generated by both models.

This first chunk loads all of the packages used in the simulation.

```r
require(mosaic)
require(tidyverse)
require(igraph)
require(sand)
require(multiplex)
require(ggplot2)
set.seed(33)
```

This is part of the code run in ‘creating_graphs.R’ that creates the one community and four community graphs from Watts-Strogatz models for simulation 1.

```r
#gws
#Creating individual graphs
g.ws1 <- watts.strogatz.game(1,25,5,0.05)
g.ws2 <-watts.strogatz.game(1,25,5,0.05)
g.ws3 <-watts.strogatz.game(1,25,5,0.05)
g.ws4 <- watts.strogatz.game(1,25,5,0.05)
gws <- g.ws1 + g.ws2 + g.ws3 + g.ws4
gws <- as.directed(gws)
x <- 50
#linking the graphs together randomly
gws <- add_edges(gws,sample(c(seq(1,100)),x,replace=T))
```

```
#Creating single larger graph
gbig <- as.directed(watts.strogatz.game(1,100,5,0.05))
```

This is part of the code run in 'overlapping_graphs_er_sm.R' that creates the one community and four community graphs from Erdős-Rényi models for simulation 3.

```
#gws
#Creating individual graphs
g.ws1 <- erdos.renyi.game(n=25,p.or.m=150,type="gnm",directed=T)
g.ws2 <-erdos.renyi.game(n=25,p.or.m=150,type="gnm",directed=T)
g.ws3 <-erdos.renyi.game(n=25,p.or.m=150,type="gnm",directed=T)
g.ws4 <- erdos.renyi.game(n=25,p.or.m=150,type="gnm",directed=T)
gws <- g.ws1 + g.ws2 + g.ws3 + g.ws4
x <- 20
#linking the graphs together randomly with ten edges
gws <- add_edges(gws,sample(c(seq(1,100)),x,replace=T))
gws <- simplify(gws)
plot(gws)
ecount(gws)
r <- 4
rem <- c(26:(26+r),76:(76+r))
for (i in rem) {
  for (e in neighbors(gws,i,mode="in")) {
    gws <- add_edges(gws,c(e,i-(r+1)))
  }
  for (e in neighbors(gws,i,mode="out")) {
    gws <- add_edges(gws,c(i-(r+1),e))
  }
}
gws <- delete_vertices(gws,rem)

#gbig
#Creating single larger graph
gbig <- erdos.renyi.game(n=90,p.or.m=540,type="gnm",directed=T)
```

The rest of the scripts that generate graphs (there are 6 in total) are similar and can be seen in the Github repository. The next several code chunks are taken from the file 'full_simulation.Rmd', and run most of the simulation. There are different ways to generate graphs, one 1000 edge Watts-Strogatz (z=1), one 1000 edge Erdős-Rényi (z=2), one 600 edge Watts-Strogatz (z=3), one 600 edge Erdős-Rényi (z=4), one overlapping 600 edge Watts-Strogatz (z=5), and one overlapping 600 edge Erods-Renyi (z=6).

```r
z <- 6

if (z == 1) {
  source("NetworkSimulation/simulation_generation/creating_graphs.R")
}
if (z == 2) {
  source("NetworkSimulation/simulation_generation/creating_graphs_er.R")
}
if (z == 3) {
  source("NetworkSimulation/simulation_generation/creating_graphs_sm.R")
}
if (z == 4) {
  source("NetworkSimulation/simulation_generation/creating_graphs_er_sm.R")
}
if (z == 5) {
  source("NetworkSimulation/simulation_generation/overlapping_graphs_sm.R")
}
if (z == 6) {
  source("NetworkSimulation/simulation_generation/overlapping_graphs_er_sm.R")
}
```

Setting the topics for gws, and saving the edgelist for the structural edge weighting for gws.

```r
#Setting topics
n <- vcount(gws)
topic <- data.frame(numeric(n),numeric(n),numeric(n),numeric(n),numeric(n))
names(topic) <- c("id","prob_1","prob_2","prob_3","assign")
topic$id <- seq(1:n)
#creating label for original graph (nodes 1-25 in 1,
#nodes 26-50 in 2, nodes 51-75 in 3, and nodes 76-100 in 4)
topic <- mutate(topic,group = (id-1) %/% 25 + 1)
#picking a source node for each topic
source <- sample(1:n,3)
#sampling random uniforms to later compare to probabilities
num1 <- runif(n,0,1)
num2 <- runif(n,0,1)
num3 <- runif(n,0,1)
topic$assign[source] <- c(1,2,3)

#computing distances between the source nodes and all other nodes
dists_1 <- bfs(gws,source[1],dist=T)$dist
dists_2 <- bfs(gws,source[2],dist=T)$dist
dists_3 <- bfs(gws,source[3],dist=T)$dist
```

```
#assigning probabilities for each topic based on distance from the
#source node and a probability for the neighbors of the sources to
#get that topic
topic$prob_1 <- 0.5^dists_1
topic$prob_2 <- 0.7^dists_2
topic$prob_3 <- 0.8^dists_3
topic$assign <- ifelse(num1<topic$prob_1,1,0)
topic$assign <- ifelse(num2<topic$prob_2 & topic$assign == 0,2,topic$assign)
topic$assign <- ifelse(num3<topic$prob_3 & topic$assign == 0,3,topic$assign)
topic$assign <- ifelse(topic$assign == 0,4,topic$assign)

#saving the number of nodes with each topic
gws_topic = topic %>%
  group_by(assign) %>%
  summarise(n=n())
gws_topic

#observing topic assigment based on group membership
topic %>%
  group_by(group,assign) %>%
  summarise(n=n())

#assigning the topics as node attributes
gws$topic <- topic$assign

#visualizing the graph based on topics
V(gws)[gws$topic==1]$color <- "red"
V(gws)[gws$topic==2]$color <- "blue"
V(gws)[gws$topic==3]$color <- "green"
V(gws)[gws$topic==4]$color <- "yellow"
plot(gws)

#saving the edgelist (unweighted)
gws_edgelist <- get.edgelist(gws)
gws_edgelist_df <- data.frame(gws_edgelist) %>%
  arrange(X1,X2)

if (z == 1) {
  write.dat(gws_edgelist,"NetworkSimulation/gws/")
}
if (z == 2) {
  write.dat(gws_edgelist,"NetworkSimulation/gws_er/")
}
if (z == 3) {
  write.dat(gws_edgelist,"NetworkSimulation/sim2/gws")
}
if (z == 4) {
  write.dat(gws_edgelist,"NetworkSimulation/sim2/gws_er/")
}
if (z == 5) {
  write.dat(gws_edgelist,"NetworkSimulation/sim3/gws")
}
if (z == 6) {
  write.dat(gws_edgelist,"NetworkSimulation/sim3/gws_er/")
}
```

Setting the topics for gbig, and saving the edgelist for the structural edge weighting for gbig.

```
#Setting topics
topic <- data.frame(numeric(n),numeric(n),numeric(n),numeric(n),numeric(n))
names(topic) <- c("id","prob_1","prob_2","prob_3","assign")
topic$id <- seq(1:n)
topic$assign[source] <- c(1,2,3)

#computing distances between the source nodes and all other nodes
dists_1b <- bfs(gbig,source[1],dist=T)$dist
dists_2b <- bfs(gbig,source[2],dist=T)$dist
dists_3b <- bfs(gbig,source[3],dist=T)$dist

#assigning probabilities for each topic based on distance
#from the source node and a probability for the neighbors
#of the sources to get that topic
topic$prob_1 <- 0.5^dists_1b
topic$prob_2 <- 0.7^dists_2b
topic$prob_3 <- 0.8^dists_3b
topic$assign <- ifelse(num1<topic$prob_1,1,0)
topic$assign <- ifelse(num2<topic$prob_2 & topic$assign == 0,2,topic$assign)
topic$assign <- ifelse(num3<topic$prob_3 & topic$assign == 0,3,topic$assign)
topic$assign <- ifelse(topic$assign == 0,4,topic$assign)

#saving the number of nodes with each topic
gbig_topic <- topic %>%
  group_by(assign) %>%
  summarise(n=n())
gbig_topic

#assigning the topics as node attributes
gbig$topic <- topic$assign
```

```
#visualizing the graph based on topics
V(gbig)[gbig$topic==1]$color <- "red"
V(gbig)[gbig$topic==2]$color <- "blue"
V(gbig)[gbig$topic==3]$color <- "green"
V(gbig)[gbig$topic==4]$color <- "yellow"
plot(gbig)

#saving the edgelist (unweighted)
gbig_edgelist <- get.edgelist(gbig)

if (z == 1) {
  write.dat(gbig_edgelist,"NetworkSimulation/gbig/")
}
if (z == 2) {
  write.dat(gbig_edgelist,"NetworkSimulation/gbig_er/")
}
if (z == 3) {
  write.dat(gbig_edgelist,"NetworkSimulation/sim2/gbig/")
}
if (z == 4) {
  write.dat(gbig_edgelist,"NetworkSimulation/sim2/gbig_er/")
}
if (z == 5) {
  write.dat(gbig_edgelist,"NetworkSimulation/sim3/gbig")
}
if (z == 6) {
  write.dat(gbig_edgelist,"NetworkSimulation/sim3/gbig_er")
}
```

Creating interaction edge weighting for gws

```
#Draw from a Poisson distribution
num_tw <- rpois(n,lambda=10)

#add indicator variable to edge list "same" for whether the two
#nodes connected by an edge have the same topic or not
edge_list <- data.frame(get.edgelist(gws)) %>%
  mutate(same = ifelse(gws$topic[X1] == gws$topic[X2],1,0),
         tw_num = 0) %>%
  arrange(X1,X2)

#compute the number of edges that each node has
edge_nums <- edge_list %>%
  group_by(X1) %>%
  summarise(edge_num = n())

#use same to determine how many tweets (generated from Poisson) are
#assigned to each neighboring node (using multinomial distribution)
for (i in edge_nums$X1) {
  prob <- edge_list %>%
    filter(X1==i) %>%
    select(same)
  edge_list$tw_num[edge_list$X1==i] <-
    as.vector(rmultinom(1,num_tw[i],prob=c(prob$same+1)))
}

#weight number of tweets by total tweets that a node recieves
edge_list <- edge_list %>%
  group_by(X2) %>%
  mutate(end = n()) %>%
  ungroup() %>%
  mutate(weight = tw_num/end)

#filter out edges where the weights are equal to 0
gws_int_edge <- edge_list %>%
  select(X1,X2,weight) %>%
  filter(weight>0) %>%
  mutate(weight = ifelse(weight>1,1,weight))

if (z == 1) {
  write.dat(gws_int_edge,"NetworkSimulation/gws/")
}
if (z == 2) {
  write.dat(gws_int_edge,"NetworkSimulation/gws_er/")
}
if (z == 3) {
  write.dat(gws_int_edge,"NetworkSimulation/sim2/gws/")
}
if (z == 4) {
  write.dat(gws_int_edge,"NetworkSimulation/sim2/gws_er/")
}
if (z == 5) {
  write.dat(gws_int_edge,"NetworkSimulation/sim3/gws")
}
if (z == 6) {
  write.dat(gws_int_edge,"NetworkSimulation/sim3/gws_er")
}
```

```r
#save list of edge weights to look at distribution
dist_gws_int <- gws_int_edge %>%
  select(weight)
```

Creating interaction edge weighting for gbig

```r
#Draw from a Poisson distribution
num_tw <- rpois(n,lambda=10)

#add indicator variable to edge list "same" for whether the two nodes
#connected by an edge have the same topic or not
edge_list <- data.frame(get.edgelist(gbig)) %>%
  mutate(same = ifelse(gbig$topic[X1] == gbig$topic[X2],1,0),
         tw_num = 0) %>%
  arrange(X1,X2)

#compute the number of edges that each node has
edge_nums <- edge_list %>%
  group_by(X1) %>%
  summarise(edge_num = n())

#use same to determine how many tweets (generated from Poisson) are
#assigned to each neighboring node (using multinomial distribution)
for (i in edge_nums$X1) {
  prob <- edge_list %>%
    filter(X1==i) %>%
    select(same)
  edge_list$tw_num[edge_list$X1==i] <-
    as.vector(rmultinom(1,num_tw[i],prob=c(prob$same+1)))
}

#weight number of tweets by total tweets that a node recieves
edge_list <- edge_list %>%
  group_by(X2) %>%
  mutate(end = n()) %>%
  ungroup() %>%
  mutate(weight = tw_num/end)

#filter out edges where the weights are equal to 0
gbig_int_edge <- edge_list %>%
  select(X1,X2,weight) %>%
  filter(weight>0) %>%
  mutate(weight = ifelse(weight>1,1,weight))

if (z == 1) {
  write.dat(gbig_int_edge,"NetworkSimulation/gbig/")
}
if (z == 2) {
  write.dat(gbig_int_edge,"NetworkSimulation/gbig_er/")
}
if (z == 3) {
  write.dat(gbig_int_edge,"NetworkSimulation/sim2/gbig/")
}
if (z == 4) {
  write.dat(gbig_int_edge,"NetworkSimulation/sim2/gbig_er/")
}
if (z == 5) {
  write.dat(gbig_int_edge,"NetworkSimulation/sim3/gbig")
}
if (z == 6) {
  write.dat(gbig_int_edge,"NetworkSimulation/sim3/gbig_er")
}


#save list of edge weights to look at distribution
dist_gbig_int <- gbig_int_edge %>%
  select(weight)
```

Transfering gbig interactions to gws for interaction edge weighting

```
#Draw from a Poisson distribution
num_tw <- rpois(n,lambda=10)

#add indicator variable to edge list "same" for whether the two nodes
#connected by an edge have the same topic or not
edge_list <- data.frame(get.edgelist(gws)) %>%
  mutate(same = ifelse(gbig$topic[X1] == gbig$topic[X2],1,0),
         tw_num = 0) %>%
  arrange(X1,X2)

#compute the number of edges that each node has
edge_nums <- edge_list %>%
  group_by(X1) %>%
  summarise(edge_num = n())

#use same to determine how many tweets (generated from Poisson) are
#assigned to each neighboring node (using multinomial distribution)
for (i in edge_nums$X1) {
  prob <- edge_list %>%
    filter(X1==i) %>%
    select(same)
  edge_list$tw_num[edge_list$X1==i] <-
    as.vector(rmultinom(1,num_tw[i],prob=c(prob$same+1)))
}

#weight number of tweets by total tweets that a node recieves
edge_list <- edge_list %>%
  group_by(X2) %>%
  mutate(end = n()) %>%
  ungroup() %>%
  mutate(weight = tw_num/end)

#filter out edges where the weights are equal to 0
gws_gbigweights_int <- edge_list %>%
  select(X1,X2,weight) %>%
  filter(weight>0) %>%
  mutate(weight = ifelse(weight>1,1,weight))

if (z == 1) {
  write.dat(gws_gbigweights_int,"NetworkSimulation/gws/")
}
if (z == 2) {
  write.dat(gws_gbigweights_int,"NetworkSimulation/gws_er/")
}
if (z == 3) {
  write.dat(gws_gbigweights_int,"NetworkSimulation/sim2/gws/")
}
if (z == 4) {
  write.dat(gws_gbigweights_int,"NetworkSimulation/sim2/gws_er/")
}
if (z == 5) {
  write.dat(gws_gbigweights_int,"NetworkSimulation/sim3/gws")
}
if (z == 6) {
  write.dat(gws_gbigweights_int,"NetworkSimulation/sim3/gws_er")
}

#save list of edge weights to look at distribution
dist_gws_gbigW_int <- gws_gbigweights_int %>%
  select(weight)
```

Transfering gws interactions to gbig for interaction edge weighting

```
#Draw from a Poisson distribution
num_tw <- rpois(n,lambda=10)

#add indicator variable to edge list "same" for whether the two nodes
#connected by an edge have the same topic or not
edge_list <- data.frame(get.edgelist(gbig)) %>%
  mutate(same = ifelse(gws$topic[X1] == gws$topic[X2],1,0),
         tw_num = 0) %>%
  arrange(X1,X2)

#compute the number of edges that each node has
edge_nums <- edge_list %>%
  group_by(X1) %>%
  summarise(edge_num = n())

#use same to determine how many tweets (generated from Poisson) are
#assigned to each neighboring node (using multinomial distribution)
for (i in edge_nums$X1) {
  prob <- edge_list %>%
    filter(X1==i) %>%
    select(same)
  edge_list$tw_num[edge_list$X1==i] <-
    as.vector(rmultinom(1,num_tw[i],prob=c(prob$same+1)))
}

#weight number of tweets by total tweets that a node recieves
edge_list <- edge_list %>%
  group_by(X2) %>%
  mutate(end = n()) %>%
  ungroup() %>%
  mutate(weight = tw_num/end)

#filter out edges where the weights are equal to 0
gbig_gwsweights_int <- edge_list %>%
  select(X1,X2,weight) %>%
  filter(weight>0) %>%
  mutate(weight = ifelse(weight>1,1,weight))

if (z == 1) {
  write.dat(gbig_gwsweights_int,"NetworkSimulation/gbig/")
}
if (z == 2) {
  write.dat(gbig_gwsweights_int,"NetworkSimulation/gbig_er/")
}
```

```r
if (z == 3) {
  write.dat(gbig_gwsweights_int,"NetworkSimulation/sim2/gbig/")
}
if (z == 4) {
  write.dat(gbig_gwsweights_int,"NetworkSimulation/sim2/gbig_er/")
}
if (z == 5) {
  write.dat(gbig_gwsweights_int,"NetworkSimulation/sim3/gbig")
}
if (z == 6) {
  write.dat(gbig_gwsweights_int,"NetworkSimulation/sim3/gbig_er")
}

#save list of edge weights to look at distribution
dist_gbig_gwsW_int <- gbig_gwsweights_int %>%
  select(weight)
```

Creating topic (tag) weighting for gws

```r
#setting number of tags to propogate
n_top <- 20

#choosing n_top number of sources for the tags
source <- sample(1:n,n_top)

#sampling random uniforms to later compare to probabilities
m <- matrix(0,nrow = n,ncol=n_top)
rand_nums <- data.frame(m)
for (i in 1:n_top) {
  rand_nums[1:n,i] <- runif(n,0,1)
}

#computing distances between the source nodes and all other nodes
d <- matrix(0,nrow=n,ncol=n_top)
dists <- data.frame(d)
for (i in 1:n_top) {
  dists[1:n,i] <- bfs(gws,source[i],dist=T)$dist
}

#assigning probabilities for each tag based on distance from the
#source node and a probability for the neighbors of the sources
#to get that tag
probs <- runif(n_top,0.25,0.75)
tags <- matrix(0,nrow=n,ncol=n_top)
tags <- data.frame(tags)
#loop over nodes
for (i in 1:n) {
  #loop over tags
  for (j in 1:n_top) {
    #if randomly generated number is less than the probility to
    #the power of the distance from the source for that tag,
    #assign the tag to that node
    if (rand_nums[i,j] < probs[j]^dists[i,j]) {
      tags[i,j] <- 1
    }
  }
}

#saving the number of nodes with each topic
gws_tag = colSums(tags)

#calculating h(u) (vector of tag frequencies weighted by how often
#they appear in dataset) for each node, where h_i(u) = freq_i(u)log(N/n_i)
#where n_i is number of users that use a specific hashtag
x <- data.frame(matrix(0,nrow=n,ncol=n_top))
for (i in 1:n_top) {
  x[1:n,i] <- rep(log(n/gws_tag[[i]]))
}
h_gws <- tags*x
#creating vector to use in denominator of weight computation
h_sq <- h_gws^2

#function to compute the topic weighting
compute_top_wgt <- function(h,h_sq,node1,node2) {
  num <- sum(h[node1,1:n_top]*h[node2,1:n_top])
  denom <- sqrt(sum(h_sq[node1,1:n_top]))*sqrt(sum(h_sq[node2,1:n_top]))
  return(num/denom)
}

#adding topic information to edge list and weights based on topics
edge_list <- data.frame(get.edgelist(gws)) %>%
  rowwise() %>%
  mutate(weight = compute_top_wgt(h_gws,h_sq,X1,X2))

#filtering out edges with weights of 0
gws_tag_edge <- edge_list %>%
  select(X1,X2,weight) %>%
  filter(weight>0)
```

```
if (z == 1) {
  write.dat(gws_tag_edge,"NetworkSimulation/gws")
}
if (z == 2) {
  write.dat(gws_tag_edge,"NetworkSimulation/gws_er")
}
if (z == 3) {
  write.dat(gws_tag_edge,"NetworkSimulation/sim2/gws")
}
if (z == 4) {
  write.dat(gws_tag_edge,"NetworkSimulation/sim2/gws_er")
}
if (z == 5) {
  write.dat(gws_tag_edge,"NetworkSimulation/sim3/gws")
}
if (z == 6) {
  write.dat(gws_tag_edge,"NetworkSimulation/sim3/gws_er")
}

#save list of edge weights to look at distribution
dist_gws_tag <- gws_tag_edge %>%
  select(weight)
```

Creating topic (tag) weighting for gbig

```
#computing distances between the source nodes and all other nodes
d <- matrix(0,nrow=n,ncol=n_top)
dists <- data.frame(d)
for (i in 1:n_top) {
  dists[1:n,i] <- bfs(gbig,source[i],dist=T)$dist
}

#assigning probabilities for each tag based on distance from the
#source node and a probability for the neighbors of the sources
#to get that tag
probs <- runif(n_top,0.25,0.75)
tags <- matrix(0,nrow=n,ncol=n_top)
tags <- data.frame(tags)
#loop over nodes
for (i in 1:n) {
  #loop over tags
  for (j in 1:n_top) {
    #if randomly generated number is less than the probility to the
    #power of the distance from the source for that tag, assign the
    #tag to that node
    if (rand_nums[i,j] < probs[j]^dists[i,j]) {
      tags[i,j] <- 1
    }
  }
}

#saving the number of nodes with each topic
gbig_tag = colSums(tags)

#calculating h(u) (vector of tag frequencies weighted by how often
#they appear in dataset) for each node, where h_i(u) = freq_i(u)log(N/n_i)
#where n_i is number of users that use a specific hashtag
x <- data.frame(matrix(0,nrow=n,ncol=n_top))
for (i in 1:n_top) {
  x[1:n,i] <- rep(log(n/gbig_tag[[i]]))
}
h_gbig <- tags*x
#creating vector to use in denominator of weight computation
h_sq <- h_gbig^2

#adding topic information to edge list and weights based on topics
edge_list <- data.frame(get.edgelist(gbig)) %>%
  rowwise() %>%
  mutate(weight = compute_top_wgt(h_gbig,h_sq,X1,X2))

#filtering out edges with weights of 0
gbig_tag_edge <- edge_list %>%
  select(X1,X2,weight) %>%
  filter(weight>0)

if (z == 1) {
  write.dat(gbig_tag_edge,"NetworkSimulation/gbig")
}
if (z == 2) {
  write.dat(gbig_tag_edge,"NetworkSimulation/gbig_er")
}
if (z == 3) {
  write.dat(gbig_tag_edge,"NetworkSimulation/sim2/gbig")
}
if (z == 4) {
  write.dat(gbig_tag_edge,"NetworkSimulation/sim2/gbig_er")
}
if (z == 5) {
  write.dat(gbig_tag_edge,"NetworkSimulation/sim3/gbig")
}
if (z == 6) {
  write.dat(gbig_tag_edge,"NetworkSimulation/sim3/gbig_er")
}

#save list of edge weights to look at distribution
dist_gbig_tag <- gbig_tag_edge %>%
  select(weight)
```

Transfering gbig topics to gws for topic (tag) edge weighting

```
#creating vector to use in denominator of weight computation
h_sq <- h_gbig^2

#adding topic information to edge list and weights based on topics
edge_list <- data.frame(get.edgelist(gws)) %>%
  rowwise() %>%
  mutate(weight = compute_top_wgt(h_gbig,h_sq,X1,X2))

#filtering out edges with weights of 0
gws_gbigweights_tag <- edge_list %>%
  select(X1,X2,weight) %>%
  filter(weight>0)

if (z == 1) {
  write.dat(gws_gbigweights_tag,"NetworkSimulation/gws")
}
if (z == 2) {
  write.dat(gws_gbigweights_tag,"NetworkSimulation/gws_er")
}
if (z == 3) {
  write.dat(gws_gbigweights_tag,"NetworkSimulation/sim2/gws")
}
if (z == 4) {
  write.dat(gws_gbigweights_tag,"NetworkSimulation/sim2/gws_er")
}
if (z == 5) {
  write.dat(gws_gbigweights_tag,"NetworkSimulation/sim3/gws")
}
if (z == 6) {
  write.dat(gws_gbigweights_tag,"NetworkSimulation/sim3/gws_er")
}
```

```
#save list of edge weights to look at distribution
dist_gws_gbigW_tag <- gws_gbigweights_tag %>%
  select(weight)
```

Transfering gws topics to gbig for topic (tag) edge weighting

```
#creating vector to use in denominator of weight computation
h_sq <- h_gws^2

#adding topic information to edge list and weights based on topics
edge_list <- data.frame(get.edgelist(gbig)) %>%
  rowwise() %>%
  mutate(weight = compute_top_wgt(h_gws,h_sq,X1,X2))

#filtering out edges with weights of 0
gbig_gwsweights_tag <- edge_list %>%
  select(X1,X2,weight) %>%
  filter(weight>0)

if (z == 1) {
  write.dat(gbig_gwsweights_tag,"NetworkSimulation/gbig")
}
if (z == 2) {
  write.dat(gbig_gwsweights_tag,"NetworkSimulation/gbig_er")
}
if (z == 3) {
  write.dat(gbig_gwsweights_tag,"NetworkSimulation/sim2/gbig")
}
if (z == 4) {
  write.dat(gbig_gwsweights_tag,"NetworkSimulation/sim2/gbig_er")
}
if (z == 5) {
  write.dat(gbig_gwsweights_tag,"NetworkSimulation/sim3/gbig")
}
if (z == 6) {
  write.dat(gbig_gwsweights_tag,"NetworkSimulation/sim3/gbig_er")
}

#save list of edge weights to look at distribution
dist_gbig_gwsW_tag <- gbig_gwsweights_tag %>%
  select(weight)
```

Once all of the edge lists are saved, OSLOM is run using the OSLOM software downloaded and saved in the folder OSLOM2. Scripts are written (two for each simulation, one for the Watts-Strogatz graphs, and one for the Erdős-Rényi graphs), and saved as 'WS_sim1', 'ER_sim2' and etc. in the OSLOM2 folder.

Comparing edge weight distributions. The code that actually creates the histograms shown in the text are generated from creating ten different graphs and aggregating their edgeweights, but the code is similar and can be seen in the file 'edgewt_histogram.Rmd' in the Github repository.

```
cols <- c("c"="red","a"="blue","b"="green","d"="purple")
if (z==1 | z==3 | z==5) {
  ggplot(dist_gws_int,aes(x=weight,color="c")) +
    geom_density() +
    geom_density(data=dist_gbig_int,aes(x=weight,color="a")) +
    geom_density(data=dist_gbig_gwsW_int,aes(x=weight,color="b")) +
    geom_density(data=dist_gws_gbigW_int,aes(x=weight,color="d")) +
    labs(title="Interaction Weights for Watts Strogatz") +
    scale_colour_manual(name="Data Source",values=cols)
}
if (z==1 | z==3 | z==5) {
  ggplot(dist_gws_tag,aes(x=weight,color="c")) +
    geom_density() +
    geom_density(data=dist_gbig_tag,aes(x=weight,color="a")) +
    geom_density(data=dist_gbig_gwsW_tag,aes(x=weight,color="b")) +
    geom_density(data=dist_gws_gbigW_tag,aes(x=weight,color="d")) +
    labs(title="Topic Weights for Watts Strogatz") +
    scale_colour_manual(name="Data Source",values=cols)
}
if (z==2 | z==4 | z==6) {
  ggplot(dist_gws_int,aes(x=weight,color="c")) +
    geom_density() +
    geom_density(data=dist_gbig_int,aes(x=weight,color="a")) +
    geom_density(data=dist_gbig_gwsW_int,aes(x=weight,color="b")) +
    geom_density(data=dist_gws_gbigW_int,aes(x=weight,color="d")) +
    labs(title="Interaction Weights for Erdos Renyi") +
    scale_colour_manual(name="Data Source",values=cols)
}
if (z==2 | z==4 | z==6) {
  ggplot(dist_gws_tag,aes(x=weight,color="c")) +
    geom_density() +
    geom_density(data=dist_gbig_tag,aes(x=weight,color="a")) +
    geom_density(data=dist_gbig_gwsW_tag,aes(x=weight,color="b")) +
    geom_density(data=dist_gws_gbigW_tag,aes(x=weight,color="d")) +
    labs(title="Topic Weights for Erdos Renyi") +
    scale_colour_manual(name="Data Source",values=cols)
}
```

The next chunk comes from the file ‘network_stats.Rmd’ and calculate the network statistics for each simulation. The code is included for the first two simulations, and it is similar for the third, except that the graphs are generated to overlap.

Generating network statistics for simulations 1 and 2 (n_ed = 5 for sim 1, n_ed = 3 for sim 2):

```
n <- 50
node_num <- numeric(4)
edge_num <- numeric(4)
diam <- numeric(4)
avg_path <- numeric(4)
trans <- numeric(4)
l_trans <- numeric(4)
#n_ed <- 5 #use for network statistics for simulation 1
n_ed <- 3 #use for network statistics for simulation 2
for (i in 1:n) {
  #Creating individual graphs
  g.ws1 <- watts.strogatz.game(1,25,n_ed,0.05)
  g.ws2 <-watts.strogatz.game(1,25,n_ed,0.05)
  g.ws3 <-watts.strogatz.game(1,25,n_ed,0.05)
  g.ws4 <- watts.strogatz.game(1,25,n_ed,0.05)
  gws <- g.ws1 + g.ws2 + g.ws3 + g.ws4
  gws <- as.directed(gws)
  x <- 50
  #linking the graphs together randomly
  gws_ws <- add_edges(gws,sample(c(seq(1,100)),x,replace=T))

  #Creating single larger graph
  gbig_ws <- as.directed(watts.strogatz.game(1,100,n_ed,0.05))

  #Creating individual graphs
  g.ws1 <- erdos.renyi.game(n=25,p.or.m=n_ed*50,type="gnm",directed=T)
  g.ws2 <-erdos.renyi.game(n=25,p.or.m=n_ed*50,type="gnm",directed=T)
  g.ws3 <-erdos.renyi.game(n=25,p.or.m=n_ed*50,type="gnm",directed=T)
  g.ws4 <- erdos.renyi.game(n=25,p.or.m=n_ed*50,type="gnm",directed=T)
  gws <- g.ws1 + g.ws2 + g.ws3 + g.ws4
  x <- 50
  #linking the graphs together randomly
  gws_er <- add_edges(gws,sample(c(seq(1,100)),x,replace=T))

  #Creating single larger graph
  gbig_er <- erdos.renyi.game(n=100,p.or.m=n_ed*200,type="gnm",directed=T)

  node_num[2] <- node_num[2]+vcount(gws_ws)
  node_num[1] <- node_num[1]+vcount(gbig_ws)
  node_num[4] <- node_num[4]+vcount(gws_er)
  node_num[3] <- node_num[3]+vcount(gbig_er)

  edge_num[2] <- edge_num[2]+ecount(gws_ws)
  edge_num[1] <- edge_num[1]+ecount(gbig_ws)
  edge_num[4] <- edge_num[4]+ecount(gws_er)
  edge_num[3] <- edge_num[3]+ecount(gbig_er)

  diam[2] <- diam[2]+diameter(gws_ws,directed=T)
  diam[1] <- diam[1]+diameter(gbig_ws,directed=T)
  diam[4] <- diam[4]+diameter(gws_er,directed=T)
  diam[3] <- diam[3]+diameter(gbig_er,directed=T)

  avg_path[2] <- avg_path[2]+mean_distance(gws_ws,directed=T)
  avg_path[1] <- avg_path[1]+mean_distance(gbig_ws,directed=T)
  avg_path[4] <- avg_path[4]+mean_distance(gws_er,directed=T)
  avg_path[3] <- avg_path[3]+mean_distance(gbig_er,directed=T)

  trans[2] <- trans[2]+transitivity(gws_ws,type="global")
  trans[1] <- trans[1]+transitivity(gbig_ws,type="global")
  trans[4] <- trans[4]+transitivity(gws_er,type="global")
  trans[3] <- trans[3]+transitivity(gbig_er,type="global")

  l_trans[2] <- l_trans[2]+transitivity(gws_ws,type="local")
  l_trans[1] <- l_trans[1]+transitivity(gbig_ws,type="local")
  l_trans[4] <- l_trans[4]+transitivity(gws_er,type="local")
  l_trans[3] <- l_trans[3]+transitivity(gbig_er,type="local")
}
node_num <- node_num/n
edge_num <- edge_num/n
diam <- diam/n
avg_path <- avg_path/n
trans <- trans/n
l_trans <- l_trans/n
```

The next few chunks are from the file ‘plotOSLOM.Rmd’ and plot the results from OSLOM, specifically for the structural weighting for the Erdős-Rényi four community graph from simulation 3.

Writing a function to read in the results from OSLOM

```
read_tp <- function(path,num) {
  #number of nodes
  n=num
  x <- readtext(path)
  s <- x$text
  #split lines
  s1 <- strsplit(s,"\n")
  #only save the lines that describe clusters
  s2 <-s1[[1]][seq(2,length(s1[[1]]),2)]
  #make matrix to save cluster membership in
  data <- data.frame(matrix(0,ncol=length(s2),nrow=n))
  for (i in 1:length(s2)) {
    lst <- as.numeric(unlist(str_extract_all(s2[i], "[0-9]+")))
    for (j in 1:length(lst)) {
      data[lst[j],i] = 1
    }
  }
  return(data)
}
```

Plotting 1st hierarchical level

```
t <- read_tp("NetworkSimulation/sim3/gws_er/
             gws_edgelist.dat_oslo_files/tp",vcount(gws))
for (i in 1:length(t)) {
  name <- paste0("cl",i)
  x <- which(t[i]==1)
  assign(name,x)
}
t <- t %>%
  mutate(sum = rowSums(.))
ovl <- which(t$sum>1)

V(gws)[cl1]$color <- "red"
V(gws)[cl2]$color <- "blue"
V(gws)[cl3]$color <- "green"
V(gws)[cl4]$color <- "yellow"
V(gws)[ovl]$color <- "black"

plot(gws)
```

Plotting 2nd hierarchical level

```
t <- read_tp("NetworkSimulation/sim3/gws_er/
             gws_edgelist.dat_oslo_files/tp1",vcount(gws))
for (i in 1:length(t)) {
  name <- paste0("cl",i)
  x <- which(t[i]==1)
  assign(name,x)
}
t <- t %>%
  mutate(sum = rowSums(.))
ovl <- which(t$sum>1)

V(gws)[cl1]$color <- "red"
V(gws)[cl2]$color <- "blue"
V(gws)[ovl]$color <- "black"

plot(gws)
```

The last code chunks in this appendix are from the file ‘OtherCDAlgs.Rmd’, and run and visualize other community detection methods implemented in igraph on the structural weighting for the Erdős-Rényi four community graph from simulation 3.

Loading in the simulation 3 Erdős-Rényi graphs.

```
source("NetworkSimulation/simulation_generation/overlapping_graphs_er_sm.R")
```

Walktrap and spinglass work for directed graphs, edge betweenness, infomap, label propogation, and leading eigenvector work by using the underlying undirected graph, fast and greedy and louvain don’t run for directed graphs, and optimal takes too long to run.

Running walktrap (works with directed graph)

```
walk4 <- cluster_walktrap(gws) #visualized below
walk1 <- cluster_walktrap(gbig) #several communities detected
```

Visualizing walktrap results for four communities

```
V(gws)[unlist(walk4[1])]$color <- "red"
V(gws)[unlist(walk4[2])]$color <- "blue"
V(gws)[unlist(walk4[3])]$color <- "green"
V(gws)[unlist(walk4[4])]$color <- "yellow"

plot(gws)
```

Running infomap (uses undirected graph)

```
info4 <- cluster_infomap(gws) #visualized below
info1 <- cluster_infomap(gbig) #no communities detected
```

Visualizing infomap results for four communities

```r
V(gws)[unlist(info4[1])]$color <- "red"
V(gws)[unlist(info4[2])]$color <- "blue"
V(gws)[unlist(info4[3])]$color <- "green"

plot(gws)
```

Running edge-betweenness (uses undirected graph)

```r
edge4 <- cluster_edge_betweenness(gws)
#finds two larger communities, visualized below
edge1 <- cluster_edge_betweenness(gbig)
#mostly puts nodes in their own communities
#combines 6-11 nodes into 3 small communities
```

Visualizing edge-betweenness results for four communities

```r
V(gws)[unlist(edge4[1])]$color <- "red"
V(gws)[unlist(edge4[2])]$color <- "blue"

plot(gws)
```

Running leading eigenvector (uses undirected graph)

```r
lead4 <- cluster_leading_eigen(gws)
#finds four communities, visualized below
lead1 <- cluster_leading_eigen(gbig)
#finds 5 communities
```

Visualizing leading eigenvector results for four communities

```r
V(gws)[unlist(lead4[1])]$color <- "red"
V(gws)[unlist(lead4[2])]$color <- "blue"
V(gws)[unlist(lead4[3])]$color <- "green"
V(gws)[unlist(lead4[4])]$color <- "yellow"

plot(gws)
```

Running label propogation (uses undirected graph)

```r
lab4 <- cluster_label_prop(gws)
#finds 2 communities, visualized below
lab1 <- cluster_label_prop(gbig)
#finds 1 community
```

Visualizing label propogation results for four communities

```
V(gws)[unlist(lab4[1])]$color <- "red"
V(gws)[unlist(lab4[2])]$color <- "blue"

plot(gws)
```

Running spinglass

```
spin4 <- cluster_spinglass(gws)
#finds 4 communities, visualized below
spin1 <- cluster_spinglass(gbig)
#finds 6 communities
```

Visualizing spinclass results for four communities

```
V(gws)[unlist(spin4[1])]$color <- "red"
V(gws)[unlist(spin4[2])]$color <- "blue"
V(gws)[unlist(spin4[3])]$color <- "green"
V(gws)[unlist(spin4[4])]$color <- "yellow"

plot(gws)
```

Appendix B

SNAP Data Appendix

This appendix contains the code used to load, analyze, and visualize the SNAP data. It contains code stored in the ‘SNAP_network’ folder. The code is saved in the files ‘SNAP_data_to_network.Rmd’, ‘top_weight_histograms.Rmd’, and ‘SNAP_plotOSLOM.Rmd’.

This chunk loads the packages used in the SNAP data analysis.

```
require(mosaic)
require(tidyverse)
require(igraph)
require(sand)
require(multiplex)
require(readtext)
require(stringr)
```

The first code chunks are from the file ‘SNAP_data_to_network.Rmd’.

Set variable c to determine which circle to look at (c=1 for circle 100318079, c=2 for circle 102903198, and c=3 for circle 105150583)

```
c <- 3
```

Loading edge list into an igraph object

```
#putting original edge list into a dataframe
if (c==1) {
  ego <- as.numeric(100318079)
}
if (c==2) {
  ego <- as.numeric(102903198)
}
if (c==3) {
  ego <- as.numeric(105150583)
}
```

```
#circles to use - 100318079 (221), 102903198 (227), 105150583 (211)
ego_char <- as.character(ego)
y <- readtext(paste0("SNAP_network/twitter/",ego_char,".edges"))
t <- y$text
text_df <- as.data.frame(strsplit(t,"\n"))
names(text_df) <- "edges"
edges_df <- as.data.frame(str_split_fixed(text_df$edges, " ", 2))
names(edges_df) <- c("V1","V2")

net <- graph_from_data_frame(edges_df,directed=T)
plot(net)
plot(net, vertex.size=5, vertex.label=NA, edge.arrow.size=0 )
```

Calculating network statistics

```
node_num <- vcount(net)
edge_num <- ecount(net)
diam <- diameter(net,directed=T)
avg_path <- mean_distance(net,directed=T)
trans <- transitivity(net,type="global")
l_trans <- transitivity(net,type="local")
```

Saving the structural edge list

```
#saving the edgelist
edges <- as.data.frame(get.edgelist(net))
write.dat(edges,paste0("SNAP_network/structural/",ego_char,"/"))
```

Loading feature names

```
y <- readtext(paste0("SNAP_network/twitter/",ego_char,".featnames"))
t <- y$text
text_df <- as.data.frame(strsplit(t,"\n"))
names(text_df) <- "edges"
featnames_df <- as.data.frame(str_split_fixed(text_df$edges, " ", 2))
names(featnames_df) <- c("index","feat")
```

Loading ego features

```
y <- readtext(paste0("SNAP_network/twitter/",ego_char,".egofeat"))
ego_feat <- as.data.frame(y$text)
names(ego_feat) <- "vals"
ego_feat <- as.data.frame(str_split_fixed(ego_feat$vals, " ", nrow(featnames_df)))
ego_feat <- cbind(data.frame(ego),ego_feat)
names(ego_feat)[1] <- "node"
```

Loading other nodes' features

```r
y <- readtext(paste0("SNAP_network/twitter/",ego_char,".feat"))
t <- y$text
node_feat <- as.data.frame(strsplit(t,"\n"))
names(node_feat) <- "vals"
node_feat <- as.data.frame(str_split_fixed(
  node_feat$vals, " ", nrow(featnames_df)+1))
feat_map <- node_feat %>%
  select(-V1)
feat_map <- data.frame(
  lapply(feat_map, function(x) as.numeric(as.character(x))))
```

Calculating topic weights

```r
nr <- nrow(feat_map)
nc <- ncol(feat_map)
x <- data.frame(matrix(0,nrow=nr,ncol=nc))
tag_sums = colSums(feat_map)
for (i in 1:nc) {
  x[1:nr,i] <- rep(log(nr/tag_sums[[i]]))
}
h <- feat_map*x
#creating vector to use in denominator of weight computation
h_sq <- h^2

#function to compute the topic weighting
compute_top_wgt <- function(h,h_sq,node1,node2) {
  num <- sum(h[node1,1:nc]*h[node2,1:nc])
  denom <- sqrt(sum(h_sq[node1,1:nc]))*sqrt(sum(h_sq[node2,1:nc]))
  return(num/denom)
}

#adding topic information to edge list and weights based on topics
edge_list <- edges %>%
  rowwise() %>%
  mutate(weight = compute_top_wgt(h,h_sq,V1,V2))

#filtering out edges with weights of 0
edges_top <- edge_list %>%
  select(V1,V2,weight) %>%
  filter(weight>0)
write.dat(edges_top,paste0("SNAP_network/topic/",ego_char,"/"))
```

To run OSLOM, the following code is used, where 'ego_char' is replaced by the ID number for the circle's ego node:

./oslom_dir -f ../../../git/Teichman-Thesis/SNAP_network/structural/ego_char/edges.dat -uw
./pajek_write_dir ../../../git/Teichman-Thesis/SNAP_network/structural/ego_char/edges.dat
./oslom_dir -f ../../../git/Teichman-Thesis/SNAP_network/topic/ego_char/edges_top.dat -w
./pajek_write_dir ../../../git/Teichman-Thesis/SNAP_network/topic/ego_char/edges_top.dat
Saving topic weights

```r
if (c==1) {
  saveRDS(edges_top,file="SNAP_network/top_weights_100.rds")
}
if (c==2) {
  saveRDS(edges_top,file="SNAP_network/top_weights_102.rds")
}
if (c==3) {
  saveRDS(edges_top,file="SNAP_network/top_weights_105.rds")
}
```

Testing different community detection algorithms

```r
c1 <- cluster_walktrap(net)
c2 <- cluster_edge_betweenness(net)
c3 <- cluster_leading_eigen(net)
```

The next two chunks are from the file ‘top_weight_histograms.Rmd’ and create the histogram of topic weights for the circles.
Loading topic weights that were saved in `SNAP_data_to_network.Rmd`

```r
top_100 <- readRDS("SNAP_network/top_weights_100.rds")
top_102 <- readRDS("SNAP_network/top_weights_102.rds")
top_105 <- readRDS("SNAP_network/top_weights_105.rds")
```

Creating a histogram for the three weight distributions

```r
cols <- c("100318079"="red","102903198"="blue","105150583"="green")
n <- nrow(top_100)
ggplot(top_100,aes(x=weight,color="100318079")) +
  geom_density() +
  geom_density(data=top_102,aes(x=weight,color="102903198",y=..density..)) +
  geom_density(data=top_105,aes(x=weight,color="105150583",y=..density..)) +
  labs(title="Topic Weights for Three Circles") +
  scale_colour_manual(name="Circle Number",values=cols)
```

The next few code chunks are from the file ‘SNAP_plotOSLOM.Rmd’ and visualize the different hierarchical levels of the OSLOM outputs. These images are not used in the text, because images created with the software from Pajek are used, but these can easily show which nodes are placed within overlapping communities.

Writing the function to read in the OSLOM output and color the nodes based on community detection results

```
read_tp <- function(path,num,node_names) {
  #number of nodes
  n <- num
  names <- node_names
  x <- readtext(path)
  s <- x$text
  #split lines
  s1 <- strsplit(s,"\n")
  #only save the lines that describe clusters
  s2 <-s1[[1]][seq(2,length(s1[[1]]),2)]
  #make matrix to save cluster membership in
  data <- data.frame(matrix(0,ncol=length(s2)+1,nrow=n))
  data[,1] <- names
  for (i in 1:length(s2)) {
    lst <- as.numeric(unlist(str_extract_all(s2[i], "[0-9]+")))
    for (j in 1:length(lst)) {
      row <- which(data$X1==lst[j])
      data[row,i+1] = 1
    }
  }
  t <- data
  t1 <- t %>%
    select(-X1) %>%
    mutate(sum = rowSums(.))
  ovl <- which(t1$sum>1)

  t <- t %>%
    mutate(sum=t1$sum,
           color="white")
  colors <- c("red","blue","green","yellow","purple",
              "cyan","tan2","darkgreen","lightblue","pink")

  n_cl <- length(t1)-1
  for (i in 1:n_cl) {
    print(i)
    x <- which(t1[i]==1)
    print(x)
    t$color[x] <- colors[i]
  }
  t$color[ovl] <- "black"
```

```r
  return(t)
}
```

Loading information for circle 1

```r
ego_char <- as.character(100318079)
y <- readtext(paste0("SNAP_network/twitter/",ego_char,".edges"))
t <- y$text
text_df <- as.data.frame(strsplit(t,"\n"))
names(text_df) <- "edges"
edges_df <- as.data.frame(str_split_fixed(text_df$edges, " ", 2))
names(edges_df) <- c("V1","V2")
net <- graph_from_data_frame(edges_df,directed=T)
names <- V(net)$name
```

Plotting circle 1 tp

```r
path <- "SNAP_network/structural/100318079/edges.dat_oslo_files/tp"
n=vcount(net)

t <- read_tp(path,n,names)
V(net)$color <- t$color
plot(net, vertex.size=5, vertex.label=NA, edge.arrow.size=0 )
```

Plotting circle 1 tp1

```r
path <- "SNAP_network/structural/100318079/edges.dat_oslo_files/tp1"
n=vcount(net)

t <- read_tp(path,n,names)
V(net)$color <- t$color
plot(net, vertex.size=5, vertex.label=NA, edge.arrow.size=0 )
```

Plotting circle 1 tp topic edge

```r
path <- "SNAP_network/topic/100318079/edges_top.dat_oslo_files/tp"
n=vcount(net)

t <- read_tp(path,n,names)
V(net)$color <- t$color
plot(net, vertex.size=5, vertex.label=NA, edge.arrow.size=0 )
```

The same process can be done for circles 2 and 3 by changing the ego node ID within the file path.

Corrections

When originally submitted, this thesis contained some errors that have been corrected in this version. A list of corrections follows.

Title page. Changed spacing.

p. 2, l. 26 The word “between” was added.

p. 7, l. 19 The phrase “that is” was added.

p. 10, l. 24 The phrase “with our exposition following that of the paper” was added.

p. 13 The subheading “Edge Weighting Based on Community Detection” was changed to “Detecting Communities Using Different Edge Weightings.”

p. 13, l. 27 The word “parts” was replaced by the word “members.”

p. 23, l. 8 The word “large” was replaced by the word “larger.”

p. 23, l. 15 The word “of” was removed.

p. 37, l. 6 Removed a comma and added the word “foundation.”

p. 71, l. 14 Fixed a typo in the bibliography.

p. 71, l. 20 Capitalized the letter “R.”

References

Barabasi, A.-L. (2016). *Network science*. Cambridge University Press.

Batagelj, V., & Mrvar, A. (1999). Pajek - program for large network analysis. print.

Csardi, G., & Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal*, *Complex Systems*, 1695. Retrieved from `http://igraph.org`

Darmon, D., Omodei, E., & Garland, J. (2015). Followers are not enough: A multifaceted approach to community detection in online social networks. *PLOS ONE*, *10*(8).

Fortunato, S. (2009). Community detection in graphs. *Physics Reports*, *75*(174), 75–174.

Fortunato, S., & Hric, D. (2016). Community detection in networks: A user guide. *Physics Reports*, *659*, 1–44.

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

Heckathorn, D. D., & Cameron, C. J. (2017). Network sampling: From snowball and multiplicity to respondent-driven sampling. *Annual Review of Sociology*, *43*, 101–19.

Kolaczyk. (2009). *Statistical analysis of network data: Methods and models*. New York, NY: Springer.

Kolaczyk. (2014). *Statistical analysis of network data with R*. New York, NY: Springer.

Lancichinetti, A., Fortunato, S., & Kertesz, J. (2009). Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, *11*.

Lancichinetti, A., Radicchi, F., Ramasco, J. J., & Fortunato, S. (2011). Finding statistically significant communities in networks. *PLOS ONE*, *6*(4).

Leskovec, J., & Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

McAuley, J., & Leskovec, J. (2012). Learning to discover social circles in ego networks.

NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, 539–547.

Newman, M. (2003). Fast algorithm for detecting community structure in networks. *Physical Review E*, *69*(6).

Newman, M. (2010). *Networks: An introduction.* New York, NY: Oxford.

Pons, P., & Latapy, M. (2006). Computing communities in large networks using random walks. *Journal of Graph Algorithms and Applications*, *10*(2), 191–218.

Watts, D. J., & Stogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, *393*, 440–442.