A NETWORK ANALYSIS OF COVID-19 IN THE UNITED STATES

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Mathematics with Specialization in Applied Mathematics

by

Joseph McGuire

June 2022

COMMITTEE MEMBERSHIP

TITLE:	A Network Analysis of COVID-19 in the United States
AUTHOR:	Joseph McGuire
DATE SUBMITTED:	June 2022
COMMITTEE CHAIR:	Elena Dimitrova, Ph.D. Associate Professor of Mathematics
COMMITTEE MEMBER:	Charles David Camp, Ph.D. Associate Professor of Mathematics
COMMITTEE MEMBER:	Joyce Lin, Ph.D. Professor of Mathematics

ABSTRACT

A Network Analysis of COVID-19 in the United States

Joseph McGuire

Through methods in network theory and time-series analysis, we will analyze the spread of COVID-19 in the United States by determining trends in state-by-state daily cases through a network construction. Previous researchers have found frameworks for approximating the spread of the COVID-19 pandemic and identifying potential rises in cases by a network construction based on correlation of cases between regions [1]. Applying this network construction we determine how this network and its structure act as a predictor for overall COVID-19 cases in the United States by preforming a trend analysis on a variety of network statistics and US COVID-19 cases.

ACKNOWLEDGMENTS

I would like to thank my parents, Patrick and Tami, and my sister Courtney for always being there when I needed them and for being an awesome and supporting family, you all made it possible for me to get this far.

Of course, thank you to Elena Dimitrova for being an amazing mentor and advisor through this whole process, keeping my train of thought on course and making sure I don't get too far off track. Thank you to Dave Camp and Joyce Lin for also being amazing mentors and being great to work with throughout my time at Cal Poly.

Thank you to all the incredible folks in my cohort, you all were some of the most supportive and kind folks I've ever met. I'm genuinely honored to have gotten to know you and can't wait to see all the wild places y'all will end up.

With love to my amazing partner Leah, for all of her incredible support and graphical expertise. My life is so much better with you. You are so incredible and, by far the smartest, most talented person I know. And of course to my constant work companion, Zeno, you're the bestest cat.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

SARS-CoV-2 is a coronavirus variant that was first detected in a small group of patients in Wuhan, China in December of 2019, this virus being responsible for the COVID-19 disease. The subsequent national and international spread of COVID-19 has lead to the first world-wide pandemic in the 21st-century and upheaval of many social norms and practices held prior to the pandemic. At the time of writing, May 2022, the number of COVID-19 cases reported world-wide has hit 525 million with the death count reaching 6.2 million, but even with over 213 million vaccines distributed the COVID-19 virus continues to spread in complex geographical patterns around the world [3]. This has only increased with the relaxation of travel restrictions and the re-opening of borders.

The focus of this thesis is analyzing COVID-19 through the lens of networks in order to discern trends between jurisdictions that are revealed by a network construction based on COVID-19 cases in the United States. Promising results have been found that suggest such a network construction may be a good indicators and visualizations of COVID-19's geospatial spread through the course of the pandemic [1, 4]. In this thesis we conduct a thorough analysis of this network model whose network statistics may be a good indicator of the rise in COVID-19 cases, analyze the temporal evolution of the community structure resulting from this construction, and propose a network informed regression model based on this dynamical network. The python libraries pandas [5], seaborn [6], and altair [7] were used to generate all the plots seen in this thesis; for time-series analysis statsmodels [8] and scikit-learn [9] were used extensively; for network generation and analysis - networkx [10]; and for community detection - cdlib was used [11].

The outline of this thesis is as follows: Chapter 2 will act as an introduction to network theory, Chapter 3 will be an introduction to community detection methods, Chapter 4 will be an introduction to time-series analysis and time-series models, Chapter 5 will discuss the processing of the data for the project, Chapter 6 will discuss the network construction, cover the centrality analysis preformed on this network for prediction of COVID-19 in the US, a network informed model for COVID-19 in the US, as well community detection applied to this network, and Chapter 7 will conclude.

Chapter 2

NETWORK THEORY

2.1 Graph Theory Foundations

A graph, or network, gives us a natural way of understanding relationships between objects in sets. A graph relates a set of objects back to themselves via some relation, that then defines the edges of the network.

Definition 2.1. *Let V be a set and E be a set, where $E \subset \{\{v, w\} : v, w \in V\}$. We say that $G = (V, E)$ is an* ***un-directed simple network*** *on the vertex (node) set V with an edge (connection) set E.*

Equivalently, we can also define $E \subset V \times V$ as a relation on V, where $(u, v) \in E$ if and only if uEv with an additional requirement of symmetry:

$$\textit{If } (u, v) \in E, \textit{ then } (v, u) \in E.$$

The first definition of an edge set E is notationally convenient, while the second definition will allow for a simpler generalization of the concept to a non-symmetric case.

In Figure 2.1 is a un-directed simple graph with node set $V = \{0, 1, 2\}$ and edge set $E = \{\{0, 1\}, \{1, 2\}, \{0, 2\}\}$.

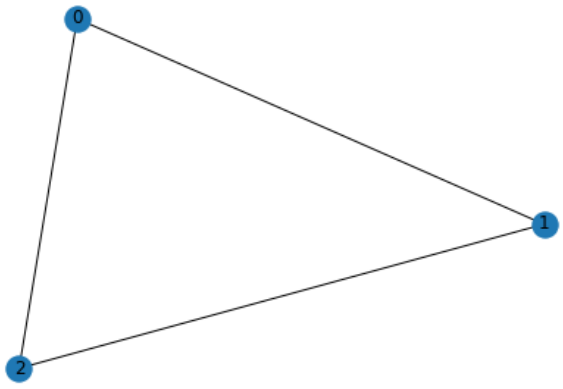

Figure 2.1: An undirected simple graph.

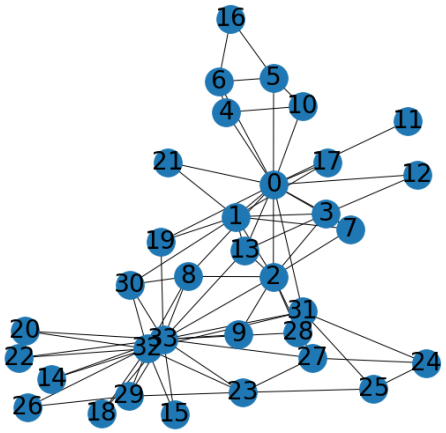

Figure 2.2: The Zachary Karate Club Network.

There are a number of different constraints and modifications that we can put on graphs, in particular, on the edge set E that will make sense for a number of different scenarios. However simple graphs are useful in describing a variety of situations, because of their links to symmetric relations. We'll see in the next example that one of the best studied graphs in network theory is a simple graph.

Example 2.2. *A PhD student, Wayne Zachary, followed the interactions of the members of a Karate club over the course of 3 years (1970 - 1972) [12]. Over this span, there was a conflict in the club, resulting in two administrators 'Mr. Hi' and 'John A' (pseudonyms) that split the group into two rival karate clubs (think 'Karate Kid'). The graph is created by treating each individual as a node, and then every edge is a social connection between two members of the club. The result being the graph in Figure 2.2.*

An informative feature of a network is the total number of connections a single node has. In Example 2.2, the number of connections that a node has tells us how connected that the node is, and thus how connected the person that node is representing. Formally, this is the notion of the degree of a node and, as with most topics in mathematics, there is a natural and elegant link from this to linear algebra.

Definition 2.3. *Let $G = (V, E)$ be a graph. For $v \in V$,*

$$\deg(v) = \#\ \textit{of edges connected to } v$$

is the ***degree*** *of node v.*

Definition 2.4. *Let $G = (V, E)$ be a graph with $|V| = n$. Define the* ***degree sequence*** *to be the decreasing sequence*

$$(\deg(v_1), \deg(v_2), \ldots, \deg(v_n)) \qquad \textit{with } \deg(v_i) \leq \deg(v_{i+1})\ \forall i.$$

The labeling of $v_1, \ldots, v_n$ maybe done arbitrarily to fit the inequality requirement above.

Example 2.5. *In Figure 2.1,*

$$\deg(0) = \deg(1) = \deg(2) = 2.$$

In the Zachary Karate Club example (2.2),

$$\deg(16) = \deg(15) = \deg(20) = \deg(22) = 2 \qquad \deg(11) = 1.$$

Definition 2.6. *Let $G = (V, E)$ be a graph with $|V| = n \in \mathbb{Z}^+$. We define the $n \times n$ matrix A by*

$$A_{ij} = \begin{cases} 1 & \textit{if } i \textit{ and } j \textit{ share an edge} \\ 0 & \textit{otherwise} \end{cases}.$$

This is called the ***adjacency matrix*** *for the graph G.*

Example 2.7. *The adjacency matrix for Figure 2.1 will be*

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

Theorem 2.8. *Let $G = (V, E)$ be a simple graph. If $i \in V$ is a node, then:*

$$\deg(i) = \sum_j A_{j,i}$$

and

$$A_{i,j} = A_{j,i}.$$

Proof. The result follows immediately from Definitions 2.3 and 2.6. □

If we relax the constraint that the adjacency matrix needs to be symmetric the result will be a 'directed' graph. Directed, in that now the relationship that an edge in E describes is no longer required to be reciprocal.

Definition 2.9. *Let V be a set and $E \subset V \times V$. $G = (V, E)$ is a **directed graph** or **di-graph**.*

To distinguish between simple and directed graphs, arrows are used for directed graphs edges. The edge $(0, 1)$ would represented by the arrow starting at 0 and landing at 1.

Definition 2.10. *Let $G = (V, E)$ be a di-graph. For $v \in V$, define*

$$\deg_O(v) = \#\ \textit{of edges pointing out of } v,$$

$$\deg_I(v) = \#\ \textit{of edges pointing in to } v,$$

*to be the **out-degree** and **in-degree** of v, respectively.*

Notice that as there are no additional requirements on directed graphs as compared to simple graphs, the edge set can be non-symmetric.

Example 2.11. *The adjacency matrix of the graph given in Figure 2.3 is*

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

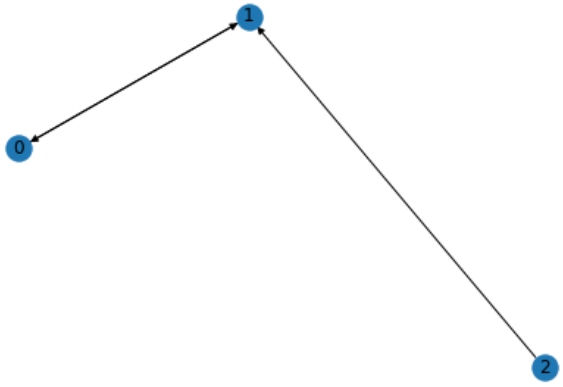

Figure 2.3: A directed graph.

This matrix has eigenvalues $-1, 0, 1$ *with associated eigenvectors of* $[-1, 1, 0]^T$, $[-1, 0, 1]^T$, $[1, 1, 0]^T$. *Notice that* $[-1, 1, 0]^T[-1, 0, 1] = 1 \neq 0$, *so these are not pair-wise orthogonal; a property we lose because the matrix is no longer symmetric.*

In many real world applications edges are not created equally; for example, the connections in a friend group might be weaker or stronger based on the quality of that friendship. So to model this, we introduce a generalization of both a simple and directed graph.

Definition 2.12. *Let* V *be a set and* $E \subset \{(u, v, w) : u, v \in V, w \in \mathbb{R}\}$. *The graph* $G = (V, E)$ *is called a* ***weighted graph****.*

The value w *for an edge* $u \to v$ *is called the* ***weight*** *of that edge pair. If the graph* $(V, E$ *weights is un-directed, then* G *is an* ***un-directed weighted graph****. Similarly, if* (V, E) *is directed, then* G *is a* ***directed weighted graph****.*

Example 2.13. *Let* $G = (V, E)$ *be a weighted di-graph with self-loops with adjacency matrix*

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 \\ 3 & 0 & 2 & 3 \end{bmatrix}. \tag{2.1}$$

The visualization of this is in Figure 2.4.

This is an example of a graph with edges such as $(4, 4, 3)$ *for* $4 \in V$ *and, in general, these are called graphs with* ***self-loops****.*

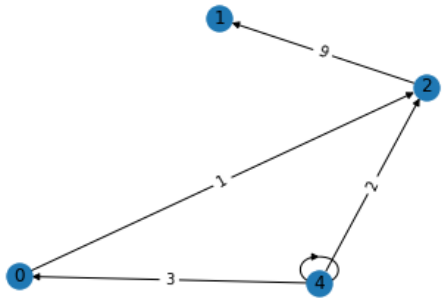

Figure 2.4: A weighted directed graph with self-loops.

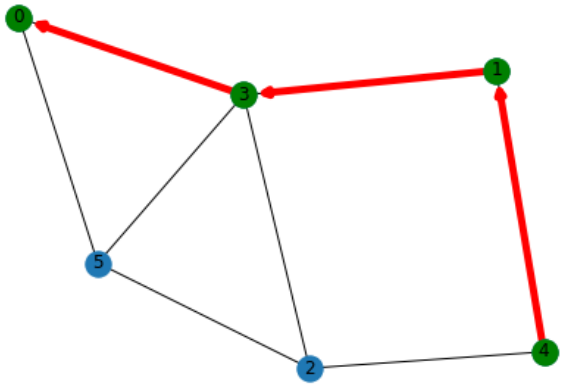

Figure 2.5: A walk of $(4, 1, 3, 0)$. Visited nodes are colored green and traversed edges are red and thicker than other edges.

Definition 2.14. *On a graph $G = (V, E)$, a **walk**, or path, of length m on V is a sequence*

$$(v_1, \ldots, v_m),$$

where v_{i+1} is jointed by an edge to v_i for all $i \in \{1, \ldots, m-1\}$.

Definition 2.15. *A graph G is **connected** if*

$$\text{for all } u, v \in V \text{ there exists a walk between } u \text{ and } v.$$

*A graph that isn't connected is **disconnected**; that is, there are two nodes $u, v \in V$ such that there is no paths between them.*

Figure 2.6: An example of a disconnected graph.

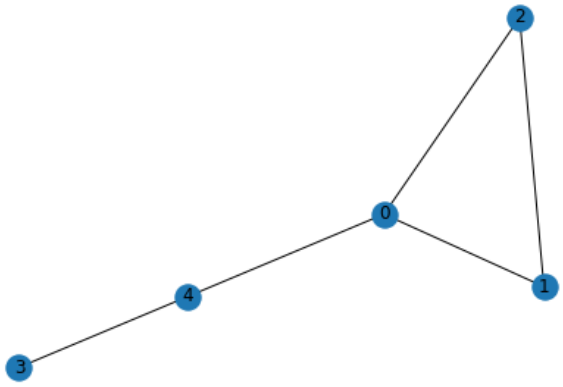

Figure 2.7: An example of a connected graph

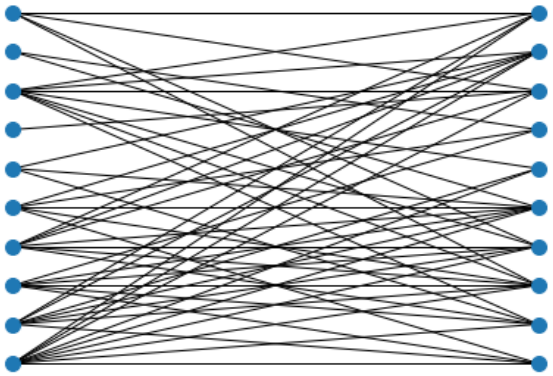

Figure 2.8: An example of a bipartite graph

Let V be an edge set. $G = (V, E)$ is a **multi-graph** if E is a multi-set (meaning multiple instances of an element are allowed; e.g. $\{(1,2),(1,2),(1,3),(3,1),(3,1)\}$ is a multi-set). Let U and V be distinct sets such that $U \cap V = \emptyset$. Then $G = (U \cup V, E)$ is a **bipartite graph** if $E \subset U \times V$. That is, any edge in E must go between an element of U and an element of V (it can't link two elements of U or two elements of of V). Example can be seen in Figure 2.8.

2.2 Stochastic Block Models

Definition 2.16. *Let V be a set, $C = \{C_1, \ldots, C_r\}$ be a partition of V, and P be an $r \times r$ matrix such that*

$$P_{i,j} \in [0,1] \qquad P_{i,j} = P_{j,i} \qquad \forall i, j \in \{1, \ldots, r\}.$$

*An element of C is called a **community** and $S = (V, C, P)$ is a **stochastic block model** (SBM).*

From S, we generate a simple graph $G = (V, E)$ by generating E with our matrix P. For all vertices $v \in C_i$ and $w \in C_j$, there is a probability of $P_{i,j}$ that $(v, w) \in E$.

- In the case where $i = j$, $P_{i,i}$ determines the internal connections of the community C_i; these are all of the edges that are between two members of the same community.
- In the case where $i \neq j$, $P_{i,j}$ determines the external connections between communities C_i and C_j; these are the connections joining the two communities.

We can allow $P_{i,j} \neq P_{j,i}$ for some $i, j \in \{1, \ldots, r\}$, however, we must drop the requirement that the generated graph be simple.

Example 2.17.

- *Figure 2.9 is a graph generated by the SBM with*

$$V = \{0, \ldots, 29\} \qquad C_1 = \{0, \ldots, 9\}, C_2 = \{10, \ldots, 19\}, C_3 = \{20, \ldots, 29\}$$

and

$$P_{i,j} = 0.5 \qquad \textit{for all } i, j \in V.$$

This is an example of an ***Erdos-Reyni model****. These satisfy*

$$P_{i,j} = p \qquad \textit{for all } i, j \in V.$$

Erdos-Reyni models are often used to generate random graphs, which act as a probability distribution over graphs.

- *Figure 2.10 has a graph generated by the SBM with*

$$V = \{0, \ldots, 29\} \qquad C_1 = \{0, \ldots, 9\}, C_2 = \{10, \ldots, 19\}, C_3 = \{20, \ldots, 29\}$$

and

$$P_{i,j} = 0.75 \ (i = j), \qquad P_{i,j} = 0.05 \ (i \neq j).$$

This is an example of a ***planted partition model****. These SBM's satisfy*

$$P_{i,j} = \begin{cases} p & \text{if } i = j \\ q & \text{if } i \neq j \end{cases} \qquad \textit{for some values } p, q \in [0, 1].$$

Note the very strong 'clustering' in this graph; this is caused by $p >> q$.

Example 2.18. *We return to the example of the famous Zachary Karate Club. In this example, we see natural communities arising in the group. Namely, the communities centered around 'Mr. Hi' (Node 0) and the communities centered around 'John A' (Node 33), colored in Figure 2.11.*

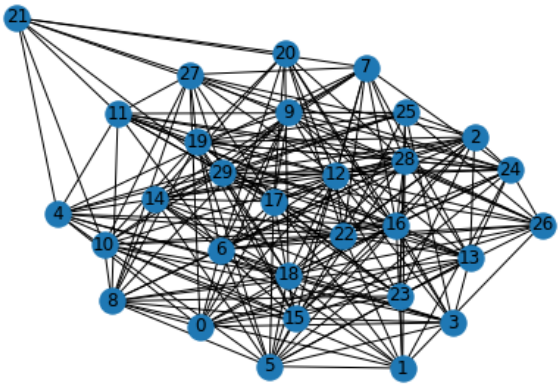

Figure 2.9: A graph generated by an Erdos-Reyni model.

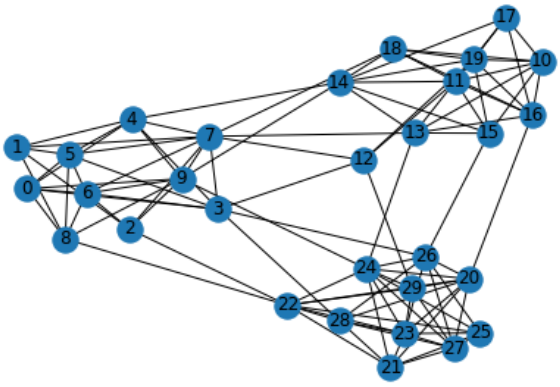

Figure 2.10: A graph generated by a planted partition model.

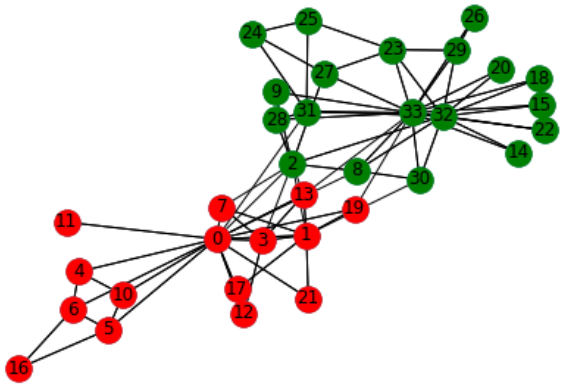

Figure 2.11: The Zachary Karate Club communities.

In a variety of social, biological, and physical situations identifying these communities has significant benefits.

- In sociology, there is the concept of *homophily*, which is the phenomenon of people with similar interests/roles/gender/age tending to establish connections. As a consequence, many social networks will naturally exhibit community structure.
- In social networks, determining communities can provide a powerful tool for marketing and recommendation algorithms; people in the same community are more likely to be friends or like the same things [13].
- In neuroscience, community structure in brain networks are being investigated for treatment of epilepsy [14]. Furthermore, variations in community structure is informing neuroimaging. [15]
- In pharmacology, community structure in protein-protein interaction networks helps reveal potential drug discoveries and treatment plans [16].

Definition 2.19. *Let $G = (V, E)$ be a graph. Assume G was generated by an SBM, $S = (V, P, C)$.* ***Community detection*** *is the process of recovering the partition C.*

As we discuss community detection in this thesis, our goal is to find a partition C that gives the strongest possible clustering behavior such as we see in the Zachary Karate Club example (Figure 2.11) or as in the planted partition model (Figure 2.10).

2.3 Community Structure

In this section we define the 'clustering' behavior in graphs. We'll restrict our discussion to undirected and unweighted graph. This '*clustering*' behavior is strongest when given a graph $G = (V, E)$ and a partition C of V, if $C_i, C_j \in C$ with $v \in C_i$ and $w \in C_j$ then

- if $i = j$, then w and v are likely to be connected by an edge;
- if $i \neq j$, then w and v are not likely to be connected by an edge.

So to formalize this, we'll compare this to the so-called **null model** assumption.

Assumption 2.20. *A randomly and uniformly generated graph has no community structure.*

That is, a graph generated by an Erdos-Reyni model has no community structure.

Definition 2.21. *A **null model** for a simple graph $G = (V, E)$, with $|V| = n$ and $|E| = m$, is a multi-graph $G' = (V, E')$, defined by:*

1. *For each edge, $(v, w) \in E$, cut the edge in half, so we now have $2m$ 'dangling' **stubs**.*
2. *Reconnect the stubs by randomly and uniformly pairing two stubs together and joining the stubs to form an edge; store the edge in E'.*

Theorem 2.22. *Let $G' = (V, E')$ be the null model of $G = (V, E)$. Then G' and G have the same degree sequence and*

$$|E| = |E'|.$$

Proof. Let $G = (V, E)$ with v being a node in G. We'll show the result by going through the construction of G'.

If v has degree k, then v has k adjacent edges. Now create the stubs attached to v: there will be precisely k stubs attached to v. Take a stub from v: there are $2|E| - 1$ choices of other stubs to attach to since the stub can't connect to itself but it can connect to any other stub created in the null model. If the stub attaches to another stub of v, then this creates a self loop and that is a connection. If the stub attaches to a stub not attached to v, then this is a new connection for v. We do this k times for all nodes $v \in V$ and we have

$G' = (V, E')$, the null model for G. By construction, the degree of v has not changed and since this is true for any $v \in V$, the degree sequence of G is the same as the degree sequence of G'. □

The intuition is that a graph generated by a Erdos-Reyni model has no clustering behavior: the entire graph is one big cluster. So this null model acts as a null hypothesis for our community testing.

To summarize the assumptions we've made so far:

Assumption 2.23.

1. *Graphs with community structure are generated by a stochastic block model.*
2. *Two nodes are much more likely to be connected inside a community than out.*

And with Assumption 2.20:

3. *A null model of a graph has no community structure.*

Lemma 2.24. *Let $G' = (V, E')$ be the null model of $G = (V, E)$, where $|E| = |E'| = m$ and $v, w \in V$. Define $I_i^{(v,w)} = 1$ if v and w are paired by a stub $i = 1, \ldots, k_v$. Then*

$$p(I_i^{(v,w)} = 1) = \frac{k_w}{2m - 1}.$$

Proof. Let $G' = (V, E')$ be the null model of $G = (V, E)$, where $|E| = |E'| = m$ and $v, w \in V$. Define $I_i^{(v,w)} = 1$ if v and w are paired by a stub $i = 1, \ldots, k_v$. For the stubs starting at v, there are a total of $2m-1$ choices, since stubs can't connect to themselves but they can connect to any other stub in the network. These number $2m$ by their construction in the null model. Out of those choices there are k_w many stubs that will connect v to w and hence make $I_i^{(v,w)} = 1$. Thus

$$p(I_i^{(v,w)} = 1) = \frac{k_w}{2m - 1}.$$

□

Theorem 2.25. *Let $G' = (V, E')$ be the null model of $G = (V, E)$, where $|E| = |E'| = m$ and $v, w \in V$. Then*

$$\mathbf{E}[\ \#\ \textit{of edges between } v \textit{ and } w\ ] = \frac{k_w k_v}{2m-1}.$$

Proof. Let $G' = (V, E')$ be the null model of $G = (V, E)$, where $|E| = |E'| = m$ and $v, w \in V$. Note that

$$\#\text{ of edges between v and w} = \sum_{i=1}^{k_w} I_i^{(v,w)},$$

so that

$$\begin{aligned}
\mathbf{E}[\ \#\text{ of edges between v and w }] &= \mathbf{E}\left[\sum_{i=1}^{k_v} I_i^{(v,w)}\right] \\
&= \sum_{i=1}^{k_w} \mathbf{E}[I_i^{(v,w)}] \\
&= \sum_{i=1}^{k_w} \frac{k_v}{2m-1} \\
&= \frac{k_w k_v}{2m-1},
\end{aligned}$$

where we used the linearity of $\mathbf{E}[\cdot]$ in the second equality. □

For large networks, where $|E| = m >> 1$, it is typical to just approximate this expected value as

$$\frac{k_w k_v}{2m}.$$

Therefore, this gives us an average number of links between two nodes v and w in a null model.

Definition 2.26. *For a graph $G = (V, E)$ with a partition C on V, define the* ***modularity of G by the partition*** *C to be*

$$Q(G, C) = \frac{1}{2m} \sum_i \sum_j \left(A_{i,j} - \frac{k_i k_j}{2m}\right) \delta(C_i, C_j) \qquad \delta(C_i, C_j) = \begin{cases} 1 & \textit{if } C_i = C_j \\ 0 & \textit{otherwise} \end{cases}$$

- If $Q(G, C) \leq 0$, then partition C gives no community structure at all, it might as well have been completely randomly generated.
- If $Q(G, C) > 0$, then the graph and partition exhibit community structure.

Modularity will be used as the working definition of community structure, and we'll add this on to our list of assumptions for community detection:

Assumption 2.27.

1. *Graphs with community structure are generated by stochastic block models.*
2. *Two nodes are much more likely to be connected inside a community than out.*
3. *A null model of a graph has no community structure.*
4. *Community structure will correspond to a graph with high modularity.*

With Assumptions 2.27 our task of community detection can then be formulated as a maximization problem:

$$\arg\max_C Q(G, C) = \arg\max_C \frac{1}{2m} \sum_i \sum_j \left(A_{i,j} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j)$$

A random brute force method is not practicable in most cases: even in the case of finding two communities C_1 and C_2 of equal size ($|C_1| = N_1$, $|C_2| = N_2$ where $N_1 = N_2 = \frac{N}{2}$), we have

$$\frac{N!}{(N_1!)^2}$$

total possible partitions. Using Stirling's approximation,

$$\frac{N!}{(N_1!)^2} \sim \exp\left((N+1)\ln(2) - \frac{1}{2}\ln(N) \right) \qquad N \to \infty.$$

Thus even for a reasonably small graph ($N = 100$) with these assumptions, there are approximately 10^{29} total partitions of this graph [17].

So we need smart algorithms to find these partitions in a computationally efficient manner, as in real world networks it's not uncommon to see networks with the number of

nodes exceeding 10^6. In the case of Facebook or other social media networks, the number of nodes might exceed 10^9 (as of 2021 Facebook has 2.91 billion users).

It has been shown that finding a global maximum for modularity is an NP-complete problem [18]. So all the algorithms that we'll be discussing in the next section for the task of community detection can only lead to local maximuma of modularity.

2.4 Centrality Measures

In this section, we'll define a number of commonly used network statistics. The following texts were used for these definitions: [17, 19, 20].

We already discussed several basic structural features of networks, such as degree and connectedness. However, one thing these features fail to capture is the topological importance of nodes to the overall network. For example, in the case of the Zachary Karate Club on Figure 2.2, we may ask which nodes are most important for communication between the two groups? Or which nodes are possible leaders in the group? To answer these questions we introduce the concept of *centrality* to networks.

Definition 2.28. *Let $G = (V, E)$ be a network with node $v \in V$. The **degree centrality** of v is*

$$\frac{1}{|V|}\deg(v).$$

Degree centrality simply measures the importance of a node by how connected a node is in a network.

Definition 2.29. *Let $G = (V, E)$ be a network with node v. The **closeness centrality** of v is*

$$g_v = \frac{1}{\sum\limits_{C}\sum\limits_{\{v,w\in C: v\neq w\}} \frac{|V|-1}{|C|-1} l_{v,w}},$$

where $l_{i,j}$ is the length of the shortest path between v and w, and C is a connected component of G.

Note that this measure is defined for disconnected networks. It is the reciprocal of the average distance from v to any node in G, so the further v is from other nodes, the larger the denominator, and so the smaller g_v is. This measures the importance of a node by how close to all other nodes that node is [21].

Definition 2.30. *Let $G = (V, E)$ and $v \in V$. The **betweennness centrality** of v is*

$$\frac{1}{\binom{|V|-1|}{2}} \sum_{h,j \in V} \frac{\sigma_{hj}(i)}{\sigma_{hj}}$$

where C is a connected component of G, σ_{hj} is the number of total shortest paths between nodes h and j, $\sigma_{hj}(i)$ is the number of total shortest paths between nodes h and j that go through i, and $\binom{|V|-1}{2}$ is the number of total paths that can go through node i [22].

This is easily extended to edges to examine the importance of an edge to a networks communications [23].

Definition 2.31. *Let $G = (V, E)$ and $e \in E$. The **edge-betweennness centrality** of e is*

$$\frac{1}{\binom{|V|-1|}{2}} \sum_{h,j \in V} \frac{\sigma_{hj}(e)}{\sigma_{hj}}$$

where C is a connected component of G, σ_{hj} is the number of total shortest paths between nodes h and j, $\sigma_{hj}(e)$ is the number of total shortest paths between nodes h and j that go through the edge e, and $\binom{|V|-1}{2}$ is the number of total paths that can go through node i [22].

Definition 2.32. *Let $G = (V, E)$ be a network and $v \in V$. The **harmonic centrality** of v is*

$$\sum_{\{u \in V : u \neq v\}} \frac{1}{l_{v,w}}$$

where $l_{v,w}$ is the length of the shortest path between v, w. If v and w are not connected by a path, then we let $l_{v,w} = \infty$ and $\frac{1}{l_{v,w}} = 0$.

Notice this is similar to, but not equal to, the definition of closeness centrality [24]. This centrality differs from closeness, in that we're summing over the reciprocals of the shortest path lengths and not taking the reciprocals of the sum of the shortest path lengths, the effect being that the measure can now easily handle disconnected graphs if we use the convention of letting $l_{v,w} = \infty$.

Definition 2.33. *Let $G = (V, E)$ be a network and $v \in V$. The* ***sub-graph centrality*** *of v is*

$$\left(e^A\right)_{v,v} = \sum_{k=0}^{\infty} \frac{A_{v,v}^k}{k!}$$

where A is the adjacency matrix of G.

Using the fact that $A_{v,w}^k$ counts the number of paths from v to w of length k, this is the sum over all possible lengths of paths that begin and end at v. Under this centrality, a node is important if it has many paths back to itself, capturing path importance of v to the overall graph G [25].

Definition 2.34. *Let $G = (V, E)$ be a network and $v \in V$. Let $H = (V', E') \subseteq G$ be a subgraph of G. Then the* ***k-core*** *induced by H is*

$$\{v \in V : \deg_H(v) \geq k\}, \tag{2.2}$$

where $\deg_H(v)$ is the degree of v in the subgraph H. The ***core-number*** *of a node v is the smallest k for which v meets the requirement of (2.2).*

That is, the k-core induced by H on G is the sub-graph of H that contains all nodes with degree greater than or equal to k. In social-media analysis, this measure has been used as an extension of the concept of density for determining the structure of graphs and $O(|E|)$ algorithms have been implemented for this in the python package `networkx` [26, 27, 10].

Finally, the following measure is defined on a graph, rather than a node.

Definition 2.35. *Let $G = (V, E)$ be a network. The* ***S-metric*** *of G is*

$$\sum_{(u,v)\in E} d(u)d(v). \tag{2.3}$$

The S-metric, as defined in [28], is a measure of the hub-and-spoke nature of a graph; graphs with many nodes ($d(v) >> 1$) connected to many highly connected nodes ($d(u) >> 1$) will maximize the S-metric.

Chapter 3

COMMUNITY DETECTION

In this section, we'll describe a number of the community detection algorithms used to find community structure in the network produced, as described in *Networks* by Mark Newman [20].

3.1 The Louvain Algorithm

In this subsection, we'll discuss one of the most widely used community detection algorithms called the **Louvain Algorithm** [29].

The goal of the Louvain algorithm and many other community detection algorithms is to maximize the modularity of a network by finding a local optimal community partition C [30, 31].

Definition 3.1. *Let $G = (V, E, W)$ be a simple weighted graph and C be part of a partition of V. We define*

$$\Sigma_{in} = \sum_{i \in C} \sum_{j \in C} A_{i,j}$$

as total number of edges that land inside the community,

$$\Sigma_{tot} = \sum_{i \in C} k_i$$

as the total number of edges in a community, including those between a node in the community and a node in another community,

$$\Sigma_{out} = \Sigma_{tot} - \Sigma_{in}$$

as the total number of edges starting in C and landing outside of C, and

$$k_{i,in} = \sum_{j \in C} A_{i,j} + \sum_{i \in C} A_{i,j}$$

as the number of edges a node has that land in C.

Algorithm 3.2 (The Louvain Algorithm)**.**

Let $G = (V, E)$ be a simple weighted graph.

Phase 1

1. *Assign an initial guess of community structure of G:*

$$C = \{C_1, \ldots, C_r\}.$$

A typical guess is that each node is a community; i.e. if $V = \{v_i\}_{i=1}^N$, then $C_i = \{v_i\}$ for all $i = 1, \ldots, N$.

2. *Start with a random community. For each $v \in V$ with $v \in C_i$ move $v \to C_j$ (for some C_j that neighbors v) and calculate the change in modularity $\Delta Q_{i\to j}(v)$. We want to calculate each term of*

$$\Delta Q_{i\to j}(v) = \Delta Q_{i\to\{v\}}(v) + \Delta Q_{\{v\}\to j}(v),$$

that is, the modularity from moving v from C_i to C_j is the same as making an intermediate step of moving v to its own community $\{v\}$.

We'll prove these steps rigorously below.

We'll show that

$$\begin{aligned}\Delta Q_{i\to j}(v) = &\left[\frac{\Sigma_{in} + 2k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m}\right)^2\right] \\ &- \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right].\end{aligned}$$

For each node v, choose C_j such that

$$C^* = \arg\max_j \Delta Q_{i\to j}(v),$$

that is, pick the community C_j that maximizes the increase in modularity.

3. *Form the new partition C' by moving v to C^*.*

If the quantity in the previous step is $\Delta Q_{i\to j}(v) \leq 0$ (we lose or don't gain any modularity by moving v), then we don't move v.

4. *Repeat (1-3) for all $v \in V$ until no modularity can be gained.*

Phase 2

1. *Construct the new graph G' by taking $C_p \in C$, for each $p \in \{1, \ldots, r\}$ and assign a self-loop to C_p with weight*

$$w(C_p, C_p) = \sum_{v \in C_p} k_{v,in}.$$

Then for each pair $C_i, C_j \in C$ assign an edge with weight

$$w(C_i, C_j) = \sum \text{ of edges starting in } C_i \text{ and ending in } C_j.$$

2. *Repeat Phase 1 with the new graph G'.*

Note that the result of this algorithm will be a local maximum attained by the 'greedy' gathering of modularity by each community.

Lemma 3.3. *Let $G = (V, E, W)$ be a simple weighted graph with node partition $C = \{C_1, \ldots, C_r\}$. Then the modularity is*

$$Q(G, C) = \frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2.$$

Proof.

$$\begin{aligned} Q(G, C) &= \frac{1}{2m} \sum_i \sum_j \left(A_{i,j} - \frac{k_i k_j}{2m}\right) \delta(C_i, C_j) \\ &= \frac{1}{2m} \sum_i \sum_j A_{i,j} \delta(C_i, C_j) - \frac{1}{2m} \sum_i \sum_j \frac{k_i k_j}{2m} \delta(C_i, C_j) \\ &= \frac{\Sigma_{in}}{2m} - \frac{1}{4m^2} \left(\sum_i k_i\right) \left(\sum_j k_j\right) \\ &= \frac{1}{2m} \Sigma_{in} - \left(\frac{\Sigma_{tot}}{2m}\right)^2. \end{aligned}$$

□

Theorem 3.4. *Let $G = (V, E, W)$ be a simple weighted graph with node partition $C = \{C_1, ..., C_r\}$. Suppose $v \in V$ and $C_p \in C$. Then*

$$\Delta Q_{\{v\} \to p}(v) = \left[\frac{\Sigma_{in} + k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m} \right)^2 \right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right].$$

Proof. By definition, this is

$$\Delta Q_{p \to \{v\}}(v) = Q_{After} - Q_{Before},$$

where Q_{After} is the modularity after we place v in C_p, and Q_{Before} is before this occurs and v is in its own isolated community $\{v\}$.

We have that

$$Q_{Before} = \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m} \right)^2 \right] + \left[0 - \left(\frac{k_i}{2} \right)^2 \right]$$

which is the sum of the modularity of the graph without the community $\{v\}$ and the modularity of the isolated community $\{v\}$.

Then

$$Q_{After} = \left[\frac{\Sigma_{in} + k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m} \right)^2 \right]$$

so that

$$\Delta Q_{\{v\} \to p}(v) = \left[\frac{\Sigma_{in} + k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m} \right)^2 \right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right].$$

□

Example 3.5. *Let's use this with the Zachary Karate Club network:*

- *The original graph is in Figure 3.1.*
- *Phase 1's result is Figure 3.2.*
- *Phase 2's result is Figure 3.3.*

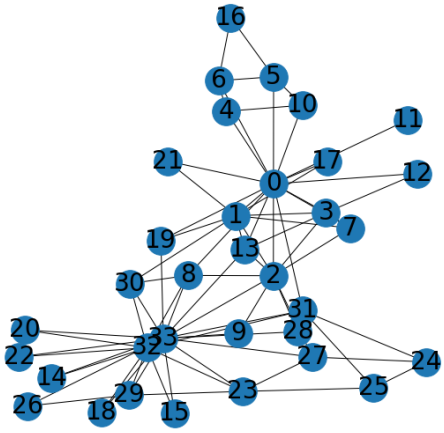

Figure 3.1: The Zachary Karate club network

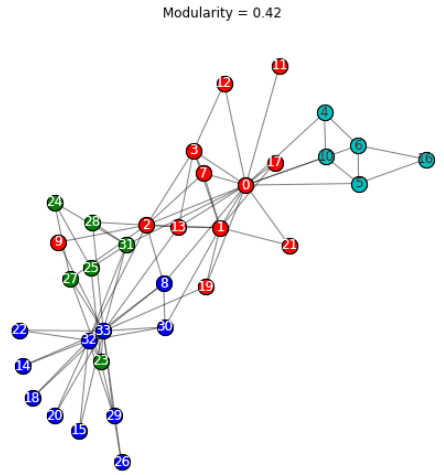

Figure 3.2: Phase 1 of the Louvain Algorithm on the Zachary Karate Club Network

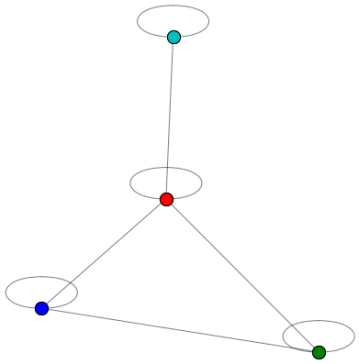

Figure 3.3: Phase 2 of the Louvain Algorithm on the Zachary Karate Club Network

3.2 Classical Spectral Clustering

As we've seen with the Louvain algorithm, the problem of community detection can be a difficult problem. One way to tackle the problem of community detection is to find a mapping

$$f : G \to \mathbb{R}^m$$

for some small $m \in \mathbb{Z}^+$, analyze the problem in $\mathbb{R}^m$, then map the solution back to the graph G.

The advantage of this is that clustering in continuous spaces such as $\mathbb{R}^m$ is a well studied area; there exist dozens of techniques in unsupervised machine learning that tackle this problem [32]. The tricky problem we have to solve before we can leverage this machinery against community detection, is that we need the map $f : G \to \mathbb{R}^m$ to decide what information about the graph G we need to preserve.

Definition 3.6. *Given a weighted simple graph $G = (V, E, W)$, define the **Laplacian matrix** of G to be*

$$L = [D_{i,j} - W_{i,j}]_{i,j},$$

where W is the weighted adjacency matrix of G and D is the diagonal degree matrix of G:

$$D_{i,j} = \begin{cases} \sum_j w_{i,j} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}.$$

The Laplacian matrix both encodes degree information of the graph, as well as the adjacency matrix of the graph (since $W_{i,i} = 0$ for graphs without self-loops). So the diagonal entries will always tell us the degree of a node, while the off-diagonal entries tell us about the connections between nodes.

The properties of the Laplacian matrix are well-studied in *spectral graph theory* [33].

Theorem 3.7. *Let G be a simple graph and L be the Laplacian of the graph. Then*

1. *L is positive semi-definite and symmetric;*
2. *0 is an eigenvalue of L with eigenvector $\mathbf{1}$ (the vector of all 1's).*

Proof. Let $G = (V, E)$ be a simple graph, so the Laplacian matrix is defined by

$$L = [D_{i,j} - A_{i,j}]_{i,j},$$

where A is the adjacency matrix of G and D is the diagonal degree matrix of G:

$$D_{i,j} = \begin{cases} \deg(i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}.$$

1. Since $L = D - A$, and both A and D are symmetric for simple graphs, we have that L is symmetric.

   And now we'll show that L is positive semi-definite, but first we need to show there is a matrix B such that $L = BB^T$. Define the **incidence matrix** of the graph G to

be the $|V| \times |E|$ matrix given by

$$B_{v,(v_i,v_j)} = \begin{cases} 1 & \text{if } v = v_i \\ -1 & \text{if } v = v_j \\ 0 & \text{otherwise} \end{cases}.$$

Then we have

$$\begin{aligned}(BB^T)_{i,j} &= \sum_k B^T_{i,k} B_{k,j} \\ &= \sum_k B_{k,i} B_{k,j}.\end{aligned}$$

If $i = j$, this is

$$\begin{aligned}(B^T B)_{i,i} &= \sum_k B^2_{k,i} \\ &= \deg(i),\end{aligned}$$

and if $i \neq j$, then

$$(B^T B)_{i,j} = \sum_k B_{k,i} B_{k,j}.$$

Notice that whenever $B_{k,i} = 1$, that is k is the 'beginning' point of an edge, then $B_{k,j} = 0$, unless $(i,j) \in E$ where $B_{k,i} B_{k,j} = -1$. Hence

$$(B^T B)_{i,j} = -A_{i,j},$$

and so $B^T B = BB^T = D - A = L$.

So then, let $x \in \mathbb{R}^{|V|}$ and hence

$$\begin{aligned}x^T L x &= x^T BB^T x \\ &= (B^T x)^T (B^T x) \\ &= \|B^T x\|^2 \geq 0.\end{aligned}$$

Thus L is positive semi-definite.

2. Finally, let $\mathbf{1} \in \mathbb{R}^{|V|}$ be the vector of all one's. Then consider the following:

$$\begin{aligned} L\mathbf{1}_i &= \sum_j L_{i,j} \\ &= \sum_j D_{i,j} - A_{i,j} \\ &= D_{i,i} - \sum_j A_{i,j} \\ &= \deg(i) - \deg(i) \\ &= 0. \end{aligned}$$

So that $L\mathbf{1} = 0\mathbf{1}$, and so 0 is an eigenvalue of L with eigenvector $\mathbf{1}$.

□

We relate community detection back to the Laplacian by posing community detection as a problem of finding a *cut* that minimizes the number of edges it passes through.

Definition 3.8 (**Cut & Cut Size**). *Let $G = (V, E, W)$ be a simple weighted graph and let l be a cut on the edges of G; i.e $l \subset E$ that defines a partition $A \cup B = V$, $A \cap B = \emptyset$. Let $X, Y \subseteq V$.*

*For any X, Y, define the **weight** between X and Y as*

$$w(X, Y) = \sum_{i \in X, j \in Y} w_{i,j},$$

*and the **normalized cut size** between X and Y as*

$$ncut(X, Y) = \frac{w(X, Y)}{w(X, V)} + \frac{w(X, Y)}{w(Y, V)}.$$

Note that $\text{ncut}(X, Y)$ is normalized so that $0 \leq \text{ncut} \leq 2$.

The following theorem relates the eigenvector of this matrix L to a suitable map $G \to \mathbb{R}^m$.

Theorem 3.9. *Let $G = (V, E)$ be a simple graph.*

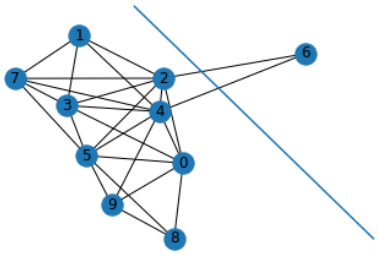

Figure 3.4: A cut equalling $\{(2,6),(4,6)\}$ and defines a partition $A = \{6\}$ and $B = V \setminus \{6\}$.

Suppose $V = A \cup B$ is a partition of V, with $|A| = |B|$ where $A \cap B = \emptyset$. Define

$$y = \begin{cases} 1 & \text{if } i \in A \\ -b & \text{if } i \in B \end{cases}, \qquad b = \frac{k}{1-k}, \qquad k = \frac{\deg(A)}{\deg(V)}.$$

Then

$$\min_{(A,B)} ncut(A,B)$$

is equivalent to

$$\begin{cases} \min_y \dfrac{y^T L y}{y^T D y} \\ y^T D\mathbf{1} = 0, \end{cases}$$

where $\mathbf{1}$ is the vector of all 1's [34].

We note that if D is a diagonal matrix, $D = \text{diag}\{d_1, \ldots, d_M\}$, then for $\alpha \in \mathbb{R}$, $D^\alpha = \text{diag}\{d_1^\alpha, \ldots, d_M^\alpha\}$.

To solve

$$\begin{cases} \min_y \dfrac{y^T L y}{y^T D y} \\ y^T D\mathbf{1} = 0 \end{cases}$$

we are going to transform this problem into

$$\begin{cases} \min_z \dfrac{z^T B z}{z^T z} \\ z^T z_0 = 0 \end{cases}, \qquad B = D^{-1/2} L D^{-1/2}$$

where $z = D^{1/2} y$ and $z_0 = D^{1/2}\mathbf{1}$. This new matrix $B = D^{-1/2} L D^{-1/2}$ is called the **normalized Laplacian**. The reason for this transformation is that we can now pose this as an eigenvalue problem:

$$Bz = \lambda z \iff z^T B z = \lambda z^T z \iff \lambda = \frac{z^T B z}{z^T z}$$

and the condition

$$z^T z_0 = 0$$

ensures that z is orthogonal to the eigenvector z_0. Then this is the minimization problem

$$\begin{cases} \min_z \dfrac{z^T B z}{z^T z} \\ z^T z_0 = 0 \end{cases},$$

with the eigenvalues of B satisfying $\lambda_0 = 0 < \lambda_1 < \ldots$. The minimization problem is solved by the eigenvector with eigenvalue $\lambda_1 \neq 0$ [34].

This method then can be generalized to k-cuts in a number of different ways, outlined in [34]. The general algorithm for spectral clustering is then given below.

Algorithm 3.10 (**Spectral Clustering for Graphs**)**.**

Let $G = (V, E)$ be a simple weighted graph with $|V| = n$.

1. *Calculate the Laplacian for G, L.*
2. *Find the first k eigenvalues of L and their corresponding eigenvectors.*
3. *Form a matrix U where the column vectors of U are the k-eigenvectors of L; U is $n \times k$.*
4. *For each $i = 1, \ldots, n$:*
   - *Map the node $i \in V$ to the row vector $v_i \in \mathbb{R}^k$ obtained from U.*

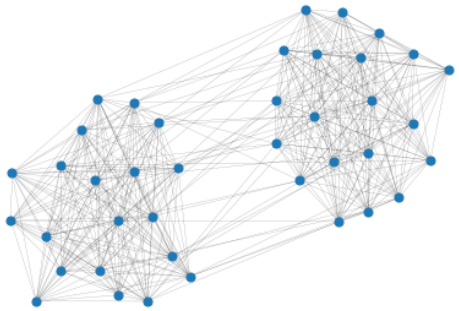

Figure 3.5: The graph we'll be preforming spectral clustering on.

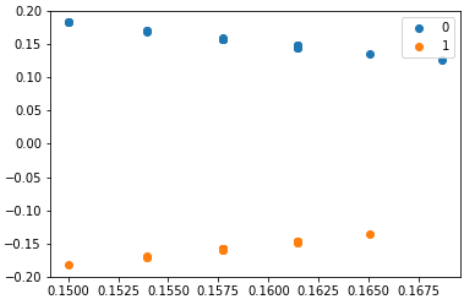

Figure 3.6: The eigenvectors of the normalized Laplacian of the graph in Figure 3.5 with cluster labels $\{0, 1\}$.

5. *Use K-means clustering or other clustering algorithms in $\mathbb{R}^k$ to find clustering.*

6. *Map cluster membership in $\mathbb{R}^k$ back to the graph G.*

Example 3.11. *Let's take a planted partition model with inter-community probability $p = 1$ and an intra-community probability $q = 0.1$ with two communities of equal size and a vertex set of size 20; that is, nodes inside the same community must be connected and nodes from the two different communities have a one-in-ten chance of being linked. The graph can be seen in Figure 3.5. Choosing just the two smallest non-zero eigenvectors of the Laplacian for G, and using 2-means clustering we end up with Figure 3.6. Finally, we map this clustering back to our graph we end up with Figure 3.7.*

Spectral clustering is also applied to problems of clustering in $\mathbb{R}^n$ as a dimensional reduction method, where a mapping is found from $\mathbb{R}^n \to G$, where G is typically a graph defined by a similarity function in $\mathbb{R}^n$, (e.g. $f(x, y; \sigma) = \exp\left(\frac{-\|x-y\|^2}{2\sigma}\right)$ is a common choice). Then a second map $G \to \mathbb{R}^m$ is found with $m \leq n$. So spectral clustering can also

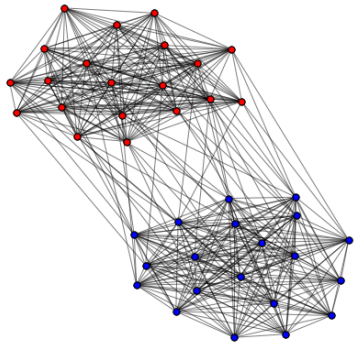

Figure 3.7: The result of spectral clustering on the graph 3.5, the modularity of this clustering is approx 0.4**.**

be used a dimensional reduction technique where we map data in $\mathbb{R}^n \to G \to \mathbb{R}^m$ where $m \leq n$.

3.3 K-Means Clustering

We'll briefly describe K-means clustering in this subsection, as used in the spectral clustering algorithm. Let X be a data set $X = \{x^{(i)}\}_{i=1}^N$ such that $x^{(i)} \in \mathbb{R}^m$ for each $i = 1, \ldots, N$. **K-means clustering** deals with the problem of finding clusters of densely packed groups in this data set. This is a topic in *unsupervised machine learning* ;*supervised machine learning* deals with prediction of a target variable usually denoted $y^{(i)}$ for each $x^{(i)}$, while unsupervised machine learning has no target variables for the data points $x^{(i)}$. This algorithm is sometimes referred to as Lloyd's algorithm or *naive k-means* [35].

Algorithm 3.12. *Let $k \in \mathbb{N}$ be fixed.*

1. *Partition your data set into k-groups, $S_1, \ldots, S_k$.*

2. *For each $i = 1, \ldots, k$:*

   Calculate the mean vector for S_i, denoted $m^{(i)}$; that is

$$m^{(i)} = \frac{1}{|S_i|} \sum_{x^{(j)} \in S_i} x^{(j)}$$

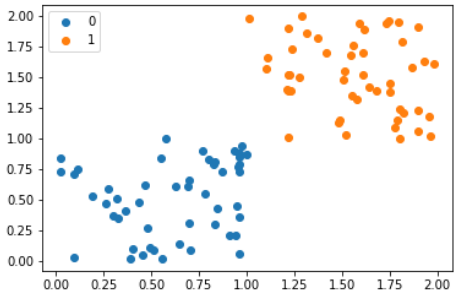

Figure 3.8: K**-means ran with** $k = 2$**.**

3. *For each* $x^{(i)}$, $i = 1, \ldots, k$*:*

   Assign each $x^{(i)}$ *to the set* S_{j^*} *where* $j^* = 1, \ldots, k$ *and*

   $$j^* = \arg\min_j \|x^{(i)} - m^{(j)}\|_2^2$$

4. *Repeat steps until in previous step no change occurs.*

Example 3.13. *We generate a random data set* $X = \{x^{(i)}\}_{i=1}^{1000}$ *where* $x^{(i)} \in \mathbb{R}^2$ *for each* $i = 1, \ldots, 1000$ *using a uniform distribution on* $[0, 1] \times [0, 1]$*. One half of the data is kept in the rectangle* $[0, 1] \times [0, 1]$*, the other half is shifted diagonally to be contained in* $[1, 2] \times [1, 2]$*. We run* K*-means with* $k = 2$ *using the Python library* `sklearn` *and we end up with the result seen in Figure 3.8.*

Chapter 4

TIME-SERIES ANALYSIS

In this chapter, we'll cover some basic definitions and terms used in time-series data analysis all leading to the definition of 'cross-correlation' as a measure of statistical significance in the trends that are often seen in time-series data. The definitions and explanation included come from the "Forecasting: Principles and Practice" by R. Hyndman and G. Athanasopoulos [2] as well as "Time-Series Analysis: With Applications in R" by J. Cryer and J. Chan [36].

4.1 Stationary Time-Series

A time-series is a data set where the data points have an inherent ordering to them, namely time. COVID-19 cases, temperature, and stock market prices are examples of time-series. These differ from regular data sets, in that ordering in the data matters and values are potentially dependent on previous values but not future values. Time-series can behave extremely irregularly and chaotically, so we'll discuss some methods of how to find trends in time-series data as well as how to forecast or predict future values in a time-series. In particular, we'll discuss the importance of normally distributed time-series.

Definition 4.1. *A time-series* $\{X_t : t \in \{1, \dots, T\}\}$ *is said to be* ***stationary*** *if*

$$X_t, X_{t+1}, \dots, X_{t+k}$$

has the same distribution for all $t \in \{1, \dots, T\}$ *and* $k \in \mathbb{Z}^+$.

Example 4.2. *Defining a time-series by* $X_t \sim \mathcal{N}(0, \sigma^2)$ *for all* t*, this is known as a* ***white-noise process****. That is, at each time step we're sampling from a normal distribution* $\mathcal{N}(0, \sigma^2)$ *with zero-mean and* σ^2 *variance. This has a probability-density function given by*

$$f(x) = \frac{1}{\sqrt{2\sigma^2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right), \tag{4.1}$$

where σ^2 *is the variance of the random-variable.*

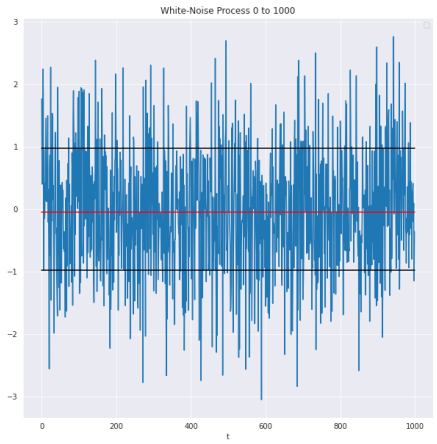

Figure 4.1: A white-noise process with mean zero and unit variance.

Illustrating this below, we take $\sigma^2 = 1$ *and construct a white-noise process with 1000 data points. First looking at the overall mean and variance in Figure 4.1, we see that this is approximately* 0 *and* 1*, respectively, as expected. We now partition this data set into 4 equally sized subsets and we plot the distribution of the overall process in Figure 4.3, as well as the distributions of the four subsets in Figures 4.4, 4.5, 4.6, 4.7. Observe that there is some differences in the distributions of the subsets, but these are minor perturbations around the probability density function in (4.1). Looking now at the means and variances over four subsets of the white-noise process, each* 250 *time-steps in width, we see that the mean and variance does differ but only slightly. This would be satisfactorily called a stationary process.*

When data is not stationary, one culprit that is typically behind it is there is a clear increasing or decreasing trend in the data. We'll explore this through an example, as well as address how to transform the data to be stationary.

Example 4.3. *Let* $X_t = t + \epsilon_t$ *where* $\epsilon \sim \mathcal{N}(0, 1)$*; that is* X_t *is following a linear trend. This is called a* ***white-noise process with drift****. In Figure 4.8 we see that both the variance and the mean are increasing with time – they are far from constant.*

However after preforming a backwards-difference on X_t*,*

$$\Delta X_t = X_t - X_{t-1},$$

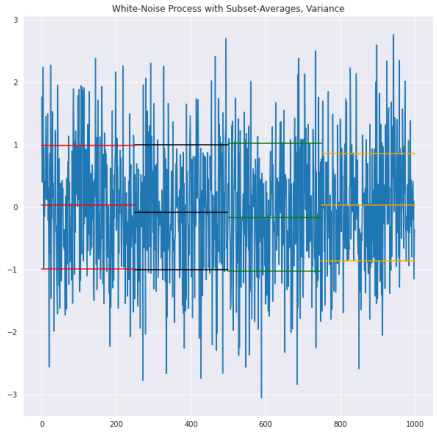

Figure 4.2: White-process with subset means and variance.

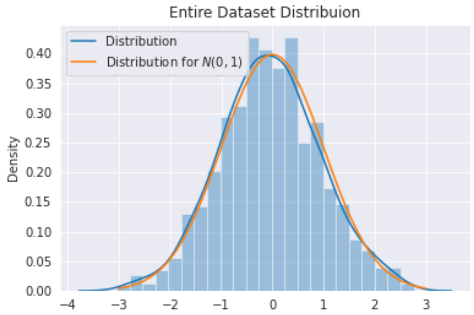

Figure 4.3: Distribution of white-noise process.

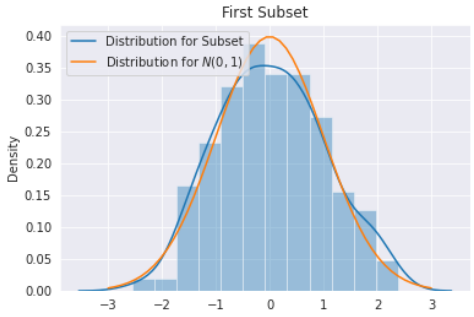

Figure 4.4: Distribution of white-noise process, subset one.

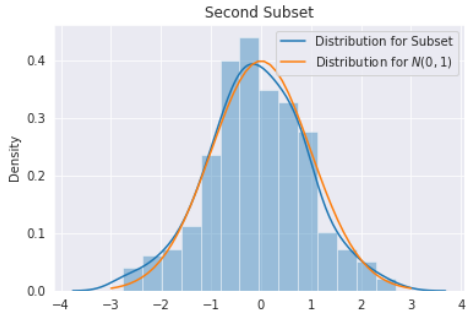

Figure 4.5: Distribution of white-noise process, subset two.

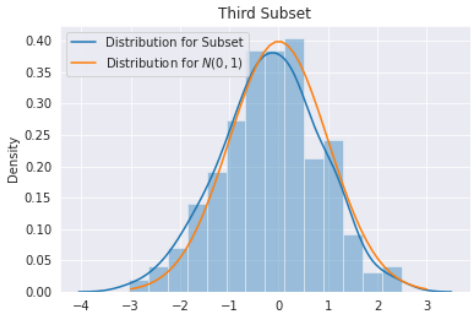

Figure 4.6: Distribution of white-noise process, subset three.

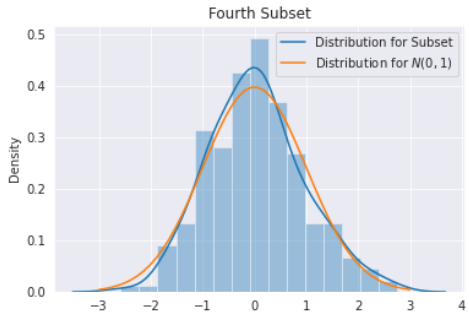

Figure 4.7: Distribution of white-noise process, subset four.

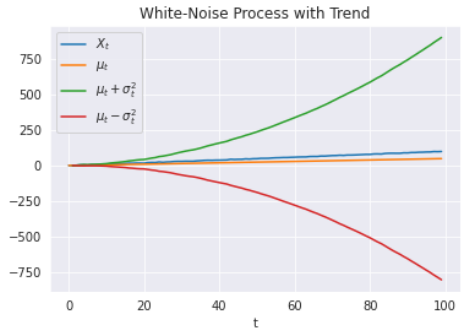

Figure 4.8: White-Noise process with trend.

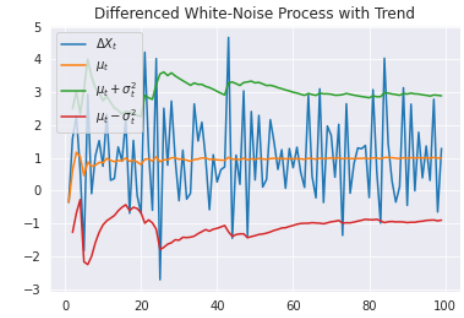

Figure 4.9: Backwards differenced white-noise process with trend.

the differencing evens out the variance and mean as can be seen in Figure 4.9. Differencing makes quite the difference, both the mean and variance eventually even out over time. Enough for this to be considered a stationary series.

4.2 Auto-Regression and Moving Average Models

As to how we can model time-series data, there are classical four methods we'll discuss here: moving-average models, auto-regressive models, auto-regressive integrated moving average models, then finally the seasonal auto-regressive integrated moving average model.

Definition 4.4. *Let $\{Y_t : t = 1, \ldots, T\}$ be a time-series. The* ***back-shift or lag operator*** *is defined as*

$$L^n Y_t = Y_{t-n} \qquad L^0 = I$$

for all $n \in \mathbb{Z}^+$. The time-series Y_{t-n} is said to have lag n.

Similarly the ***forward-shift or lead operator*** *is defined as*

$$F^n Y_t = Y_{t+n} \qquad F^0 = I,$$

for all $n \in \mathbb{Z}^+$*. The time-series* Y_{t+n} *is said to be have lead of* n*.*

Example 4.5. *Let* $Y_t = (1, 2, 3)$*, then*

$$\begin{aligned} LY_t &= Y_{t-1} \\ &= (2, 3, 0) \\ FY_t &= Y_{t+1} \\ &= (0, 1, 2). \end{aligned}$$

Notice that we can define differencing in terms of lag operators

$$\begin{aligned} \Delta Y_t &= Y_t - Y_{t-1} \\ &= Y_t - LY_t \\ &= (1 - L)Y_t \end{aligned}$$

where 1 is treated as the identity operator on Y_t, i.e $1Y_t = Y_t$.

Definition 4.6. *Let* $\{Y_t : t = 1, \ldots, T\}$ *be a time-series. An* ***auto-regressive model*** *of order* p *on* Y_t *is defined to be*

$$\hat{y_t} = \hat{a_{t-1}} y_{t-1} + \ldots + \hat{a_{t-p}} y_{t-p} + \epsilon_t, \qquad = (a_{t-1}L + \ldots + a_{t-p}L^p)y_t \tag{4.2}$$

where $\epsilon_t \sim N(0, \sigma^2)$*. This model is denoted AR(p) and*

$$\phi(L) = a_{t-1}L + \ldots + a_{t-p}L^p$$

is the ***lag-polynomial****.*

With an auto-regressive model we attempt to estimate future values of the time-series by the previous p values of the time-series.

Notice that the auto-regressive model of order p gives us a recurrence relation, that is, we're assuming that y_t is a linear combination of $\{y_{t-1}, \ldots, y_{t-p}\}$. One way to analyze the stationarity of y_t is through an auto-regressive process, known as the augmented Dickey-Fuller test or the ADF test [37]. A simple form of this test is known as the Dickey-Fuller test, and starts with a regression where we assume that y_t is modeled by the previous entry in the time-series plus some trend

$$y_t = \rho y_{t-1} + \beta t + c + \phi \Delta y_{t-1} + \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(0, \sigma^2).$$

If $\rho = 0$ then this becomes $\Delta y_t = \epsilon_t$. The ADF test then has the following hypotheses:

$$N_0 : \rho = 1$$
$$N_A : \rho \neq 1,$$

that is, the null-hypothesis is $\rho = 1$, and the alternative hypothesis is that $\rho \neq 1$. If the null hypothesis is true, then

$$y_t - y_{t-1} = \beta t + c + \phi \Delta y_{t-1} + \epsilon_t \tag{4.3}$$
$$\Delta y_t - \Delta y_{t-1} = \beta t + c + \epsilon_t \tag{4.4}$$
$$\Delta^2 y_t = \beta t + c + \epsilon_t, \tag{4.5}$$

that is, the time-series is non-stationary and y_t is said to have a unit root. This is because y_t must be differenced twice to transform it to a stationary series, and even then it may not be a strictly stationary series if $\beta \neq 0$, so one more difference may be required. The full ADF test generalizes this to a larger order auto-regression

$$y_t = \rho y_{t-1} + c + \beta t + \sum_{s=1}^{k} \phi_s \Delta y_{t-s},$$

where y_{t-s} are lagged versions of y_t. The full ADF test adjusts for possible spurious conclusions if y_t is auto-correlated, while still testing for the presence of a unit root. That is, the presence of a unit root implies a process is not stationary so if our p-value satisfies $p < \alpha$, then with $(1 - \alpha)\%$ confidence we can say that y_t is not stationary. We will commonly use values of $\alpha = 0.1$ in this thesis.

Similarly the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test will be used for testing stationarity [38]. Similar to the ADF test, this test looks for the stationarity of a time-series about a possible trend. That is, the KPSS tests for

$$y_t = \beta t + c + \epsilon_t \qquad \epsilon \sim \mathcal{N}(0, \sigma^2).$$

However, this differs from the ADF test in that the null hypothesis and alternative hypothesis are

- N_0 : y_t is stationary
- N_A : y_t is not stationary.

Both these tests will be used for testing for stationarity in time-series.

Definition 4.7. *Let $\{y_t : t = 1, \ldots, T\}$ be a stationary time-series. A **moving-average model** of y_t of order q is of the form*

$$\begin{aligned} y_t &= \mu_t + \epsilon_{t-1}\theta_{t-1} + \ldots + \epsilon_{t-q}\theta_{t-q} \\ &= \mu_t + (\theta_{t-1}L + \ldots + \theta_{t-q}L^q)\theta_t \end{aligned}$$

where μ_t is the average function for the time-series and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ for all i. This is model is denoted MA(q).

Moving-average models are typically more difficult to fit to a time-series. We'll explore a simple example in Example 4.8.

Example 4.8. *To actually calculate a moving-average model, we'll show there is a connection from an AR(1) model to an MA(∞) . Using this connection, we'll discover when exactly an MA(1) model can be calculable.*

Assume that y_t is a stationary time-series and follows an AR(1) model

$$y_t = \phi_1 y_{t-1} + \epsilon_t.$$

Using this we can write

$$\begin{aligned} y_t &= \phi_1(\phi_1 y_{t-2} + \epsilon_{t-1}) + \epsilon_t \\ &= \phi_1^2 y_{t-2} + \phi_1 \epsilon_{t-1} + \epsilon_t \\ &= \phi_1^2(\phi_1 y_{t-3} + \epsilon_{t-2}) + \phi_1 \epsilon_{t-1} + \epsilon_t \\ &\vdots \\ y_t &= \sum_{i=1}^{\infty} \phi_1^i \epsilon_{t-i} + \epsilon_t. \end{aligned}$$

For this power-series to converge we must have $|\phi_1| < 1$ [2]. Similar arguments can show that this is the case for any AR(p)-model.

For an MA(1) model we can arrive at a similar result

$$\begin{aligned} y_t &= \theta_1 \epsilon_{t-1} + \epsilon_t \\ \epsilon_t &= y_t - \theta_1 \epsilon_{t-1} \end{aligned}$$

so that

$$\begin{aligned} y_t &= \theta_1(y_{t-1} - \theta_1 \epsilon_{t-2}) + \epsilon_t \\ &= \theta_1 y_{t-1} - \theta_1^2 \epsilon_{t-2} + \epsilon_t \\ &= \theta_1 y_{t-1} - \theta_1^2(y_{t-2} - \epsilon_{t-3}) + \epsilon_t \\ &\vdots \\ &= \sum_{i=1}^{\infty} (-1)^{i-1} \theta_1^i y_{t-i} + \epsilon_t. \end{aligned}$$

If this power-series converges, then we say that the MA(1) model is ***invertible****. This is the calculation used for MA(q) models in practice.*

Combining both differencing to account for non-stationary data, the AR(p) and MA(q) models give us the robust method for modeling time-series.

Definition 4.9. *Let $\{y_t : t = 1, .., T\}$ be a time-series, possibly non-stationary. The* ***Auto-regressive Integrated Moving Average (ARIMA) model of order*** *(p, d, q) is defined*

as

$$\begin{aligned}\Delta^d y_t &= c + \phi_1 \Delta^d y_{t-1} + \ldots + \phi_p \Delta^d y_{t-p} + \theta_1 \epsilon_{t-1} + \ldots + \theta_q \epsilon_{t-q} + \epsilon_t \\ \Delta^d y_t &= c + \phi(L)\Delta^d y_t + \theta(L)\epsilon_t + \epsilon_t \\ (1-L)^d y_t &= c + \phi(L)(1-L)^d y_t + (\theta(L)+1)\epsilon_t.\end{aligned}$$

This can be written as

$$(1 - \phi(L))(1-L)^d y_t = c + (\theta(L) + 1)\epsilon_t$$

where $\Delta^d y_t = (1-L)^d y_t$*,* $\phi(L)$ *is the lag-polynomial for the auto-regressive model,* $\theta(L)$ *is the lag-polynomial of the moving-average model, and* $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ *for* $t \in \{1, \ldots, T\}$*.*

This is a combination of the auto-regressive and moving average models preformed on a difference-stationary time-series.

One component we haven't spoken of so far with time-series that vary with some sort of seasonal behavior. For example a time-series of average temperature would exhibit strong seasonal behavior. Other time-series such as daily revenue for a clothing company might also exhibit this seasonal behavior, but also possibly an overall trend upwards or downwards depending on the success of the company. To account for this formally, we must decompose our time-series into different components. We can attempt to decompose a time-series y_t into four components

1. Trend: Long-term behaviour such as an overall increase or decrease.
2. Seasonal: Fluctuations that occur on a regular or semi-regular frequency, such as seasons in a temperature time-series.
3. Cyclic: Fluctuations, such as a rise and fall, that do not occur on a regular frequency.
4. Residual: Random effect or noise.

Suppose that x_t is a time-series. An **additive decomposition** is finding seasonal S_t, trend T_t, cyclic C_t and residuals R_t components such that

$$x_t = S_t + T_t + C_t + R_t.$$

With ARIMA(p,d,q) models, the base assumption of the model is that the data can be differenced to become stationary using first-order differences only; that is $\Delta y_t = y_t - y_{t-1}, \Delta^2 y_t = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) = y_t - 2y_{t-1} + y_{t-2}, \ldots$ will become stationary after enough differences. This is not necessarily the case with the data that exhibits a strong seasonal component [2]. To account for this, we modify the ARIMA model by accounting for seasonal differencing.

Definition 4.10. *Let $\{y_t : t = 1, \ldots, T\}$ be a time-series, possibly non-stationary. The* ***Seasonal Auto-Regressive Integrated Moving Average (SARIMA) model of order*** $(p, d, q)(P, D, Q)_m$ *is defined as*

$$(1 - \phi(L))(1 - \Phi(L^m))(1 - L)^d(1 - L^m)^D y_t = \\ c + (\theta(L) + 1)(\Theta(L^m) + 1)\epsilon_t$$

where $\Phi(L^m), \Theta(L^m)$ are the seasonal-lag polynomials of order P and Q, respectively.

SARIMA $(p, d, q)(P, D, Q)_m$ models offer a robust approach to modeling time-series, but only attempts to model y_t with lagged versions of itself. To account for other possible explanatory variables we introduce a time-series model that account for explaining variables other than y_t [39].

Definition 4.11. *Let $\{y_t : t = 1, \ldots, T\}$ be a time-series, possibly non-stationary. The* ***Seasonal Auto-Regressive Integrated Moving Average with Exogenous Variables (SARIMAX) model of order*** $(p, d, q)(P, D, Q)_m$ *is defined as*

$$(1 - \phi(L))(1 - \Phi(L^m))(1 - L)^d(1 - L^m)^D y_t = \tag{4.6}$$

$$c + (\theta(L) + 1)(\Theta(L^m) + 1)\epsilon_t + \eta(L)X_t \tag{4.7}$$

where $\eta(L)$ is the lag-polynomial of our exogenous variables X_t. If the above model lacks the seasonal polynomials $\Phi(L^m), \Theta(L^m)$, this is a ***ARIMAX*** *model.*

The SARIMAX offers the support of explanatory, or exogenous, variables to assist in the forecasting of future values of y_t. However, often with SARIMA or SARIMAX models, to fit the seasonal-order of the model $(P, D, Q)_m$ this increases the dimension of the parameter space being searched which leads to computational challenges [40]. To circumvent this

problem, we can instead fit a Fourier series to the seasonal component of time-series

$$S(t) = \sum_{k=1}^{N} a_k \cos\left(\frac{2\pi kt}{m}\right) + \beta_k \sin\left(\frac{2\pi kt}{m}\right)$$

and treat this as an exogenous variable for the time-series y_t. With the Fast-Fourier transform this can be done very quickly and eliminates searching over the parameter space for $(P, D, Q)_m$, with fixed m [41].

4.3 Causality and Cross-Correlation

The process of determining whether two time-series are interrelated is quite an important and complex task, but quite an important process when determining exogenous variables for a SARIMAX model. An even more complex problem is determining if one time-series predicts another time-series. The underlying question that we seek to ask is: does a time-series X_t and Y_t have a 'cause-and-effect' relationship? In this section, we'll discuss two such metrics that can assist in this analysis.

The first such measure of a 'causal' relationship between two time-series that we'll discuss is that of Granger causality [42]. Granger causality attempts to find any relationship between time-series X_t and Y_t, such that X_t can help in the prediction of Y_t; formally, Granger causality attempts an AR-model of the following form:

$$Y_t = a_1 Y_{t-1} + \ldots + a_p Y_{t-p} + b_1 X_{t-1} + \ldots + b_q X_{t-q} + \epsilon_t.$$

From here an F-value is calculated for the coefficients $\{b_i : i \in \{1, \ldots, q\}\}$ and if the F-value is above some critical value, then we say that X_t G-causes Y_t [42]. While the name 'Granger causality' might imply we're capturing causality with this measure, we are simply capturing predictive power. Granger causality gives us an idea of how X_t influences Y_t in the sense of modeling the two time-series. There are well documented problems with Granger causality, especially in the case where both X_t and Y_t are found to 'G-cause' each other, but it still is a primary tool in multi-variate time-series analysis for a first pass at feature selection.

To evaluate these regressions a common measure used in time-series analysis is the Bayesian Information Criterion (BIC), or Schwarz Information Criterion, defined below.

Definition 4.12. *Let M be a model under consideration. The* ***Bayesian Information Criterion*** *(BIC) is defined to be*

$$BIC = -2\ln(\hat{L}) + k\ln(n)$$

where $\hat{L}$ is the likelihood function of the model $p(x|\hat{\theta}, M)$, $\hat{\theta}$ are the parameters associated with M, k is the number of parameters of M, and n is the size of the data set.

A more familiar approach can be obtained through a simple correlation of X_t and left-shifted Y_t, $Y_{t+\tau}$, this is *cross-correlation.*

Definition 4.13. *Let $\{x_t : t = 1, \ldots, T\}$ and $\{y_t : t = 1, \ldots, T\}$ be two stationary time-series. The* ***cross-correlation function*** *between x_t and y_t is defined by*

$$\rho_{xy}(\tau) = \frac{\sum_{i=1}^{T} y_{i+\tau} x_i}{\sqrt{Var(x_T)}\sqrt{Var(y_T)}},$$

where $y_{t+\tau}$ is a lead on y_t, and we define $y_m = 0$ if either $m < 0$ or $m > T$.

For the denominator to standardize the cross-covariance, this necessitates that the data is stationary. If it is not, then this normalization will not remain constant and we lose the fact that $\rho_{xy}(\tau) \in [-1, 1]$ along with the interpretation of $\rho_{xy}(\tau)$ as a true correlation measure between x_t and $y_{t+\tau}$. Moreover, a number of papers in a variety of fields point out the dangers of using cross-correlation when using time-series on two time-series that are significantly auto-correlated when conducting a trend-analysis [43, 44, 45, 46, 47]. To adjust for this, we utilize the method outlined in [48], described below.

Algorithm 4.14.

1. *Fit an SARIMA model to x_t and y_t using* `auto_arima`.
2. *For the model with the lowest BIC between x_t and y_t, store the residuals. Without loss of generality, suppose this is the SARIMA model fitted to y_t:*

$$(1 - \phi(L))(1 - \Phi(L_m))(1 - L)^d(1 - L_m)^D y_t = c + (\theta(L) + 1)(\Theta(L_m) + 1)\epsilon_t \quad (4.8)$$

3. *Store the residuals of (4.8) as $\hat{y}_t$, such that $\hat{y}_t \sim \mathcal{N}(0, \sigma^2)$ for all t.*

4. Use the model 4.8 to filter x_t and store the result as $\hat{x_t}$

$$(1-\phi(L))(1-\Phi(L_m))(1-L)^d(1-L_m)^D x_t = \\ c + (\theta(L)+1)(\Theta(L_m)+1)\epsilon_t.$$

The last step should have the effect of reducing $\hat{x_t} \sim \mathcal{N}(0, \sigma^2)$ for all t. The intention of this algorithm in trend-analysis is to erase any spurious correlation that may be occurring because of auto-correlation by filtering the two time-series through a SARIMA model, thus reducing the two series to white-noise but still retaining characteristics of the original series. Code for this in python is provided in Appendix A.

Chapter 5

PREPROCESSING

In this chapter, we'll describe the source of the data used in this thesis, how it was obtained, how it was cleaned, and how missing or incoherent values were replaced.

5.1 Description of Data

The data used was obtained from the Center of Disease Control of the United States of America (CDC), and in particular the "United States COVID-19 Cases and Deaths by State over Time" data set was used [49]. This data is updated twice daily and covers the jurisdictions of all 55 US states, territories, and commonwealths, 3 freely associated states (Federated States of Micronesia, Republic of Marshall Islands, and Republic of Palau) as well as Washington D.C., with a total of 60 reporting jurisdictions; New York City is treated as its own jurisdiction. The time covered by the data set goes from 2020-01-21 to 2022-04-17, the day the first confirmed COVID-19 case was found in the US, to the present. This is an aggregated data set determined by cases and deaths reported to the CDC, adjustments do appear retroactively in cases and deaths. However, any adjustments for a day are retroactively included in that days counts.

This data is collected by the CDC through the National Notifiable Diseases Surveillance System (NNDSS), which acts as a common reporting system for US jurisdictions on COVID-19 cases and deaths. The cases reported to the NNDSS include confirmed and probable cases, although probable cases were not included in the analysis. The reporting mechanisms for cases of COVID-19 by jurisdictions to the NNDSS is typically done by reporting the date that this case information was reported to the health department, however some jurisdictions report the date of submission to the NNDSS as the case date. Furthermore, 17 reporting jurisdictions use a combination of the previous methods as well as approximate date of diagnosis or first positive test as well. The reporting mechanisms for deaths related to COVID-19 via NNDSS are done in a similar fashion; that is, using the reporting date as the date submitted to NNDSS. Florida is the only jurisdiction that reports the date of death of a patient as the recorded date to the NNDSS. Additionally reporting on Saturdays and

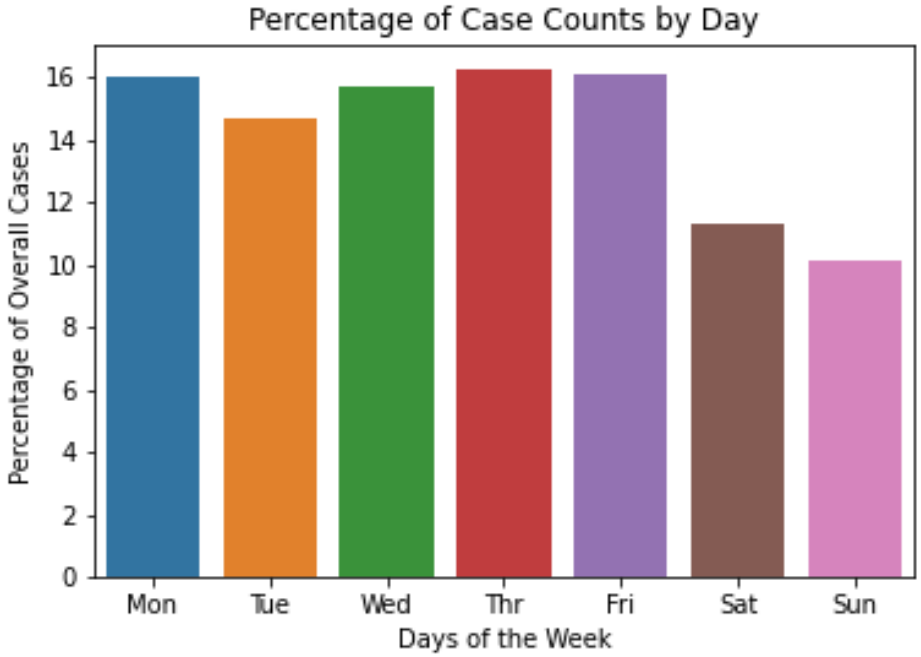

Figure 5.1: Total percentage of covid-19 cases by weekday

Sundays tend to be somewhat sparser than weekdays because of the reporting mechanisms, as can be seen in Figure 5.1. Due to this inconsistency, our analysis will be preformed on a 7-day moving average of new cases in jurisdictions, similar to studies in [50]. This has the effect of smoothing out the curve. While this might have an effect on the time-series analysis, this adjusts for non-reporting on weekends and holidays. The occasional negative values for new cases are interpolated as we'll discuss in the next section; this is done before a moving average is applied to the time-series.

We'll go through the columns used and not used in the analysis, and why certain columns were chosen over others. This data contains 15 columns and is downloaded via the CDC's Socrata web interface in a '.csv' file format. Theses columns include:

- '*state*': the reporting jurisdiction abbreviation; this is used as the ID for the reporting jurisdiction.
- '*submission_date*': the reporting jurisdiction's reporting date. This is the date that the jurisdiction's health department collected the information or this was reported by the health department to the CDC (note: a number of jurisdictions attempt to

combine this date with the actual date of a confirmed laboratory cases or diagnosis. However, no jurisdiction reports solely on the date of diagnosis or confirmed laboratory case). This was used as the date ID for COVID-19 cases and deaths.

- '*tot_cases*': the cumulative number of confirmed and probable cases going back to the beginning of the data set to the *submission_date*. This data was not consistent among jurisdictions and not used in analysis.

- '*new_case*': total number of confirmed new COVID-19 cases in the reporting jurisdiction on the *submission_date*. This was used as the daily count of COVID-19 cases in the analysis.

- '$conf\_cases$', '*prob_cases*', '*pnew_case*':

  '$conf\_cases$' and '*prob_cases*' were cumulative counts of confirmed and probable cases of COVID-19 for the *submission_date*. These column was dropped in the analysis, as the values given were often negative or empty. Similarly, *pnew_case* accounted for probable new cases of COVID-19 in the reporting jurisdiction and was summarily dropped from the analysis due to the irregularity of the data.

- '*tot_death*': cumulative number of confirmed and probable deaths related to COVID-19 for the reporting jurisdiction; because of the inclusion of probable deaths this was not used in the analysis.

- '*new_death*': total number of confirmed deaths related to COVID-19 for the reporting jurisdiction for a given day. This column was used in the analysis because of the lack of missing values and the integrity of the reporting.

- '$conf\_death$', '*prob_death*', '*pnew_death*': '$conf\_death$' and $'prob\_death'$ were cumulative counts of confirmed and probable deaths, respectively. However because of large number of missing values in both columns these were not included in the analysis. Similarly, '*prob_death*' was also dropped because of inconsistent reporting.

- '*created_at*': marked the date that the data was made available on the CDC website. This was deemed as not being useful, and not included in the analysis.

- '*consent_cases*', '*consent_deaths*': aggregation descriptions on how *tot_cases*, *tot_death* were aggregated. If these were true, then total cases/deaths included probable and confirmed cases/deaths; otherwise they did not.

To summarize, only columns 'state', 'submission_date', 'new_case', and 'new_death' were analyzed due to all values being present and their nature of counting only confirmed cases and deaths. Despite the inconsistencies in the reporting of confirmed COVID-19 cases and deaths, this data was chosen over other similar data sets because of the directness of the reporting, the relative lack of data smoothing and cleaning, and the daily updates.

These characteristics meant that a meaningful time-series analysis could be preformed on this data as opposed to other data sets that typically smoothed the data or aggregated by month or week. Aggregation by jurisdiction was decided as it allowed for a more complex graph structure that might have been too computationally complex when done on the basis of counties.

5.2 Interpolation

Some inconsistencies in the data, such as negative case numbers and obvious adjustments in the cumulative COVID-19 cases and deaths, were observed. In this section we'll discuss how these challenges were addressed. Counts and total number of negative cases can be seen in Figures 5.2 and 5.3, respectively. After applying a 7-day rolling average to the data

$$X_t = \frac{1}{7}\sum_{\tau=0}^{6} X_\tau$$

a number of data points remained negative. To address this issue, we removed these negative data points and interpolated them back based on $SARIMA(p,d,q)(P,D,Q)_m$ model fit to the data. Additionally, because of the nature of the correlations to be covered in later chapters, it was important to have complete data for each jurisdiction under consideration.

Based on a review of time-series interpolation [2, 51], the measure of fitness was chosen to be the Bayesian Information Criterion. Fitting our model, we will seek to minimize BIC with respect to the parameters to fit. In this case, M = $SARIMA(p,d,q)(P,D,Q)_m$ and $(p,d,q)(P,D,Q)_m$ are the hyper-parameters that we're seeking to optimize over. Based on observations that were seen in the data (before applying a rolling average), we found that

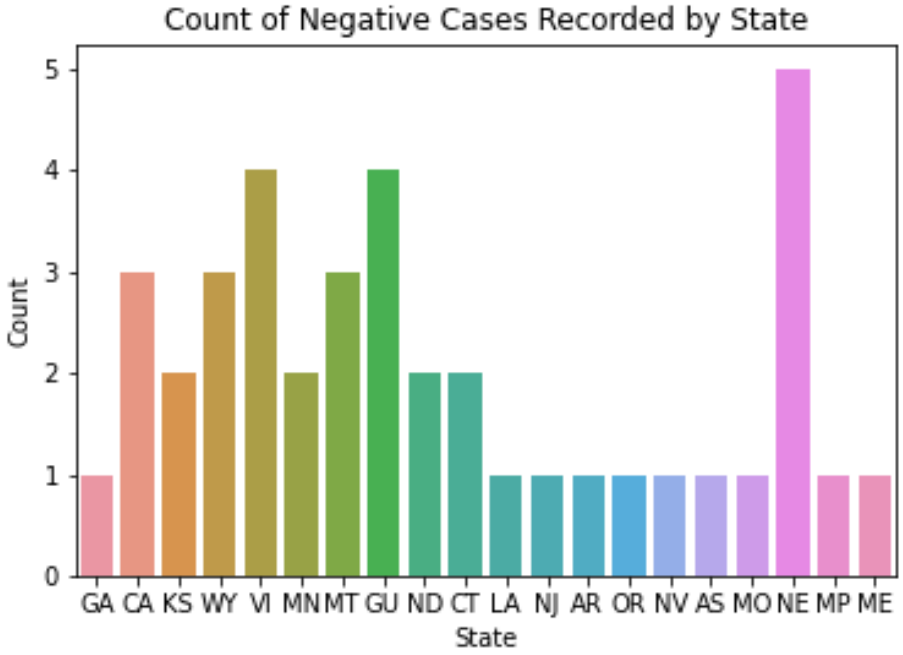

Figure 5.2: The occurrences of negative case counts by state.

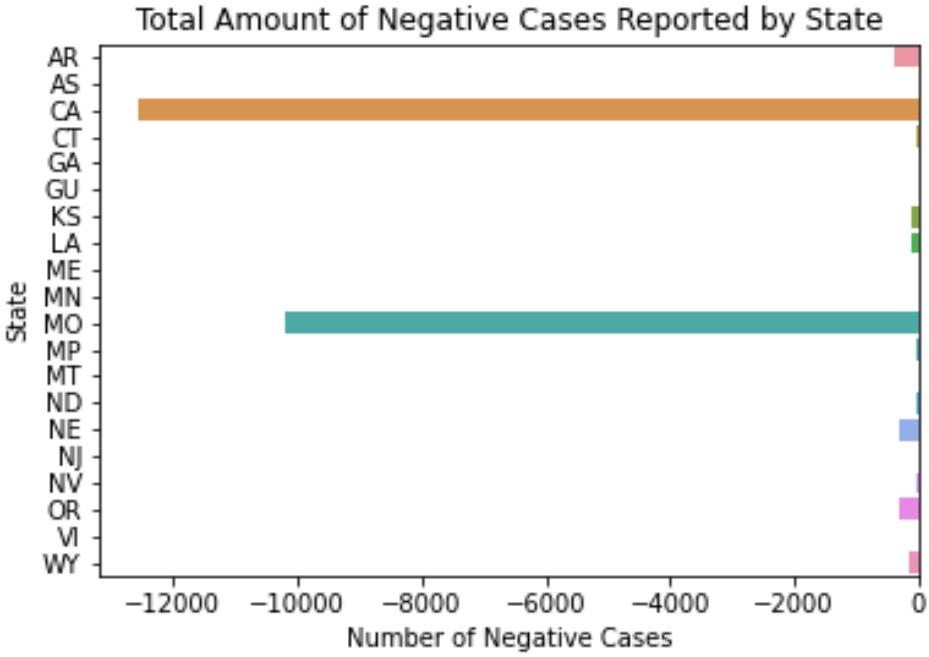

Figure 5.3: Total number of negative cases reported by state.

the data typically presented a periodicity of $m = 7$. Since our goal is minimize the BIC

$$\arg\min_{\hat{L},k} \left\{ -2\ln(\hat{L}) + k\ln(n) \right\},$$

the objective will be to maximize $\ln(\hat{L})$ while minimizing $\ln(n)k$; this is a form of maximum-likelihood estimation, since maximizing $\ln(\hat{L})$ is equivalent to maximizing $\hat{L}$. The $\ln(n)k$-term acts as a penalization or regularization term for the model, as this disincentives an overly complex model with a large number of parameters; to minimize $k\ln(n)$, keep the number of parameters to a minimum. We'll be using BIC in a number of capacities in this thesis. Here we will be applying it to the testing region for our interpolation model.

Finally, to keep the values positive for the forecasting the following transformation was applied to the data: for each data point x, evaluate

$$f(x) = \log(x + 2).$$

Then to smooth the variance and mean we use the box-cox transformation

$$h(x) = \frac{x^{\lambda} - 1}{\lambda},$$

where λ is a parameter chosen to maximize the log-likelihood function of the data set.

Since $\log(x + 2) > 0$ for $x \geq 0$, the forecasts will be strictly non-negative; note the '+2' accounts for issues forecasting a series where $X_t = 0$ for all t, which occurs for series that are all zero after the log transformation. For model-selection, that is the choice of $(p, d, q)(P, D, Q)_7$, we use the `auto_arima` function to fit the models in the `pmdarima` python package [52].

The models selected and their associated BIC on the data can be seen in Table 5.2 and an example for the state of Nebraska (NE) can be seen in Figure 5.4

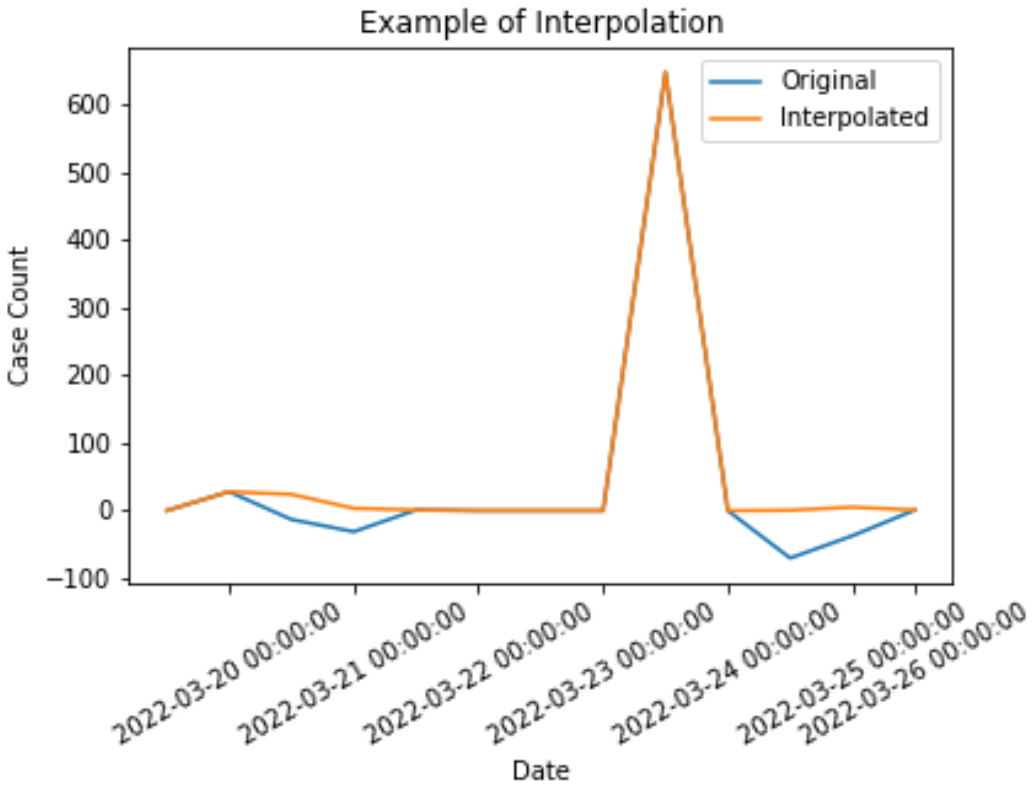

Figure 5.4: Interpolation around a set of points for Nebraska

Table 5.1: Interpolation Model and their resulting BIC's.

	state	model	BIC
0	GU	SARIMAX(1, 2, 4)x(1, 0, [1], 7)	3374.799
1	HI	SARIMAX(1, 2, 4)x(1, 0, [1], 7)	3199.651
2	IA	SARIMAX(5, 2, 0)x(2, 0, [1, 2], 7)	3387.325
3	ID	SARIMAX(5, 2, 0)x(2, 0, [1, 2], 7)	3470.185
4	IL	SARIMAX(3, 2, 4)x(1, 0, [1], 7)	3899.843
5	IN	SARIMAX(5, 2, 0)x(2, 0, [1, 2], 7)	3945.753
6	CA	SARIMAX(1, 1, 2)x(2, 0, [1], 7)	874.103
7	CO	SARIMAX(1, 1, 2)x(2, 0, [1], 7)	903.312
8	CT	SARIMAX(1, 1, 2)x(2, 0, [1], 7)	885.835
9	DC	SARIMAX(1, 1, 2)x(2, 0, [1], 7)	849.445
10	DE	SARIMAX(1, 1, 2)x(2, 0, [1], 7)	919.596
11	FL	SARIMAX(1, 1, 3)x(1, 0, [1], 7)	1131.854
12	GA	SARIMAX(3, 1, 2)x(1, 0, [1], 7)	1165.020
13	MO	SARIMAX(1, 1, 4)x(1, 0, [1, 2], 7)	4278.864
14	MP	SARIMAX(1, 1, 4)x(1, 0, [1], 7)	2099.196
15	MS	SARIMAX(1, 1, 3)x(1, 0, [1], 7)	1364.323
16	MT	SARIMAX(1, 1, 3)x(1, 0, [1, 2], 7)	1380.815
17	NC	SARIMAX(3, 1, 2)x(1, 0, [1], 7)	1483.792
18	ND	SARIMAX(1, 1, 3)x(1, 0, [1], 7)	2151.554
19	NE	SARIMAX(1, 1, 3)x(1, 0, [1], 7)	2416.559

Chapter 6

A NETWORK MODEL OF COVID-19

First outlined in [1] a network theoretic analysis of COVID-19 based on running correlations between new cases in jurisdictions in China was proposed, then this was applied to an international level between countries. This was done by calculating a Pearson-r correlation coefficient on a 14-day rolling window of the back-differenced data: $X_t := \sqrt{X_t} - \sqrt{X_{t-1}}$. A network was constructed by treating the provinces as nodes and if the correlation coefficient was greater than 0.5 for a 14 day interval, then theses states were joined by an edge. Let $X_{i,t}$ and $X_{j,t}$ be the number of COVID-19 cases in states i, j, respectively, at time t, then let

$$r_{i,j}(t;\tau) = \frac{\sum_{t=0}^{\tau-1} X_{i,t}X_{j,t}}{\sqrt{\mathrm{Var}[X_i]\mathrm{Var}[X_j]}}$$

be the Pearson-r correlation coefficient between $\{X_{i,t-k}\}_{k=0}^{\tau}$ and $\{X_{j,t-k}\}_{k=0}^{\tau}$. We define the adjacency matrix of G to be

$$A_{i,j} = \begin{cases} 1 & \text{if } r_{i,j}(t;\tau) \geq r \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

for some $\alpha \geq 0$.

Variants of this approach have been adapted and used to analyze a variety of differing topics related to COVID-19 [1, 53, 54, 4]. Authors have found that certain topological features might be good indicators of pandemic risk and financial risk associated with COVID-19 [1, 4]. To that end, we'll consider a number of different configurations of this networks with differing values of τ and r and see to what extent network statistics can give us an insight on possible rises in COVID-19 cases. In particular, we'll seek to address the question: can this network help in the prediction of COVID-19 cases in the United States?

6.1 Centrality Analysis

Constructing the network similar to that used in [1] with jurisdictions in the United States, we set $\tau = 14$ and $r = 0.5$ with the data backwards differenced and transformed for stationarity, $X_t := \sqrt{X_t} - \sqrt{X_{t-1}}$, we define the adjacency matrix as given in (6.1).

In this section, we'll study the cross-correlation between network statistics and overall trends in COVID-19 in the US for a number of different network configurations, with the goal being to obtain a network construction that acts as a reliable indicator of national and state-wide COVID-19 trends. We address the questions

- Do network statistics hold additional information about COVID-19 surges?
- What is the optimal configuration of the network for the purposes of predicting possible spikes in COVID-19?

We use a 95% confidence interval for statistical significance of our cross-correlation coefficients. For the entire pandemic this is $\pm\frac{1.96}{\sqrt{T}} \approx \pm 0.06$. We construct a variety of networks for $\tau = 14, 30, 90, r = 0.5, 0.9, p = 0, 0.5, 1$, where p is the order of differencing $X_t := X_t^p - X_{t-1}^p$, the results can be seen in Figures C.1 through C.18. The node size and color are proportional to the nodes sub-graph centrality, with the edge widths proportional to their betweenness centrality. A sample video can be seen in the video at [55] for network configuration $p = 1, \tau = 90, r = 0.9$.

With many of the $r = 0.5$, at different points in the pandemic the graph becomes very connected and dense. With such a low threshold value, network statistics are unlikely to give good patterns of COVID-19 spread as these networks are largely homogeneous, similar to an Erdos-Reyni model. So the result that few $r = 0.5$ statistics appear to be significantly correlated with overall COVID-19 cases in the United States is unsurprising, Figure 6.1. On the other hand, as might be expected, the threshold value of $r = 0.9$ causes the network to become sparse in most scenarios allowing for more dynamic network statistics. As can be seen in Figure 6.2.

These top preforming metrics have good predictive power as determined by cross-correlation. Notably the model configuration of $p = 1, r = 0.9$ and window $= 90$ seems to suit itself well to short-lag correlations meaning that this network configurations and net-

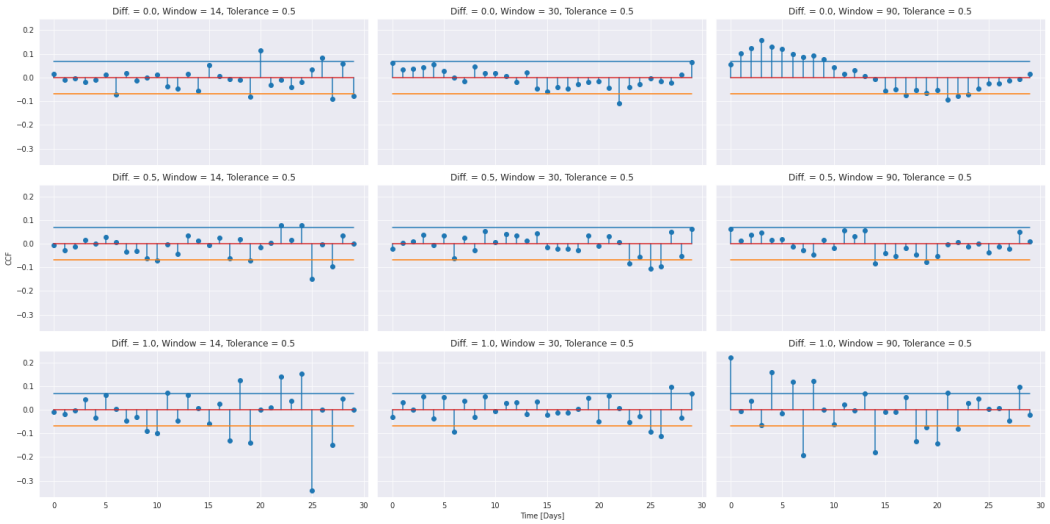

Figure 6.1: Mean sub-graph centrality ccf coefficients with US Cases versus CCF lag, $r = 0.5$

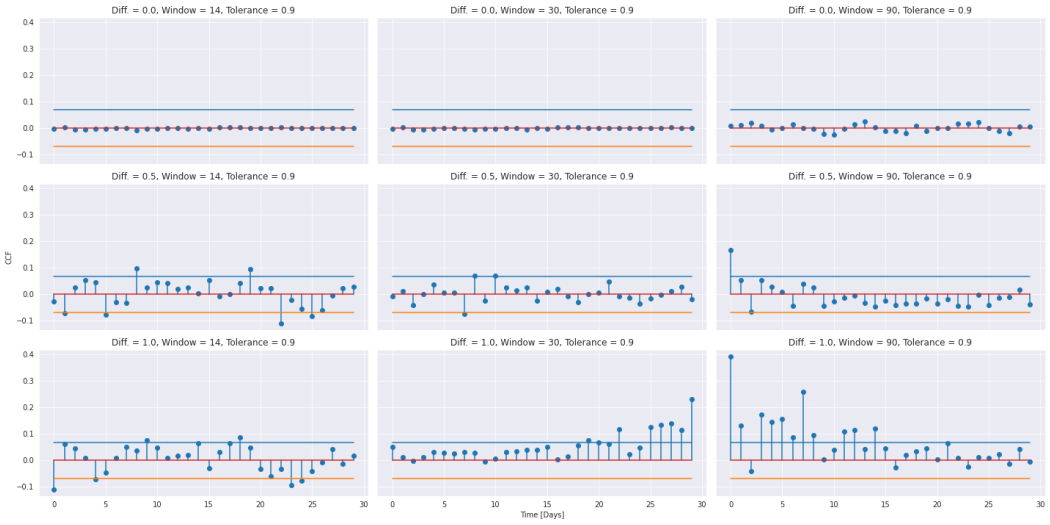

Figure 6.2: Mean sub-graph centrality ccf coefficients with US Cases versus CCF lag, $r = 0.9$

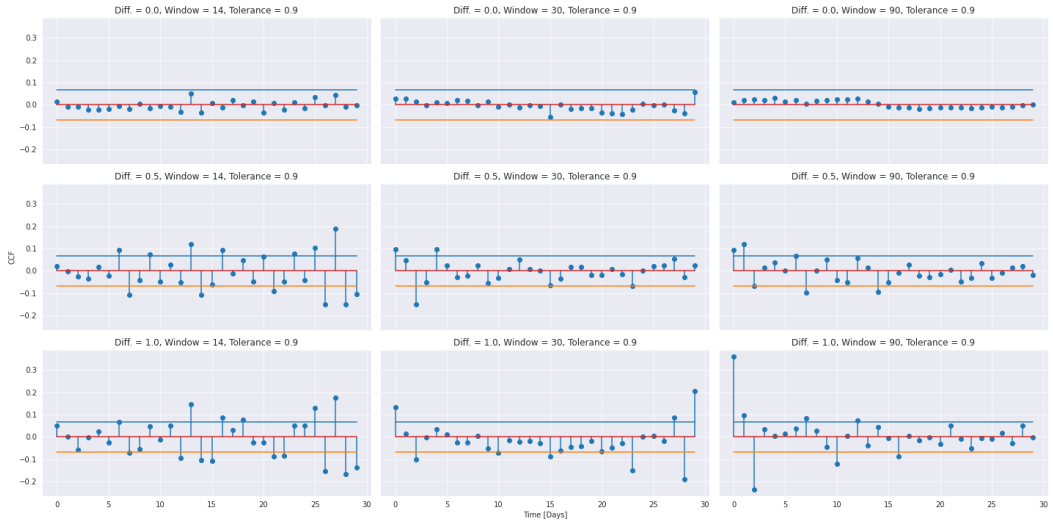

Figure 6.3: S-Metric ccf coefficients with US Cases verse CCF lag, $r = 0.9$

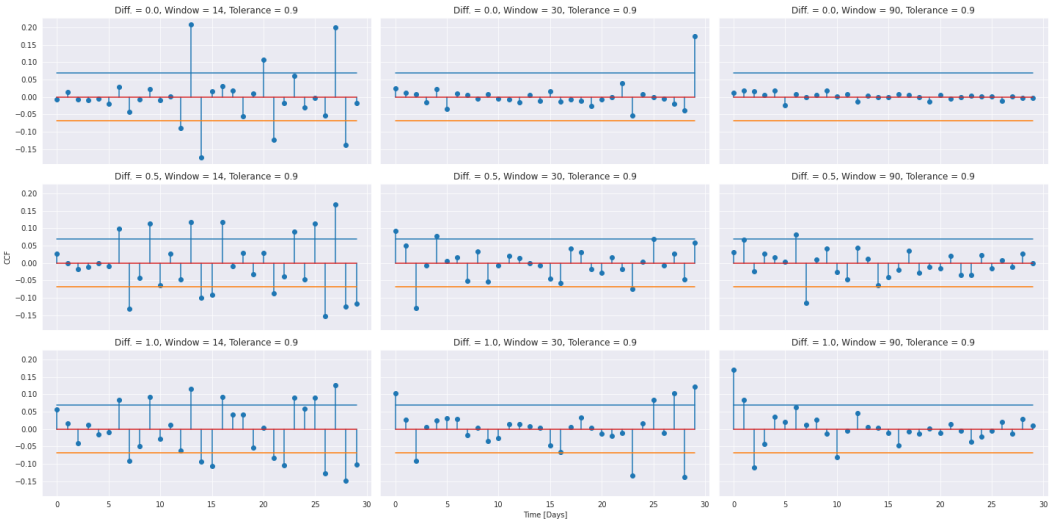

Figure 6.4: Mean core number ccf coefficients with US Cases verse CCF lag, $r = 0.9$

work statistics could be good candidates for visualizing the geographical spread of COVID-19 in the US, Figures 6.4 6.3 6.2. Our analysis suggest that

- Mean sub-graph centrality, the s-metric, and mean core number have the strongest cross-correlation coefficients with US COVID-19 cases
- The model configuration of $p = 1, \tau = 90, r = 0.9$ consistently shows significant cross-correlation with US COVID-19 cases at lag 0 for a variety of network statistics
- Tolerance values of $r = 0.5$ have weak correlation compared to network statistics with $r = 0.9$

6.2 Network Informed Time-Series Model

Using the network construction in the previous section, we propose a network-informed model for COVID-19 cases in the United States, Figure 6.5. By taking four top preforming statistics outlined in Section 6.1, average core number, s-metric, average sub-graph centrality, and average harmonic centrality (largest clique size was excluded due to time constraint), we'll explore in what circumstances these can assist in the forecasting of overall COVID-19 cases in the United States. The time-series of these data points can be seen in Figures 6.7 and 6.8.

Let $G(t)$ be our network construction at time t and y_t be overall COVID-19 cases in the US. We propose a SARIMAX model of the form

$$(1 - \phi(L))(1 - \Phi(L^m))(1 - L)^d(1 - L^m)^D y_{i,t} =$$
$$c + (\theta(L) + 1)(\Theta(L^m) + 1)\epsilon_{i,t} + \eta(L)X_{i,t}$$

where $X_{i,t}$ are network statistics and Fourier-features. In this approach we considered a combination of different exogenous variables based on the network structure:

- The average sub-graph centrality of $G(t)$
- The average harmonic centrality of $G(t)$
- The average core number of $G(t)$
- The s-metric of $G(t)$.

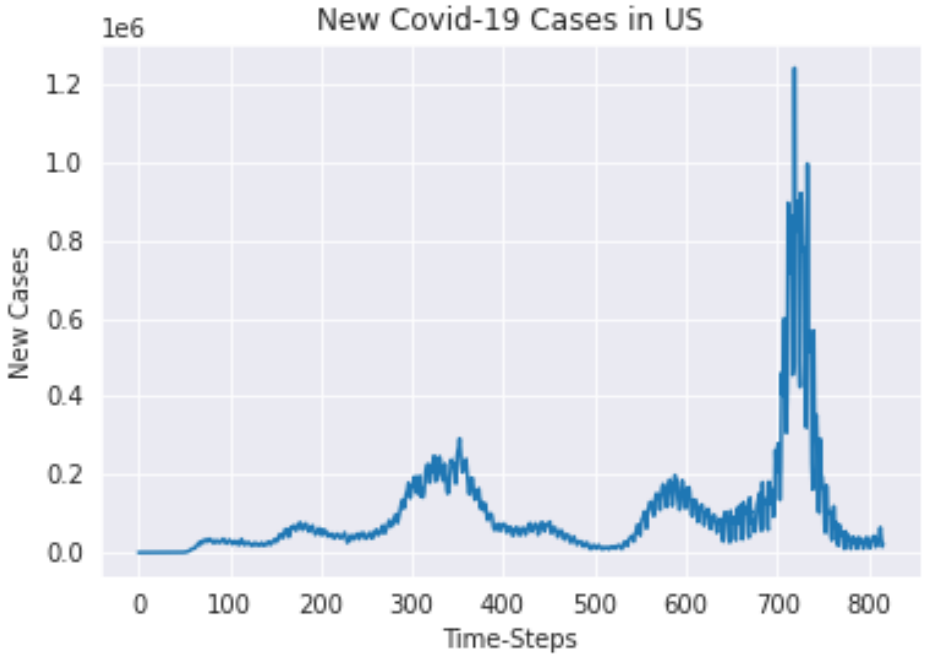

Figure 6.5: New Cases of COVID-19 in US.

As a baseline of comparison, we'll be comparing the results of this SARIMAX model with that of a SARIMA model, both fitted using the `auto_arima` function from the `pmdarima` library [52].

We compare the test set errors for 8 time steps over the course of the pandemic, testing a number of configurations of the network construction with varying values for α, p, training window T, and forecast/test window h. We test the SARIMAX method only with one statistic at a time to gauge the effectiveness of each variable in COVID-19 forecasting. In addition, for error analysis we use a time-series training and test split as described in Figure 6.9, for training/test windows: $\{(14,1),(14,3),(30,1),(30,7),(90,7),(90,14)\}$.

We plot the root-mean squared error (RMSE) on the test set against time-steps ($t = 0$ is 01/22/2020 and $t = 816$ is 04/17/2022),

$$\text{RMSE} = \sqrt{\frac{1}{h}\sum_{i=1}^{h}(\hat{y}_i - y_i)^2},$$

where $\hat{y}$ is the predicted value of y and y is the actual value on the test set.

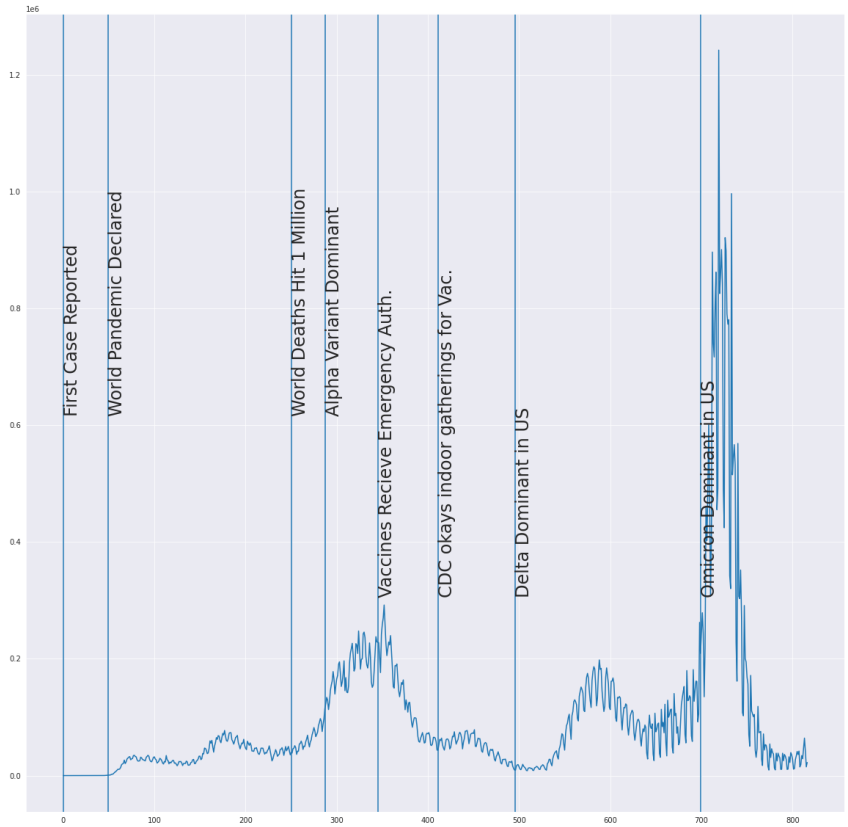

Figure 6.6: New Cases of COVID-19 in US, with major events.

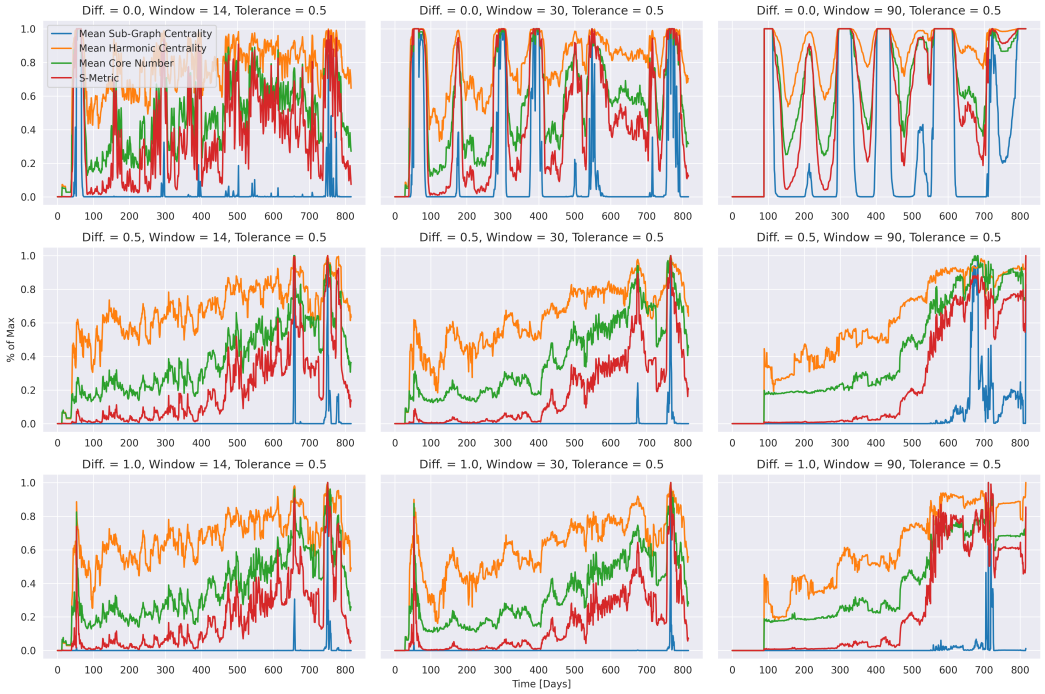

Figure 6.7: Network Statistics with $r = 0.5$

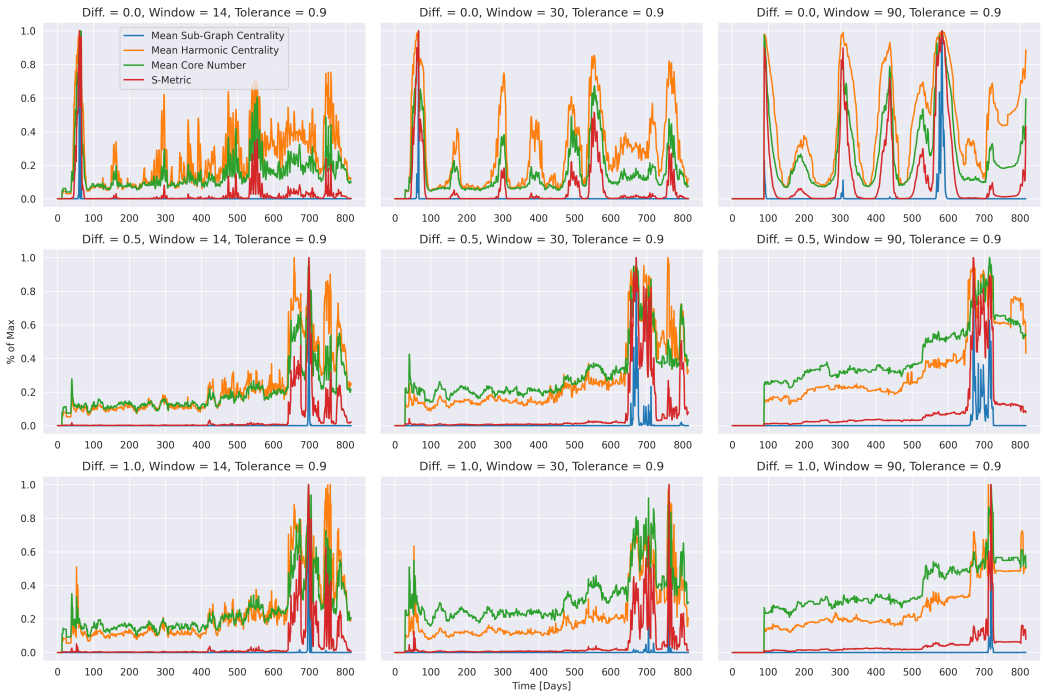

Figure 6.8: Network Statistics with $r = 0.9$

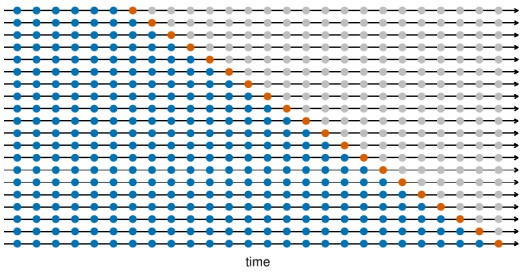

Figure 6.9: Time-series cross-validation scheme, image courtesy of [2]. Orange are Testing Sets, Blue are Training Sets.

For a majority of the network configurations we saw that with network statistics, the SARIMAX model outperform the baseline SARIMA model for time steps $t = 300, 700$, which correspond to the Alpha and Omicron surges in COVID-19 cases in the US. Suggesting that with accurate network statistics a SARIMAX model can help in the prediction of COVID-19 cases during large spikes in COVID-19, Figures D.10 D.8 D.11.

The statistic with the consistently worse performance is sub-graph centrality, this is counter to our hypothesis that sub-graph centrality's large cross-correlation may indicate better predictions with a SARIMAX model for COVID-19 in the US. It should be noted that the SARIMAX model relies on predictions for the network statistics, as the test set is perfect information for the network statistics. This means that a reliable model for these network statistics is required, so either a time-series model or network model must be developed for the network statistics to be utilized in this method for the SARIMAX model.

6.3 Community Detection with COVID-19

By the graph construction outlined at the beginning of this chapter, the edges of our network encode the similarity of COVID-19 trends from state to state. This encodes no explicit information about causal relationships, although may act as a proxy for such relationships. The graphical information that we retrieve from these networks tell us characteristics of the topology of the graph, and thus of COVID-19 trend similarities in the US. Thus it is natural that clusters in this graph coincide with regions of high inter-connectivity with respect to their trends in COVID-19, and hence the policies in one jurisdiction may work well in another jurisdiction. Hence the task of community detection in these networks may

Figure 6.10: The Time-Series of Modularity of $G(t)$ with Louvain Algorithm, $r = 0.5$

well lend itself to determining common response regions when responding to outbreaks or surges of COVID-19.

Before we apply community detection to these networks we test to see how the modularities of these graphs correspond to overall COVID-19 cases in the US; that is, do dips or spikes in the modularity correspond to overall spikes or dips in COVID-19? We will test this hypothesis with one such community detection algorithm outlined in Chapter 3, the Louvain algorithm.

Comparing the modularities obtained from the Louvain algorithm with new cases in the US we get the ccf-coefficients for lags, i.e $\text{ccf}_{XY}(\tau)$, where X is modularity and Y is new cases of COVID-19 in US (with order-p differencing) in Figures 6.12 and 6.13.

The cross-correlation coefficients for $r = 0.5$ are almost all below levels of statistical significance. This may be due to the fact that the networks with this threshold value typically display low levels of modularity, as seen in Figure 6.10. This in turn may be due to the fact that the graph is amalgamating into one large cluster for small $r = 0.5$. In Figures E.1, E.2, E.3 we plot the top three communities by size. The density is noticeable, as many of these graphs have only one or two communities. We see sparser behavior for graphs with similar configurations, Figures E.4, E.5, E.6.

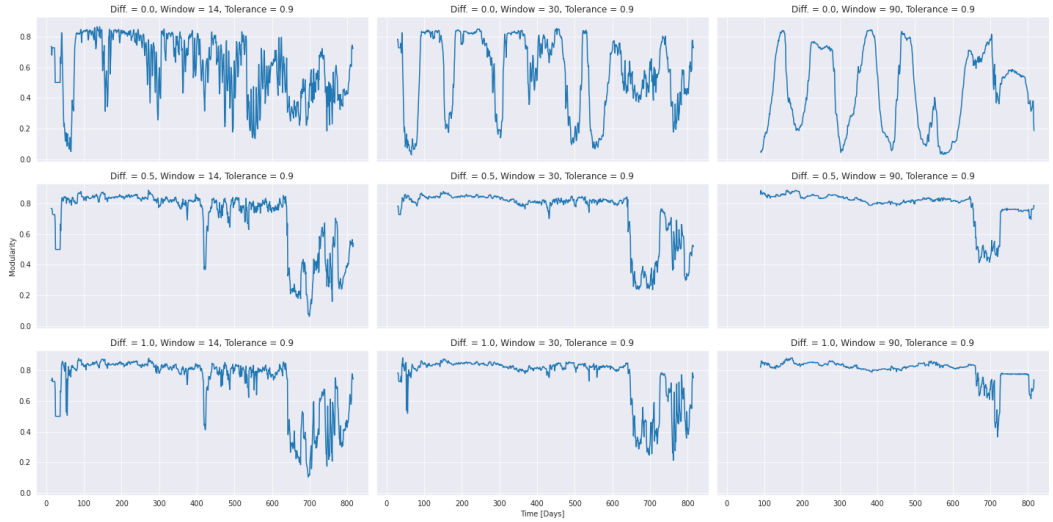

Figure 6.11: The Time-Series of Modularity of $G(t)$ with Louvain Algorithm, $r = 0.9$

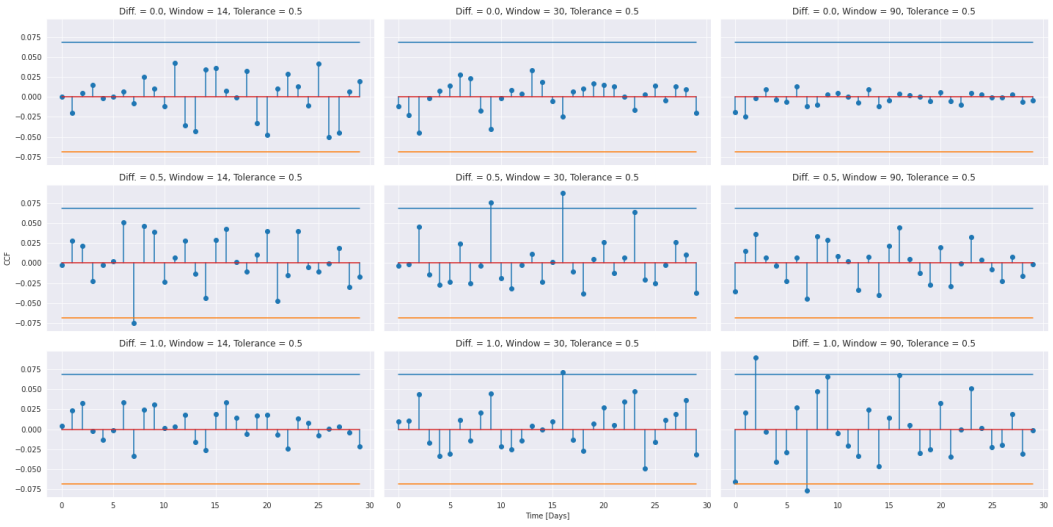

Figure 6.12: The Cross-Correlation Coefficients for $r = 0.5$ versus CCF lag Blue and orange lines are the 95% confidence intervals for statistical significance

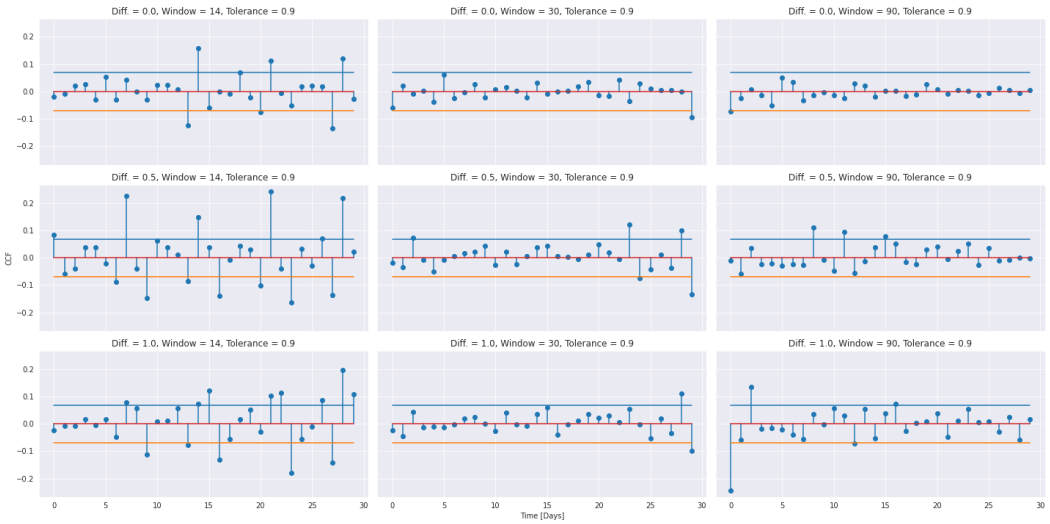

Figure 6.13: The Cross-Correlation Coefficients for $r = 0.9$ versus CCF lag Blue and orange lines are the 95% confidence intervals for statistical significance

For the tolerance level of $r = 0.9$, modularity shows higher levels of cross-correlation, with many peaks occurring for a differencing-order of $p = 0.5$ and the largest peak occurring with the network configuration $p = 0.5, \alpha = 0.9$, window $= 14$. We see this in Figure 6.11, with the spike around $t = 700$, corresponding to the spike in Omicron cases in the United States, with a major dip around $t = 400$ just after the drop in new cases after the first wave of vaccines began to be dispersed. In this configuration the increases in community structure under partitions obtained via the Louvain algorithm may lead as indicators of predictions of spikes in COVID-19 cases in the United States. Additionally, there's one considerable peak at 0 lag with the network configuration of $p = 1.0.\alpha = 0.9$, window $= 90$. Indicating that this configuration's modularity may also be an immediate indicator of the decrease of COVID-19 in the US; that is, decreases in modularity are correlated to spikes in COVID-19.

Chapter 7

CONCLUSION AND FURTHER RESEARCH

We have conducted a through analysis of the network construction given by the adjacency matrix defined in Equation 6.1, as well as lay out a framework for a rigorous analysis of centrality measurements and time-series models informed by network statistics via cross-correlation. The three questions we asked with our analysis were:

1. Are network statistics correlated with COVID-19 cases in the United State?
2. Can this network help in the prediction of COVID-19 cases?
3. What is the optimal configuration of the network for the purposes of predicting possible spikes in COVID-19?

Through our analysis we can make the following conclusions

1. Mean sub-graph centrality, mean core number, and the s-metric showed the strongest cross-correlations with COVID-19 cases for the following network configurations:
   - $p = 1, \tau = 90, \alpha = 0.9$ at lags 0 and 7
   - $p = 0, \tau = 14, \alpha = 0.9$ at lags 13 and 27
   - $p = 1, \tau = 90, \alpha = 0.9$ at lags 0 and 2

   respectively.
2. Comparing a SARIMA model to a SARIMAX model with network statistics as exogenous variables, the SARIMAX model out-preformed the SARIMA model at large spikes in COVID-19 cases in the US. Namely the SARIMAX model preformed better than the SARIMA model at times $t = 300, 700$, corresponding to the Alpha and Omicron variant surges of COVID-19. Indicating the SARIMAX model with network statistics may assist in the prediction of large spikes in COVID-19.
3. We saw the best improvement for the following configurations:
   - $p = 1, \alpha = 0.9$, (Train, Test) $= (30, 1)$

- $p = 1, \alpha = 0.5$, (Train, Test) $= (30, 7)$
- $p = 1, \alpha = 0.9$, (Train, Test) $= (30, 7)$

With the results from Section 6.2, it's hopeful that a deeper network construction of COVID-19 trends can help in facilitating simple and fast time-series models for COVID-19, as well as unveil possible structure in the spread of the virus as suggested in Section 6.3. The advantages of this network construction allow for fast and easily understandable representations of COVID-19's spread and geographical distribution, as well as facilitating better decision making as determined by the clusters of the network. Additional fuzzy or other community detection algorithms could be employed to improve the accuracy of community detection in this network.

The network centrality's have proven effective in helping with the predictions of SARIMA models through inclusion via a SARIMAX model given perfect network information. However, the assumption of perfect information is quite a large one. Any deployment of this model would require the forecasting or prediction of future network structures, a non-trivial task as can be seen from the dynamic behavior of many of these statistics.

Possible solutions to this exist in applying a SARIMA or other time-series model to the time-series of these network statistics, possibly a SARIMAX model with other exogenous variables, as well as a link prediction algorithm. In this vein of link prediction, the networks we have analyzed in this thesis are generally sparse networks ($|V| \sim |E|$) with many link deletions occurring during the time-span covered. This means that a robust link prediction algorithm would need to account for new links being generated as well as links being deleted (i.e jurisdictions trends lining up and trends diverging, respectively). Additionally, introducing a variety of node and edge attributes may allow for a deeper network representation of COVID-19, possibly using the software Neo4j or python library PyTorch.

The sample sizes for our parameter spaces p, τ, α were relatively small due to computational limitations. We suggest that a further analysis of these parameter spaces be conducted. Expanding the options for p to include $\log(\cdot)$ and other transformations may allow for better performance. An analysis of the network construction for weighted and

possibly signed adjacency matrices would also provide a richer network for analysis; that is:

$$A_{i,j}(t) = \begin{cases} 1 & \text{if } r_{ij}(t) \geq \alpha \\ -1 & \text{if } r_{ij}(t) \leq -\alpha \\ 0 & \text{otherwise} \end{cases}.$$

Similarly, a weighted network given by

$$A_{i,j} = r_{ij}$$

could allow for a weighted network analysis of COVID-19. The inclusion of a network construction based on cross-correlation could facilitate multivariate time-series model selection based on the matrix

$$A_{i,j}(t;\tau) = \begin{cases} 1 & \text{if } CCF_{i,j}(t) \geq \alpha \\ -1 & \text{if } CCF_{i,j}(t) \leq -\alpha \\ 0 & \text{otherwise} \end{cases}.$$

This could lead to a deeper study of the geographical spread of COVID-19, as this adjacency matrix would necessitate a directed graph, as the CCF is not symmetric.

An important factor that we did not include in our study is the inclusion of demographic variables in the model. A number of researchers have found that groups with lower levels of food security, historical levels of inequality, and higher levels of historical discrimination have suffered disproportionately during the pandemic [56, 57]. The incorporation of this information into COVID-19 models and the analysis of how these communities are effected by COVID-19 could lead to a stronger defense against endemic waves of COVID-19 in the future.

In conclusion, networks provide a visualization and comprehension tool for very complex behaviors and moreover they come with many important features that tell us the local and global structure of the phenomena being modeled. Networks applied to COVID-19 have the potential of revealing new information about the geographic spread o f COVID-19 and in our decision making around COVID-19. The further analysis of this network configuration and other pandemic networks has deep potential and offers a font of knowledge for discovery.

BIBLIOGRAPHY

[1] M. K. So, A. Tiwari, A. M. Chu, J. T. Tsang, and J. N. Chan, "Visualizing covid-19 pandemic risk through network connectedness," *International Journal of Infectious Diseases*, vol. 96, pp. 558–561, 2020.

[2] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2018.

[3] E. Dong, H. Du, and L. Gardner, "An interactive web-based dashboard to track covid-19 in real time," *The Lancet infectious diseases*, vol. 20, no. 5, pp. 533–534, 2020.

[4] M. K. So, A. M. Chu, A. Tiwari, and J. N. Chan, "On topological properties of covid-19: Predicting and assessing pandemic risk with network statistics," *Scientific Reports*, vol. 11, no. 1, pp. 1–14, 2021.

[5] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[6] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. [Online]. Available: https://doi.org/10.21105/joss.03021

[7] J. VanderPlas, B. E. Granger, J. Heer, D. Moritz, K. Wongsuphasawat, A. Satyanarayan, E. Lees, I. Timofeev, B. Welsh, and S. Sievert, "Altair: Interactive statistical visualizations for python," *The Journal of Open Source Software*, vol. 3, no. 32, 2018. [Online]. Available: http://idl.cs.washington.edu/papers/altair

[8] S. Seabold and J. Perktold, "statsmodels: Econometric and statistical modeling with python," in *9th Python in Science Conference*, 2010.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[10] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[11] G. Rossetti, pyup.io bot, Letizia, C. Remy, dsalvaz, BAKEZQ, O. Lloyd, fossabot, and GilCampos, "Giuliorossetti/cdlib: Arthur dent," Jan. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3632345

[12] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of anthropological research*, vol. 33, no. 4, pp. 452–473, 1977.

[13] S. Papadopoulos, Y. Kompatsiaris, A. Vakali, and P. Spyridonos, "Community detection in social media," *Data mining and knowledge discovery*, vol. 24, no. 3, pp. 515–554, 2012.

[14] L.-E. Martinet, M. Kramer, W. Viles, L. Perkins, E. Spencer, C. Chu, S. Cash, and E. Kolaczyk, "Robust dynamic community detection with applications to human brain functional networks," *Nature communications*, vol. 11, no. 1, pp. 1–13, 2020.

[15] S. Paul and Y. Chen, "A random effects stochastic block model for joint community detection in multiple networks with applications to neuroimaging," *The Annals of Applied Statistics*, vol. 14, no. 2, pp. 993–1029, 2020.

[16] H. Mahmoud, F. Masulli, S. Rovetta, and G. Russo, "Community detection in protein-protein interaction networks using spectral and graph approaches," in *International meeting on computational intelligence methods for bioinformatics and biostatistics*. Springer, 2013, pp. 62–75.

[17] A.-L. Barabási, "Network science," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 371, no. 1987, p. 20120375, 2013.

[18] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner, "Maximizing modularity is hard," *arXiv preprint physics/0608255*, 2006.

[19] F. Menczer, S. Fortunato, and C. A. Davis, *A first course in network science*. Cambridge University Press, 2020.

[20] M. Newman, *Networks*. Oxford university press, 2018.

[21] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social networks*, vol. 1, no. 3, pp. 215–239, 1978.

[22] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[23] ——, "On variants of shortest-path betweenness centrality and their generic computation," *Social networks*, vol. 30, no. 2, pp. 136–145, 2008.

[24] Y. Rochat, "Closeness centrality extended to unconnected graphs: The harmonic centrality index," Tech. Rep., 2009.

[25] E. Estrada and J. A. Rodriguez-Velazquez, "Subgraph centrality in complex networks," *Physical Review E*, vol. 71, no. 5, p. 056103, 2005.

[26] S. B. Seidman, "Network structure and minimum degree," *Social networks*, vol. 5, no. 3, pp. 269–287, 1983.

[27] V. Batagelj and M. Zaversnik, "An o (m) algorithm for cores decomposition of networks," *arXiv preprint cs/0310049*, 2003.

[28] L. Li, D. Alderson, J. C. Doyle, and W. Willinger, "Towards a theory of scale-free graphs: Definition, properties, and implications," *Internet Mathematics*, vol. 2, no. 4, pp. 431–523, 2005.

[29] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[30] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[31] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[32] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on neural networks*, vol. 16, no. 3, pp. 645–678, 2005.

[33] D. A. Spielman, "Spectral graph theory and its applications," in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*. IEEE, 2007, pp. 29–38.

[34] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[35] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[36] J. D. Cryer and K.-S. Chan, *Time series analysis: with applications in R*. Springer, 2008, vol. 2.

[37] D. A. Dickey and W. A. Fuller, "Distribution of the estimators for autoregressive time series with a unit root," *Journal of the American statistical association*, vol. 74, no. 366a, pp. 427–431, 1979.

[38] D. Kwiatkowski, P. C. Phillips, P. Schmidt, and Y. Shin, "Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?" *Journal of econometrics*, vol. 54, no. 1-3, pp. 159–178, 1992.

[39] A. Jain, T. Sukhdeve, H. Gadia, S. P. Sahu, and S. Verma, "Covid19 prediction using time series analysis," in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*. IEEE, 2021, pp. 1599–1606.

[40] R. J. Hyndman and Y. Khandakar, "Automatic time series forecasting: the forecast package for r," *Journal of statistical software*, vol. 27, pp. 1–22, 2008.

[41] P. Heckbert, "Fourier transforms and the fast fourier transform (fft) algorithm," *Computer Graphics*, vol. 2, pp. 15–463, 1995.

[42] A. Seth, "Granger causality," *Scholarpedia*, vol. 2, no. 7, p. 1667, 2007.

[43] S. Razavi and R. Vogel, "Prewhitening of hydroclimatic time series? implications for inferred change and variability across time scales," *Journal of hydrology*, vol. 557, pp. 109–115, 2018.

[44] H. Fuenzalida and B. Rosenblüth, "Prewhitening of climatological time series," *Journal of climate*, vol. 3, no. 3, pp. 382–393, 1990.

[45] R. T. Dean and W. Dunsmuir, "Dangers and uses of cross-correlation in analyzing time series in perception, performance, movement, and neuroscience: The importance of constructing transfer function autoregressive models," *Behavior research methods*, vol. 48, no. 2, pp. 783–802, 2016.

[46] R. W. Katz, "Use of cross correlations in the search for teleconnections," *Journal of Climatology*, vol. 8, no. 3, pp. 241–253, 1988.

[47] M. Bayazit and B. Önöz, "To prewhiten or not to prewhiten in trend analysis?" *Hydrological Sciences Journal*, vol. 52, no. 4, pp. 611–624, 2007.

[48] M. Y. Mahan, C. R. Chorn, and A. P. Georgopoulos, "White noise test: detecting autocorrelation and nonstationarities in long time series after arima modeling," in *Proceedings 14th python in science conference (Scipy 2015), Austin, TX*, 2015.

[49] "United states covid-19 cases and deaths by state over time." [Online]. Available: https://data.cdc.gov/Case-Surveillance/United-States-COVID-19-Cases-and-Deaths-by-State-o/9mfq-cb36

[50] H. S. Badr, H. Du, M. Marshall, E. Dong, M. M. Squire, and L. M. Gardner, "Association between mobility patterns and covid-19 transmission in the usa: a mathematical modelling study," *The Lancet Infectious Diseases*, vol. 20, no. 11, pp. 1247–1254, 2020.

[51] A. A. Neath and J. E. Cavanaugh, "The bayesian information criterion: background, derivation, and applications," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, no. 2, pp. 199–203, 2012.

[52] "Pmdarima." [Online]. Available: https://pypi.org/project/pmdarima/

[53] A. Tiwari, M. K. So, A. C. Chong, J. N. Chan, and A. M. Chu, "Pandemic risk of covid-19 outbreak in the united states: An analysis of network connectedness with air travel data," *International Journal of Infectious Diseases*, vol. 103, pp. 97–101, 2021.

[54] A. M. Chu, J. N. Chan, J. T. Tsang, A. Tiwari, and M. K. So, "Analyzing cross-country pandemic connectedness during covid-19 using a spatial-temporal database: Network analysis," *JMIR Public Health and Surveillance*, vol. 7, no. 3, p. e27317, 2021.

[55] "Thesis defense video, www.youtube.com, www.youtube.com/watch?v=1via6igu9ww," Jun 2022. [Online]. Available: www.youtube.com,www.youtube.com/watch?v=1viA6IGU9ww

[56] E. Sirotich and J. S. Hausmann, "Removing barriers and disparities in health: lessons from the covid-19 pandemic," *Nature Reviews Rheumatology*, vol. 17, no. 3, pp. 125–126, 2021.

[57] D. M. Gray, A. Anyane-Yeboa, S. Balzora, R. B. Issaka, and F. P. May, "Covid-19 and the other pandemic: populations made vulnerable by systemic inequity," *Nature Reviews Gastroenterology & Hepatology*, vol. 17, no. 9, pp. 520–522, 2020.

APPENDICES

Appendix A

CODE

```
def pre_whiten(input_1, input_2):
  import numpy as np
  import pmdarima as pmd
  import statsmodels.api as ts
  ## Input : Two data sets (input_1, input_2)##
  ## Output : Pre-whitened versions of (input_1, input_2) ##
  ## Check to see if data has to be pre-whitened ##
  if check_auto(input_1, input_2):
    ## Copy the data ##
    data = [input_1, input_2]
    ## Calculate the number of differences
    ## (first-order and seasonal with period 7) to become stationary ##
    ## This saves on computation time for the SARIMA model fit ##
    diffs = [pmd.arima.utils.ndiffs(data[0], alpha = 0.1),
               pmd.arima.utils.ndiffs(data[1], alpha = 0.1)]
    sdiffs = [pmd.arima.utils.nsdiffs(data[0], m = 7),
                pmd.arima.utils.nsdiffs(data[1], m = 7)]

    ## Dummy variable ##
    differenced = [input_1, input_2]

    ## Preform the necessary differencing ##
    ## Seasonal Differencing ##
    if sdiffs[0] > 0:
      differenced[0] = pmd.arima.utils.diff(differenced[0],
                lag = 7, differences = sdiffs[0])
    if sdiffs[1] > 0:
      differenced[1] = pmd.arima.utils.diff(differenced[1],
                lag = 7, differences = sdiffs[1])
    ## Non-Seasonal Differencing ##
    if diffs[0] > 0:
      differenced[0] = pmd.arima.utils.diff(differenced[0],
                differences = diffs[0])
    if diffs[1] > 0:
      differenced[1] = pmd.arima.utils.diff(differenced[1],
                differences = diffs[1])

    ## fit 2 models, and choose the better one out of the two, based on BIC ##
    models = [pmd.AutoARIMA(m = 7, d = 0, D = 0,
            information_criterion = 'bic', alpha = 0.1,
            method = 'nm',
            maxiter = 20).fit(differenced[0]).model_ ,
              pmd.AutoARIMA(m = 7, d = 0, D = 0,
              information_criterion = 'bic', alpha = 0.1,
```

```
            method = 'nm',
            maxiter = 20).fit(differenced[1]).model_]

    optimal = np.argmin([models[0].bic(), models[1].bic()])

    # filter the second model using the more successful model
    filter = int(abs(optimal - 1))
    model = models[optimal]

    ## Difference to the same order as optimal model ##
    seasonal_diff_filter = data[filter]
    diff_filter = data[filter]
    ## difference the data to be filtered to the
    ## same order as the optimal data ##
    if sdiffs[optimal] > 0:
      diff_filter = pmd.arima.utils.diff(diff_filter,
                    lag = 7, differences = sdiffs[optimal])
    if diffs[optimal] > 0:
      diff_filter = pmd.arima.utils.diff(diff_filter,
                    differences = diffs[optimal])
    ## Extract the lag-polynomials from the SARIMA model ##
    ar_poly = model.arima_res_.polynomial_ar
    sar_poly = model.arima_res_.polynomial_seasonal_ar
    combined_ar = np.polymul(ar_poly, sar_poly)
    ma_poly = model.arima_res_.polynomial_ma
    sma_poly = model.arima_res_.polynomial_seasonal_ma
    combined_ma = np.polymul(ma_poly, sma_poly)
    ## Use the arma_innovations filter to filter
    ## the data using the above polynomials ##
    innovations = ts.tsa.innovations.arma_innovations(diff_filter,
                                        ar_params = -combined_ar[1:],
                                        ma_params = combined_ma[1:])[0]

    ## Set the data to the right values ##
    data[optimal] = models[optimal].resid()
    data[filter] = innovations

    return data
  else:
    return [input_1, input_2]


def build_adj(data,column,r = 0.5, weighted = False,
    differencing = 0, two_sided = False,
    window = 14, expanding = False):
  ## Input
  ##  data(pandas data frame) = pandas data frame to
  ##        build an adjacency matrix for, X_t
  ##  column(string, column name from data) = the column
  ##        of the data frame we wish to construct our network for
  ##  r(float) = the cut-off value for the adjacency matrix
  ##  differencing(float) = the order of differencing we wish
  ##        to impose on X_t := X_t^{p} - X_{t - 1}^{p}
  ##  two_sided(Boolean) = whether to use an absolute-value for
  ##        the correlation tolerance
```

```
##  window(int) = the rolling window on which we're calculating correlations
##  expanding(Boolean) = whether to using an expanding verses rolling
##        window for the adjacency matrix
## Output
##  A(numpy_array, shape = (N,N,T)) = The adjacency matrix for G(t),
##        where N is the number of states, T is the number of time-steps

import numpy as np
import scipy as sci
## Define Variables
df = data.copy(deep = True)
N = df.state.unique().size
T = df.submission_date.unique().size
A = np.zeros(shape = (N,N,T))
## Loop through each state
for i in range(0,N):
  ## Loop through each state, up to i
  for j in range(0,i):
    ## If differencing == 0, then do now differencing
    if differencing == 0:
      df_i = df[df['state'] ==
          df['state'].iloc[i]].set_index('submission_date')
      df_j = df[df['state'] ==
          df['state'].iloc[j]].set_index('submission_date')
    ## Else if, differencing ==
    ##    int, then repeat first-order differencing that many times
    elif differencing.is_integer():
      df_i = df[df['state']
          == df['state'].iloc[i]].set_index('submission_date')
      df_j = df[df['state']
          == df['state'].iloc[j]].set_index('submission_date')
      for d in range(int(differencing)):
        df_i.loc[:, column] = df_i[column].diff().fillna(0)
        df_j.loc[:, column] = df_j[column].diff().fillna(0)
    ## Else, apply the operation X_t := X_{t}^p - X_{t-1}^p
    else:
      df_i = df[df['state']
          == df['state'].iloc[i]].set_index('submission_date')
      df_j = df[df['state']
          == df['state'].iloc[j]].set_index('submission_date')
      df_i.loc[:, column] = df_i[column].apply(lambda row: row**(differencing))
      df_j.loc[:, column] = df_j[column].apply(lambda row: row**(differencing))
      df_i.loc[:, column] = df_i[column].diff().fillna(0)
      df_j.loc[:, column] = df_j[column].diff().fillna(0)
    ## If not expanding, then use a rolling window for correlations
    if (not expanding):
      corr = df_i[column].rolling(window).corr(df_j[column])
    ## Else, apply a expanding filter to the data
    else:
      corr = df_i[column].expanding().corr(df_j[column])
    ## Loop through each time-step, k
    for k in range(0,T):
      ## If two_sided == True,
      ##  then apply absolute value and
```

```
      ##  create edge if it's greater than tolerance,
      ##  otherwise it's 0
      if two_sided:
        if np.abs(corr.iloc[k]) >= r:
          A[i,j,k] = corr.iloc[k]
          ## If weighted == False, then just edge weight to 1
          if not weighted:
            A[i,j,k] = 1
        else:
          A[i,j,k] = 0
      ## Else, just apply for positive correlations
      else:
        if corr.iloc[k] >= r:
          A[i,j,k] = corr.iloc[k]
          if not weighted:
            A[i,j,k] = 1
        else:
          A[i,j,k] = 0
## Return the (N,N,T) numpy-array
return A
```

Appendix B

CROSS-CORRELATION RESULTS

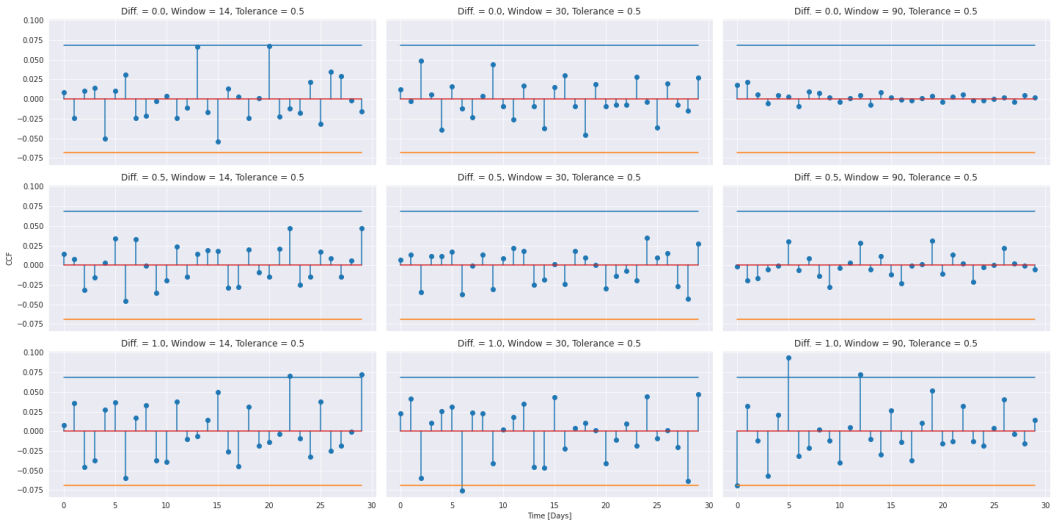

Figure B.1: Mean harmonic centrality ccf coefficients with US Cases versus CCF lag, $r = 0.5$

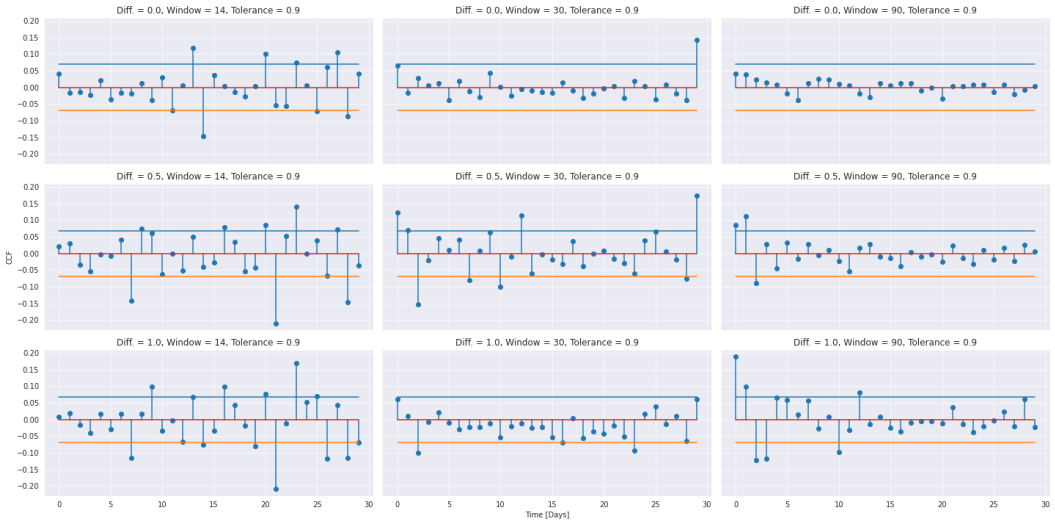

Figure B.2: Mean harmonic centrality ccf coefficients with US Cases versus CCF lag, $r = 0.9$

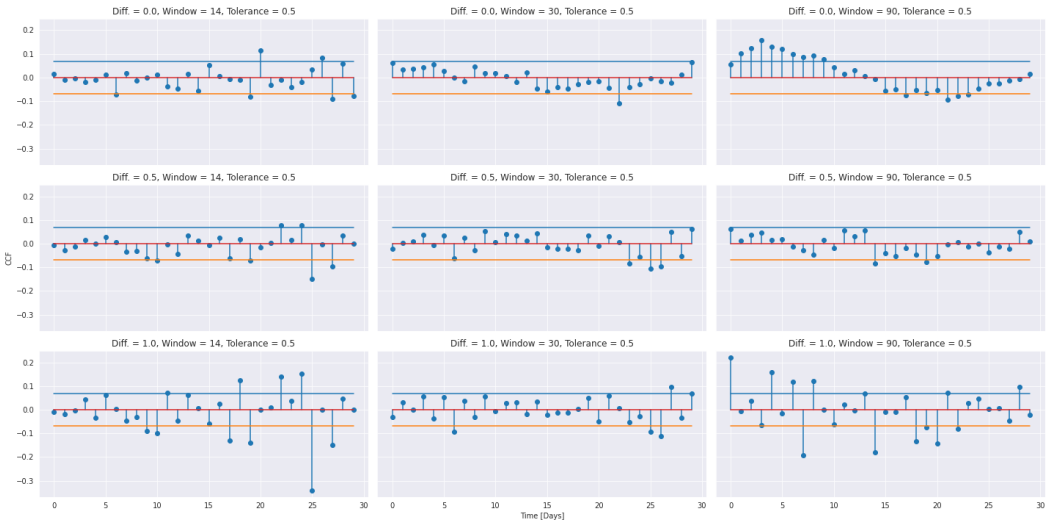

Figure B.3: Mean sub-graph centrality ccf coefficients with US Cases versus CCF lag, $r = 0.5$

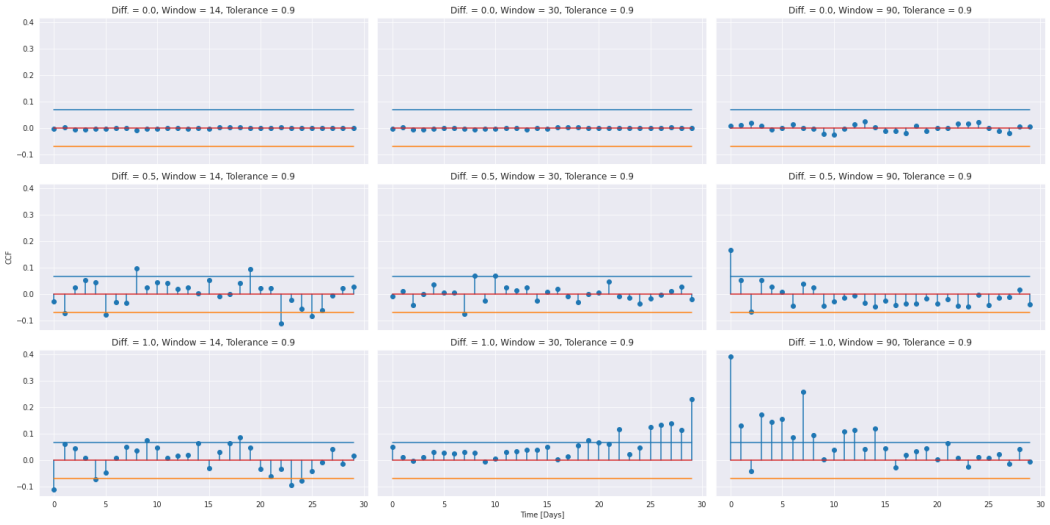

Figure B.4: Mean sub-graph centrality ccf coefficients with US Cases versus CCF lag, $r = 0.9$

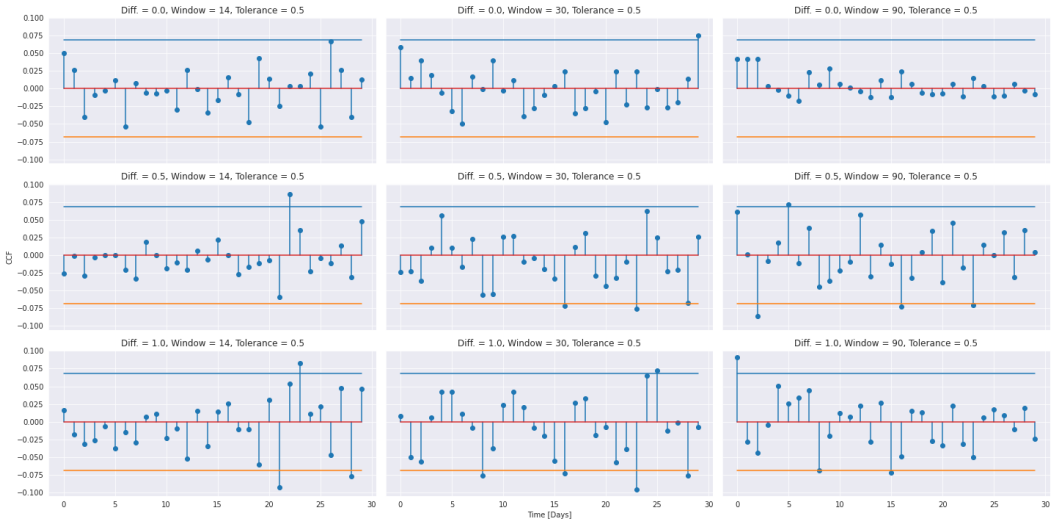

Figure B.5: Largest clique size ccf coefficients with US Cases versus CCF lag, $r = 0.5$

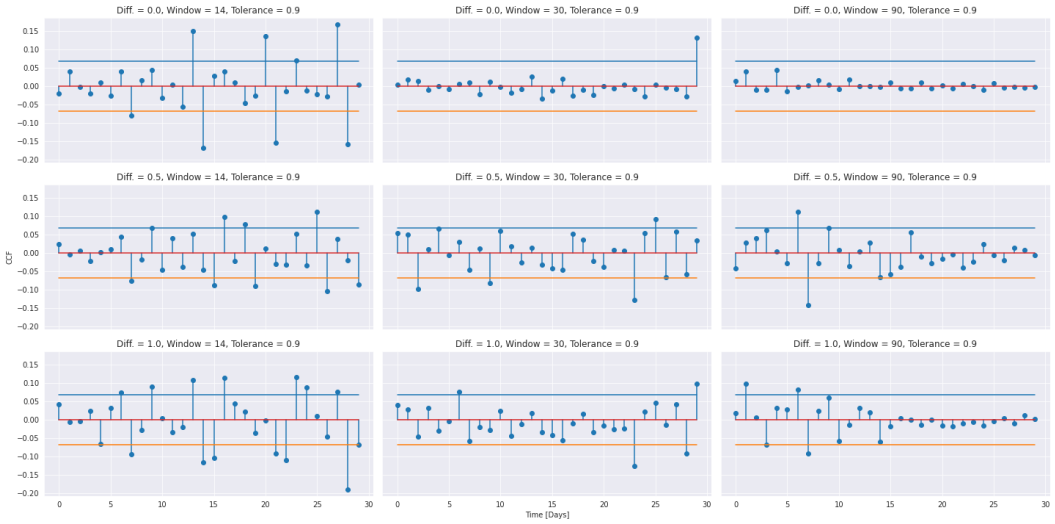

Figure B.6: Largest clique size ccf coefficients with US Cases versus CCF lag, $r = 0.9$

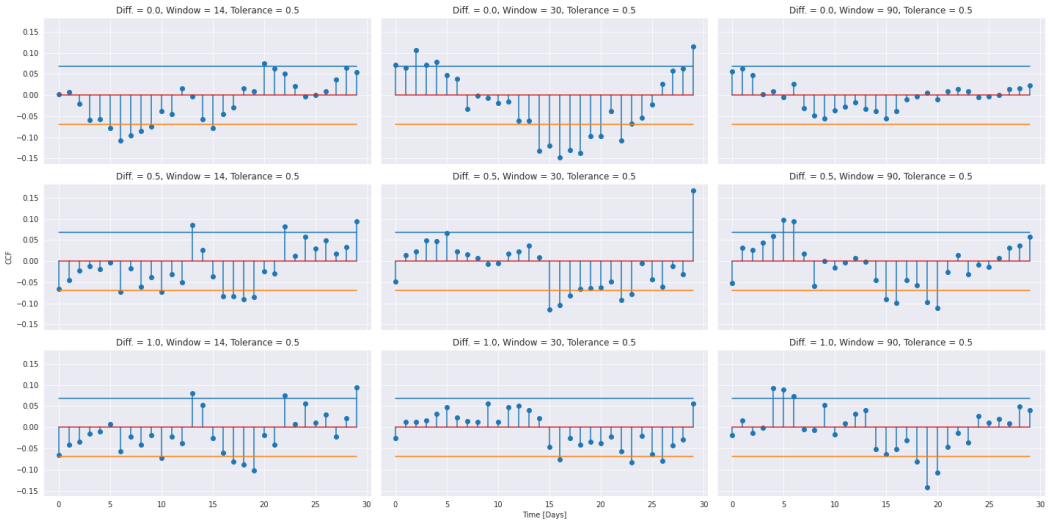

Figure B.7: S-Metric ccf coefficients with US Cases versus CCF lag, $r = 0.5$

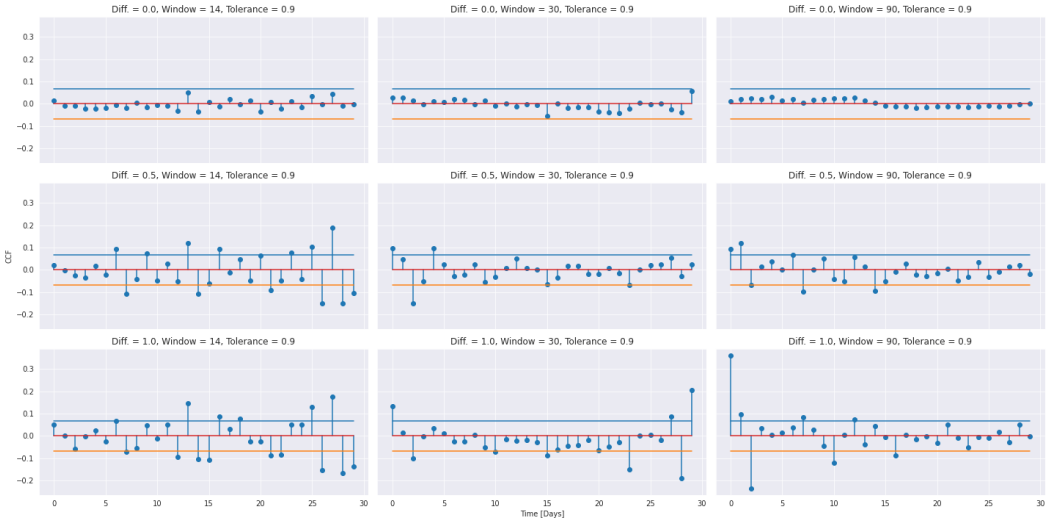

Figure B.8: S-Metric ccf coefficients with US Cases versus CCF lag, $r = 0.9$

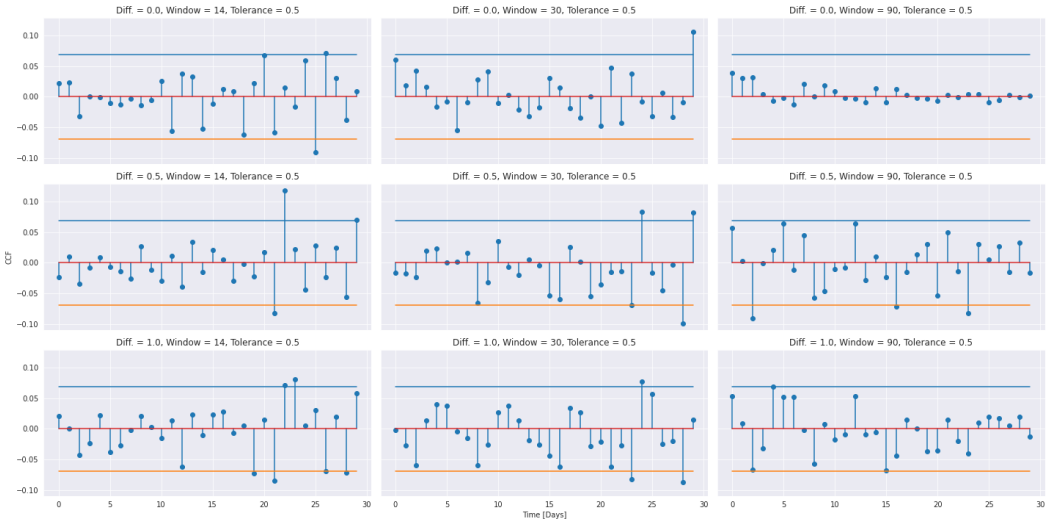

Figure B.9: Mean core number ccf coefficients with US Cases versus CCF lag, $r = 0.5$

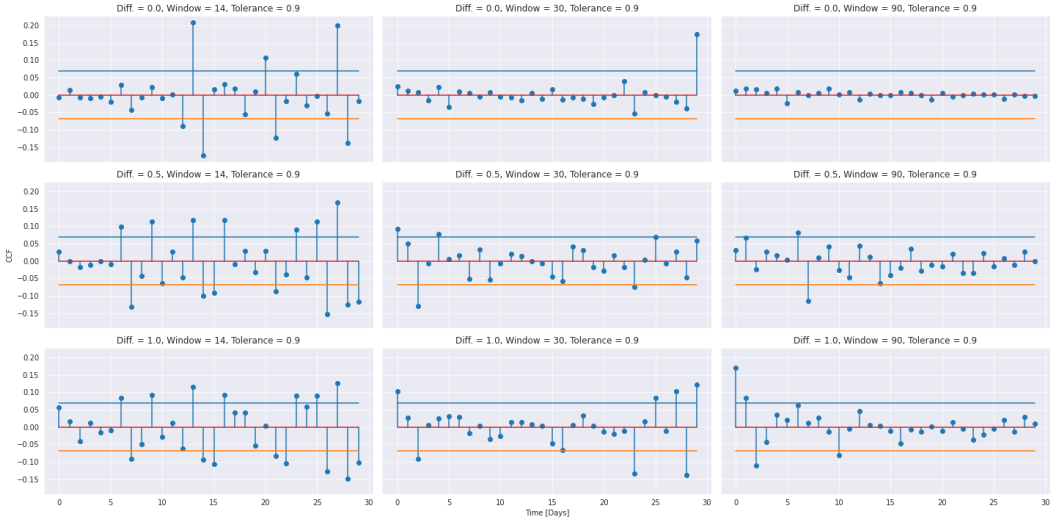

Figure B.10: Mean core number ccf coefficients with US Cases versus CCF lag, $r = 0.9$

Appendix C

NETWORK GRAPHS

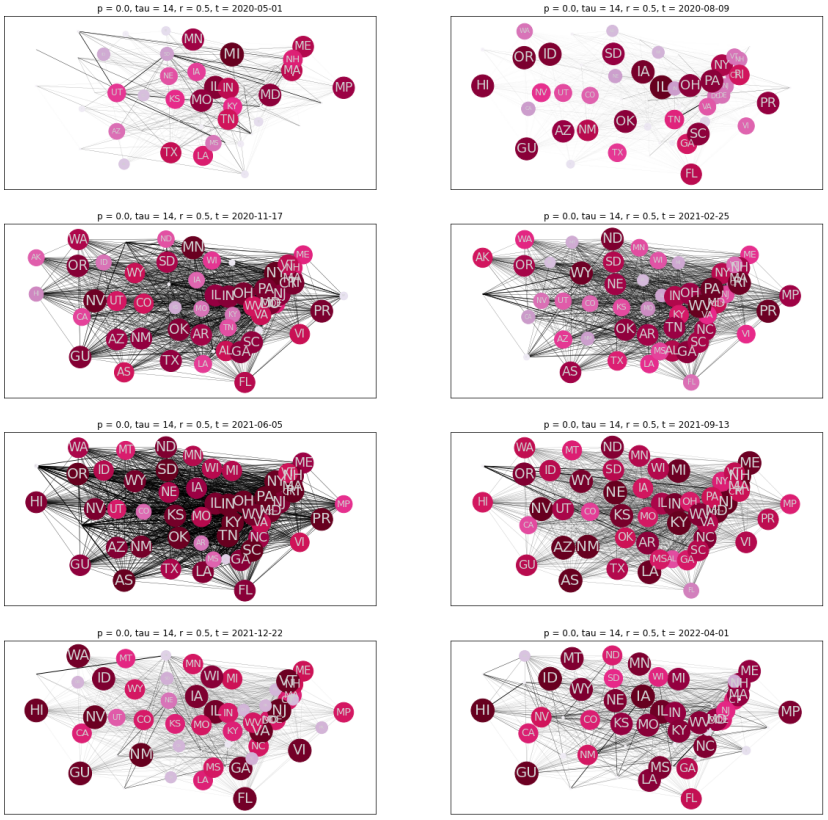

Figure C.1: p = 0.0, tau = 14, r = 0.5

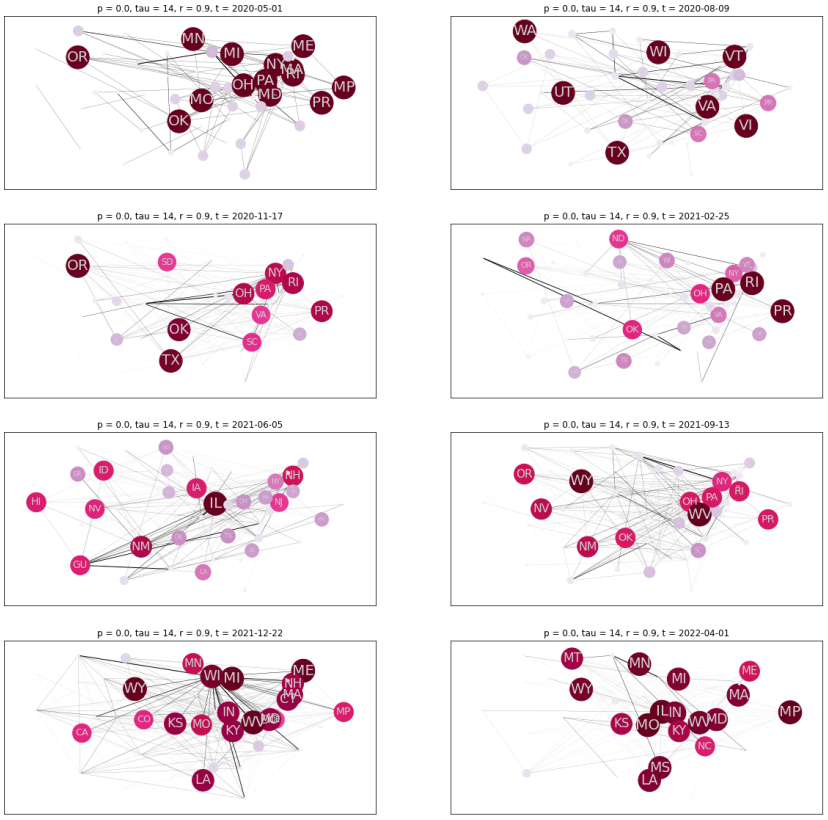

Figure C.2: p = 0.0, tau = 14, r = 0.9

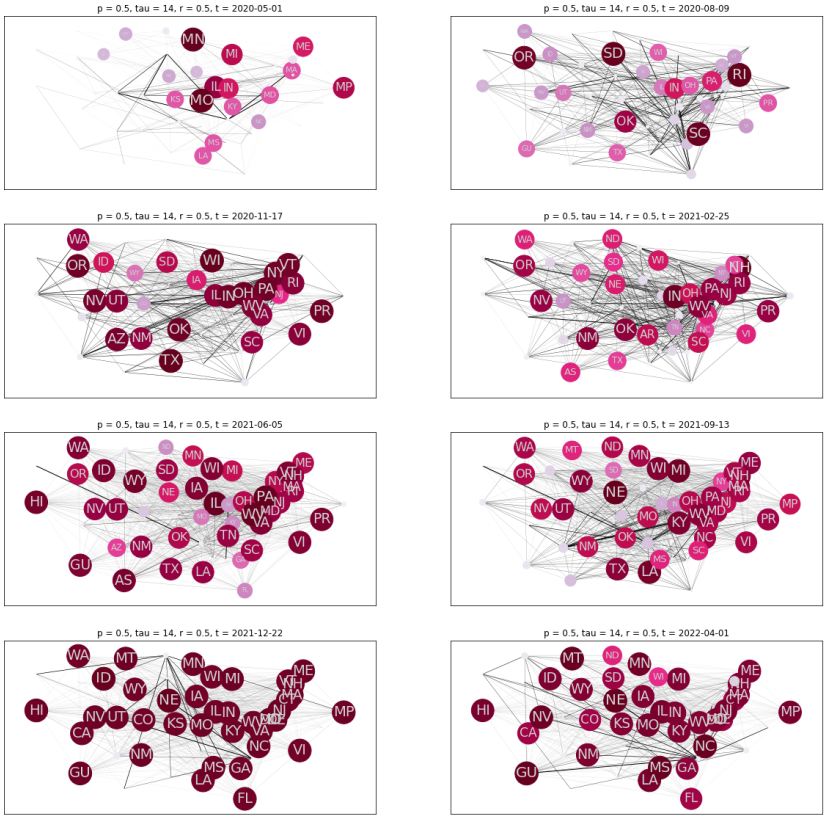

Figure C.3: p = 0.5, tau = 14, r = 0.5

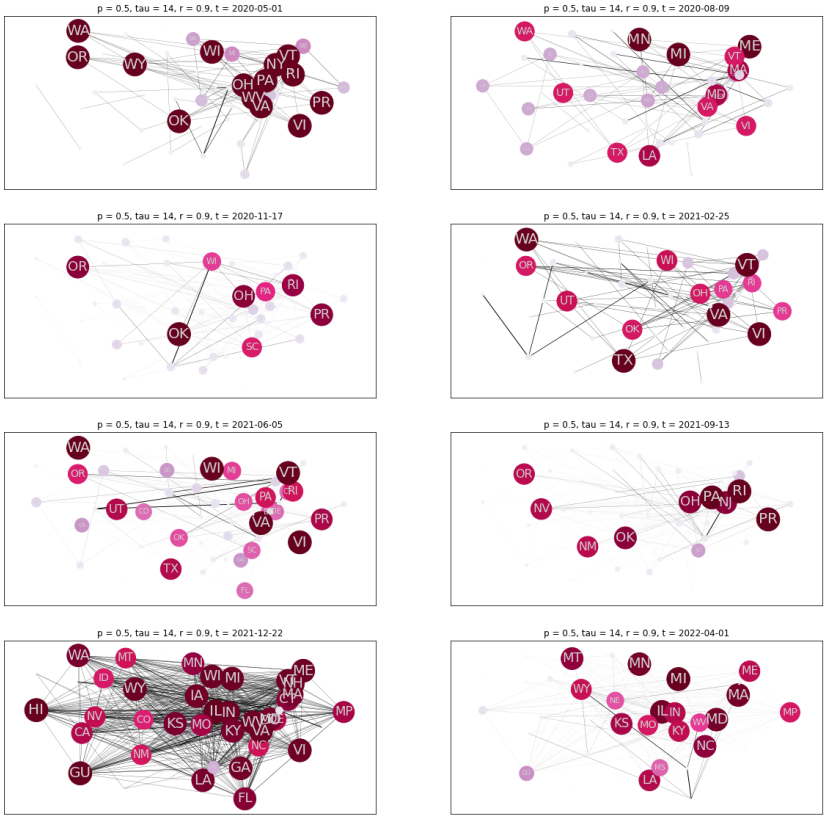

Figure C.4: p = 0.5, tau = 14, r = 0.9

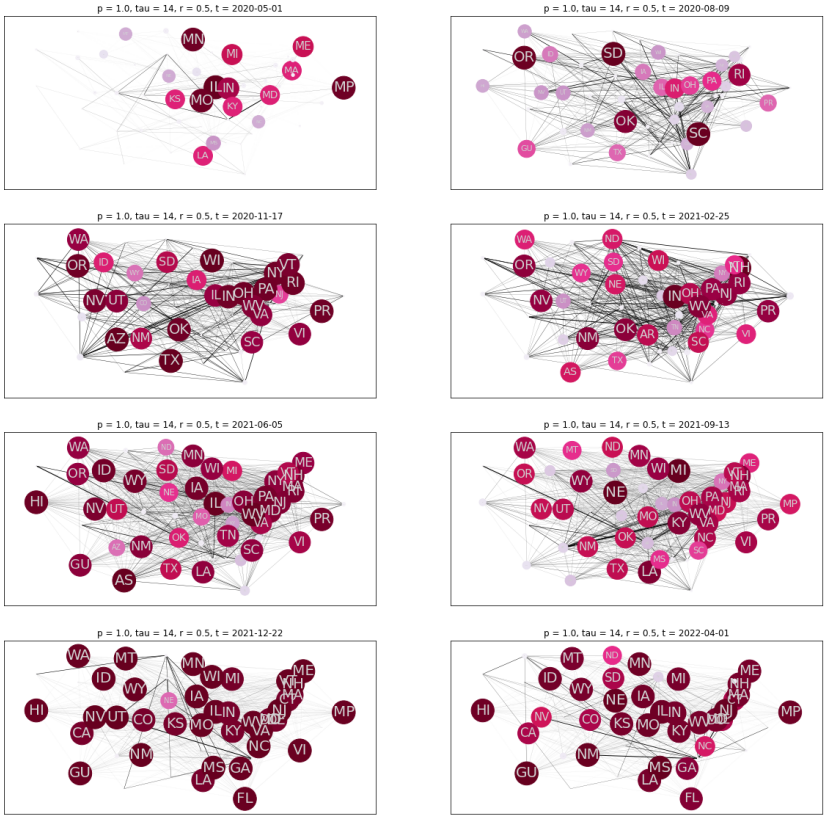

Figure C.5: p = 1.0, tau = 14, r = 0.5

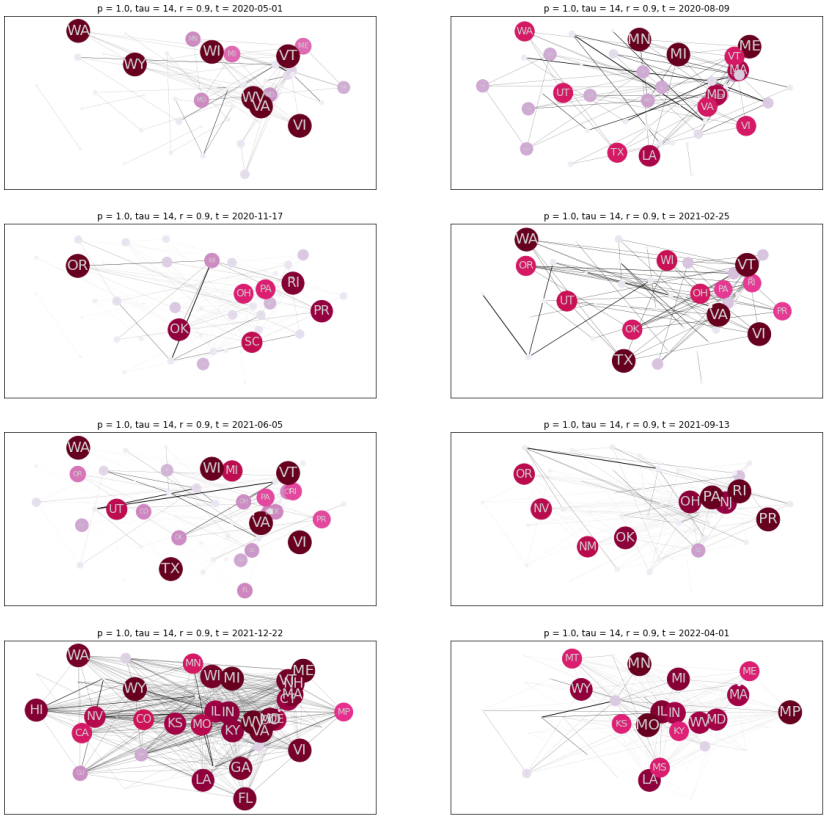

Figure C.6: p = 1.0, tau = 14, r = 0.9

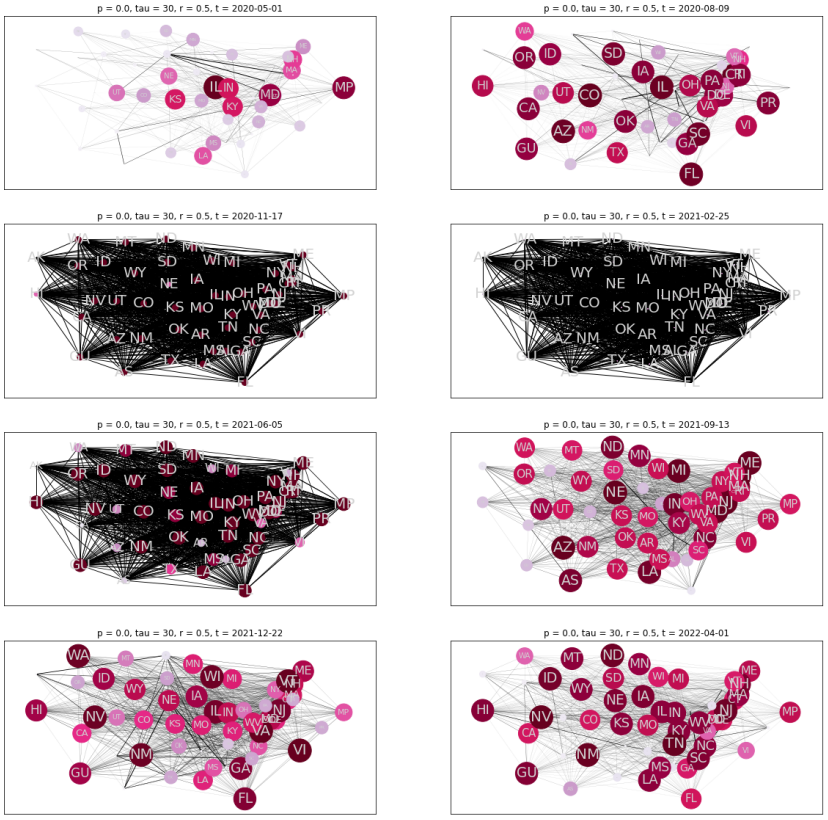

Figure C.7: p = 0.0, tau = 30, r = 0.5

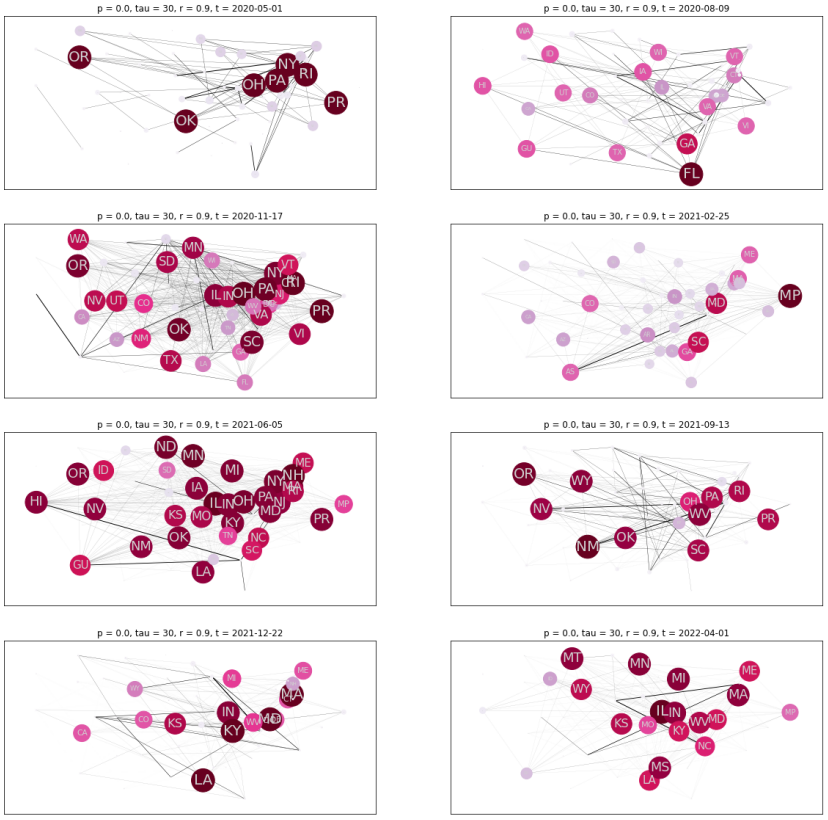

Figure C.8: p = 0.0, tau = 30, r = 0.9

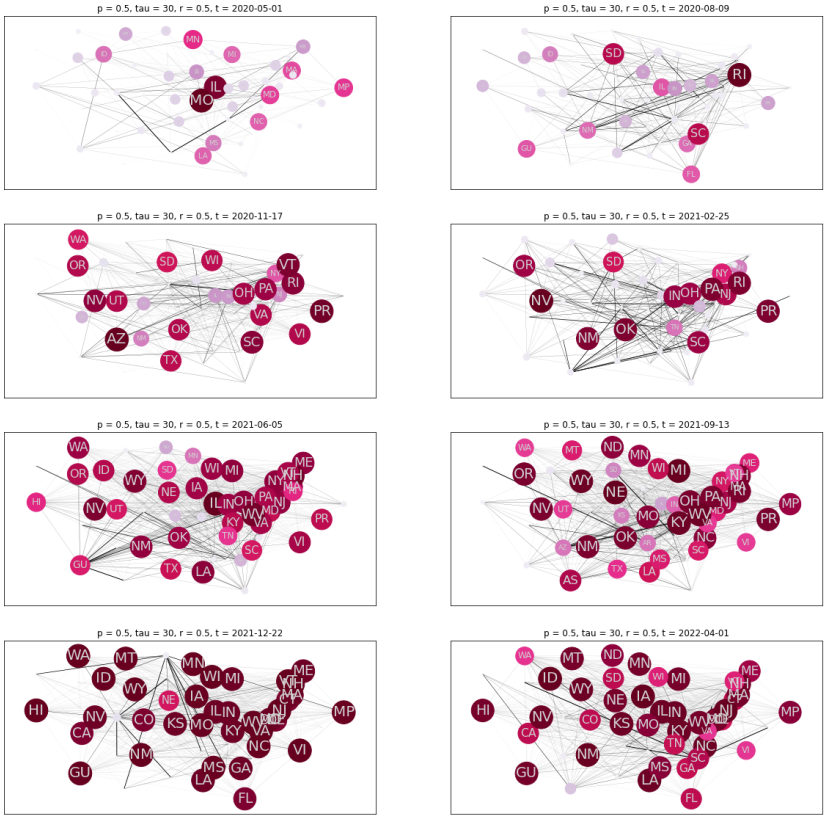

Figure C.9: p = 0.5, tau = 30, r = 0.5

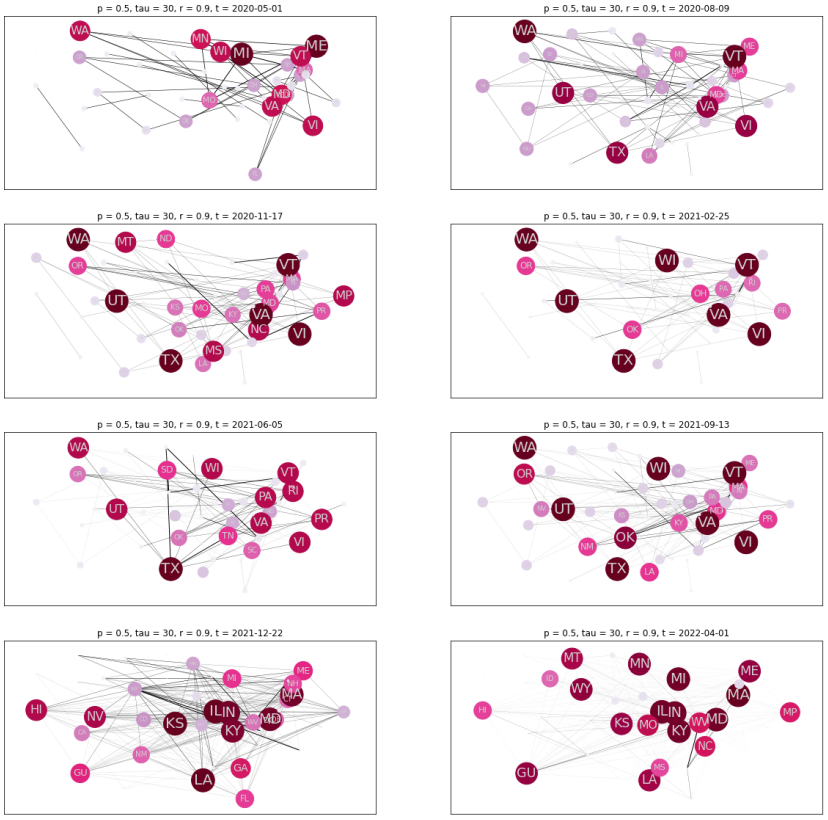

Figure C.10: p = 0.5, tau = 30, r = 0.9

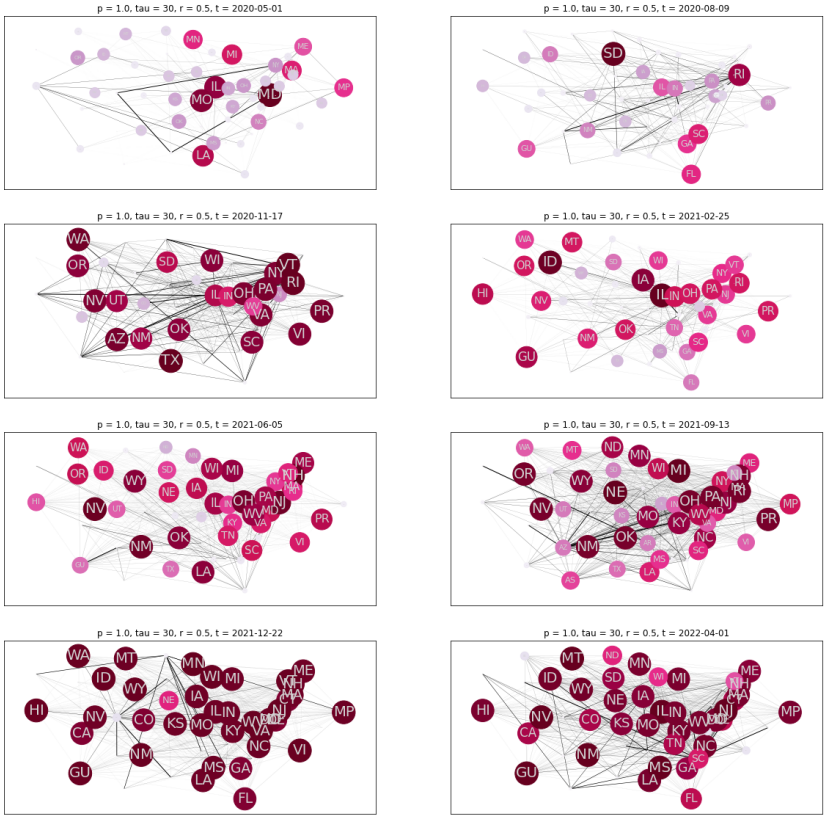

Figure C.11: p = 1.0, tau = 30, r = 0.5

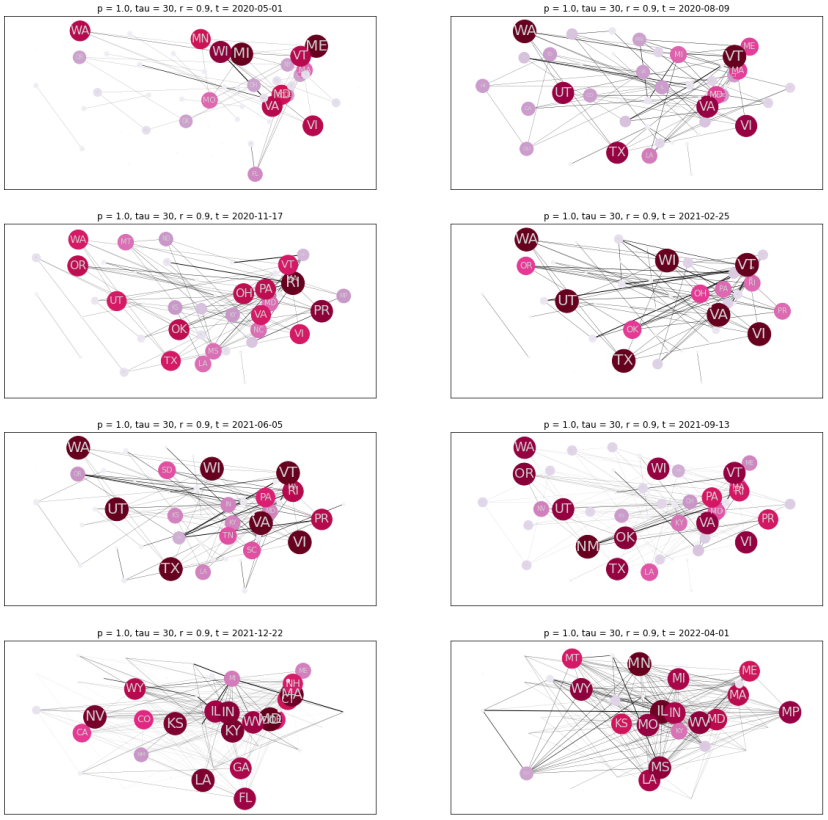

Figure C.12: p = 1.0, tau = 30, r = 0.9

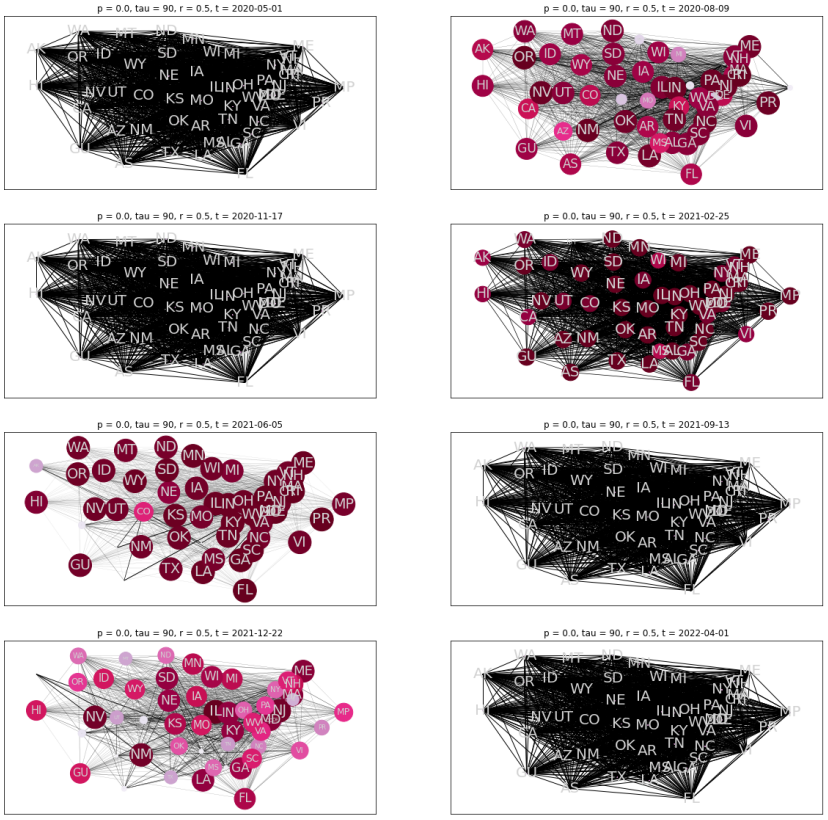

Figure C.13: p = 0.0, tau = 90, r = 0.5

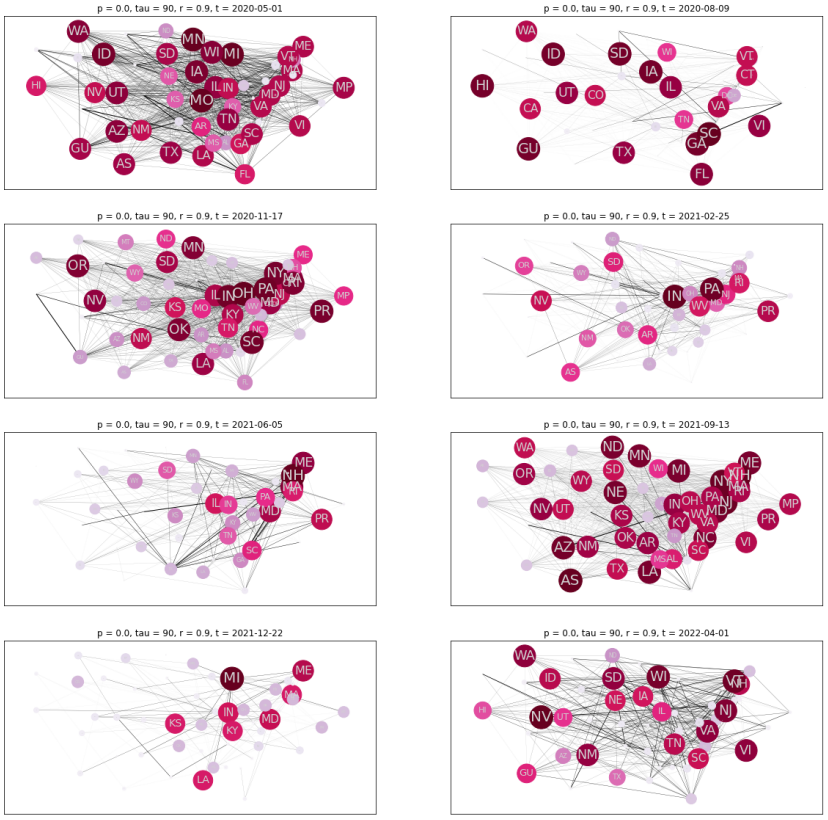

Figure C.14: p = 0.0, tau = 90, r = 0.9

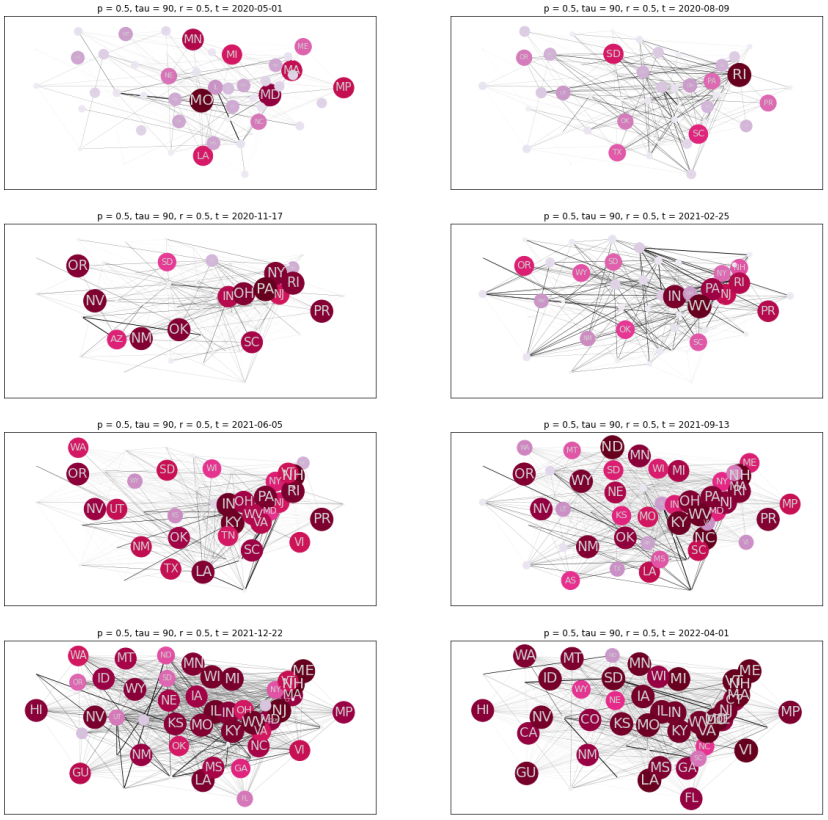

Figure C.15: p = 0.5, tau = 90, r = 0.5

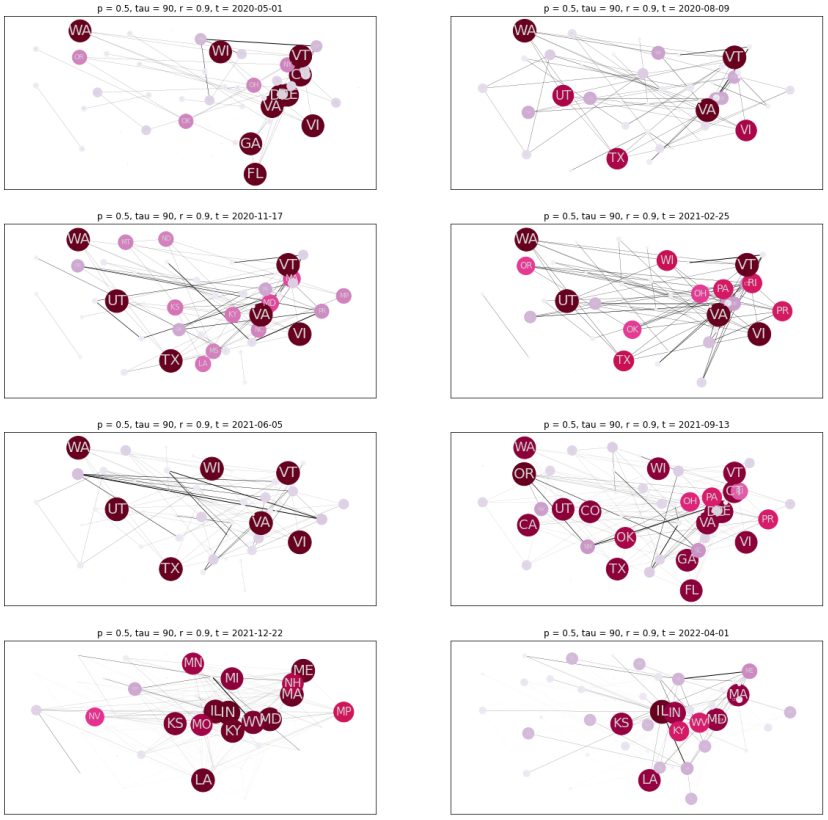

Figure C.16: p = 0.5, tau = 90, r = 0.9

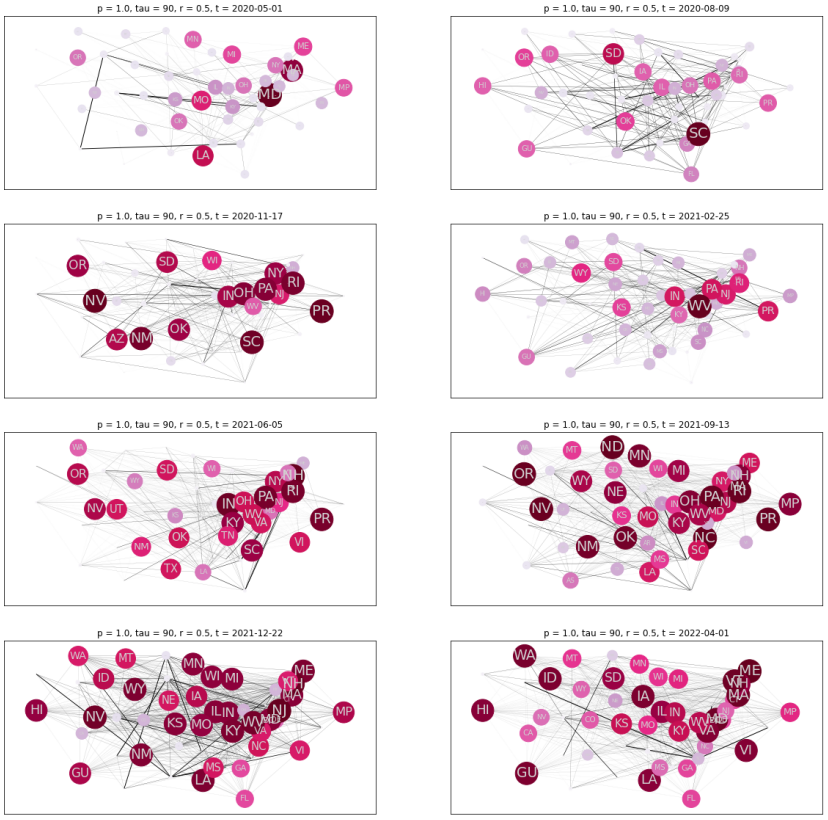

Figure C.17: p = 1.0, tau = 90, r = 0.5

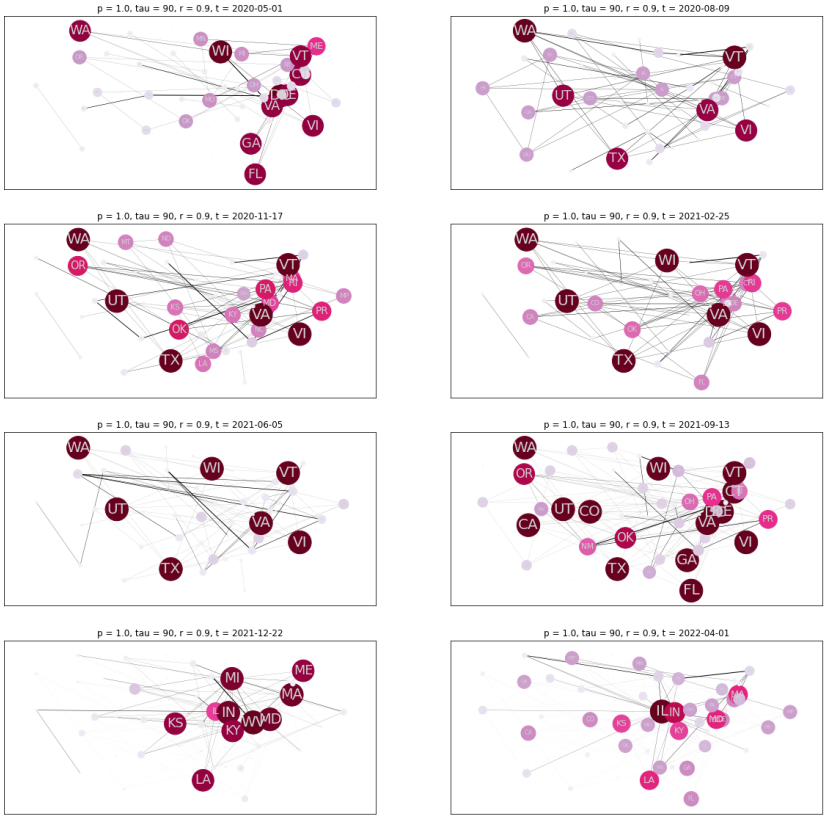

Figure C.18: p = 1.0, tau = 90, r = 0.9

Appendix D

MODEL TEST ERROR

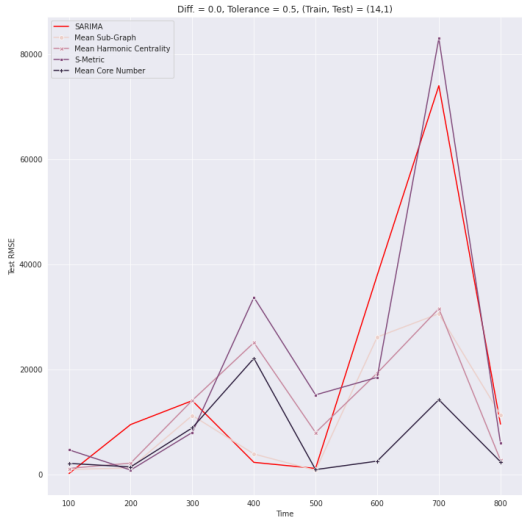

Figure D.1: Diff. = 0.0, Tolerance = 0.5, (Train, Test) = (14,1)

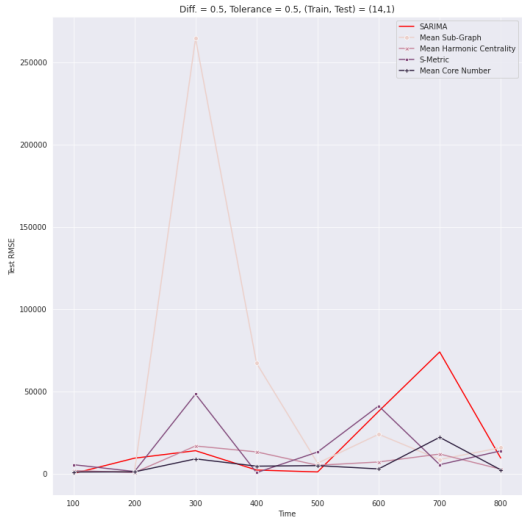

Figure D.2: Diff. = 0.5, Tolerance = 0.5, (Train, Test) = (14,1)

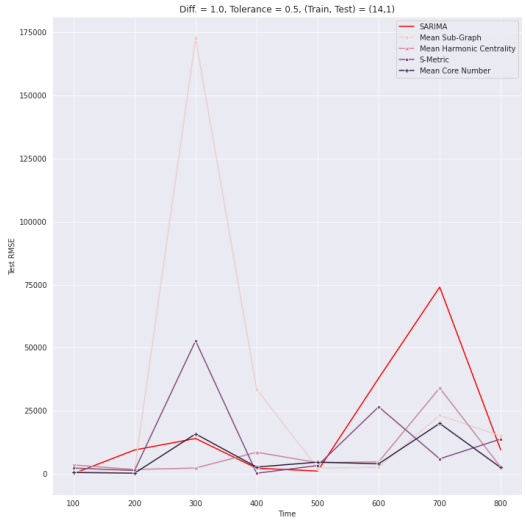

Figure D.3: Diff. = 1.0, Tolerance = 0.5, (Train, Test) = (14,1)

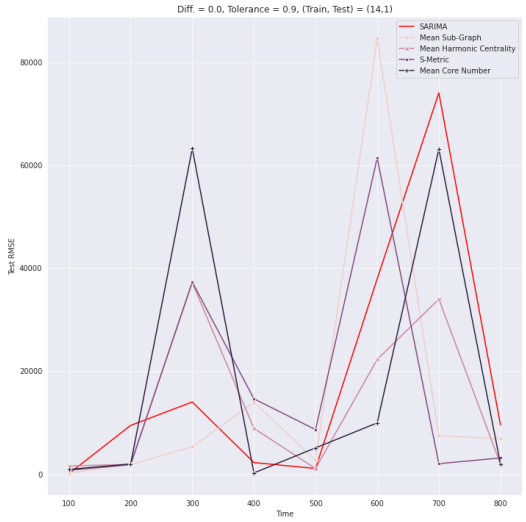

Figure D.4: Diff. = 0.0, Tolerance = 0.9, (Train, Test) = (14,1)

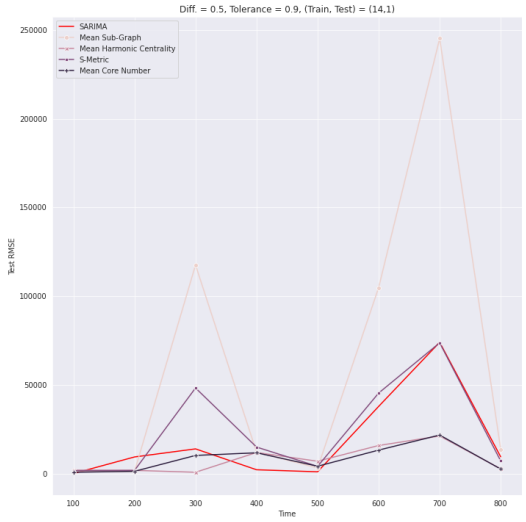

Figure D.5: Diff. = 0.5, Tolerance = 0.9, (Train, Test) = (14,1)

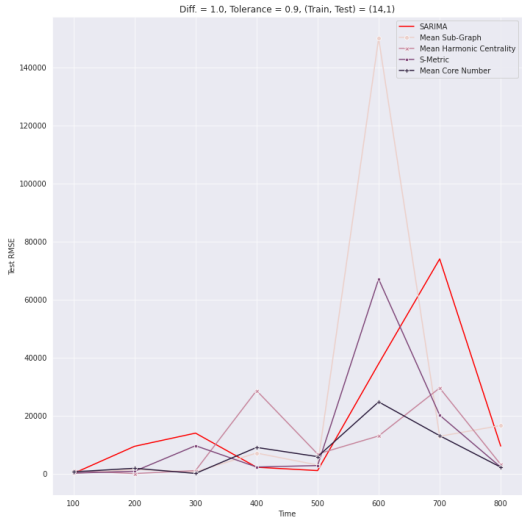

Figure D.6: Diff. = 1.0, Tolerance = 0.9, (Train, Test) = (14,1)

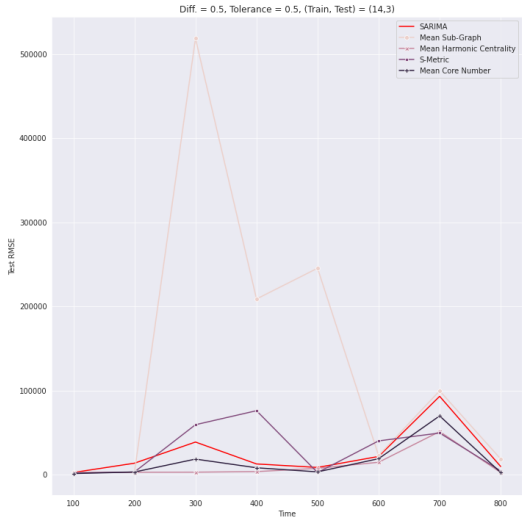

Figure D.7: Diff. = 0.5, Tolerance = 0.5, (Train, Test) = (14,3)

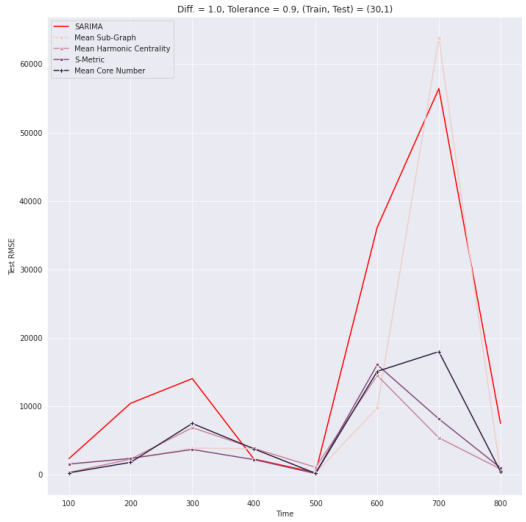

Figure D.8: Diff. = 1.0, Tolerance = 0.9, (Train, Test) = (30,1)

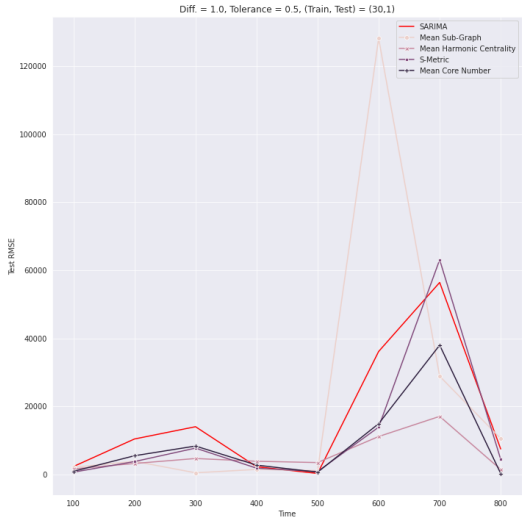

Figure D.9: Diff. = 1.0, Tolerance = 0.5, (Train, Test) = (30,1)

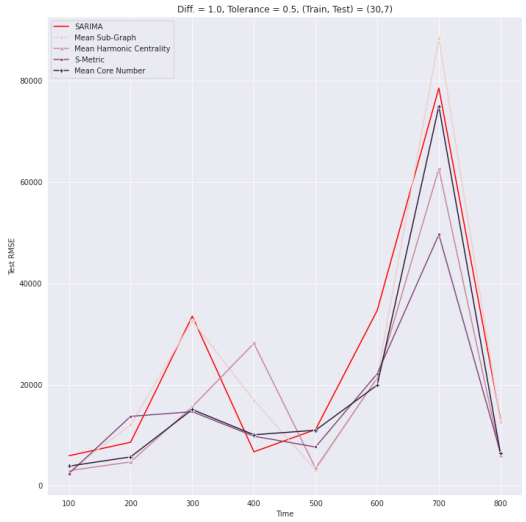

Figure D.10: Diff. = 1.0, Tolerance = 0.5, (Train, Test) = (30,7)

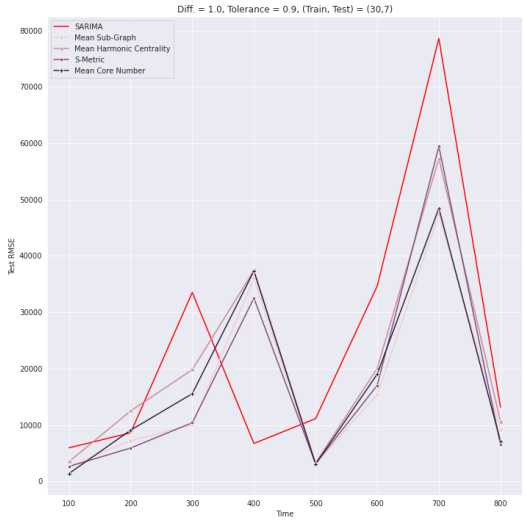

Figure D.11: Diff. = 1.0, Tolerance = 0.9, (Train, Test) = (30,7)

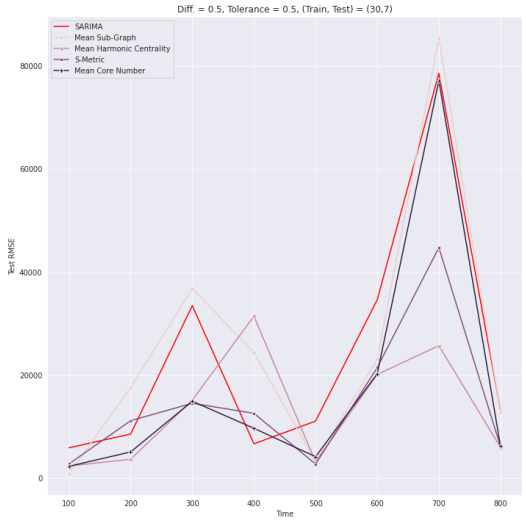

Figure D.12: Diff. = 0.5, Tolerance = 0.5, (Train, Test) = (30,7)

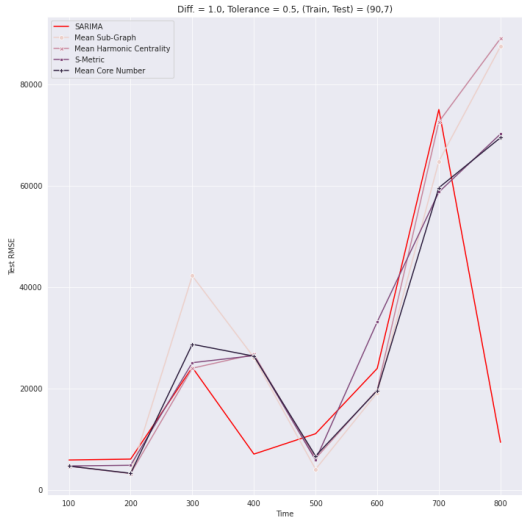

Figure D.13: Diff. = 1.0, Tolerance = 0.5, (Train, Test) = (90,7)

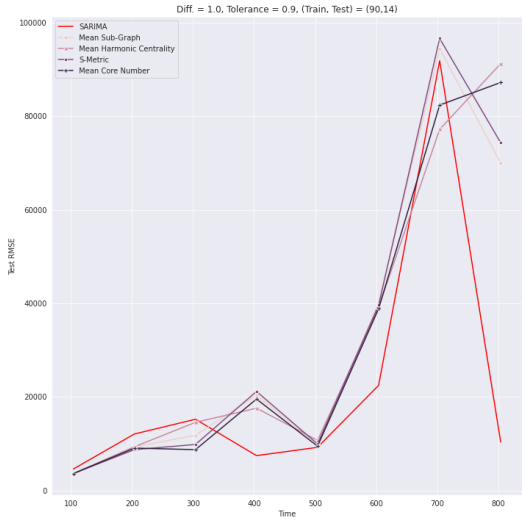

Figure D.14: Diff. = 1.0, Tolerance = 0.9, (Train, Test) = (90,14)

Appendix E

COMMUNITY DETECTION RESULTS

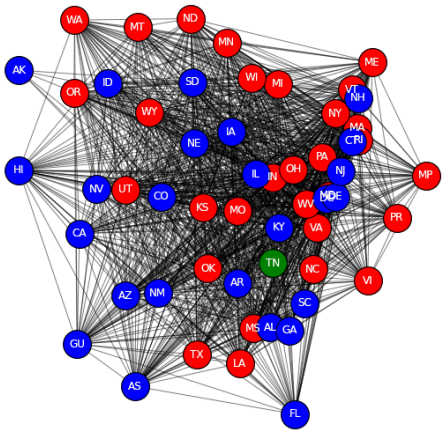

Figure E.1: Top 3 Communities, p = 0.0, tau = 14, r = 0.5,t = 2021-09-13

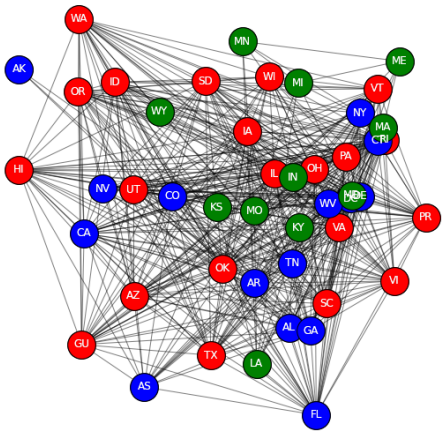

Figure E.2: Only 1 community, p = 0.0, tau = 30, r = 0.5, t = 2020-08-09

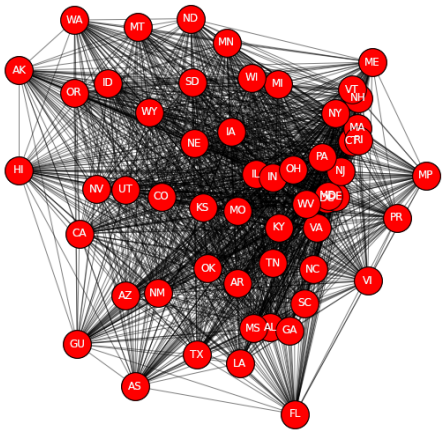

Figure E.3: Only 1 Community, p = 0.0, tau = 30, r = 0.5,t = 2021-09-13

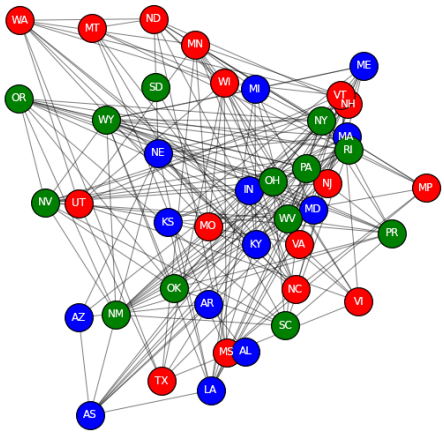

Figure E.4: Top 3 Communities, p = 0.0, tau = 14, r = 0.9,t = 2021-09-13

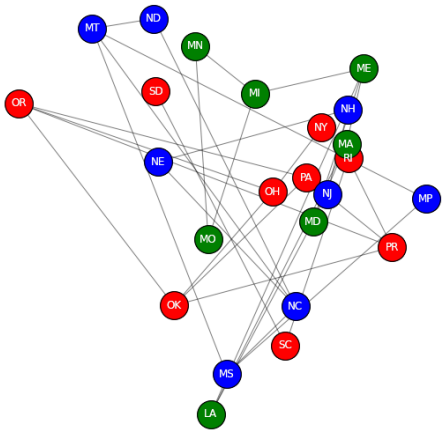

Figure E.5: Top 3 Communities, p = 0.0, tau = 30, r = 0.9,t = 2020-08-09

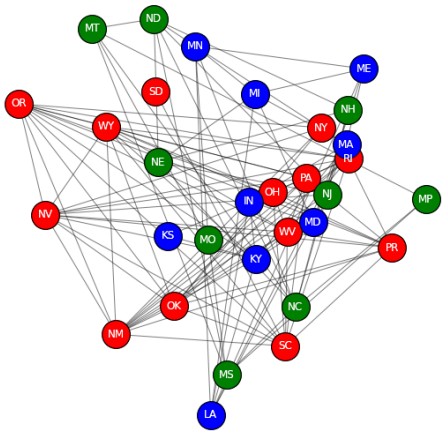

Figure E.6: Top 3 Communities, p = 0.0, tau = 30, r = 0.9,t = 2021-09-13