On Systems of Dynamic Graphs: Theory and Applications

देवव्रत विवेक दाबके
Devavrat Vivek Dabke

A dissertation
presented to the faculty
of Princeton University
in candidacy for the degree
of Doctor of Philosophy

Recommended for acceptance by the
Program in Applied and Computational Mathematics

Adviser: Bernard Chazelle

May 2023

Abstract

Graphs are powerful mathematical structures that pose deep theoretical questions and adapt to fascinating applications. Simple graphs have a rich history and many important open problems, but contemporary research in graphs now involves a wide range of extensions: hypergraphs, attributed graphs, graph neural networks, graph algorithms, network analysis, and more. Each of these topics is its own active research area. This dissertation focuses on **dynamic graphs**, namely graphs whose structure depends on time. We will study traditional simple graphs, as well as some of these extended graph structures, but all through the lens of dynamical systems, where our state space is of graphs or their many variations.

To properly study dynamic graphs, we have to leverage techniques from graph theory, algorithms, probability, machine learning, topology, geometry, and other mathematical and computational disciplines. Part of the excitement of dynamic graphs comes from the seemingly unlimited connections to other important areas of study. As an opus of applied mathematics, this work will cover dynamic graphs that arise naturally from a wide range of applications in virology, sociology, sports, biology, electrical engineering, satellite communication, and more.

While no document can be complete, this dissertation furnishes a survey on innovative ongoing research in dynamic graphs, insight into their key constructions, a presentation of our contributions to this area with collaborators, strong evidence for their utility in a wide range of applications, and a hint at possible future directions for these elegant structures.

Contents

Abstract 3

I Background 13

1 Overview 14
- 1.1 Euler and the Origins of Graph Theory 15
- 1.2 Moving Bridges 17
- 1.3 Key Themes and Contributions 21

2 Mathematical Introduction 25
- 2.1 Graphs and their Variations 25
- 2.2 Applied Topology & Geometry 28
- 2.3 Basic Graph Operations 32

3 Brief Survey 34
- 3.1 Graph Dynamical Systems 35
- 3.2 Applied Topology 37
- 3.3 Machine Learning with Graph Neural Networks 39
- 3.4 Dynamic Networks So Far 41

II Main Results 43

4 Tracking Virality in Connected Populations 44
- 4.1 Information as a Virus 46
- 4.2 Estimating Asymptomatic Viral Spread 65
- 4.3 Key Takeaways 79

5 Fast-Moving Natural Networks 80
- 5.1 Basketball through Applied Topology 82
- 5.2 Basketball through Geometry and Machine Learning 110
- 5.3 Lessons on Dynamism 131

6 Final Frontier: Inspired by Space 132
- 6.1 Temporal Graphs in Lunar Networks 134
- 6.2 Routing Problems and Dynamic Graphs 161
- 6.3 Final Notes 190

III Looking Forward 191

7 Conclusion 192
7.1 Overview of Results 193
7.2 The Bridges of Kaliningrad 195

8 Future Directions 196
8.1 Theoretical Considerations 197
8.2 Connections to Related Disciplines 203
8.3 Application Areas of Interest 205

Appendix A Code 209
A.1 Viral Networks 211
A.2 Basketball 218
A.3 Code in Space 238
A.4 Exploratory Code 245

References 252

Author List

Several sections in this dissertation include work completed with other contributors and is reflected in various publications. All figures in this work that come from publications have been reproduced with permission.

Section 4.1 includes work conducted with Eva E. Arroyo and Anita T. Layton. It is primarily reflected in a publication entitled *Rumors with Personality: A Differential and Agent-Based Model of Information Spread through Networks*[64].

Section 4.2 includes work conducted with Kritkorn Karntikoon, Chaitanya Aluru, Mona Singh, and Bernard Chazelle. It is primarily reflected in a forthcoming publication (not yet published at the time of this writing) entitled *Network-Augmented Compartmental Models to Track Asymptomatic Disease Spread*.

Section 5.1 includes work conducted with Erin Taylor, Christopher J. Tralie, and John Harer.

Section 5.2 includes work conducted with Bernard Chazelle. It is primarily reflected in a publication entitled *Extracting Semantic Information from Dynamic Graphs of Geometric Data*[65].

Section 6.1 includes work conducted at NASA with Alan Hylton, Robert Short, Jacob Cleveland, Olivia Freides, Zander Memon, Robert Cardona, Robert Green, Justin Curry, Sriram Gopalakrishnan, Brittany Story, Michael Moy, and Brendan Mallery. It is primarily reflected in a publication entitled *A Survey of Mathematical Structures for Lunar Networks*[136].

Section 6.2 includes work conducted at NASA with Jacob Cleveland, Alan Hylton, Robert Short, Brendan Mallery, Robert Green, Justin Curry, and Olivia Freides. It is primarily reflected in a publication entitled *Introducing Tropical Geometric Approaches to Delay Tolerant Networking Optimization*[56].

While each of these sections contains work with other authors, sections are not in one-to-one correspondence with publications. Some material within a publication is not reflected in this dissertation and much of each section is substantially revised, refreshed, and contains new material.

Listing of figures

1.1 Königsberg and its layout when Euler would have seen it. The Pregel River weaves throughout the city and has seven bridges that cross it. The bridges are highlighted in green. This figure is reproduced with permission from[102]. 15
1.2 A graph that represents the bridges of Königsberg. Each vertex is a landmass within the city and the edges represent a bridge between two pieces of land. While it would not be unusual to see such a figure today, this 18th-century abstraction was a novel idea by Euler. This figure is in the public domain and reproduced from[91]. . . 16
1.3 A simple, alternating graph sequence: G is the graph on the left and G' is on the right. We imagine that at odd timesteps, the graph goes clockwise (left), while at even timesteps, the graph goes counterclockwise. There are a variety of real-world edges that behave this way, e.g. roads that switch direction depending on the time to account for rush hour traffic. Electrical current can also alternate within a wire. In both cases, violating the direction of flow can have catastrophic results. . . 18
1.4 The time-extended version of the alternating sequence from Figure 1.3. The first column of nodes represent G, while the second column represents G', and so on and so forth. Between the columns, we draw the edges of the former column to the matched vertices of the latter column. For example, edge (A, B) is represented by the edge from the top vertex in the left-most column to the second vertex in the second column. 19
1.5 The time-extended version of the trivial sequence of repeating G over and over again. This graph is constructed in the same way as Figure 1.4, but with a different underlying graph. 20

4.1 The ISTK Model. This network-driven model is inspired by the traditional SIR viral model and then Daley-Kendall model. There are four compartments: the **I**gnorant, **S**preader, s**T**ifler, and **K**nowledgeable classes. The arrows in this diagram characterize the flow between the classes and the relevant controlling parameters. 49
4.2 The result of numerically solving the differential model over 22 days. Each line represents a particular population class, as indicated by the legend. The total population does not change over time and is normalized to 1. 56
4.3 Results of the sensitivity analysis of parameter α_1 in the differential model. For each value of α_1 the time at which 75% of the population had been exposed to the rumor is recorded. 57
4.4 Sensitivity analysis of parameter c in the differential model. For each credibility value the time at which 50% of the population heard the rumor. 57

4.5 Results of the sensitivity analysis of parameter d (average days to forget) in the differential model. The hour that 50% of the population heard the rumor is recorded for each value of d. 58
4.6 Results of the agent-based model. Solid line indicates median proportion of population across the 400 trials; shadow indicates IQR. 59
4.7 Box-and-whisker plot comparing the steady states in Figure 4.2 (differential model) and the end states in Figure 4.6 (Agent-based model). 59
4.8 Results of the most and least similar feature vectors to the population in the agent-based model (average across 300 trials). 60
4.9 Density of the proportion of the population who heard the rumor after 22 days with the most and least similar rumors (300 trials). 61
4.10 Linear model of the relationship between the final percentage of the population heard rumor and average similarity score of the feature vector ($r = 0.538$). Shading designates the 95% confidence interval. 61
4.11 This network represents the counties of the New York City metropolitan area. Each state is colored differently with Manhattan at the center in black. The node size corresponds to the population in that county. The nodes are positioned according to the geographic center of each county. 72
4.12 This figure demonstrates the predictive power of our estimation techniques per parameter. Each plot compares the actual and predicted value of a parameter for many different combinations of the two others. As expected, the estimation of ρ degrades as the actual value gets large. Ultimately, if no one feels sick, behavior does not change and the method cannot pick up ρ. 76
4.13 This graph displays real COVID data against TraSIR-simulated data. The blue ragged line is the reported daily case count across the New York City metro area, summed across the 44 counties considered here. The orange smooth line is the simulated daily symptomatic case count with our estimated parameters. . . . 77

5.1 A basketball court with with defense in blue and offense in orange. The court picture is reproduced from `http://printablediagram.co`. 95
5.2 A basketball court with a regular rectilinear gridding. This figure is the same as Figure 5.1 but has a regular grid on top of it. The defense is in blue and the offense is in orange. 96
5.3 A basketball court with a position-based rectilinear gridding. This figure is the same as Figure 5.1 but has a position-based rectilinear grid on top of it. In particular, for each defense player (in blue), we place a vertical and horizontal line at their position, e.g. for a player at (x_0, y_0), we place the lines $x = x_0, y = y_0$ on top of the basketball court. We do not do anything for the offense (in orange). 97
5.4 A basketball court with a regular polar gridding, emanating from the basket that the offense (orange) is pursuing and the defense (blue) is defending. The polar grid image is from `http://etc.usf.edu`. 98
5.5 A basketball court with a Delaunay triangulation gridding. We take the Delaynay triangulation of the players on the defense (in blue), along with points at each corner of the half-court and the basket. We do not compute a gridding from the offense (in orange). 99

5.6 A basketball court with the convex hulls of the defense (blue) and offense (orange). 100
5.7 The raw trajectory data of the ten basketball players in a particular play within a particular Duke University basketball game. 102
5.8 A bar graph that displays the size of each of the clusters detected by the spectral clustering method. 103
5.9 Dendrogram of the relationship between plays. Play indices are indicated along the x-axis, while relative similarity is indicated across the y-axis. Clustering is based on Delaunay Triangulation for gridding. 104
5.10 Cross-similarity matrix showing aggregate similarity across experiments. Play numbers are reordered to correspond to the order in the dendrogram. 106
5.11 Cross-similarity matrix with corresponding dendrogram along with a manual labeling of possession clusters by well-known play types. 107
5.12 Overview of the data analysis pipeline presented in this section. First, the raw trajectory data is converted into interaction networks. Then, by comparing graphs up to isomorphism, we can construct a "library" of possible configurations. We can then construct a jump Markov model by taking the empirical maximum likelihood estimator with graphs as the state space. Finally, we can feed in the raw trajectory data and the graph data from the jump Markov model into a Transformer model for prediction. This article opens the door to future work on inferring game semantics and strategies from actual games. 111
5.13 A deeper look at a snapshot of a basketball game. Green nodes with Roman characters are offensive players. Red nodes with Greek letters are defensive players. On the left, we see player positions with occlusion fields. In the center, we see all offensive lines of sight. On the right, we see the occlusion network with only the offensive edges. 116
5.14 A frame of the basketball game with the constructed passing network. Green circles with Latin letters are the offense, while red circles with Greek letters are the defense. 117
5.15 The architecture diagram of a Transformer, as presented in[287]. The Transformer has an encoder system (left) and a decoder system (right) and uses self-attention in three ways to accurately capture semantic information from both sequences. 122
5.16 Sample library of graphs for a possession, descending in order of frequency from left to right and top to bottom. 123
5.17 Two players on the offense (black points on either side of the line segment) with a member of the defense in between them (large orange ball in the center). . . 124
5.18 The number of edges for a particular possession in the game. Mathematically, the number of edges are a random walk over the frames of data within a possession. 125
5.19 Probability transition matrix for a sample possession. This transition matrix assumes a naive Markov chain and therefore has most of the density for each row on the diagonal. 127

5.20 Sample hold time distribution as a cumulative density function of a representative graph in a representative possession. The blue dots are the true data, which have an average of 5, giving an exponential distribution with parameter $\frac{1}{5}$ The orange dots represent an ideal exponential distribution with the same parameter. 128

6.1 A picture of an example satellite network from a simulation tool used at NASA. Note, connections are denoted by a line between two devices. 153
6.2 An example space network that connects a system of satellites with communication nodes to Earth. 163
6.3 An example of a parametric graph. 164
6.4 An analytical solution of cell decomposition and associated shortest path trees corresponding to Figure 6.3. Results from applying the Joswig algorithm described below. Each interval shows the values of parameter x for which each tree $(T0, T1, T2)$ optimal. Note that there is a fourth tree omitted, which is the same as $T0$ except with the path to v_1 going along the leftmost edge with weight 5. It is omitted because there are no values of x that make it optimal. 168
6.5 An example sequence of steps in the binary search. 170
6.6 Example output of binary (first) implementation applied to Figure 6.3. Note that this agrees with the solution given in Figure 6.4. 172
6.7 Example graph with two parameters. Note the separability. 173
6.8 Lines for two parameter binary search, $n = 5$ sample lines, corresponding to Figure 6.7 174
6.9 Example output of binary (first) implementation applied to Figure 6.7. 174
6.10 Example output of our Joswig (second) implementation applied to Figure 6.3. Note that this agrees with the solution given in Figure 6.4. 175
6.11 Waveform associated with the Verilog Tropical ALU implementation. 176
6.12 High level representation of program being demonstrated in the waveform of Figure 6.11. 177

8.1 A simple graph sequence parsed via renormalization. 202
8.2 The alternating graph sequence with eight nodes (analogous to the graph in Figure 1.3) parsed with renormalization. 203
8.3 A snapshot of a mouse embryo in development. It displays a variety of cells in color and their estimated geometry. Although difficult to discern from this image, it is possible to tell which cells physically adjoin each other and which cells may have inter-cell signaling. Furthermore, it is possible to track cells across time and, as they split, retain ancestry information. 206
8.4 A snapshot of the subway system in New York City. Each line represents a subway line and the colors with varying intensity display the congestion across these pathways. 207

To my family, with all my love.

Acknowledgments

I HAVE BEEN CONTINUOUSLY EDUCATED since Kindergarten, which started when I was five years old. To finish a dissertation—overcoming both personal and professional challenges—I have many people to acknowledge, which I will attempt to do here. I would like to broadly thank my family, friends, and colleagues in the wide range of endeavors I have been blessed to participate in. From all-nighters to group projects, from long video chats to shared beverages, I am profoundly grateful to everyone who has helped me grow, learn, and live.

For their help in my education, I would like to thank my favorite teachers: Ms. Brown, Ms. Butler, and Ms. LaBrosse; and to Ms. Galuska and Ms. A, for pushing me to be the best student I could be. At Duke, I had the privilege of learning from Profs. Pierce and Calderbank, who instilled in me much needed mathematical discipline; Prof. Harer and Dr. Tralie taught me how to conduct research; and Profs. Mösenbichler-Bryant, Tufts, Siegel, Gillespie, and Wood Crowley opened my eyes to the world outside of math. I have a special place in my heart for Xiaobai, who always believed in me. And, to my very first professor, whom I met in her 10:05 AM Monday class, I simply cannot enumerate how many ways in which she has provided her advice, mentorship, and kindness: Anita, thank you beyond measure.

To Elaine, Joni, Analese, Chris, Ryan, Liz H., Salman, Jeff, and John, I appreciate our work to make the world a better place through technical education. To Ken, Kyle, and Liz P., thank you for showing me the lifelong joy of learning. I have enjoyed research activity with my brilliant collaborators, all of whom have ideas represented directly or indirectly here: Eva, Brendan, Michael, Jacob, Olivia, Zander, Robert C., Sriram, Brittany, Kritkorn, Mona, and Justin. I would also like to thank Peter, Sam, Rohith, Jess, Erin, and Emily for always picking up my phone calls for both technical and personal conversations. And to Robby, I hope we never stop exploring our ideas together.

I would like to express gratitude to my program, especially Bernadeta, Victoria, Gina, Tina and, the one who holds everyone together, Audrey. I have had the privilege of studying with two mentors at NASA: Alan and Bob. I would like to thank them for creating most intellectual and inviting space within the atmosphere to study the one without. Of course, the most important influence on my dissertation has been my adviser, Bernard. To him, I have infinite gratitude for his mathematical insights and unyielding support.

To Alice and Mike, I hope that you would have been proud to see this dissertation; to David and Helen, I am glad you helped me cross the finish line at every step. I would also like to thank Suparna for her patience and Avanti, Tanvi, Vetra, and Shirish, for all of the years they have indulged my mathematical education in their company. I would not be here today (for a wide variety of reasons) without my parents: I love you. And, without the love from Sugar, Spice, Daisy, and Macaroni (Max), I would not have learned anything at all.

Part I

Background

The more I think about language, the more it amazes me that people ever understand each other at all.

Kurt Friedrich Gödel

1 Overview

UNSURPRISINGLY, the roots of graph theory can be traced to Leonhard Euler, the brilliant and indefatigable mathematician, who solved the famous *Bridges of Königsberg* problem[40]. This problem concerns itself with seven bridges in the city of Königsberg and their particular arrangement around the Pregel River.

1.1. Euler and the Origins of Graph Theory

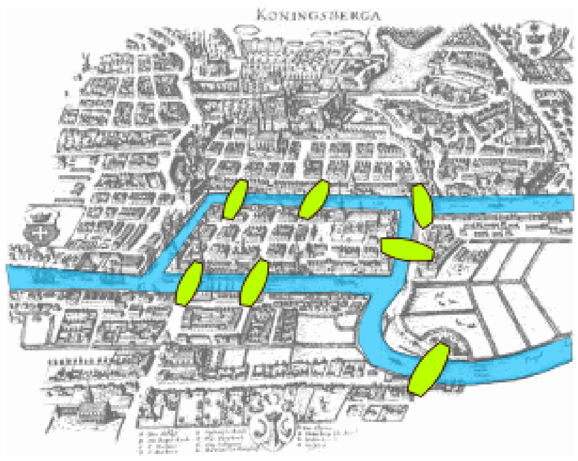

Figure 1.1: Königsberg and its layout when Euler would have seen it. The Pregel River weaves throughout the city and has seven bridges that cross it. The bridges are highlighted in green. This figure is reproduced with permission from [102].

In Figure 1.1, we see the location of the seven bridges in and around the city, with their respective crossings over the Pregel. The key problem posed: is it possible to take a walk through the city and cross each of the seven bridges exactly once? Being the astute problem-solver, Euler solved this problem, but more importantly than efficiently sightseeing the bridges of a tranquil albeit obscure European city, he generalized his technique via defining the first recorded instance of a graph.

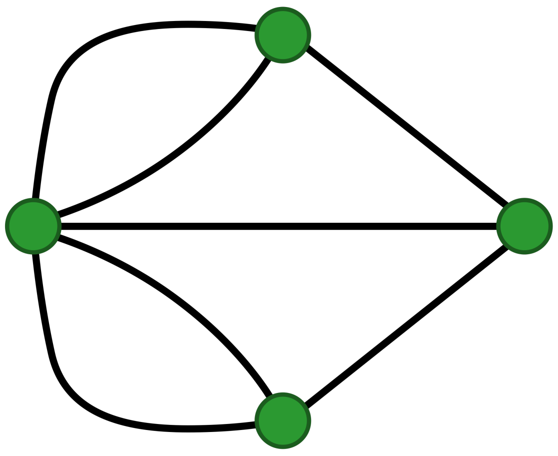
Figure 1.2: A graph that represents the bridges of Königsberg. Each vertex is a landmass within the city and the edges represent a bridge between two pieces of land. While it would not be unusual to see such a figure today, this 18th-century abstraction was a novel idea by Euler. This figure is in the public domain and reproduced from[91].

By removing all irrelevant structures and focusing on the topological properties of the problem, Euler was able to conceptualize the problem as a graph, as seen in Figure 1.2. Putting this together, Euler proposed the following theorem[87]:

> *Thus for any configuration that may arise the easiest way of determining whether a single crossing of all the bridges is possible is to apply the following rules:*
>
> > *If there are more than two regions which are approached by an odd number of bridges, no route satisfying the required conditions can be found.*
> >
> > *If, however, there are only two regions with an odd number of approach bridges the required journey can be completed provided it originates in one of the regions.*
> >
> > *If, finally, there is no region with an odd number of approach bridges, the required journey can be effected, no matter where it begins.*
>
> *These rules solve completely the problem initially proposed.*

We can rewrite Euler's observations in contemporary graph theoretic language.

Definition 1.1.1 (Euler Path)**.** Given a (simple, undirected) connected graph G, an Euler path is a path that contains each edge in the graph exactly once.

With this definition, we can write the Euler Theorem, as Theorem 1. (For further definitions, see Chapter 2.)

Theorem 1 (Euler Theorem). A (simple, undirected) connected graph G has an Euler path if and only if it has exactly zero or two vertices of odd degree. Additionally, for graphs with exactly two vertices of odd degree, every Euler path in G has to start from one of these vertices of odd degree and end at the other.

This neat historical problem illustrates the motivation behind graph theory. By excavating the core structure to a set of vertices and edges, we can deftly analyze our situation. In his case, Euler was dealing with ancient bridges in a static city, but our modality is a bit different.

1.2. Moving Bridges

Euler's key innovation was the construction of a graph, but three hundred years or so later, we will now update his problem and allow the bridges to move. One obvious case is the road network in a city. Consider someone living in Manhattan, New York, NY: roads are continually closed, the city places haphazard pedestrian detours, and there are accidents among the cyclists, taxis, and tourists all the time. We are faced with a wider class of graph problems, but our accessible routes are constantly shifting over time. We are thus not just interested in the topological insights of a static graph, but rather the abstraction of a dynamic graph.

To motivate why this abstraction is needed, we consider a simple example that became the thorn in my side that motivated this entire thrust of research. In Figure 1.3, we see an example of an alternating graph. For a formal characterization, we define the graphs $G =$

$(V, E), G' = (V, E')$ where

$$V = \{A, B, C, D\}$$
$$E = \{(A, B), (B, D), (D, C), (C, A)\}$$
$$E' = \{(B, A), (D, B), (C, D), (A, C)\}$$

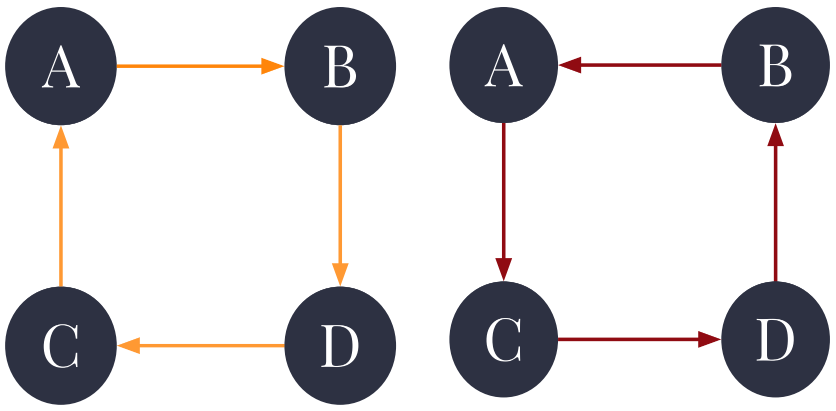

Figure 1.3: A simple, alternating graph sequence: G is the graph on the left and G' is on the right. We imagine that at odd timesteps, the graph goes clockwise (left), while at even timesteps, the graph goes counterclockwise. There are a variety of real-world edges that behave this way, e.g. roads that switch direction depending on the time to account for rush hour traffic. Electrical current can also alternate within a wire. In both cases, violating the direction of flow can have catastrophic results.

From here, let us consider the "alternating" discrete-time graph sequence, where we switch from the left graph, G, to the right graph, G', at every timestep. More precisely, we can define the sequence $\mathbf{G} = (G^t)_{t \in \mathbb{N}}$, where

$$G_t = \begin{cases} G & t \text{ is odd} \\ G' & t \text{ is even} \end{cases}$$

From here, we can define a simple dynamical system. We will place a particle on node A at the first timestep and force this particle to move across an edge at every timestep. Mathematically, we define x^t to be the position of our particular at every timestep with the following restrictions. We can write $x^t \in V$ with our positional sequence being $\mathbf{x} = (x^t)_{t\in\mathbb{N}}$. We then can define the rules of our dynamical system as

$$x^t = \begin{cases} A & t = 1 \\ \in N^{t-1}(x) & \text{otherwise} \end{cases}$$

where we define

$$N^{t-1}(u) \triangleq \{v : (u, v) \in E^{t-1}\}$$

Now, we can ask the natural question: how does this dynamical system behave? As a static graph theorist, we could invoke a simple lemma: each graph in the sequence is strongly connected. The maximum possible diameter of a graph of n nodes is $n - 1$ and we are thus tempted to conclude that we will start and return to A in $\mathcal{O}(n)$ time. Of course, we would be baldly misapplying this fact; simply looking at this sequence tells us that we will never move past A and B.

Admittedly, there may be a simple way to convert this dynamic sequence into a static one.

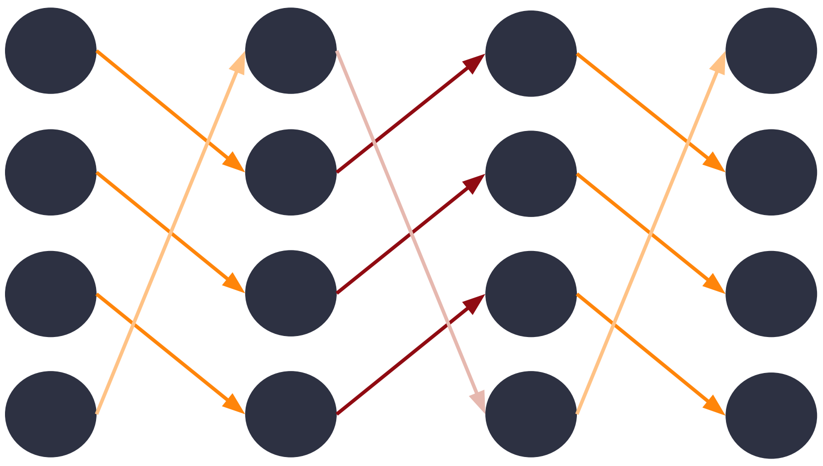

Figure 1.4: The time-extended version of the alternating sequence from Figure 1.3. The first column of nodes represent G, while the second column represents G', and so on and so forth. Between the columns, we draw the edges of the former column to the matched vertices of the latter column. For example, edge (A, B) is represented by the edge from the top vertex in the left-most column to the second vertex in the second column.

One possible resolution is the creation of the time-extended graph, which we can see in Figure 1.4. This graph "unrolls" the alternating sequence into a larger graph with copies of each node for each timestep. From this, we can see that we have four disconnected components, which suggest to us that perhaps a disconnected time-extended graph implies a dynamical system that is disconnected. (n.b. time-extended graphs are inherently directed because of causality, so they can never be strongly connected, though they can be weakly connected.) However, this proposition is blatantly false.

To see why this is false, let us consider the sequence of just G, i.e. we repeat the same graph over and over again as our sequence, which is perhaps not an interesting sequence.

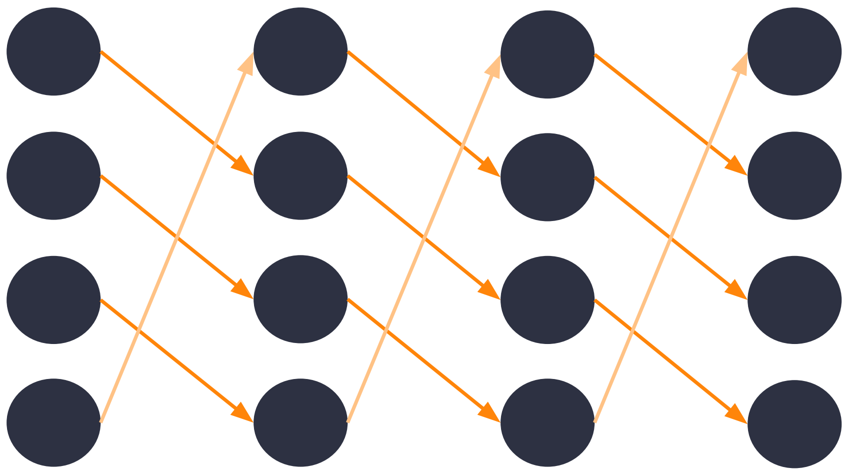

Figure 1.5: The time-extended version of the trivial sequence of repeating G over and over again. This graph is constructed in the same way as Figure 1.4, but with a different underlying graph.

In Figure 1.5, we see the result of this process and we get another disconnected time-extended graph, but our sequence will allow our particle to move across it without issue. Time-extended graphs are certainly a useful analytical tool, but static notions of connectivity cannot and do not fully characterize the dynamic case. Moreover, we have some obvious issues with this construction: if we have T total timesteps and n original vertices in our graph, our constructions will operate over $\mathcal{O}(n \cdot T)$ total vertices, which is unsatisfying for the theorists (because $T = \infty$ in many cases) and impractical for the realists (because n may already be somewhat large). Looking at even this simple example, we start to see the shortcomings of our tools from static graph theory. While these tools may have helped Euler solve the problem of bridge crossings, our contemporary world is full of dynamical systems that require much more sophisticated tools.

So what do we do from here? It is not a trivial task to extend static notions to their dynamic counterparts, which is what we endeavor to do in later parts of this dissertation. We will indeed see a solution to our problem and it involves the construction of novel results with associated proofs.

1.3. Key Themes and Contributions

Even in this overview, we can see a hint at what is on offer to us by studying dynamic graphs. There are three key themes that recur consistently throughout the study of dynamic graphs and will be highlighted throughout this dissertation.

1. **Non-locality**: traditional static properties that may explain a local part of a dynamic graph do not trivially extend to global properties. We can already see this in our example of the simple alternating graph in Figure 1.3.

2. **Naturalness**: dynamic graphs arise naturally in a wide range of contexts. As we pursue the various applications and future directions for this work, we will see many examples of life modelled crisply by dynamic graphs.

3. **Tractability**: while there are many benefits of purely theoretical areas of study, dynamic graphs are practical. When we define certain algorithms over them, they are indeed tractable and computationally feasible. We can also apply cutting-edge techniques in machine learning to them to find interesting and surprising results.

1.3.1. Structure of this Dissertation

In this dissertation, we will cover a variety of theoretical tools, applications, and techniques in machine learning that help uncover this rich area of study. Chapter 2 gives an outline of the mathematical notation and concepts that are invoked throughout this work and Chapter 3 provides a survey of the state of affairs as of the time of this writing. In the second part of this dissertation, Chapters 4, 5 & 6 cover dynamic graphs in a wide variety of settings: Chapter 4 describes traditional discrete- and continuous-time models with dynamic graphs and uses analytical and agent-based techniques to characterize these systems; Chapter 5 provides mechanisms for dynamic graphs in extracting the embedded geometric information within basketball games and leverages techniques in applied topology, algorithms, and machine learning; and Chapter 6 uncovers the main theoretical results (directly addressing our alternating sequence), as well as a plethora of applications to space networks using exotic mathematical structures found in algebraic topology and tropical geometry. Finally, Chapter 7 ties together the value of dynamic graphs and Chapter 8 looks ahead to future research directions. This dissertation has been completed under the *Program in Applied and Computational Mathematics*, so many of the results are powered by code, which is described in Appendix A.

1.3.2. Main Contributions

In this work, as indicated by the title, we contribute both theoretical and practical results. From a theoretical perspective, even with existing tools, we aggregate several areas of mathematics in novel ways, provide a detailed review of relevant literature, and demonstrate novel connections between different constructions within mathematics and computer science. We also contribute several novel ideas, mainly:

1. In Section 4.1, we define a feature-based dynamic network model that captures the propagation of viral data across a population. While we use this model for viral information, this model can handle any viral particle spreading through any notion of a connected population. We give a sketch of a theoretical analysis of this model and construct both a differential and agent-based version of this model.

2. In Section 4.2, we define a dynamic network model, TraSIR, that captures viral spread across a population with a binary separating class. The particular application is for

asymptomatic and symptomatic spread in COVID-19. This model can also capture feature that induces a binary partition the population. We also give a theoretical analysis of this model.

3. In Section 5.1, we define the novel concept of a "crossing" and connect it to important definitions with geometry and topology. We demonstrate its capacity to serve as a metric space with appropriately chosen metrics and connect this to the existing literature on clustering.

4. In Section 5.2, we construct a novel pipeline for analyzing geometric data using dynamic networks and machine learning. We use this in the context of basketball, but this pipeline is generic enough to handle many different types of geometric data that is tractable for "semantic" analysis.

5. In Section 6.1, we provide a robust set of definitions for dynamic networks. We provide a translation of common static graph definitions (e.g. connectivity) and elevate them to the dynamic setting. We prove several results about these constructions. We also introduce the notion of summary graphs, describe their utility in the general case, construct two examples, and provide some results about our constructions. Finally, we also make connections to zigzag persistence, algebraic topology (through sheaves), and game theory.

6. In Section 6.2, we provide an overview of tropical geometry and max-plus algebra in the context of graphs. We also construct the notion of parameterized graphs, a notion of tropical graphs, a notion of optimality, and demonstrate the utility of optimization problems in this setting with relevant connections to existing graph algorithms. We prove results about these tropical graphs and some results with respect to well-known *graph centrality measures*, which we define later on. We finally show stability results of these optimization problems.

From a practical perspective, we give concrete implementations of important classes of algorithms, write code to simulate our ideas, analyze data, and demonstrate several empirical results. We connect our ideas to fun and important applications areas, namely sociology, epidemiology, sports, and satellite networks. Our main contributions are as follows:

- In Section 4.1, we simulate our differential feature-vector model and provide a thorough sensitivity analysis. We then take real network data from Facebook and simulate the spread of viral information. We demonstrate the ability of our network to capture viral spread and its invariance to different network topologies. We also have a codebase for this model in MATLAB.

- In Section 4.2, we run simulations of our TraSIR model against real COVID-19 data. We display the results of our simulation and demonstrate the ability of our model to capture real-world data. Finally, we perform parameter estimation and derive a first-principles estimate of the asymptomatic COVID-19 spread rate that we validate from the literature. We also have a codebase for this model in `python`.

- In Section 5.1, we deliver a pipeline to cluster basketball plays based on the underlying basketball player trajectory data. We perform manual review to demonstrate that our clustering technique is correct and we have code to support this technique.

- In Section 6.1, we construct some useful notions of graph summarization that are computationally tractable and we maintain an open-source codebase that implements these ideas for general use.

- In Section 6.2, we use our theoretical contributions to generalize shortest-path algorithms in the context of dynamic satellite networks. We run simulations on synthetic satellite data (n.b. real satellite data is restricted by *International Traffic in Arms Regulations (ITAR)* and illegal to distribute in many cases, so we do not provide real data) to demonstrate the utility of our work.

The science of operations, as derived from mathematics more especially, is a science of itself, and has its own abstract truth and value

Augusta Ada King, Countess of Lovelace

2 Mathematical Introduction

DEFINITIONS are the foundation of any mathematical theory. In this section, we provide a set of definitions, basic results, and constructions that are pervasive throughout this work. We strive to balance formal rigor and practical working results.

2.1. Graphs and their Variations

For completeness and as an homage to the roots of this dissertation, we start with our first definition.

Definition 2.1.1 (Graph)**.** Given a (finite) vertex set V, a **graph** G is at tuple

$$G = (V, E)$$

where $E \subseteq V \times V$. We generally say this is a *simple* graph.

Additionally, if $(u, v) \in E \iff (v, u) \in E$, then we say the graph is **undirected** and **directed** otherwise (we also say *digraph*). We can also construct undirected graphs by letting $E \subseteq \mathcal{P}(V) : \forall e \in E, |e| = 2$ where $\mathcal{P}$ is the power set. In other words, E is some subset of the set of all subsets of size two. This second definition aligns well with the notion of a hypergraph.

Definition 2.1.2 (Neighborhood)**.** Given a graph $G = (V, E)$, the **neighborhood** $N(u) \subseteq V$ of a vertex $u \in V$ is defined as

$$N(u) \triangleq \{v : (u, v) \in E\}$$

For a directed graph, we may have the *inbound* and *outbound* neighborhoods defined as

$$N_{\text{OUT}}(u) \triangleq N(u)$$
$$N_{\text{IN}}(u) \triangleq \{v : (v, u) \in E\}$$

Definition 2.1.3 (Degree)**.** Given a graph $G = (V, E)$, the *out-degree* and *in-degree* of a vertex are defined as

$$\deg_{\text{OUT}}(u) = |\{v : (u, v) \in E\}| \deg_{\text{IN}}(u) \qquad = |\{v : (v, u) \in E\}|$$

In other words, the out-degree (in-degree) of a vertex is simply the number of edges that start from (resp. end at) a given vertex. In an undirected graph, the out-degree and in-degree are equal for every vertex, so we unambiguously just say the **degree** and define it to be the out-degree.

Definition 2.1.4 (Path). In a graph $G = (V, E)$, we say a **path** P is a sequence of vertices, i.e. $P = (v_{i_0}, v_{i_1}, \dots, v_{i_k})$, such that for $1 \leq j \leq k$, $(v_{i_{j-1}}, v_{i_j}) \in E$. In other words, a path is a sequence of vertices where each consecutive pair of vertices in the sequence forms an edge in the digraph. P has *length* k.

Definition 2.1.5 (Shortest Path Length). Given a digraph G, let $\mathcal{P}$ be the set of all finite paths and $\mathcal{P}(i, j)$ be the set of all finite paths where the first vertex is v_i and the last vertex is v_j. We define the shortest path length $\mathrm{SP}(i, j)$ as

$$\mathrm{SP}(i, j) = \min_{P \in \mathcal{P}(i,j)} \mathrm{length}(P).$$

Definition 2.1.6 (Adjacency Matrix). Given a graph $G = (V, E)$, we can define its **adjacency matrix** A, which is an $|V|$-by-$|V|$ matrix satisfying

$$A_{ij} \triangleq \begin{cases} 1 & (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Definition 2.1.7 (Graph Diameter). Therefore, we can now define the diameter of G as

$$\mathrm{diameter}(G) = \max_{i,j \in |V|} \mathrm{SP}(i, j).$$

Notably, $\mathrm{SP}(i, j)$ can be ∞, so the diameter can be, too; in particular, this happens when no path exists.

Definition 2.1.8 (Connected Graph). A graph is a **connected** graph if its diameter is finite. A directed graph $G = (V, E)$ is *strongly connected* if it is connected and *weakly connected* if $G' = (V, E')$ is connected, where

$$E' = E \cup \{(v, u) : (u, v) \in E\}$$

Definition 2.1.9 (Weighted Graph). A graph $G = (V, E)$ can be promoted into a **weighted**

graph with a *weight function* ω where

$$\omega : E \to \mathbb{R}$$

We generally write $G = (V, E, \omega)$.

From here, we can define the core object of study quite quickly.

Definition 2.1.10 (Dynamic Graph). Given a (finite) vertex set V and a totally ordered indexing set $\mathbb{T}$, we define the sequence of edge sets $(E_t \subseteq V \times V)_{t \in \mathbb{T}}$. We can then define the dynamic graph $\mathcal{G}$ as

$$\mathcal{G} = (G_t)_{t \in \mathbb{T}}$$

where $G_t = (V, E_t)$. If for all t, $(u, v) \in E_t \iff (v, u) \in E_t$, then the dynamic graph is **undirected**, and we define **directed** analogously to a simple graph.

2.2. Applied Topology & Geometry

2.2.1. Simplicial Complexes

Definition 2.2.1 (Convex Hull). A set of points in Euclidean space (or any real affine space) is **convex** if for every two points in the set, the line segment between the two points is also in the set. Given a set X of points in Euclidean space, its **convex hull** is the (unique) minimal convex set that contains X. See Figure 5.6 for a real example of two convex hulls we produced from data.

Definition 2.2.2 (Simplex). Given a set of $k + 1$ points $V = \{v_0, \ldots, v_k\}$, the k-simplex is the k-dimensional convex hull of V. We write this simplex σ as $\sigma = [v_0, v_1, \ldots, v_k]$ A simplex is thus a combinatorial geometric object for a set of points within a geometric space and is generalization of our natural notion of a "point," "line segment," "face," or "polytope" for a set of points. For example, a tetrahedron has several simplices:

1. Each point of the tetrahedron is a 0-simplex.
2. Each line segment connecting two points is a 1-simplex
3. Each triangular face defined by three points is a 2-simplex
4. The whole tetrahedron is itself a 3-simplex.

Definition 2.2.3 (Face). A **face** τ of a simplex σ is a simplex formed by a non-empty subset of the vertices of σ, i.e.

$$\tau \subseteq \sigma$$

Definition 2.2.4 (Simplicial Complex). A **simplicial complex** K is a finite collection of simplices such that:

1. Every face of a simplex in K is also in K.
2. The intersection of any two simplices in K is a face of both simplices.

2.2.2. Computational Geometry

Definition 2.2.5 (Triangulation). A **triangulation** of a set of points X in $\mathbb{R}^d$ is a subdivision of the convex hull of X into a set of non-overlapping simplices, such that each vertex of the simplices belongs to X, and the intersection of any two distinct simplices is either empty, a shared vertex, or a shared edge.

Formally, given a set of points $X = \{x_1, x_2, \ldots, x_n\}$, a triangulation T is a collection of simplices $\{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ satisfying the following conditions:

1. The union of all simplices in T is equal to the convex hull of X. In other words, if we combine all of the simplices, then we recover the convex hull of all points.
2. The intersection of any two distinct simplices in T is either empty, a shared vertex, or a shared edge, i.e., $\sigma_i \cap \sigma_j = \emptyset$, a vertex, or an edge for all $i \neq j$.

Definition 2.2.6 (Voronoi Diagram)**.** Given a set of (distinct) points $X = \{x_1, x_2, \ldots, x_n\}$ in $\mathbb{R}^d$, a **Voronoi diagram** is a partition of the space into cells, where each cell corresponds to a point in X. The cell associated with point x_i consists of all points in $\mathbb{R}^d$ that are closer to x_i than to any other point in P.

Formally, the Voronoi cell $V(x_i)$ corresponding to point x_i is defined as:

$$V(x_i) = \{x \in \mathbb{R}^d : \forall j \neq i, \|x - x_i\| \leq \|x - x_j\|\}. \tag{2.1}$$

To create the Voronoi Diagram, we draw the set of Voronoi Cells for each point x, which is defined to be the locus of points closest to p (usually using the Euclidean distance, though any metric will suffice).

Definition 2.2.7 (Delaunay Triangulation)**.** Given a set of points $X = \{x_1, x_2, \ldots, x_n\}$ in $\mathbb{R}^d$, a Delaunay triangulation is a triangulation of X that satisfies the empty circumcircle property, i.e. the circumcircle (or circumsphere) of each simplex in the triangulation contains no other points of X in its interior. Formally, a triangulation T of X is a **Delaunay triangulation** if for each simplex $\sigma \in T$, the open ball circumscribing σ contains no points of X.

Notably, a Delaunay triangulation can be constructed using Voronoi cells for a set of (distinct) points: we construct the Voronoi diagram for our points and then we draw a straight line segment between every pair of points whose Voronoi cells are adjacent or "touch." In other words, we connect points whose Voronoi cells are next to each other. This set of points and line segments forms the Delaunay Triangulation, since the set of line segments will in fact be a triangulation over the convex hull of the set of points.

See Figure 5.5 for a real example of a Delaunay triangulation.

2.2.3. Persistent Homology

Definition 2.2.8 (Filtration). A **filtration** of a simplicial complex K is a nested sequence of subcomplexes K_i such that:

$$K_0 \subseteq K_1 \subseteq \ldots \subseteq K_n = K$$

Definition 2.2.9 (Persistence Module). **A persistence module** V is a collection of vector spaces V_i and linear maps $f_{i,j} \colon V_i \to V_j$ such that:

1. $f_{i,i} = \mathrm{id}_{V_i}$ for all i.

2. $f_{j,k} \circ f_{i,j} = f_{i,k}$ for all $i \leq j \leq k$.

Definition 2.2.10 (Simplicial Homology). Given a topological space X, the k^{th} homology group $H_k(X)$ is a group that captures the k-dimensional "holes" or "cycles" in the space. Intuitively, $H_0(X)$ represents the connected components of X, $H_1(X)$ represents loops or one-dimensional holes, $H_2(X)$ represents two-dimensional voids, etc.

The *k-th chain group*, denoted $C_k(K)$, is the free abelian group generated by the set of k-dimensional simplices in K. Elements of $C_k(K)$ are called k-chains and can be represented as formal linear combinations of k-simplices with integer coefficients. The *boundary operator*, denoted $\partial_k : C_k(K) \to C_{k-1}(K)$, is a linear map that takes a k-chain and returns its $(k-1)$-dimensional boundary. The boundary operator is defined as follows:

$$\partial_k(\sigma) = \sum_{i=0}^{k} (-1)^i [v_0, \ldots, \hat{v}_i, \ldots, v_k]$$

where $\sigma = [v_0, v_1, \ldots, v_k]$ is a k-simplex and $\hat{v}_i$ indicates that the vertex v_i is omitted from the sum.

The k^{th} **simplicial homology group**, denoted $H_k(K)$, is defined as the quotient group of the k^{th} cycle group by the k^{th} boundary group:

$$H_k(K) = Z_k(K)/B_k(K)$$

where $Z_k(K) = \mathrm{kernel}(\partial_k)$ is the k-th cycle group (chains with zero boundary), and $B_k(K) = \mathrm{image}(\partial_{k+1})$ is the k^{th} boundary group.

Definition 2.2.11 (Persistent Homology Groups). For each dimension k, the **persistent homology groups** are the homology groups of the filtered simplicial complex.

$$H_k(K_i) \xrightarrow{f_{i,j}} H_k(K_j)$$

Definition 2.2.12 (Persistence Diagram). A **persistence diagram** is a multiset of points in $\mathbb{R}^2$ that represents the birth and death times of topological features in the filtration.

$$D = \{(b_1, d_1), (b_2, d_2), ..., (b_n, d_n)\}$$

Definition 2.2.13 (Persistence). The persistence of a topological feature is the difference between its birth time and death time, i.e. $p_i = d_i - b_i$.

2.3. Basic Graph Operations

Given a graph, one classical algorithm to find the shortest path from one vertex to the others is Dijkstra's Algorithm, see in Algorithm 1.

Algorithm 1 Dijkstra's Algorithm

Require: A connected, weighted graph $G = (V, E, \omega)$ such that $\omega \geq 0$ and a source vertex s

Ensure: A list of shortest path distances d from s to all other vertices

```
procedure DIJKSTRA(G, s)
    $Q \leftarrow V, d[s] \leftarrow 0$
    for $v \in V \setminus \{s\}$ do
        $d[v] \leftarrow \infty$
    while $Q \neq \emptyset$ do
        $u \leftarrow \arg\min_{v \in Q} d[v]$
        for $v \in N(u)$ do
            alt $\leftarrow d[u] + \omega(u, v)$
            if alt $< d[v]$ then
                $d[v] \leftarrow$ alt
    return $d$
```

My brain is open.

Paul Erdős

3 Brief Survey

As with many other fundamental mathematical structures, graphs have found utility across the scientific and literary disciplines. Without exaggeration, graphs have become a defining mathematical structure in nearly all science, engineering, mathematical, and computational disciplines. Part of their usefulness comes from the ease of visualization: graphs are simple and quick to draw. While we cannot cover everything in our survey, we touch upon the important foundations of our work: we are but people standing on the shoulders of giants and here, we try to cover the giants standing right below us. More precisely, we consider a brief range of other topics that directly entertain dynamic networks.

In this survey, we look at:

1. Graph dynamical systems: dynamical systems that are defined with respect to a graph

structure.

2. Applied topology: we consider the foundational tools of applied topology (and topological data analysis) with respect to dynamic networks.

3. Machine learning: we look at graph neural networks and what we can say about them with tools from machine learning and, appropriately, graph neural networks.

3.1. Graph Dynamical Systems

Graph dynamical systems (GDS) are mathematical models that describe the evolution of the state of a graph over time. They have become a powerful tool for modeling complex networks in various fields, including biology, social sciences, and engineering. This survey aims to provide an overview of the key concepts, methods, and applications of graph dynamical systems, with a focus on the different types of dynamical systems, their properties, and their relevance to real-world problems. The cited works cover a wide range of topics and offer a comprehensive view of the field.

3.1.1. Continuous-Time Dynamical Systems

Continuous-time dynamical systems describe the evolution of the state of a graph as a continuous function of time. Examples of continuous-time GDS include the Kuramoto model[160], which describes the synchronization of coupled oscillators, and the Lotka-Volterra model[129], which models the dynamics of interacting populations.

3.1.2. Discrete-Time Dynamical Systems

Discrete-time dynamical systems describe the evolution of the state of a graph as a discrete function of time, with updates typically occurring at fixed time intervals. Examples of discrete-time GDS include cellular automata[297], which model spatially distributed systems, and the Ising model[138], which describes the dynamics of magnetic systems.

3.1.3. Stability, Controllability, and Observability

Stability is a crucial property of GDS, as it determines the long-term behavior of the system. Lyapunov stability[180] is a common concept used to analyze the stability of GDS, while more specialized stability notions, such as synchronization stability[225] and consensus stability[217], have been developed for specific classes of GDS.

Controllability and observability are fundamental concepts in the study of GDS, as they describe the ability to manipulate and monitor the state of a graph. The structural controllability and observability of GDS have been extensively studied[171,176], with applications in network design and control.

3.1.4. Biological Systems

GDS have been widely applied in the study of biological systems, such as gene regulatory networks[151], metabolic networks[219], and neural networks[131]. These models have provided valuable insights into the mechanisms underlying various biological processes and have guided experimental research in the life sciences.

3.1.5. Social and Economic Systems

GDS have been used to model social and economic systems, such as opinion dynamics[125], spreading processes[223,222], and traffic flow[66]. These models have helped researchers understand complex phenomena in these fields, such as the emergence of consensus or polarization, the spread of information or diseases, and the optimal management of transportation networks.

3.1.6. Engineering and Technology

In engineering and technology, GDS have found numerous applications, including power grid stability[203], communication networks[86], and robotic swarms[31]. These models have been used to design more efficient and robust systems, as well as to develop control strategies for their optimal operation.

Graph dynamical systems provide a versatile and powerful framework for studying the behavior of complex networks in a wide range of domains. This survey has presented an overview of graph dynamical systems, with a focus on their properties and relevance to real-world problems. The cited works offer a broad view of the field, and future research is expected to continue expanding our understanding of GDS and their applications.

3.2. Applied Topology

As a subfield of computational topology, applied topology and topological data analysis (TDA) provides a framework for studying the multi-scale topological properties of real-world datasets. In particular, persistent homology and related techniques, enables the detection and quantification of topological features in networks and other complex systems. It has become a powerful tool for analyzing dynamic networks in various fields, including biology, social sciences, and engineering. This survey aims to provide an overview of the key ideas, constructions, and uses of applied topology in dynamic networks, with a focus on their potential to uncover hidden structures and properties in complex systems. The cited works cover a wide range of topics and offer a comprehensive view of the field. Notably, this is a robust and exciting area of research with many active researchers. Future research is expected to continue expanding our understanding of applied topology and its applications in the analysis of dynamic networks.

3.2.1. Simplicial Complexes and Persistent Homology

Simplicial complexes are combinatorial structures used to represent the topology of a network[81]. They provide a natural framework for studying topological properties, such as connected components, loops, and voids. Persistent homology, a method developed by (among others) Edelsbrunner and Harer[80,82], is a technique for quantifying topological features in simplicial complexes and their persistence across different scales. This technique has been widely applied to the analysis of dynamic networks, providing insights into their structure and evolution.

3.2.2. Mapper Algorithm

The Mapper algorithm, introduced by Singh, Mémoli, and Carlsson[261], is another technique in applied topology that has been used for analyzing dynamic networks. The algorithm constructs a simplified representation of the network by clustering nodes based on a given function and then connects these clusters to form a topological graph. This method has been employed in various applications to reveal the underlying structure and organization of dynamic networks.

3.2.3. Biological, Sociological, and Economical Systems

Applied topology has been used to analyze dynamic networks in biological systems, such as protein-protein interaction networks[229], gene regulatory networks[46], and brain networks[103]. These techniques have been successful in identifying topological features associated with specific biological functions and processes, as well as revealing the organization and dynamics of these networks.

In social and economic systems, applied topology has been used to study the structure and evolution of social networks[300], collaboration networks[224], and financial networks[235]. These analyses have provided valuable insights into the formation and stability of communities, the impact of network topology on information diffusion, and the identification of critical nodes and links.

3.2.4. Sensor and Transportation Networks

Applied topology has also been employed in the analysis of dynamic networks in engineering and technology, such as sensor networks[68], communication networks[30], and transportation networks[168]. These applications have led to the development of new methods for network design, optimization, and control, as well as improved understanding of the impact of network topology on system performance and resilience.

3.3. Machine Learning with Graph Neural Networks

Machine learning techniques, particularly graph neural networks (GNNs), have emerged as powerful tools for learning complex patterns and making predictions in dynamic networks. Machine learning is a truly enormous area of study, representing some of the most cited scientific works of all time. No person (but perhaps well-trained neural network) could hope to keep up with the literature in this area. The term itself has many different meanings and covers everything from neural networks to Bayesian learning, genetic algorithms, and so much more. Even restricting our scope to GNNs in the context of dynamic networks provides a voluminous literature, so we present a small sampling to address both theoretical and practical aspects, and highlighting the potential of these techniques to advance our understanding and analysis of complex dynamic systems.

As a general matter, machine learning is moving towards large, pre-trained networks and then constructing task-specific fine-tuned, zero-shot, or few-shot networks. This paradigm would be the gold standard for dynamic networks, but this setup does not yet exist in this area: it would require the training of a large-scale general network for extracting features from dynamic networks.

3.3.1. Machine Learning and Dynamic Networks

Machine learning is a subset of artificial intelligence that focuses on developing algorithms capable of learning patterns from data and making predictions or decisions. In the context of dynamic networks, machine learning techniques have been employed to address various tasks, including link prediction[318], community detection[304], and anomaly detection[233]. These techniques have been successful in uncovering hidden structures, predicting network evolution, and identifying critical events in dynamic networks.

3.3.2. Graph Neural Networks

Graph neural networks (GNNs) are a class of machine learning models specifically designed to handle graph-structured data[299]. GNNs combine node features with the network's topology to learn powerful representations capable of capturing complex patterns in the data. GNNs have been employed to address a wide range of tasks in dynamic networks, including node classification[108], link prediction[310], and graph generation[306].

GNNs have been adapted to handle dynamic networks through several approaches, such as recurrent graph neural networks (R-GNNs)[255], spatio-temporal graph convolutional networks (ST-GCNs)[302], and temporal graph attention networks (TGATs)[301]. These models incorporate temporal information into the graph learning process, allowing them to capture the evolution of the network's structure and features over time.

3.3.3. Scientific Systems

Machine learning and GNNs have been applied to dynamic networks in biological systems, such as protein-protein interaction networks[93], gene regulatory networks[182], and neuronal networks[220]. These techniques have contributed to the identification of novel biomarkers, the prediction of disease progression, and the understanding of the dynamic processes underlying cellular functions.

In social and economic systems, machine learning and GNNs have been used to analyze and predict the behavior of dynamic networks, such as social networks[282], financial networks[309], and online user behavior[319]. These techniques have enabled the identification of influential individuals, the prediction of market trends, and the understanding of user preferences and interactions over time.

Dynamic networks are also prevalent in various engineering systems, such as sensor networks[170], transportation networks[313], and communication networks[50]. Machine learning and GNNs have been employed in these systems to optimize network design, improve system performance, and enhance resilience against failures and attacks.

3.4. Dynamic Networks So Far

As a general matter, there is no unified framework for studying dynamic graphs. Instead, we have piecemeal techniques in tackling specific tasks or applications. In conferences like *Complex Networks* or through the use of graph tools like graph motifs and kernels, we have made much progress in the past decade in analyzing dynamic networks. This work seeks to provide a unified view on the subject.

Dynamic networks go by many names: temporal networks, spatiotemporal networks, graph dynamical systems, etc. As a class of complex systems where nodes and edges may change over time[130], we provide a survey with an overview of networks from a dynamical systems and complex systems perspective. We discuss their characteristics, modeling approaches, analysis methods, and various application domains.

3.4.1. Characteristics and Challenges

Dynamic networks capture the evolving relationships between entities over time. These networks are characterized by time-varying structures and properties[38], which pose unique challenges in their analysis, such as handling incomplete data, scalability, and the detection of temporal patterns[1].

Several approaches have been developed to model dynamic networks, including time-aggregated graphs[206], time-varying graph models[273], and continuous-time models[105]. Time-aggregated graphs represent networks as a sequence of static graphs, while time-varying graph models consider discrete time steps with varying node and edge sets. Continuous-time models use mathematical frameworks, such as stochastic processes or differential equations, to describe network evolution.

Methods for analyzing dynamic networks range from traditional graph analysis techniques to more recent machine learning-based approaches[121]. Techniques such as temporal centrality[154], community detection[115], and link prediction[292] have been adapted to handle the temporal nature of dynamic networks. Additionally, machine learning methods, particularly graph neural networks, have shown promising results in tasks like node classification and graph generation[318].

3.4.2. Application Domains

Dynamic networks are present in various domains, including social networks[263], biological networks[21], chemical networks, financial networks[187], physical networks, transportation networks[313], sociological networks, communication networks[50], etc. Analyzing dynamic networks in these domains has enabled insights into network resilience, influence maximization, trend prediction, and the identification of hidden structures. While there is no comprehensive technique for analyzing all of these different network types, dynamic networks arise naturally in nearly every scientifc (and many non-scientific) domain. As systems become more complicated with long-range interactions and as we continue to generate massive datasets in our era of *Big Data*, we have more and more opportunities to analyze real-world systems through the lens of dynamic networks.

Part II

Main Results

Rumor, that which no other evil thing is faster.

Publius Vergilius Maro (Virgil)

4

Tracking Virality in Connected Populations

VIRALITY is an informal and formal emergent property in a wide range of dynamical systems. This chapter covers two studies of viral systems over dynamic graphs. Each study involves a collection of people, connected in various ways, with some "viral" element traveling across this population. In both studies, we construct a parametric model to track this spread and then we answer several typical questions about each model:

1. What does the theory predict for the dynamics of these models?
2. How well do these models explain real-world data?
3. Given real-world data, what are our estimates for the underlying model parameters?

While no model can capture the world around us with perfect fidelity, each model contributes to our understanding of dynamical systems, especially where there is an underlying dynamic network. Each model builds upon existing literature to capture some novel or unexplored element of viral spread. While each model has increased computational costs, mathematical operations over networks can be vectorized and dispatched on specialized computing devices, e.g. a GPU. Moreover, each model as implemented uses a simple notion of dynamic networks; while there are certainly additional dynamics that could be incorporated into each model, the main contribution for each study is the introduction of a dynamic network.

The first study examines the nature of viral information propagating through a social network with constantly shifting connections and interests. In it, we provide a flexible framework that has both global model parameters and local model parameters that are specific to the "people" within our system, all sitting on top of a graph that can also be controlled through additional parameters. Then, for practical purposes, we further constrain our system under certain regularity assumptions and select base model parameters based on real-world observations. The second study takes a similar approach, but for analyzing an actual virus (COVID-19) spreading through a country. Just as with the first study, we create a flexible framework that has global model parameters and local control parameters for the specific graph structure, which we initialize with real-world data and certain simple assumptions. Taken together, these studies provide two different but related models on studying viral spread throughout a population, each providing a different set of control parameters. Arising from particular applications in mind, we test each model on a paragon example and demonstrate the capacity to capture salient properties of real-world data, the first hinging on Facebook data and the second predicting asymptomatic spread characteristics of COVID-19.

4.1. Information as a Virus

This project focuses on the spread of viral information in a dynamic social network, as if it were a virus being transmitted through a population. Based on the Daley-Kendall model from 1965, this study provides an updated perspective in this setting. The key innovation is the introduction of personalized feature vectors that control spread in an ever-changing social graph based on Facebook data. We provide both a differential and agent-based model, perform some analysis typical of dynamical systems, and then provide experimental evidence of the model's ability to capture a real information system. Building upon a previous publication, we extend these ideas by providing a more detailed theoretical analysis and then extending these ideas in Section 4.2.

4.1.1. Introduction

Background

A rumor is defined as a "proposition for belief of topical reference disseminated without official verification,"[155] a notion that lends itself quite well to the imagination of applied mathematicians. The mathematics of rumor spread is somewhat explored, beginning with the epidemic model applied to information spread in a population by Daley and Kendall in 1965[67]. This model's assumptions of homogeneous interactions and its lack of well-defined parameters likely caused "the superficial similarity between rumors and epidemics to break down on closer scrutiny"[67]. Nonetheless, the similarities of knowing and spreading a rumor, and having and spreading a disease, share parallels that only deviate in some of the intricacies of their mechanism. In both, an "infected" individual in a network desires to (or inadvertently spreads) their "condition." With disease, one of the mechanisms of suppressing spread is vaccination; with rumors it is an individual's eventual boredom and desire for novel information.

More recent models of rumor spread in a population examined the dynamics through randomized networks[150], examining the rumor transmission in exponentially distributed networks[197], and the time of rumor spread given contacts with the initial spreader in a regular network[92]. There is increasing emphasis on the structure of networks themselves and how this affects model dynamics[312,228,227,18,317]. Many of these models derive their structure and dynamics from complex yet internally homogeneous simulations of how individuals interact with rumors. However, with the advent of massive social media networks that allow sharing of information on a large scale, more analyses are focusing on behaviors (such as rumors) that are frequent in social media networks. Simulations of rumor spread through social media are becoming increasingly realistic by including "forgetting" mechanisms typical of social media[315], comparing time of rumor spread in random networks as compared to structured networks[173], methods for combatting rumor spread in social networks[281], and examining rumor spread on gaming networks[110].

The ISTK Model

In this work, we consider a stochastic rumor spread model with four categories of individuals: the "ignorant" individuals, those who have never heard the rumor; the "spreaders," those that have heard the rumor and are actively spreading it; the "stiflers," those who have heard the rumor and actively suppress further transmission (either because they now consider the rumor old news, or they never believed the rumor in the first place); and finally the "knowledgeable" population, those who have heard, but have subsequently forgotten the rumor. The rumor initializes in only a small fraction of the population, and spreads as the individuals interact. The "ignorant," "spreader," and "stifler" populations were presented in the Daley-Kendall model[67], but we have added a "knowledgeable" population, which has been postulated before as necessarily distinct from the ignorant population[314,315]. Assuming otherwise presumes that the attitude of an individual who has forgotten a rumor is identical to the behavior of an individual who had not yet heard the rumor. We account for this distinction with the addition of the knowledgeable population to the Daley-Kendall model. This model is henceforth referred to as the **I**gnorant, **S**preader, s**T**ifler, and **K**nowledgeable (ISTK) model. We use three variations of this model: one differential, and two agent-based. The differential ISTK model simulates a homogenous group of people, and has no awareness of the concept of individuals; it simply "moves" proportions of the group of people from one population to another over time. The first agent-based model, the "Simple model," simulates individuals through several iterations (rounds) over time. The model incorporates a network that represents the connections between individuals, which in this case is based off of Facebook friends. The second agent-based model, the "Feature-vector model," incorporates demographic data of these Facebook users.

In the "feature vector model," we further consider 1. how a rumor might be targeted towards a certain demographic, and 2. how the "similarity" between a rumor and an individual affects a user's behavior. The original social network dataset included many different types of "features:" education level, gender, and language. Instead of assuming that every individual is equally likely to spread any rumor, we assumed that the rumor's targeted characteristics and the demographic information of each individual affected the likelihood of the rumor to spread. In this way, we equipped the rumor with a *personality*. If the individual from whom they heard the rumor was more similar to them, they were more likely to believe the rumor, and if the rumor's characteristics was more similar to theirs they were more likely to spread the rumor. There is evidence to suggest that people are more likely to believe information that comes from others with similar values[100]. The similarity of the rumor's personality to that of the individual's influenced the individual's probability to spread the rumor. The theory of confirmation bias suggests this behavior, insofar as we are more likely to accept information that confirms our previous beliefs[293]. The characteristics of the rumor itself contribute to how the rumor is spread between individuals in the Feature vector model, Section 4.1.3.

ISTK Model Equations

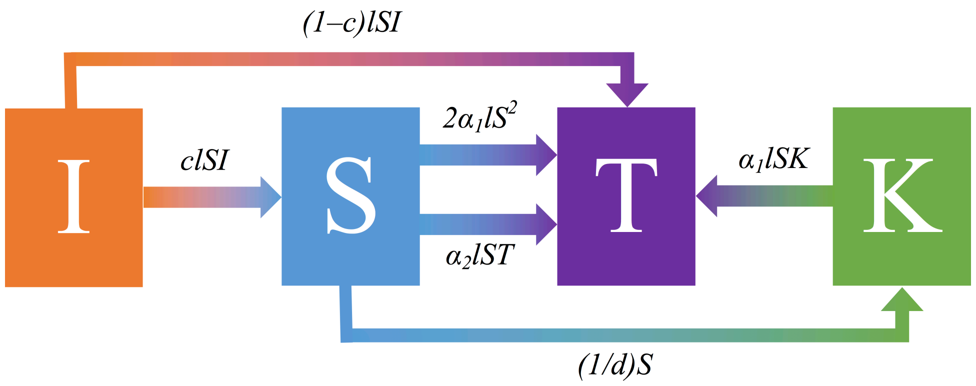

Figure 4.1: The ISTK Model. This network-driven model is inspired by the traditional SIR viral model and then Daley-Kendall model. There are four compartments: the **I**gnorant, **S**preader, s**T**ifler, and **K**nowledgeable classes. The arrows in this diagram characterize the flow between the classes and the relevant controlling parameters.

I, S, T, and K, represent the total Ignorant, Spreader, Stifler, and Knowledgeable populations respectively. We also designate N to represent the total size of the population and enforce the invariant

$$N = I + S + T + K$$

(n.b. we assume no one dies or is born, so N stays constant, i.e. $\frac{\mathrm{d}N}{\mathrm{d}t} = 0$).

There are several common parameters to all four differential equations, as follows:

- We represent the "credibility" of the rumor, expressed as a probability that the Ignorant believes the Spreader, as c. Therefore $(1 - c)$ the complement of c is equivalent to being incredulous of the rumor.

- To represent the chance per day of interaction, l, we take the complement of the overall probability that an individual does not talk with a single Spreader. It is computed by a set of Bernoulli trials with success probability ρ where $\rho = 1 - \frac{S}{N}$ and number of trials to be τ (i.e. $l = 1 - \rho^\tau$).

- We use d to represent the number of days after which a population spontaneously forgets a rumor. Note that in our equations, we use $\frac{1}{d}$ because $d \in [1, \infty)$, so $\frac{1}{d} \in [0, 1]$.

We use α_1 to describe the loss of novelty of the rumor and α_2 to describe the chance that the Spreader becomes a Stifler upon interacting with a Stifler.

The equations for each population follow:

$$\frac{\mathrm{d}I}{\mathrm{d}t} = -clSI - (1-c)lSI \tag{4.1}$$

In Equation 4.1, the first term describes the interaction between Ignorants and Spreaders. It is dependent on both the size of the Ignorant and Spreader classes, and is proportional to parameters c and l. The second term of Equation 4.1 accounts for the complement of "believing the rumor" (being "incredulous" of the rumor) and is proportionate to $(1-c)$. Both terms remove some of the population from the Ignorant class and lead to the Spreader class and Stifler class respectively. Although we can simplify this equation to $(-lSI)$, we want to distinguish the credulous (c) and incredulous ($1-c$) group of people.

$$\frac{\mathrm{d}S}{\mathrm{d}t} = clSI - \frac{1}{d}S - 2\alpha_1 lS^2 - \alpha_2 lST \tag{4.2}$$

The first term of Equation 4.2 is the addition of members from the Ignorant class who believed the rumor. The second term characterizes the population which spontaneously forgets the rumor, and hence is inversely proportionate to the Spreader population and d. The third term of Equation 4.2 accounts for two Spreaders who interact with each other and foster disinterest in the rumor (since the rumor has lost its novelty). When these two Spreaders interact (S^2), we account for the chance that **each** Spreader could become a Stifler by multiplying the term by 2. The final term represents the disillusionment power of a Stifler when interacting with a Spreader.

$$\frac{\mathrm{d}T}{\mathrm{d}t} = 2\alpha_1 lS^2 + \alpha_1 lSK + \alpha_2 lST + (1-c)lSI \tag{4.3}$$

In equation 4.3, the first of which describes the removal of members from the Spreader class into the Stifler as defined above. The second term describes the population of Knowledgeable individuals who become a Stifler, as described in Equation 4.2. The third and fourth terms of Equation 4.3 describe the addition of members to the Stifler population from the Spreader and Ignorant populations, respectively.

$$\frac{\mathrm{d}K}{\mathrm{d}t} = \frac{1}{d}S - \alpha_1 lSK \tag{4.4}$$

Finally, Equation 4.4 describes the individuals in the Spreader class who forget the rumor and become Knowledgeable; and the population which loses the novelty of the rumor and become Stiflers.

In our model, we do not consider the interaction between a Stifler and an Ignorant because neither one has a reason to broach the subject of a rumor (the Ignorant because they do not know and the Stifler because they no longer care). Moreover, by a similar logic, when a Knowledgeable and Stifler interact, there is no change in populations.

Our equation differs from the Daley-Kendall model[67] primarily through augmenting the system of the Knowledgeable class; individuals in this category would have been in the Ignorant of Stifler classes in the original model. The Daley-Kendall model also assumes all individuals who hear a rumor will believe it, whereas in our case, this is not required to be true and can be controlled through parameter c. Finally, note that in the Daley-Kendall model it is possible to become ignorant after hearing the rumor; in the ISTK model, this is not possible.

4.1.2. Differential Model

Modeling Applications

By solving the differential model, we can compare a continuous model to a stochastic agent-based model. We can further how parameters based on face-to-face interaction had an impact on rumor spread versus interaction over a network (described by an adjacency matrix). More precisely, we examine how long it takes a rumor to reach a significant proportion (90%) of the population and the effect on the amount of time until steady states were reached with perturbations in initial parameters. To do this, we leverage data on a model of consumer goods.

Estimating Parameters

The parameters necessary to estimate were *credibility*, the *loss of novelty*, and the *number of close interactions* for an individual. Using consumer statistics on perceptions of reliability of information from different sources, we initially estimated the credibility $c = \frac{2.8}{7}$[149]. Estimations in number of close contacts varied from 12–26 people per day, varying based on age[39,201,83]. We took the average number of close contacts to be 22 (i.e. $\tau = 22$, which determines our parameter of interaction l). Although the differential model itself does not change with the medium of Facebook, the meaning of τ changes. Instead of 22 close interactions per day, we selected an appropriate analog, in that we assumed the average individual reads approximately 22 posts per day. We estimated the value representing loss of novelty at $\alpha_1 = .01$, and $\alpha_2 = .02$, since the spreaders will have a stronger effect on the stiflers. In all cases, these were the "baseline" parameters, and were only modified for the Feature vector model, in which case parameters c and α_1 were based on the features vectors of agents, and rumors. Sensitivity analyses for c, and δ were run on the interquartile ranges of the studies on which they were based. α_1 was an approximated variable, so we simply run as small of an α_1 variable that the differential model was capable of processing, up to an α_1 value of .25.

4.1.3. Agent-Based Models

Simple Model Method

In order to incorporate a network into our model, we constructed an agent-based model (a useful stochastic technique for modeling dynamics with graphs)[264]. We discretized the data we used for the differential model, in essence using our parameterized proportions as the probabilities that certain individuals would move between populations. Using data from the Facebook network, we allowed individuals to communicate only with those to whom they are connected. Since agent-based models are based on probabilities, and are inherently nondeterministic, we essentially have to perform many “trials” of the model. Each trial consists of initializing a set of “agents” into one of the four populations: Ignorant, Spreader, Stifler, or Knowledgeable. In each trial, there are several time-steps, at which point each agent has a “turn.” At each turn, an agent can interact with other agents, and move from one population to another. The rules that define what an agent can or cannot do on a turn are described by the ISTK model. For example, in the differential ISTK model, an Ignorant becomes a Spreader by the term $clSI$. Translating this term to the agent-based model: lSI represents the chance that an Ignorant and Spreader interact (as characterized by the network), and c represents the credibility of the rumor, as expressed as a probability. We performed 400 distinct trials, where each trial constituted 22 days.

Each person began as ignorant, except for a randomly selected subset of the population, who became spreaders. The chance of becoming a spreader was set at 5% distributed randomly (i.e. without considering the network). Because it is a large population (4 039 individuals), each trial would have had around 202 spreaders, but the actual number of spreaders varies from trial to trial. Additionally, it was possible for all of the spreaders to be concentrated in a subnetwork or a “pocket of friends.”

To begin each day, every user was assigned an amount of time logged in by picking randomly from a normal distribution with a mean of 23 minutes and a standard deviation of 4 minutes, bounded above 0. We made the assumption that the majority of people will probably be logged on during an 8 hour period of the day; therefore, we only modeled 480 minutes per day, within which the users select their logon time (n.b. users also could not log on in the last 23 minutes of the day, as 23 minutes a day was set as the mean browsing time). Each user had a probability of $14/365$ to “post” in a given day. Then, based on the time that they are “logged on,” users were assigned a “time” which they made their post form a uniform random distribution. This occurred on each of the 22 days that constituted a trial.

Each “day,” after determining the logon time, posting order, and post time, the simulation of rumor spread began. Every minute, each user could “view” posts written at that minute from people to whom they were **directly** connected. Users were also capped at reading 10

Reader State \ Poster State	I	S	T	K
I	—	$\mathbb{P}(S) = c = 0.8$ $\mathbb{P}(T) = 1 - c = 0.2$	—	—
S	—	$\mathbb{P}(T) = \alpha_1 = 0.01$	$\mathbb{P}(T) = \alpha_2 = 0.02$	—
T	—	—	—	—
K	—	$\mathbb{P}(T) = \alpha_1 = 0.01$	—	—

Table 4.1: Next reader state for possible interactions between reader and poster. $\mathbb{P}(X)$ denotes probability that reader changes to class X

posts a minute. If a poster was a spreader, they had chance $\delta = \frac{1}{d} = \frac{1}{22*480}$ of forgetting the rumor. Then, based off of the probabilities in Table 4.1, the state of each person was immediately recalculated. Therefore, if a person changed state at a particular minute within a day, then that person would interact as that state with other users in every minute after that. Finally, after 22 days, the trial ended.

Feature Vector Model Method

This model followed a similar logic as the preceding agent-based model. However, the different interactions accounted for the similarity between two agents or the similarity between an agent and the rumor. First, a baseline feature space of dimension $D = 195$ was taken as a subsample from the Facebook dataset[190]. Each feature corresponds to some data from the original Facebook profiles, like language, identified gender, etc. Though we can access realistic demographic data, the individual features are not particularly important. We are simply finding a metric for personality similarity to demonstrate that we can model how targeted rumors spread in the context of a (social) network. It is a reasonable assumption that a piece of viral information can be in fact targeted with demographic information. We then examine how assuming personality similarity influences credibility of rumors impacts rumor spread in a population. Each feature is boolean, taking either "true," "false," or "N/A" if the value is unknown. Any "N/As" for a given feature were filled in randomly with some probability

p, where $p = \frac{x_t}{x_f + x_t}$, x_t is the number of *true* values there were for a particular feature across the population, and x_f is the corresponding number of *false* values. The rumor itself was also initialized with a particular feature vector, each term generated randomly. A "most similar" rumor vector was generated, which was created by rounding every p to 0 or 1 for each feature. The "most dissimilar" rumor vector was the logical complement of the "most similar" rumor vector.

Next, pairwise angular similarity $S_{p,r}$ was taken between the two interacting agents, poster and reader, where

$$S_{p,r} = \cos(\theta) = \frac{\mathbf{v_r} \cdot \mathbf{v_p}}{||\mathbf{v_r}||\,||\mathbf{v_p}||}$$

where the poster has feature vector $\mathbf{v_p}$ and the reader has feature vector $\mathbf{v_r}$. Angular similarity between the feature and the reader was also determined, where

$$F_r = \cos(\theta) = \frac{\mathbf{v_r} \cdot \mathbf{f}}{||\mathbf{v_r}||\,||\mathbf{f}||}$$

and $\mathbf{f}$ is the feature vector of the particular rumor.

We also determined a "baseline" $b = 0.5$, which is the "influence" of an original parameter, and where $1 - b = 0.5$ represents the influence of the interaction of feature vectors; this baseline is simply the expectation of the Bayesian uniform prior. This baseline determined how much each parameter was affected by the similarity scores of feature vectors, and guaranteed the values would be at least half of the original model values. The simple agent-based model was run again, with $bc + (1 - (bc))F_k$ substituted for c and $b\alpha + (1 - (b\alpha))S_{p,r}$ substituted for the respective α values and agents i and j.

We tested 86 different feature vectors, with 100 trials each. In addition for our simulated "most similar" rumor and the "most dissimilar" rumor, we ran 300 repetitions with each rumor, of the stochastic agent based model, with the same population.

4.1.4. Results

Differential Model and Simple Agent-Based Model

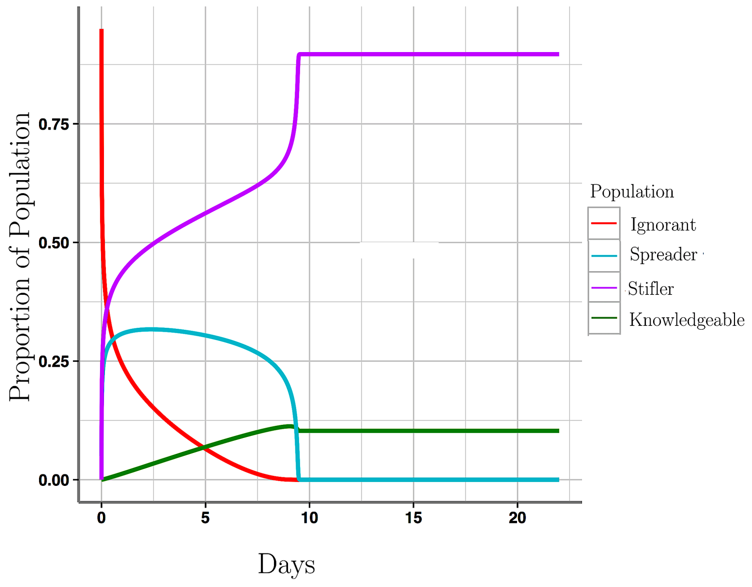

Figure 4.2: The result of numerically solving the differential model over 22 days. Each line represents a particular population class, as indicated by the legend. The total population does not change over time and is normalized to 1.

For the differential model, as is demonstrated in Figure 4.2, the spreader and ignorant populations become negligible by the end of the 22 days, and the knowledgeable and stifler populations stabilize above 0. The ignorant population declines, as the spreader population initially grows, and then declines as the stifler population grows. In the differential model, essentially all individuals learn about the rumor. Varying the parameters impacts how quickly the population hears of the rumor, but not the ignorant and spreader populations.

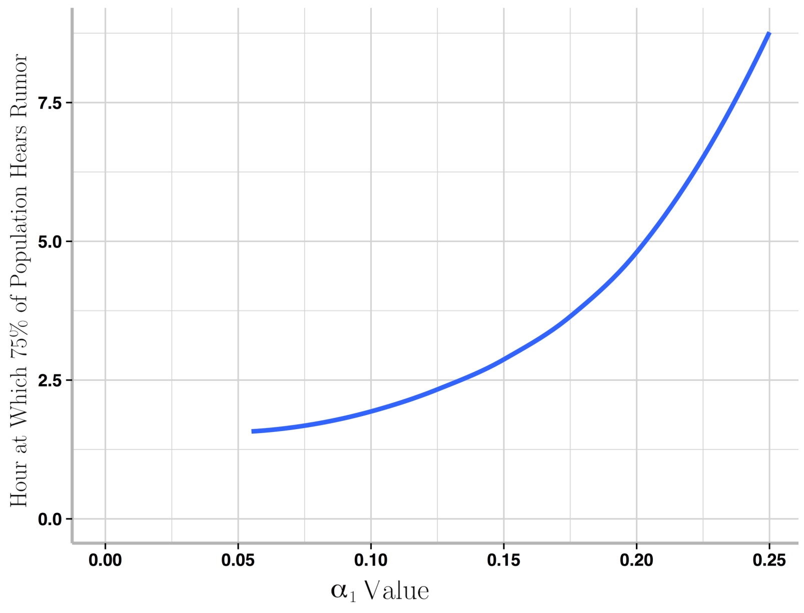

Figure 4.3: Results of the sensitivity analysis of parameter α_1 in the differential model. For each value of α_1 the time at which 75% of the population had been exposed to the rumor is recorded.

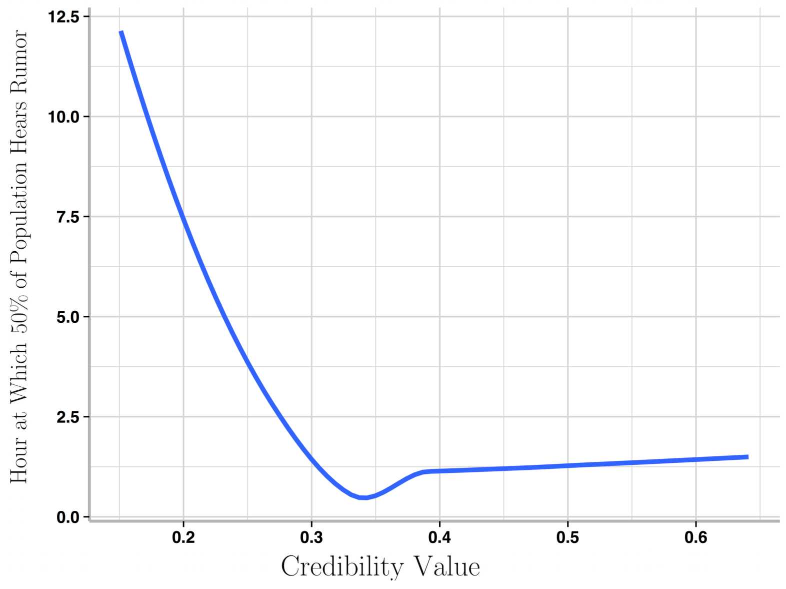

Figure 4.4: Sensitivity analysis of parameter c in the differential model. For each credibility value the time at which 50% of the population heard the rumor.

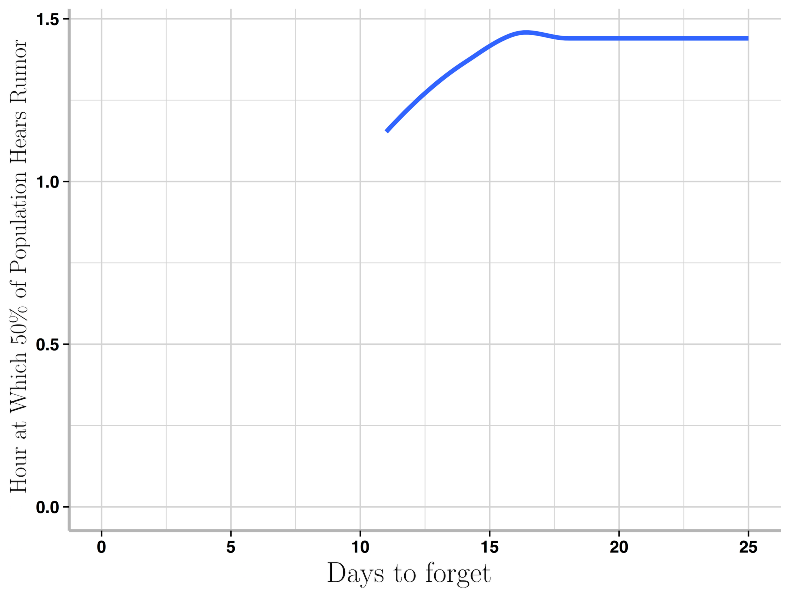

Figure 4.5: Results of the sensitivity analysis of parameter d (average days to forget) in the differential model. The hour that 50% of the population heard the rumor is recorded for each value of d.

Looking at Figures 4.3, 4.4, and 4.5, we can see that increases in credibility decrease the amount of time until the rumor spreads to the majority of the population. The increase in average days to forget increases the time by which half of the population has been exposed to the rumor. The sensitivity analysis indicated that the time to reach the steady state depends most heavily on the α values. As α increases, the time at which 75 percent of the population hear the rumor increases. When varying parameters, the α_2 value was a constant double of the α_1 value. (See Table 4.1 for definitions.)

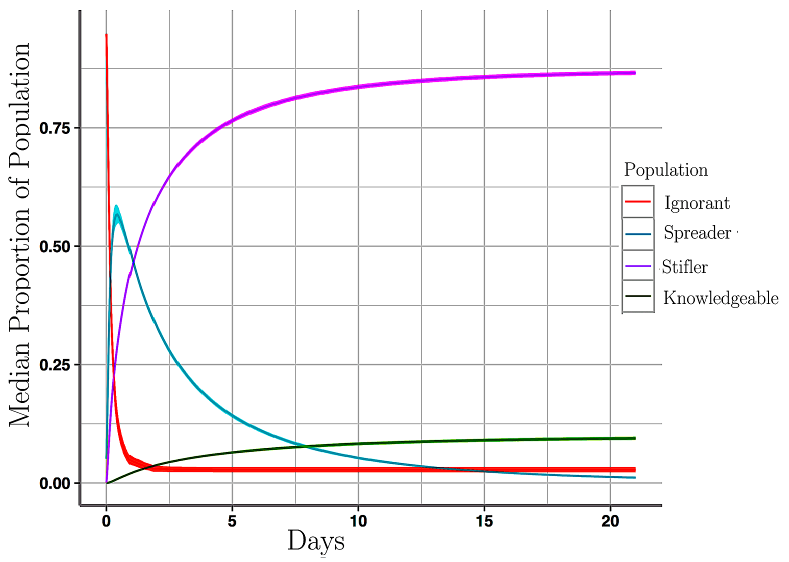

Figure 4.6: Results of the agent-based model. Solid line indicates median proportion of population across the 400 trials; shadow indicates IQR.

By comparison, in the analogous agent-based model (Figure 4.6) there are some individuals who do not hear the rumor at all by the end of the simulation.

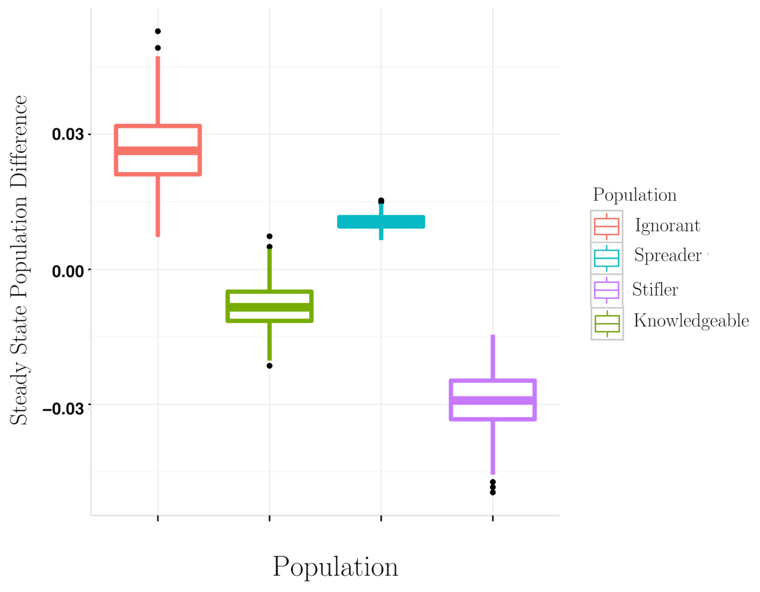

Figure 4.7: Box-and-whisker plot comparing the steady states in Figure 4.2 (differential model) and the end states in Figure 4.6 (Agent-based model).

In this agent-based model, as expected in a stochastic model, there are pockets of "ignorance" that remain. Additionally, the slopes of the population graphs are more gradual, rather than moving sharply, as in the differential model. While the dynamics of how the differential model and the simple agent-based model reached their steady-states is different, the end states are generally similar, as seen in Figure 4.7. This shows a direct comparison of the steady states from the differential model shown in Figure 4.2 and the stochastic end states from the agent-based model in Figure 4.6. Though the differential model has no network, and is deterministic, we find that the agent-based model ends with essentially the same steady states. Both models seem to confirm that with the baseline α values we chose, most people end up as rumor stiflers and, most people will be exposed to the rumor. Very few people end up forgetting the rumor entirely.

Agent-Based Model with Feature Vectors

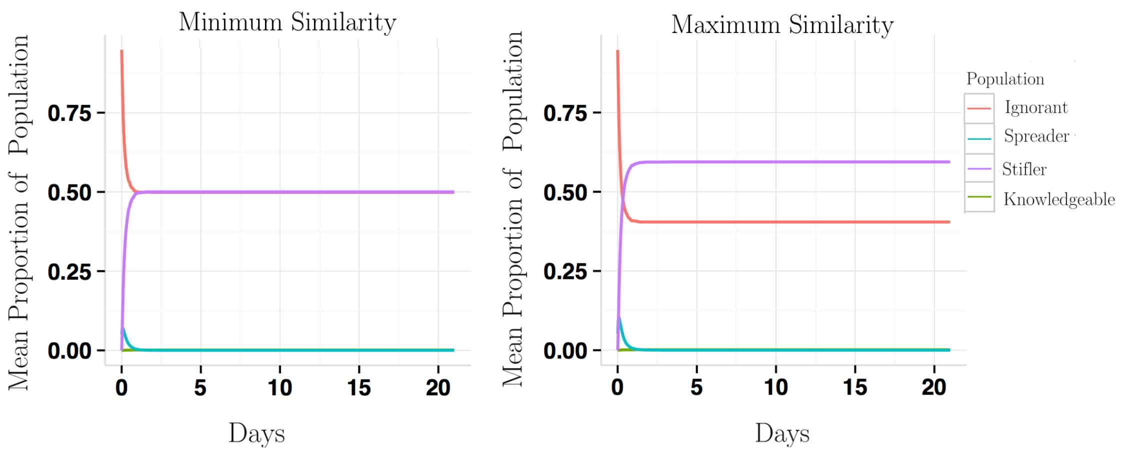

Figure 4.8: Results of the most and least similar feature vectors to the population in the agent-based model (average across 300 trials).

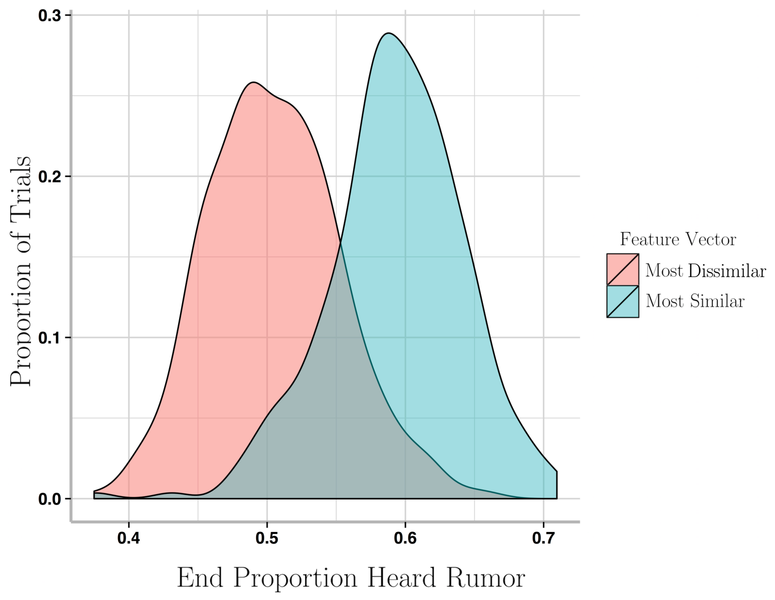

Figure 4.9: Density of the proportion of the population who heard the rumor after 22 days with the most and least similar rumors (300 trials).

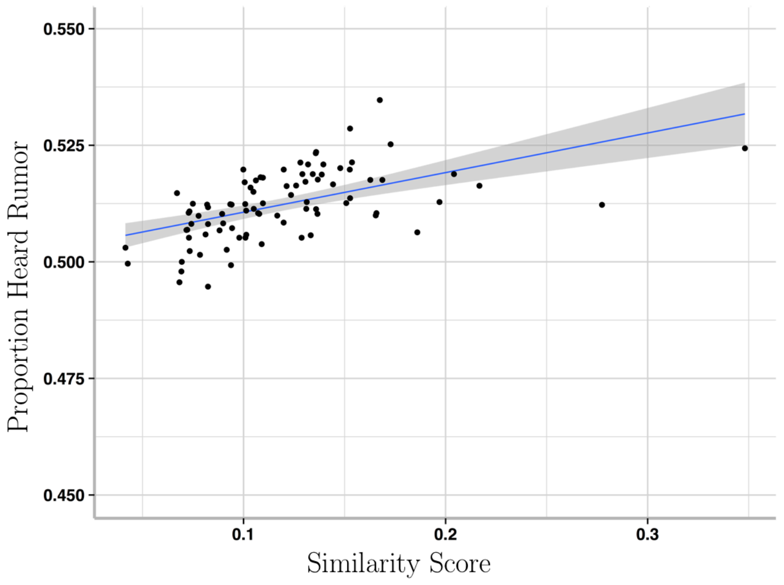

Figure 4.10: Linear model of the relationship between the final percentage of the population heard rumor and average similarity score of the feature vector ($r = 0.538$). Shading designates the 95% confidence interval.

When the similarity score is added, even the most similar rumor dies out, as seen in Figure 4.8. This plot is identical to Figure 4.2 and Figure 4.6, but is the stochastic agent-based case. However, the average similarity score of a rumor with the population does affect the spread (Figure 4.9). The most similar rumor to the population spreads to more of the population than does the least similar. The trend from the other feature vectors supports this claim, as demonstrated by Figure 4.10. The predictive power of the similarity in predicting the number of individuals who heard the rumor is decent where the bulk of the data lies. (n.b. the way that we generated most and least similar feature vectors did not guarantee that they were the absolute most or least similar to the population. To make the most similar vector, we rounded the total proportion of each feature in the entire population to either 1 or 0 to make it binary. The least similar vector is the logical complement of the most similar one. Therefore, in randomly generating feature vectors, we ended up with a few that were less similar than the "least similar" feature vector.)

4.1.5. Discussion

There was relatively little difference between the end states of the differential and simple agent-based models, despite the fact that the former aggregates the population and the latter provides more granularity. As previously noted by Chierchetti et al.[53], in a fully-connected network with push-pull interactions, a rumor will spread to the majority of the population with high probability. Though our model has a significantly different setup, we came to similar conclusions as this previous study's findings, though our model had a fully-connected network and assumed different interactions: individuals only had the opportunity to interact with the same individuals at every time step, as opposed to choosing new "partners" each time[53]. In the agent-based model, not every individual learned about the rumor, and the addition of some structured social network causes a delay in rumor spread. That is to say, the effects on one cluster are not immediately transferred to another cluster, as effects upon individuals must travel through the other individuals in a complex network in order to have large-scale effects on the population. Thus, the curves are less dramatic, change more gradually, and there is no guarantee every individual will hear the rumor: by the end of the 22 days essentially none of the population remains ignorant in the differential model, whereas in the agent-based model 2.8% of the population remains ignorant. However, the trajectories of the two models are qualitatively similar, suggesting that the agent-based model tends to a vanishing of an ignorant population, save for a small connected subnetwork. Just like the claim so well supported in push models, eventually there is a high probability all individuals will hear the rumor[230,8].

The incorporation of Feature vectors in the agent-based model changes the overall spread of the rumor. Even in the case in which most people hear the rumor (the most similar feature vector) there remains a significant population that never hears the rumor, a factor of the similarity between individuals and the rumor. Facebook friendships are a relationship that we take here to model real-world social networks. However, Facebook friends are likely to be more superficial. In fact, the average number of Facebook friends is 338[262], yet Dunbar's number suggests that humans cannot maintain more than 150 relationships due to neocortex size[78]. The social network we use shows the spread of a rumor in people who are not necessarily close, but do interact. Perhaps the feature-vector-based spread in a Facebook network is less effective in spreading the rumor due to this superficiality of relationships. Perhaps if individuals in a Facebook network cluster based on features, it would explain how a rumor could die out trying to navigate a dissimilar subnetwork. As indicated by our results, even where the rumor spreads, individuals become stiflers so quickly that the rumor dies out before reaching a large proportion of the population. This behavior is familiar to anyone who has been on social media, and had friends who relentlessly post stories that bear no significance to their personal beliefs or preferences.

Perhaps networks with clusters of similar people (by their feature vectors) would aid in the rapid transmission of a rumor across a network. In the feature vector model, spread of the rumor is a factor of both the similarity of individuals to each other, and similarity of the rumor to each individual. We speculate that in a community with many highly similar individuals one could much more easily engineer a rumor to spread through the whole network. However, an individual hearing the rumor has less to do with their individual traits, than the similarity of individuals to each other in the population. In Section 4.2, we explore this idea further by considering a homogenuous group of people and exclude any considerations of subtle individual variability; instead, we only consider one trait with a sharp difference: if someone is sick, do they have symptoms or not. In other words, we collapse the feature vector into a single dimension with a binary value instead of providing multiple orthogonal features with scalar values. This approach, called "TraSIR," is well-supported by the observations and results from the approach presented in this section.

Finally, this study also shows that there is nothing in the topology of the network that prevents rumor spread in the simple agent based model, so inoculation against hearing the rumor is a factor of the general dissimilarity of individuals to each other in the population. We suspect that the inevitable "death" of our rumors may be due to a population of individuals with a great variety of different feature vectors. Future models should investigate how the similarity of individuals' feature vectors impacts the spread of any rumor.

4.1.6. Conclusions

Since the rumor tends to spread rapidly at the start of the simulation (resulting in a corresponding boost in the stifler population), our results inspire the consideration of different network configurations. However, a rumor spreads rather more quickly in preferentially connected real-world graphs than in common theoretical graphs[74]. In our case, even the "best-performing" rumor—one that maximized spread—still died out, but it may be possible to engineer a rumor that saturates the network. In all, it would seem as though our model, in part thanks to the advent of increased computing power for simulations, can begin to unravel the nuances and intricacies of information spread through a social network. By arriving at a model that uses feature vectors and graphs, we have greater control and specificity in looking at the spread of viral information, possibly leading us to mathematically "perfect" viral information.

4.2. Estimating Asymptomatic Viral Spread

This project focuses on the spread of COVID-19 in a dynamic commute network. Based on the classical SIR model, this study incorporates a simple dynamic network to study the spread of COVID-19. The key innovation is the introduction of an asymptomatic spread parameter and fusing a commute network with several small SIR models at the county-level. The key result is the estimation of asymptomatic spread in the New York City Metropolitan Area.

4.2.1. Introduction

At the outset of the COVID-19 pandemic, the prevalence of asymptomatic cases among infections was estimated to lie anywhere between 17% and 81%[214]. Given the importance of this parameter for early health policy decisions[213], such a high level of uncertainty was a major roadblock. With testing now widely available, this issue has largely dissipated, with estimates of asymptomatic rates between ∼30% and ∼40%[181,256]. To prevent such difficulties in future epidemics, it would be highly beneficial to have computational tools for estimating the asymptomatic rate of infected individuals right at the beginning of an epidemic.

Infectious disease spread is classically modeled using compartmental models. The population is assigned to distinct compartments (for example, the susceptible, infectious, and recovered compartments in the widely studied SIR model)[152], with rates at which individuals move from one compartment to another. When applying these compartment-based epidemiological models, it is impossible to predict the true prevalence of a virus early on in a pandemic without widespread random testing: indeed, even a tiny fraction of a population showing symptoms for the disease is compatible with a widespread infection. To estimate via computational modeling the fraction of infectious individuals that are asymptomatic, known as the *asymptomatic rate* ρ, requires additional information. Here, we show that considering information about how a virus spreads in a spatial manner—not just between compartments at a single location—can be leveraged to estimate ρ. The intuition is that, while individuals travel between locations and this contributes to viral spread, individuals who feel sick (i.e., are symptomatic) tend to curb travel, which in turn yields a distinguishing observable between symptomatic and asymptomatic carriers.

Here, we introduce traSIR (pronounced "tracer"), a network traffic-based SIR model, which combines the classic SIR compartmental model with network modeling. As with the ISTK model introduced in Section 4.1, we incorporate a dynamic network to more precisely model spread dynamics[64]. In traSIR, we have a network where each node is a location (e.g., a county or ZIP Code), each location is associated with a compartmental model, and edges in the network represent frequent travel between the locations (e.g., commuting). TraSIR additionally models asymptomatic and symptomatic infections, together with a dampening effect on viral spread for symptomatic infections. Our primary contribution is to demonstrate the utility of traSIR in estimating the asymptomatic rate of an infectious disease using only knowledge about symptomatic infections across geographic locations, as well as information about typical travel between locations.

We begin with theoretical results relating the asymptomatic rate of infection to other key parameters of the model (e.g., infection and recovery rates). Since these key parameters are not known *a priori* and must be estimated from the data, we next assess how well parameters of a traSIR model can be estimated using only knowledge about symptomatic infections. In particular, we simulate disease spread using traSIR, and then perform empirical parameter estimation using the number of symptomatic infections over time across locations to estimate the asymptomatic rate ρ. Across a wide range of parameters, we find excellent agreement between the actual and estimated ρ values. Finally, we analyze the number of reported COVID-19 infections across the New York metropolitan area during the first wave.

The method behind traSIR seeks to combine topological flow information with diagnostic data and behavioral variations. It makes use of a number of observable nonlinearities: (i) in the absence of public health measures, a multiplicative decrease in the symptomatic rate causes a forward time-shift in the infection curve relative to its measurable baseline; (ii) detection of carriers grow superlinearly in the number of symptomatic cases; (iii) the number of newly symptomatic cases is largely determined by the asymptomatic neighbors in the network; (iv) asymptomatic carriers have a different transmissibility rate[169]. Our platform, traSIR, is the first of its kind to integrate heterogeneous data on a large scale to recover critical epidemiological characteristics directly from network dynamics, in particular the asymptomatic rate.

Further Background

Standard epidemiological models have previously been extended to account for disease spread across space, but the medium has typically been assumed to be homogeneous[32,85], leading to a diffusive process. Typically, the speed of a wave across the population grows in proportion to the square root of the reproduction number and the diffusion coefficient. Epidemics have also previously been studied in random graphs and scale-free networks[3,22]. Previous work has also considered the correlation of viral spread with changing commuting patterns as well as signals from social media or search engines[35,169,231,268,269,308]; other approaches have integrated network effects into compartmental models[7,20,73,76,174] and some models explicitly entertain asymptomatic spread and vaccine dynamics[164]. However, our model integrates both network dynamics and symmetry-breaking mechanisms to estimate ρ, which is what the traSIR platform offers.

4.2.2. Methods

The Model

We show how to embed the classic SIR epidemiological model[152] within a geographic network with known travel rates. The network $G = (V, E)$ is a directed graph joining N nodes (typically, counties), whose edges are annotated with the corresponding mean traffic rates of commuters. The edge set E includes all the pairs (i, j) such that residents of county i commute to work in county j. We assume the availability of an N-by-N stochastic "commute" matrix M, such that M_{ij} indicates the probability that someone commutes from county i to county j on a typical workday.

On day t, we denote the number of susceptible and recovered individuals in county i by $s_i(t)$ and $r_i(t)$, respectively. Among the $f_i(t)$ carriers of the virus in the county, we distinguish between the $c_i(t)$ of them who show symptoms and the $a_i(t) = f_i(t) - c_i(t)$ who do not. The population size in county i is denoted by $n_i = f_i(t) + s_i(t) + r_i(t)$ and is assumed fixed over the period under investigation. For convenience, we may write the right-hand side as $\sum_{x\in\{f,s,r\}} x_i(t)$.

The commute matrix M is insensitive to the health status of commuters. However, symptomatic people tend to travel less and this change has great effect on contagion. To capture this phenomenon, we introduce the *decommute rate* $\delta \in [0,1]$ as a measure of the propensity of people feeling sick to stay home:

$$M^c = (1-\delta)M + \delta\,\mathbb{I}\,. \tag{4.5}$$

Note that, if $\delta = 0$, being symptomatic has no bearing on commuting. The matrix M^c is a symmetry-breaking device which allows to distinguish between sick virus carriers and the rest. This difference creates observable nonlinearities in the viral dynamics that we can exploit to estimate the asymptomatic rate ρ.

The chronology of infection

Instead of stating the model all at once, we introduce it one piece at a time, following its natural chronology. We fix a county i and trace the changes in the main state variables s, c, a, r, f, and n. We use specific times for illustrative purposes only.

- **Step 1** At 8am on day t, all commuters are ready to go to work. We have $f_i(t) = c_i(t) + a_i(t)$ and $\sum_{x\in\{s,c,a,r\}} x_i(t) = n_i(t) = n_i$.

- **Step 2** At 9am, commuters are at work. This changes the local population into a transient one, which we denote with a "hat." By definition of the commute matrix, $\hat{x}_i(t) = \sum_j M^x_{ji} x_j(t)$ for $x = s, c, a, r$, with $\hat{f}_i(t) = \sum_{x\in\{c,a\}} \hat{x}_j(t)$, $\hat{n}_i(t) =$

$\sum_{x\in\{s,f,r\}} \hat{x}_j(t)$, and $M^x = M$ for $x \neq c$. The transient population at county i will now get to mix all day at work and spread the infection among itself.

- **Step 3** At 5pm, commuters go home. The new population at county i is denoted with a bar over the symbol. It consists of the same n_i people present at 8am, but with a different health status distribution. Take the set of infected individuals: it includes the $f_i(t)$ carriers from 8am plus the newly infected. The latter consist of the subset of the $s_i(t)$ susceptible individuals who caught the virus by commuting to county j and got exposed to a carrier in the transient population of j. Note that this includes the case $j = i$ of non-commuters who were exposed to infected visitors. The chance of anyone getting sick in this fashion is $\varphi_j(t) := \beta \hat{f}_j(t)/\hat{n}_j(t)$, where $0 < \beta < 1$ measures the transmission rate: it is the average number of contacts per person per day times the probability of transmission in a contact between an infected person and a susceptible one.

  The number of newly infected residents of county i is the sum, over all j, of the number of commuters who went to county j and got infected there: therefore, it is equal to $s_i(t)\psi_i(t)$, where $\psi_i(t) := \sum_j M_{ij}\varphi_j(t) < 1$ denotes the *worktime infectivity rate*: it is the probability that a commuter from i catches the virus at work. We have $\bar{f}_i(t) = f_i(t) + s_i(t)\psi_i(t)$. Since a fraction ρ of these new infections are asymptomatic, we have

$$\begin{cases} \bar{s}_i(t) = s_i(t)\big(1 - \psi_i(t)\big) \\ \bar{c}_i(t) = c_i(t) + (1-\rho)s_i(t)\psi_i(t) \\ \bar{a}_i(t) = a_i(t) + \rho s_i(t)\psi_i(t)\,. \end{cases} \tag{4.6}$$

- **Step 4** At 8am on day $t + 1$, further mixing will have occurred in county i since the previous evening. A fraction γ of the infected people will have recovered by then. Writing

$$u_i(t) = \beta \bar{s}_i(t) \left(\frac{\bar{c}_i(t) + \bar{a}_i(t)}{n_i} \right),$$

we have

$$\begin{cases} s_i(t+1) = \bar{s}_i(t) - u_i(t) \\ c_i(t+1) = (1-\gamma)\bar{c}_i(t) + (1-\rho)u_i(t) \\ a_i(t+1) = (1-\gamma)\bar{a}_i(t) + \rho u_i(t) \\ r_i(t+1) = r_i(t) + \gamma\bar{c}_i(t) + \gamma\bar{a}(t). \end{cases} \tag{4.7}$$

We note that traSIR involves two rounds of mixing: the first one in the daytime accounts for inter-county infection (via commuting); the second one (nighttime) models intra-county infection (within each county). For simplicity, we model recovery in the latter only. (For this reason, our value of γ might differ from the standard one by a factor of 2. In our experiments, we set δ to $8/9$.)

Parameter Estimation

Given a commute network and daily symptomatic infections across each node in the network, we develop an approach for estimating the asymptomatic rate ρ. The estimation algorithm can be viewed as a two-player game in which participants take turns updating their current estimates of (β, γ) and ρ, respectively. The updating is driven by grid search (and gradient descent) with respect to a normalized mean-square loss function, which is computed for a node k across all time points as follows:

$$\mathcal{L}(c, \hat{c}) = \sum_{t=1}^{T} \left(\frac{c(t)}{\|c\|_\infty} - \frac{\hat{c}(t)}{\|\hat{c}\|_\infty} \right)^2 .$$

We slightly abuse notation and use c here to mean a vector in $[0,1]^T$. We write $c_k(t)$ to be the recorded rate of symptomatic cases in the population in some given county k at time t. If the county is clear from context, we just write c and $c(t)$.

The normalization makes the loss invariant under scaling. This is a necessary feature given the noise in the data. Of highest concern is the corruption of the official figures caused by the inclusion of reported asymptomatic cases via testing and the exclusion of symptomatic patients who do not seek a diagnosis. We assume that the signal-to-noise ratio remains constant over time; hence that the time series c is available up to an unknown scaling factor. The normalization factors out that uncertainty.

The vector $\hat{c} = \hat{c}(\beta, \gamma, \rho)$ is the traSIR-predicted counterpart to the factual vector c; the matrix M and the decommute rate, defined in (4.5), are treated as hyperparameters. We assume that the infection is seeded at county i_0. With ρ expected to exert a relatively minor influence on the transmission/recovery parameters at the seeded node, it is natural to base the estimate of (β, γ) on the time series c_{i_0}.

Algorithm 2

```
procedure Estimate($c$)
    $\rho \leftarrow 0.5$
    for $\ell = 1, 2, \ldots, j_{\max}$ do
        $(\beta, \gamma) \leftarrow \arg\min_{(x,y)} \mathcal{L}\big(c_{i_0}, \hat{c}_{i_0}(x, y, \rho)\big)$   [via grid search]
        $\rho \leftarrow \arg\min_z \sum_{i=1}^N \mathcal{L}\big(c_i, \hat{c}_i(\beta, \gamma, z)\big)$   [via grid search]

        $g(x) := \sum_{i=1}^N \mathcal{L}\big(c_i, \hat{c}_i(\beta, \gamma, x)\big)$
        $k \leftarrow 0;\ \tau \leftarrow \infty$
        while $\tau > \tau_{\min}$ & $k < k_{\max}$ do
            $\tau \leftarrow \varepsilon(dg/dx)(\rho)$
            $\rho \leftarrow \max\{0, \rho - \tau\}$
            $k \leftarrow k + 1$

    return $(\beta, \gamma, \rho)$
```

We set $j_{\max} = 3$ (convergence is quick). The grid search is over a discrete space of size 10^3 for ρ and 10^4 for (β, γ). The number of gradient descent steps is $k_{\max} = 10^3$; the gradient descent threshold is $\tau_{\min} = 10^{-12}$; and the learning rate is $\varepsilon = 10^{-4}/NT$.

Real Data

For the commute network and population data, we rely on the most recent (pre-COVID) *American Community Survey* from the U.S. Census Bureau[283,284]. The nodes in the network represent the counties; the edges are directed and weighted in proportion to the number of residents who live in the source county and work in the destination county. We clean up the data by removing all the edges associated with fewer than 10 000 commuters. This cutoff threshold balances the need to make the graph more sparse to exclude trivial nodes and aid in computational feasibility, while also still preserving the significant graph structures. It was selected by looking at the node degrees and selecting a cutoff within a "gap" in these values. Graphs are inherently compositional, which means that the behavior of a significant subnetwork generally drives the behavior of a network as a whole; the formalities of this idea are beyond the scope of this work. We also performed some smaller-scale experiments to verify that selecting a lower threshold for filtering would not meaningfully change the results.

From the resulting graph, we extract the largest weakly connected component, which in this case corresponds to the New York Metropolitan Area. It consists of 44 counties: a visualization of which can be seen in Figure 4.11. For the infection data, we use the New York Times COVID-19 tracker and focus on the 200 days between March 1, 2020 and September 17, 2020[277,276,61,283,284].

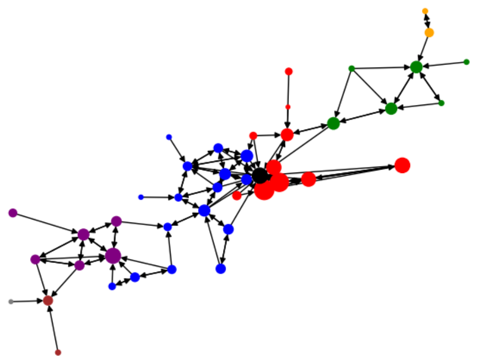

Figure 4.11: This network represents the counties of the New York City metropolitan area. Each state is colored differently with Manhattan at the center in black. The node size corresponds to the population in that county. The nodes are positioned according to the geographic center of each county.

Simulated Data

We generate 481 low-discrepancy values of β, γ and ρ, where $0.2 \le \beta, \rho \le 0.8, 0.01 \le \gamma \le 0.7$, using Sobol sequences from the SciPy package. For each of the 481 combinations of parameters, we run traSIR with the corresponding parameters for 150 timesteps on the New York Metropolitan area population and network data, assuming that there is a single infected individual in New York County (Manhattan). We further corrupt the resulting symptomatic population sizes by a fixed unknown scalar.

For validation, we run Algorithm 2 on the corrupted simulation to produce the estimated parameters $(\hat{\beta}, \hat{\gamma}, \hat{\rho})$. We evaluate the accuracy and tabulate the residuals between the estimation and actual parameters (β, γ, ρ).

4.2.3. Results

The main contribution of this work is to demonstrate empirically that a network-based epidemiological model can uncover key parameters of a contagious disease. We provided an intuitive explanation for why this might be possible as long as a symmetry-breaking mechanism is in place for distinguishing among different types of virus carriers. Before we discuss the empirical evidence and validate our approach, we provide a succinct mathematical foundation for our claim. (This next section can be skipped without affecting the readability of the rest of the article.)

Theoretical Analysis

We fix the county i and the time t and we drop all mention of t when it is understood from the context. By (4.6, 4.7),

$$\begin{aligned} f_i(t+1) &= (1-\gamma+\beta\bar{s}_i/n_i)\bar{f}_i \\ &= (1-\gamma+\beta s_i(1-\psi_i)/n_i)(f_i+s_i\psi_i) \\ &= (1-\gamma+\beta s_i/n_i)f_i \\ &\quad + (1-\gamma+\beta(s_i-f_i)/n_i)s_i\psi_i - (\beta/n_i)(s_i\psi_i)^2, \end{aligned} \tag{4.8}$$

where

$$\begin{aligned} \psi_i &= \sum_j M^s_{ij}\varphi_j(t) \\ &= \beta\sum_j M_{ij}\,\frac{\hat{f}_j}{\hat{n}_j} \\ &= \beta\sum_j M_{ij}\,\frac{\sum_k(f_k-\delta c_k)M_{kj}+\delta c_j}{\sum_k(n_k-\delta c_k)M_{kj}+\delta c_j}\,. \end{aligned}$$

Recall that $\hat{f}_i(t)$ denotes the number of infected individuals in the transient population at county i at the end of the morning commute. Let $f'_i = \sum_k f_k M_{ki}$ be the number it would have been if we had $\delta = 0$ and hence $M^c = M$; we derive $n'_i = \sum_k n_k M_{ki}$ from $\hat{n}_i(t)$ likewise. We have

$$\begin{cases} \hat{f}_j(t) = \sum_k(f_k-\delta c_k)M_{kj}+\delta c_j = f'_j - \delta(1-\rho)g_j \\ \hat{n}_j(t) = \sum_k(n_k-\delta c_k)M_{kj}+\delta c_j = n'_j - \delta(1-\rho)g_j, \end{cases} \tag{4.9}$$

where $g_j = f'_j - f_j$. This allows us to rewrite ψ_i as

$$\psi_i = \beta\sum_j M_{ij}\left(\frac{f'_j-\delta(1-\rho)g_j}{n'_j-\delta(1-\rho)g_j}\right)\,. \tag{4.10}$$

The worktime infectivity rate ψ_i plays a key role in traSIR. If $M = \mathbb{I}$, then $\psi_i = \beta f_i/n_i$ is the usual infectivity rate in the classic SIR model. Take the case of an arbitrary matrix M and set $\delta = 0$. Denote by $\mathbf{E}_{j(i)}$ the expectation operator indexed by i and defined by M_{ij}, for $j = 1, \ldots, N$. Likewise, we introduce the expectation operator $\mathbf{E}_{k(j)}$, indexed by j and defined by $n_k M_{kj} / \sum_l n_l M_{lj}$, for $k = 1, \ldots, N$. It follows that

$$\begin{aligned} \psi_i^{|\delta=0} &= \beta \sum_j M_{ij} \sum_k \left(\frac{n_k M_{kj}}{\sum_l n_l M_{lj}} \right) \frac{f_k}{n_k} \\ &= \beta \, \mathbf{E}_j \, \mathbf{E}_{k(j)} \, \frac{f_k}{n_k} \, . \end{aligned} \tag{4.11}$$

We conclude that, when decommuting is withheld ($\delta = 0$), ψ_i is an average of infection ratios f_k/n_k over counties adjacent to i or adjacent to the latter. This two-degree separation corresponds to individuals from distinct counties meeting at work in a third county. The same idea holds for $\delta > 0$, but with corrective terms that we discuss below.

At the outset of the pandemic, we can use $f_i(t+1)/f_i(t)$ as a proxy for the reproduction number associated with county i. It follows from (4.8) that

$$\begin{aligned} R_0 = 1 - \gamma + \frac{\beta s_i}{n_i} &+ \left(\frac{1 - \gamma + \beta(s_i - f_i)/n_i}{f_i} \right) s_i \psi_i \\ &- \frac{\beta}{f_i n_i} (s_i \psi_i)^2. \end{aligned} \tag{4.12}$$

Note that R_0 does not have the usual form β/γ[59]. Together, (4.10) and (4.12) form a system $\mathcal{S}(\rho) = 0$, which in theory allows us to recover the asymptomatic rate ρ from β, γ, and R_0. It is noteworthy that this requires decommuting. In fact, sensitivity analysis shows that its effect cannot be negligible. The system $\mathcal{S}$ cannot be solved for ρ in closed form. Using traSIR for estimation can thus be viewed as a numerical solver for $\mathcal{S}$.

Simulations

We demonstrate that Algorithm 2 can accurately recover the infection rate β, recovery rate γ, and asymptomatic rate ρ in simulated infections across a wide range of parameters, using just knowledge about the network and the numbers of symptomatic infected individuals. For each of simulations resulting from many combinations of parameters (see Methods), we will use the number of symptomatic individuals for each county over time. In practice, the actual

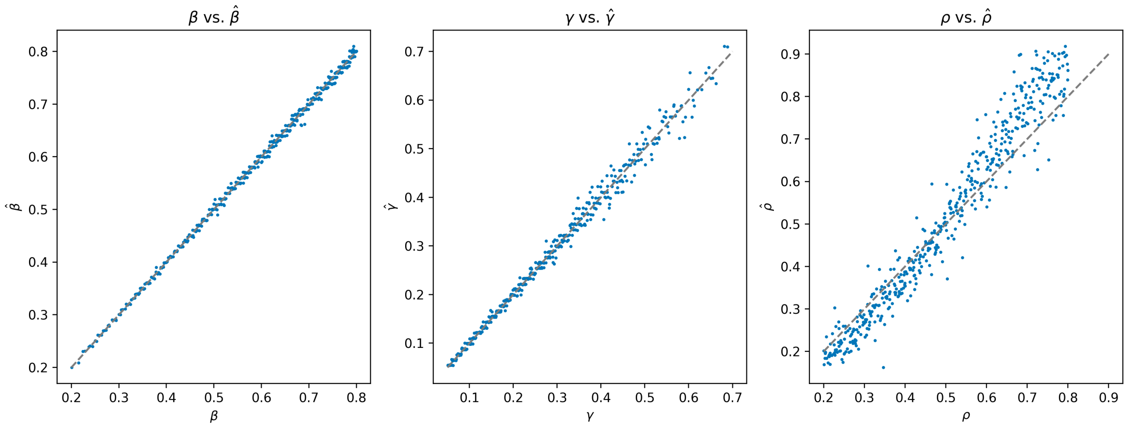

Figure 4.12: This figure demonstrates the predictive power of our estimation techniques per parameter. Each plot compares the actual and predicted value of a parameter for many different combinations of the two others. As expected, the estimation of ρ degrades as the actual value gets large. Ultimately, if no one feels sick, behavior does not change and the method cannot pick up ρ.

number of symptomatic individuals is larger than the number reported, we multiply each of the resulting symptomatic population sizes by a fixed unknown scalar, and then run Algorithm 2 to produce the estimated parameters $(\hat{\beta}, \hat{\gamma}, \hat{\rho})$.

We find excellent agreement between the actual parameters β, γ and ρ and their estimates $(\hat{\beta}, \hat{\gamma}, \hat{\rho})$ (Figure 4.12). Figure 4.12 shows a scatter plot of an estimated parameter against the corresponding synthetic parameter for the New York area. The Pearson correlation coefficient is 0.9996 between β and its predicted value. For γ and ρ, it is 0.9983 and 0.9915, respectively. The absolute residual across all starting parameters has mean 0.023 and standard deviation 0.017. The absolute residual mean and standard deviation for β is 0.0032 and 0.00246; for γ is 0.0065 and 0.0064; and for ρ is 0.0158 and 0.0163.

Applications to COVID-19 Data

Having validated our estimation technique on simulated data, we now apply Algorithm 2 to daily infection numbers from the New York Metropolitan area (see Methods), and estimate the asymptomatic rate ρ, a parameter of critical importance to health policymakers. We find:

$$\beta = 0.320 \quad ; \qquad \gamma = 0.046 \quad ; \qquad \rho = 0.345\,.$$

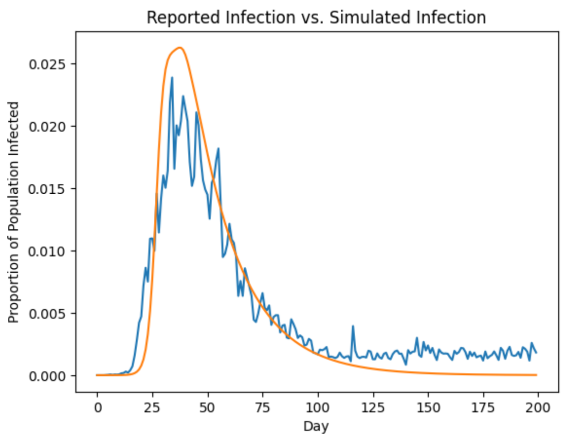

Figure 4.13: This graph displays real COVID data against TraSIR-simulated data. The blue ragged line is the reported daily case count across the New York City metro area, summed across the 44 counties considered here. The orange smooth line is the simulated daily symptomatic case count with our estimated parameters.

Using these parameters, we also compared the traSIR-simulated symptomatic infection count with the real reported infection count across the New York metropolitan area (Figure 4.13), and find good agreement.

4.2.4. Discussion

We have shown via theoretical analysis and simulation that a network-augmented compartmental model can effectively estimate the asymptomatic rate of viral infections using only data about symptomatic infections. We have applied this approach to actual COVID-19 data and derived an estimate of the asymptomatic rate that matches well with the latest estimates obtained via extensive random testing.

Our estimates for β and γ are sharper than for ρ. This is no surprise. Both the transmission rate and the recovery rate have direct influence on the local shape of the infection time series: the first one has a large effect on the ascent phase of the contagion while the other one's impact can be felt most acutely in the descent phase. The impact of the asymptomatic rate ρ is more global and subtle. It can be felt in the speed of the traveling waves and generally operates on longer time scales. TraSIR is able to leverage such information. Credit for our success must also go to sheer luck: An asymptomatic rate of $\sim 30\%$ is almost ideally sized for estimation. As we observed earlier, a rate close to 100% would make the task hopelessly difficult. This leaves open the possibility that other nonlinearities in the system can be exploited to boost accuracy when needed. While fast-changing health policy measures and medical breakthroughs (e.g., vaccination) can present traSIR with major challenges, they also create new windows of opportunity for novel estimation mechanisms. We hope that this work will plant the seeds for exciting new research on the messy, difficult, but fascinating subject of uncovering hidden epidemiological parameters.

4.3. Key Takeaways

This chapter on the spread of viruses and viral information over dynamic networks emphasizes the importance of understanding how these phenomena spread and can be controlled. One key takeaway from the chapter is the critical role of network structures in the spread of viruses and viral information. Different network structures can lead to different outcomes, such as slow or fast spread, localized outbreaks or widespread pandemics. The behavior of individuals within these networks also plays a significant role in the spread, and factors such as susceptibility, infectiousness, and adoption of protective measures can all influence it.

We have also discussed various models that have been developed to simulate the spread of viruses and viral information over dynamic networks. These models provide insights into the mechanisms behind the spread and can be used to inform public health policies and interventions. Network-based interventions, including targeted vaccination, quarantine, and information campaigns, are highlighted as effective strategies for controlling the spread of viruses and viral information. However, their implementation requires continuous monitoring and analysis of the dynamic networks to detect potential outbreaks early and take preventive measures.

Overall, these studies emphasize the complexity of the spread of viruses and viral information over dynamic networks and the need for a multifaceted approach to control their spread. By considering the various factors that influence the spread, and by using models and network-based interventions, we can work towards reducing the impact of undesiriable viral vectors: whether fake news or a deadly pathogen.

Give me a place to stand, and I will move the earth.

Archimedes of Syracuse

5

Fast-Moving Natural Networks

DYNAMISM is a term that usually captures the notion of movement over time. As a heuristic, we tend to say that something is "more" dynamic if the underlying structure changes more rapidly with respect to time. We can explore dynamical systems at the geological and cosmic scale, all of which move slowly. At the other extreme of possible scales, we have quantum systems that arise in statistical mechanics that move faster than we can make certain measurements. At the human level, these other chapters in this dissertation consider systems that operate on the scale of "days," whereas this chapter focuses on what we can study on the scale of "seconds."

We will concern ourselves with a fast, rich, and ancient system of interest: sports. In particular, we turn our attention to basketball, which is both fast-moving and has a lot of player interaction. After all, we may not find so many dynamic networks in the context of running, even if the sport moves quickly. There are a number of properties that make basketball interesting beyond its dynamism: there is rich and accurate data; we have a prior understanding of the intended dynamics (i.e. the rules of the game), and the network space is relatively small, which makes it computationally tractable. Although there is certainly a rich geometry in the dynamics of basketball, this chapter will demonstrate how the topological abstractions given by dynamic networks can sufficiently capture essential elements of the game.

We will proceed with two studies, one focusing on techniques in topological data analysis and the other leveraging statistical models and machine learning. Each study has a dynamic network at its core, but they are fraternal twins: out of the same data, we produce completely different types of dynamic networks and have two unrelated tasks that we excel in.

5.1. Basketball through Applied Topology

Here, we lay the foundation for looking at basketball data, which naturally produces trajectory data. While we could take typical geometric approaches to trajectory data, this section provides a novel pipeline using techniques in topological data analysis. In particular, we construct a dynamic network that represents how players are "connected" under some reasonable notion of a metric. Then, we consider the simplices induced by the trajectories of players as they cross these networks: in other words, we construct dynamic paths through our dynamic graphs. From here, we can use various techniques in statistical learning to understand and characterize these types of crossing sequences.

5.1.1. Introduction and Motivation

From March Madness to the NBA, basketball is an incredibly popular sport in the USA. Part of the excitement of basketball comes from how quickly the game moves: unlike American football or soccer, points are scored frequently. Players move rapidly across the court, passing the ball between each other. From a mathematical perspective, these players produce trajectories across a small bounded rectangle in $\mathbb{R}^2$. Of course, these trajectories are far from chaotic: indeed, players move in particular ways to maximize their scoring potential or to prevent the other team from scoring.

A discerning fan of the sport can also further tell which movements correspond with a specific type of "play." A play is a well-known pattern of player movement that is deployed within a larger strategy within the game. However, one of the beautiful aspects of the sport is that the boundaries between plays are loose: one play merges into the next without an obvious delineation. Moreover, plays, though well-defined, can vary tremendously because a small rotation, translation, or warping of player movements do not typically change the type of play. Finally, plays are oftentimes interrupted, curtailed, or elongated with respect to time. In other words, plays have fundamental structural properties that define them, but have a slew of invariances that make them tough to capture precisely. In this study, we leverage techniques in computational geometry to extract important features of player movements; then, in order to handle these invariances, we use constructions from topological data analysis (applied topology) to perform play classification based on topological (rather than strictly geometric) properties.

Notably, basketball plays are not strictly invariant to typical transformations in the mathematical sense. Shifting a play by one foot likely does not matter, but shifting by ten feet will. Therefore, our setting sits somewhere in between geometry and topology; we have to perform adept multi-scale analysis. In all, we construct a pipeline by defining a notion of "edge crossings" that extract topological data from trajectories via the known geometry of a basketball court. From there, we use spectral clustering to classify plays and validate our results using manual review. To this, we use a variety of tools from analysis, geometry, and topology. We being by rendering a set of useful definitions from the literature, then provide our constructions, and finally display empirical results from the data alongside some concluding notes. While not perfect, our overall aim of play segmentation and classification is somewhat successful and we can distinguish between important types of plays.

5.1.2. Background

Trajectory

Trajectory analysis is a general problem that poses interesting applications in mathematics, computer science, and electrical and computer engineering: from geometric approaches to reconstruct flight paths of lost planes[322], to tracking vehicles with heterogeneous sensor data.[280,24]. In the context of this work, we will consider the trajectories of basketball players that they make across the court. Moreover, we will attempt to uncover the underlying, salient formations that arise from a game of basketball.

In general, we consider a "trajectory" as a continuous position function, mapping from time to a plane or three-dimensional space. Moreover, "trajectory data" is collected as a discrete sample of points from an underlying trajectory.

Definition 5.1.1 (Trajectory). To formalize this concept, we can write a *trajectory* $\boldsymbol{y}$ as

$$\begin{aligned}\boldsymbol{y} &: \mathbb{R} \to \mathbb{R}^n \\ \boldsymbol{y} &: t \to \Big(y_1(t), \dots, y_n(t)\Big)\end{aligned}$$

where the y_i's are some **continuous** scalar functions and t is the *time* variable. We may define a trajectory $\boldsymbol{y}$ with an interval as its domain, rather than all of $\mathbb{R}$.

Remark (Applications). In our applications, $n = 2, 3$. When $n = 2$, we are considering a trajectory on a plane, which results in a *plane curve*. For basketball, our principal dataset provides mostly plane curves. When $n = 3$, we are considering a trajectory in space, which is also called a *space curve*. Our dataset does include some interesting z-position data; we do not explore data in this dimension in detail, but it could be salient with future study.

Typically, trajectory data is given as an ordered set of points in $\mathbb{R}^n$. In particular, we have the position data for some object represented as a time series, which we can further formalize.

Definition 5.1.2 (Trajectory Data). A *time series* is an indexed set $\mathcal{T}$ such that

$$\mathcal{T} = \{t_1, \dots, t_n\} \subset \mathbb{R} : \forall t_i, t_j,\ i < j \iff t_i < t_j$$

with the usual order on the indexing set and the real numbers.

Trajectory data, which we denote by $\mathcal{X}$, is the image of a trajectory of some time series. Or, symbolically

$$\mathcal{X} = \{\boldsymbol{x_1} \triangleq \boldsymbol{y}(t_1), \dots, \boldsymbol{x_n} \triangleq \boldsymbol{y}(t_n)\}$$

for some trajectory $\boldsymbol{y}$. Trivially, the indexing on the dataset is monotonic with respect to the underlying time series.

From characterizing an underlying trajectory for motion planning[289] to predicting the trajectory at future times[54], there are many interesting questions we can ask with trajectory data. Trajectory data also lends itself to interesting synchronization problems[274]. We focused on computing meaningful similarities between trajectories, and using this to cluster trajectory data.

Cluster

We now briefly review the definition of a cluster.

Definition 5.1.3 (Cluster)**.** A cluster is a partitioning of some data, ideally in a way that has some meaning or captures some essentially similarities within our data. Formally, given $\{\boldsymbol{x_i}\}_{i=1}^n$, a **clustering** is a *partition* of the index set $\mathbb{I} \triangleq \{1, \ldots, n\}$. In particular, we define $\mathcal{C}$ as a set of *clusters* where $\mathcal{C} = \{C_j : C_j \subseteq \mathbb{I}\}_{j=1}^k$ with three important properties:

1. Disjoint: the intersection of two clusters is empty
2. Totality: the union of the clusters is the index set
3. Non-empty: no cluster is empty (we may sometimes relax this condition, but we will be explicit when we do)

Intuitively, the definition of a cluster provides a formal framework over the idea of splitting our dataset $\{\boldsymbol{x_i}\}$ into different clusters, where each element of our dataset is in a unique cluster. Clustering data, however, is neither a general nor a precise process and requires an understanding of the underlying dataset. There exist a variety of techniques for clustering data, but there are two broad approaches: We could start with some predefined clusters and some paragon trajectory data that represents each cluster; this strategy is usually called "supervised" learning. Alternatively, we could start with some notion of "similarity" and generate clusters from this function; usually, we say this strategy is "unsupervised" learning.

Toolbox

To aid us along our trajectory through trajectory clustering, we will record some other useful concepts that will be helpful in different contexts.

Definition 5.1.4 (Metric). A *metric space* $\mathcal{M}$ is a set with a function called a *metric* d such that to each pair of elements of $\mathcal{M}$, the metric assigns a non-negative real number. More formally, we write

$$d : \mathcal{M} \times \mathcal{M} \rightarrow \mathbb{R}^+ \cup \{0\}$$

Moreover, for all elements $x, y, z \in \mathcal{M}$, the metric must also satisfy

1. Positive Definiteness: $d \geq 0$ and $d(x, y) = 0 \iff x = y$

2. Symmetry: $d(x, y) = d(y, x)$

3. Triangle Inequality: $d(x, z) \leq d(x, y) + d(y, z)$

A metric permits us to measure "distance" between elements of a set. More concretely, when considering the notion of "error" or "clustering," we need to have a method to precisely quantify a notion of closeness, which a metric provides. While the class of functions that are metrics certainly includes the familiar Euclidean distance, there are much more abstract notions of metrics that are important in all areas of the mathematical sciences (and even beyond).

Using the notion of a metric, we will mostly attempt clustering using various notions of "similarity" between trajectories. Similarity and distance are closely-related, but distinct concepts: both real-valued, positive-definite numbers, similarity takes higher numbers the "closer" two objects are. Then, we can attempt to cluster objects based on their similarity with each other. Though neither a perfect or fully formalized framework, the model of defining similarity and using it to cluster is a powerful and practical technique in trajectory clustering.

Finally, at times we will also use the notion of a "minimum-cost matching" that provides a mechanism to match elements between two sets, where matching two elements has a certain "cost." For these purposes, we usually use a metric to define the "distance" or "cost" between matching two elements, if the two sets we are matching between are both subsets of the same space.

Definition 5.1.5 (Minimum-Cost Matching). Between two (finite) sets U and V, we define a *matching* to mean a set M where $M \subseteq U \times V$ and that for each $u \in U$ there exists at most one element $m \in M$ where the first term of m is u and that for each $v \in V$ there exists at most one element $m \in M$ where the second term of m is v. A *perfect matching* is a matching where $|M| = |U| = |V|$.

We can also define a *cost function* c defined on the elements of M such that

$$c : M \to \mathbb{R}_{\geq 0}$$

We will also define the *matching cost* or *cost* of the matching (and abuse notation) by writing

$$c(M) \triangleq \sum_{m \in M} c(m)$$

We let $\mathfrak{M}(U, V)$ be the set of all matchings for U and V and let $\mathfrak{M}^P(U, V) \subseteq \mathfrak{M}(U, V)$ be the set of *perfect matchings* (which could be empty). If $\mathfrak{M}^P(U, V)$ is non-empty, then for cost function c, the *minimum cost matching*[*] or $\mathrm{MCM}_c(U, V)$ is defined to be

$$\mathrm{MCM}_c(U, V) \triangleq \operatorname*{arg\,min}_{M \in \mathfrak{M}^P(U,V)} c(M)$$

5.1.3. Background

We will explore an existing technique for comparing trajectories, the Fréchet distance.

[*]There are diverse ways to define a matching, a cost function, and a minimum cost matching, so we pick a definition that is convenient to our needs

Continuous Fréchet Distance

One intuitive notion of distance between trajectories is the Fréchet Distance.[123] The Fréchet Distance, named after Maurice Fréchet, measures how close two trajectories are with two important properties:

1. Curves that are "spatially close" are considered close, i.e. a curve that is the result of an affine shift of another curve will not be considered "close"
2. The "speed" at which we move along a trajectory is unimportant, i.e. the distance between times in the underlying time series is ignored

More formally, the Fréchet Distance is for curves defined over a metric space, of which trajectories are a special case.

To formally define the Fréchet Distance, we will note that closed intervals of the real numbers are homeomorphic to each other. Therefore, the Fréchet Distance will consider all time series data, as if the time series occurred over the standard unit interval $[0, 1]$.

Then, we inspect, over all reparametrizations of the trajectories, the one that provides the minimal distance. In that way, we try to consider the "best way" to write the trajectories that is invariant to how "fast" a particle moves over a trajectory.

Definition 5.1.6 (Curve)**.** For some metric space $\mathcal{M}$, we define a curve C such that

$$C : [0, 1] \to \mathcal{M}$$

where C is continuous.

Moreover, we define a reparametrization ϕ such that

$$\phi : [0, 1] \to [0, 1]$$

such that ϕ is continuous, non-decreasing, and surjective.

While a curve and a trajectory are related, we wish to formally distinguish the two concepts by noting that the definition of a curve is in some sense more general, as it admits a metric, rather than a trajectory, which relies on the geometry of the real numbers.

Definition 5.1.7 (Fréchet Distance)**.** Let A, B be two curves in some metric space M with distance d. Let Φ be the set of all reparametrizations of the unit interval $[0, 1]$ and $\phi, \rho \in \Phi$. Let $t \in [0, 1]$. Then, the Fréchet Distance F between A and B is

$$F(A, B) = \inf_{\phi,\rho} \max_{t} \left\{ d\left[A\left(\phi(t)\right), B\left(\rho(t)\right) \right] \right\}$$

Given some set of curves, we can compute the Fréchet Distance between them in polynomial time.[6] Then, this distance can be converted to a measure of similarity or used directly in the computation of clusters that are based on distance matrices, something we cover later.

Discrete Fréchet Distance

The Discrete Fréchet Distance is based on so-called polygonal curves. Given some trajectory data, we connect each point with the one immediately following and preceding it with a straight line. We call such curves "piecewise linear." Then, we modify the Fréchet Distance to consider matchings only at vertices[84].

Clustering

Though we have covered only metric-based notions of similarity, there are other forms of similarity that we can use to perform various clustering techniques. Examples include Spectral Clustering, Locally Linear Embedding (LLE), and through the use of neural networks. Clustering is a deep and rich topic in mathematics and computer science[2]. No doubt, any citation or description of data clustering techniques would be outdated by the time of publication.

Computational Geometry

We will use convex hulls (the unique minimal convex set that contains a set of points), a Voronoi diagram (a partition of space into cells given points), and a Delaunay triangulation (a triangulation that is the dual of a Voronoi diagram.) See Definitions 2.2.1, 2.2.6, 2.2.7 for formal definitions of these concepts. Figure 5.5 has an example of a Delaunay Triangulation.

5.1.4. Basketball Trajectories

One interesting application of trajectory clustering comes to us from basketball.

Remark (Data Set). We are currently working with raw position data of basketball players, sampled around every 5 ms. We are focused on the Duke '16-'17 basketball season. We also have events data, which record fouls, scoring attempts, rebounds, etc. We use this data to delineate possessions. We thank STATS and SportVU for providing this dataset under an academic license.

Each game is split into a series of possessions, which contains ten sets of trajectory data.

Definition 5.1.8 (Basketball Possession, Game). We define a *team trajectory* to be an indexed set of five sets of trajectory data. In particular, we define one as the "defense" $\mathcal{D}$ and one the "offense" $\mathcal{O}$ and write

$$\mathcal{D} = \{D_1, \ldots, D_5\}$$
$$\mathcal{O} = \{O_1, \ldots, O_5\}$$

We define a *basketball possession* $\mathcal{P}$ to be pair of one defense team trajectory and one offense team trajectory.

$$\mathcal{P} = (\mathcal{D}, \mathcal{O})$$

We define a *basketball game* to be an ordered set $\mathcal{B}$ of basketball possessions

$$\mathcal{B} = \{\mathcal{P}_1, \dots, \mathcal{P}_k\}$$

for some $k \in \mathbb{N}$.

However, comparing sets of trajectories is difficult, i.e. it is computationally intensive and can be too sensitive to noise. Therefore, the first step is to somehow reduce the density of our data. Thus, for each possession, we compute some form of a gridding scheme (discussed later) and fix the positions of the defense halfway through the possession; we take the position halfway through because the defense has "settled" by this point. Additionally, we add the four corners of the half-court and the basket itself. Therefore, we can write a function G that takes in a defense team trajectory and returns some gridding.

Next, given a particular gridding for a possession, we can then compute the edge crossing that each offense trajectory makes. Using this pipeline, we can write down the "crossing map" $\chi_{G(\mathcal{D})}$ which takes in trajectory data and returns us a sequence of edge crossings.

Definition 5.1.9 (Crossing)**.** Given a set of points X and a set of edges $E \subseteq X \times X$, we say that a trajectory *crosses* an edge (x_1, x_2) if the trajectory intersects with the image of the edge $\{\lambda x_1 + (1 - \lambda)x_2 : \lambda \in [0, 1]\}$. We further say that a *crossing sequence* is the sequence (ordered by time or, in the case that a trajectory crosses two or more edges at the same time, we break ties arbitrarily but stably) of edges that a trajectory crosses. With trajectory data, the crossing sequence is that of the trajectory created by linearly connecting the data.

We write

$$\chi_{G(\mathcal{D})} : \mathcal{T} \to \mathfrak{C}$$

where $\mathcal{T}$ is the space of trajectory data and $\mathfrak{C}$ is the space of crossing sequences. We can write $\mathcal{X}_{G(\mathcal{D})} = \chi_{G(\mathcal{D})}(\mathcal{T})$, i.e. the image under $\chi_{G(\mathcal{D})}$.

In sum, we first begin with some continuous trajectory created by ten players on a basketball court. Using a system of cameras, we sample each trajectory and produce one trajectory data for each trajectory. These trajectory data can be grouped into an offense and defense, which we denote as $\mathcal{D} = \{D_1, \ldots, D_5\}$ and $\mathcal{O} = \{O_1, \ldots, O_5\}$ respectively.

From some defense data, we define a gridding, namely through some gridding function G and we compute $G(\mathcal{D})$. From this particular gridding, we generate a map $\chi_{G(\mathcal{D})}$ that can take in trajectory data and transform it into a sequence of *crossings*: a map that we will use on the trajectory data of the offense. For example, we can compute $\chi_{G(\mathcal{D})}(O_1)$, which provides the crossing sequence of the offense. In all, we can create

$$\chi_{G(\mathcal{D})}(\mathcal{O}) \triangleq \{\chi_{G(\mathcal{D})}(O_1), \ldots, \chi_{G(\mathcal{D})}(O_5)\}$$

whereby, we are taking the defense trajectory data for each possession to generate some sort of gridding scheme, then using this gridding scheme to create a sparse representation of the offense by creating a crossing sequence.

Since a possession is defined as a pair of offense and defense team trajectory data, given some G, we can compute the crossing sequence C of a possession $\mathcal{P} = (\mathcal{D}, \mathcal{O})$ by defining it to be $\mathrm{C} \triangleq \chi_{G(\mathcal{D})}(\mathcal{O})$. We can also define a crossing sequence $\mathcal{C}$ to simply be some crossing sequence for a given G and $\mathcal{D}$ of some trajectory data.

Definition 5.1.10 (Crossing Similarity). Given two crossing sequences $\mathcal{C}_1, \mathcal{C}_2 \in \mathfrak{C}$, we can define various notions of similarity σ. We can use the Longest Common Subsequence (LCS) for the foundation of a definition of similarity, such as a Smith-Waterman definition, or some other sequence matching notion of similarity.[17] We define σ such that

$$\sigma : \mathfrak{C} \times \mathfrak{C} \to \mathbb{R}^+ \cup \{0\}$$

Finally, given this measure of similarity, we can then further define a similarity between possessions. We need to understand one more concept to do so.

Definition 5.1.11 (Frobenius Norm). Defining a norm over matrices is not as straightforward as over vectors. In some sense, we want to capture the magnitude of the matrix based on its entries. Therefore, we define the *Frobenius Norm* of a square matrix A as follows:

$$\|A\|_F = \sqrt{\mathrm{Tr}\,(A^\intercal A)}$$

where Tr is the trace and $\intercal$ is the matrix transpose.

Definition 5.1.12 (Possession Similarity). Given two possessions $\mathcal{P}_1 = (\mathcal{D}_1, \mathcal{O}_1), \mathcal{P}_2 = (\mathcal{D}_2, \mathcal{O}_2)$ and some G, we define the *similarity matrix* $\Sigma \in \mathbb{R}^{5\times 5}$ as follows

$$(\Sigma_{ij}) = \sigma(\chi_{G(\mathcal{D}_1)}(U_i), \chi_{G(\mathcal{D}_2)}(V_j))$$

for all $U_i \in \mathcal{O}_1$ and $V_j \in \mathcal{O}_2$.

Then, we define the *similarity* of two possessions as

$$S(\mathcal{P}_1, \mathcal{P}_2) = \|\Sigma\|_F$$

Given this definition of possession similarity, experimental results have so far validated the use of spectral clustering. In particular, taking the pairwise similarity between all possessions in a game has been successful and confirmed with video data of basketball games, as seen in Figure 5.10 below.

Summary of Pipeline

We summarize our resulting pipeline for clustering basketball possessions:

1. Input: basketball trajectory data split by possessions
2. For each possession:
   (a) grid the space by using some gridding technique
   (b) compute sequence of edge crossings for each offensive player
3. For every pair of possessions

    (a) compute the similarity score between each pair of offensive players, this becomes a similarity matrix
    (b) compute the Frobenius norm of this matrix, this is the similarity score between the pair of possessions
4. Cluster on this similarity matrix between possessions
5. Result: Clustered possessions

Gridding Techniques

Overview

One blackbox in the aforementioned algorithm is the use of a "gridding" method. The motivation of a gridding method is to reduce the density of the dataset; in other words, we can take some trajectory data that may have hundreds of points and reduce it to some smaller sequence.

In other words, by fixing a new coordinate system on the basketball court, it is possible to transform the trajectory data into a new trajectory data. Instead of tracking the position of each basketball player at each timestep, though, we instead keep a list of "cells" each player is in. A grid is simply a collection of these cells that partition the court, i.e. the cells have two important properties:

1. The cells cover the entire basketball court
2. The cells are disjoint or, in other words, they do not intersect

Therefore, we can keep a track of the sequences of cells that each basketball player enters and exits. Then, we can compare this sequence of cells for each basketball player across all of the plays.

Importantly, the gridding for a particular play can be dependent on the position of the basketball players in the play itself. For the purpose of the gridding, the position of the basketball players in the play is frozen in the middle of the play. Naturally, we must then also consider how we can compare sequences generated by these gridding techniques, especially when the gridding may change per play. Thus, several gridding techniques that were considered are presented in the following section, each with an explanation of how the gridding in one play is identified with the gridding in another play.

As a note, for the purposes of comparing a grid, we simply need to create a notion of an indicator, namely we only track if two cells are the same across different griddings, rather than trying to measure the "distance" between two cells. Therefore, we only define a way to identify if two cells are the "same" or "not the same" across two different griddings.

Several Gridding Techniques

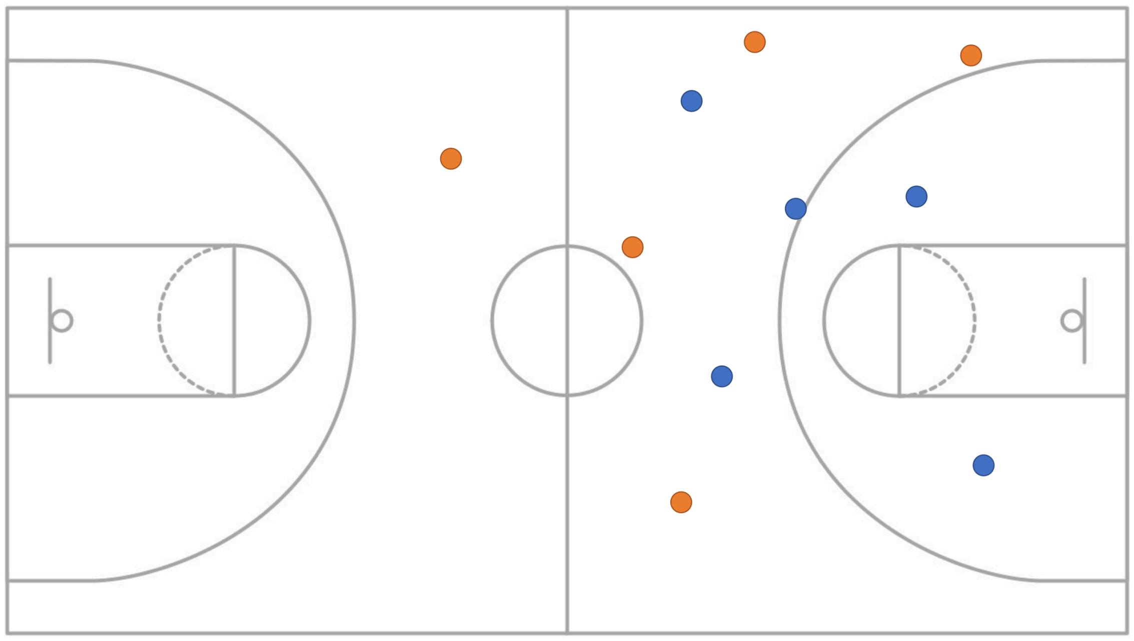

Figure 5.1: A basketball court with with defense in blue and offense in orange. The court picture is reproduced from `http://printablediagram.co`.

In the example presented in Figure 5.1, we see a basketball court with some sample player data: the offensive players marked in orange with the defensive players marked in blue. Given this data, we could consider several schemes for dividing the court into a grid: some that use the data or some that do not.

The first, and perhaps most simple gridding method, would be a *regular rectilinear* grid.

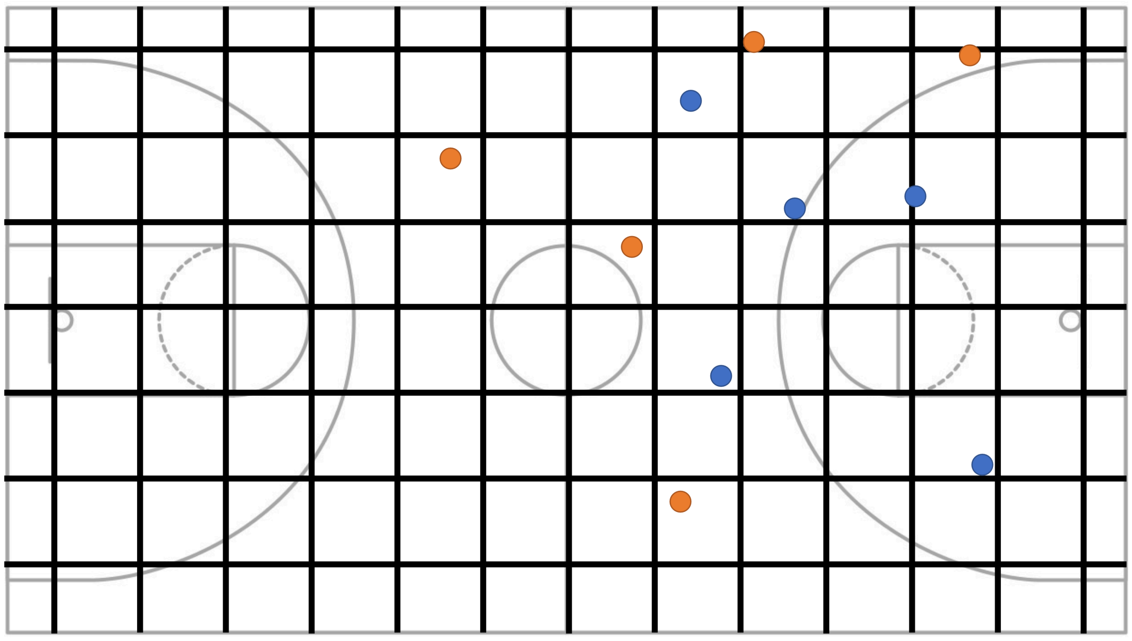

Figure 5.2: A basketball court with a regular rectilinear gridding. This figure is the same as Figure 5.1 but has a regular grid on top of it. The defense is in blue and the offense is in orange.

As evinced in Figure 5.2, we see a regular rectilinear gridding where each cell has equal area is spaced at regular intervals, thus the gridding does not change with each play. Moreover, each cell is easily identifiable by the row and column that it sits in and can be compared quickly across the plays.

The next type of gridding is the *position-based rectilinear* gridding.

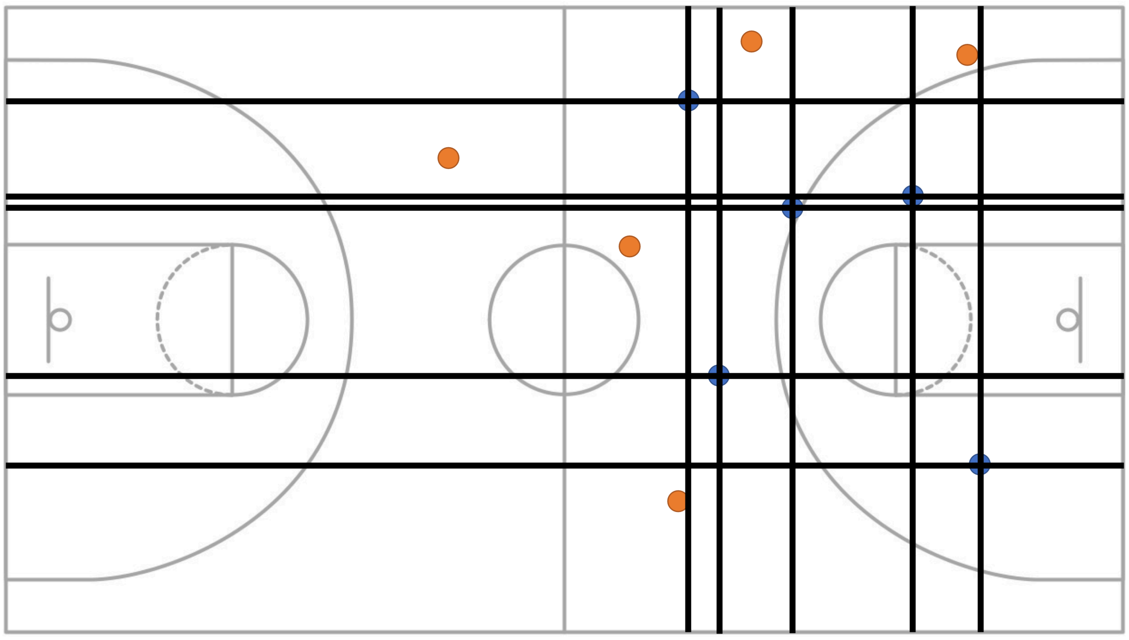

Figure 5.3: A basketball court with a position-based rectilinear gridding. This figure is the same as Figure 5.1 but has a position-based rectilinear grid on top of it. In particular, for each defense player (in blue), we place a vertical and horizontal line at their position, e.g. for a player at (x_0, y_0), we place the lines $x = x_0, y = y_0$ on top of the basketball court. We do not do anything for the offense (in orange).

As picture in Figure 5.3, the position-based rectilinear gridding places a horizontal and vertical line at each defender position. In this way, each (distinct) defender position creates four quadrants of the basketball court.

This gridding changes with each play depending on the position of the defense, so we have to somehow draw a correspondence between griddings for each play. Since the "area" of a line is 0 using the standard Lebesgue Measure, we note that there is a 0% chance of five points chosen uniformly at random from a rectangle in $\mathbb{R}^2$ (i.e. a basketball court) will lie on the same vertical or horizontal line (i.e. either their x- or y-coordinate will coincide).

Next, we note that, by this result, we expect to have five distinct vertical and horizontal lines. This selection of vertical and horizontal lines will create six columns and six rows, which implies a total of thirty-six cells that will be created.

Finally, by taking the (somewhat imprecise) assumption that no two defenders will lie on the same horizontal or vertical line (an assertion that is borne out experimentally, so we take for granted theoretically), we know that this gridding technique will always produce thirty-six cells. Thus, we can identify the cells in the same way as a regular rectilinear grid by fixing an indexing of the cells from left-to-right and then from top-to-bottom (sometimes called "raster order").

The next gridding method is a *regular polar* grid.

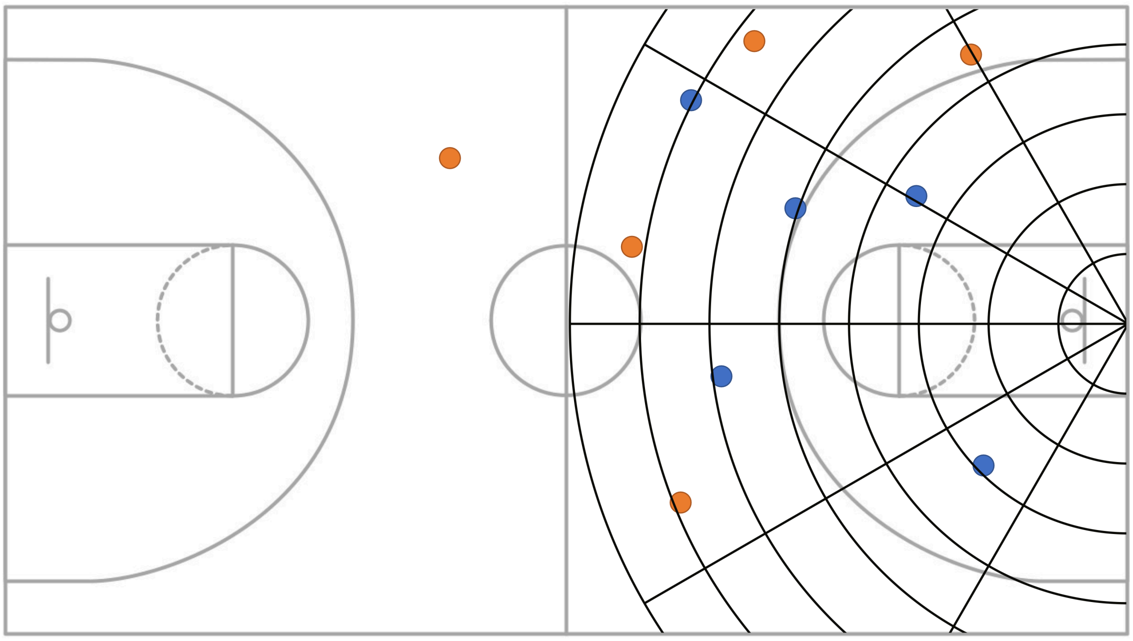

Figure 5.4: A basketball court with a regular polar gridding, emanating from the basket that the offense (orange) is pursuing and the defense (blue) is defending. The polar grid image is from `http://etc.usf.edu`.

In Figure 5.4, we have an example of a regular polar grid. In this case, a series of concentric circles are drawn centered around the origin, with each circle increasing in radius by unit length. The circle is partitioned into arcs of equal length by selecting some roots of unity for a particular number of roots. In this way, a regular polar grid is created, with each cell representing a section of an annulus.

Again, since this is a regular gridding, in that it does not rely upon the position of the defenders, it is trivial to identify each cell in the gridding with its corresponding cell in another gridding.

Finally, we consider the *Delaunay Triangulation* gridding.

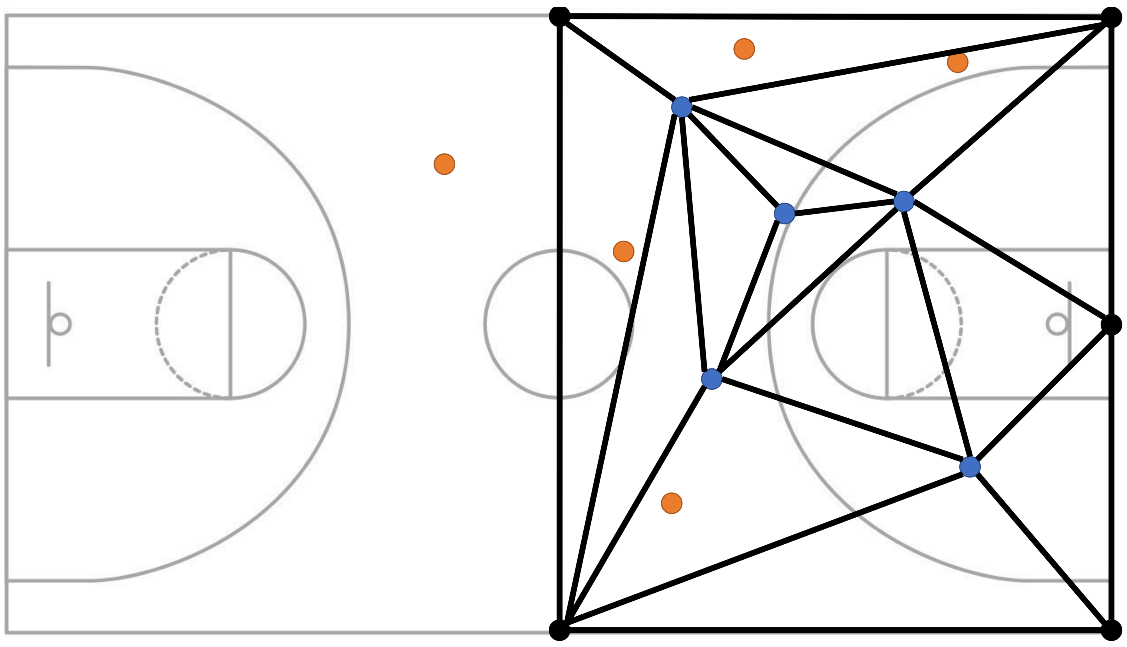

Figure 5.5: A basketball court with a Delaunay triangulation gridding. We take the Delaynay triangulation of the players on the defense (in blue), along with points at each corner of the half-court and the basket. We do not compute a gridding from the offense (in orange).

The Delaunay Triangulation is a standard method of creating a gridding from a set of points in the plane. In particular, to create this gridding, we first have to add five points that represent the outer part of the half-court. Then, we compute the Delaunay Triangulation and let this define a gridding.

To identify cells within each gridding with each other, we can use the five "fixed" points as reference. In particular, we label each reference point as $\{A, B, C, D, E\}$. Then, using a minimum-cost matching (with the cost function between points as the standard Euclidean distance), we identify each defender with a fixed point and give the label $\{A', B', C', D', E'\}$. In other words, we create a matching from the set of fixed points and the set of defenders (both are just points in a plane) with the cost function as the standard distance function. We can thus label each cell by its corresponding three points (a Delaunay Triangulation will always produce cells that are triangles) and equate cells from one gridding to the next by these labels.

Importantly, we note that in this gridding scheme, the fixed points provide a mechanism for comparing cells across griddings for each play. Namely, the fixed points provide a frame of reference and allow for a standardized way to label the defenders across plays. This technique provides a principled way of determining which cells are the "same" across plays (since cells that are the same are close to the same fixed points), while also providing the flexibility for the cells to warp with the configuration of each defense.

Convex Hulls

Some prior art by Stephen Shea and Chris Baker suggests the use of convex hulls.[257] In particular, they define the notion of the CHAD and the CHAO, or the *Convex Hull Area of the Defense* and the *Convex Hull Area of the Offense* respectively.

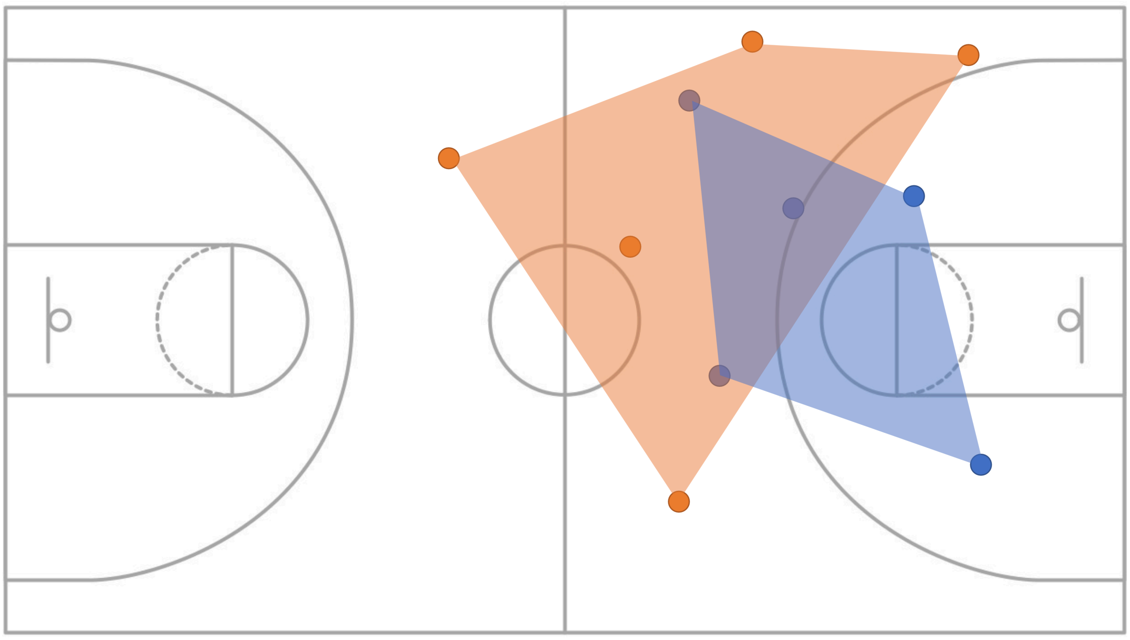

Figure 5.6: A basketball court with the convex hulls of the defense (blue) and offense (orange).

These convex hulls as seen in Figure 5.6 are fairly easy to compute, along with their respective areas. The CHAD is the area of the convex hull of the defense (as seen in blue) and the CHAO is the are of the convex hull of the offense (as seen in orange).

In particular, given that each possession is made of trajectory data, Shea et al. pick a single moment in a possession defined by the first time that the ball was on the three-point line above the break. For our purposes, we shift that moment in time to be consistent with the computation of the triangulation, namely halfway through the possession.

Shea et al. suggest that the ratio of the CHAD and the CHAO can be used to predict the scoring potential for a particular possession. However, these results are preliminary and do not take into account the dynamics of the game (e.g. they are based off of a single moment in a possession). Moreover, even if their correlation is statistically significant, the correlation itself is weak and does not provide strong quantitative predictive power. Finally, the use of the CHAD and the CHAO may be misleading; the intuition is that a spread out defense is a weak one, but the extreme example of all defenders standing in one corner illustrates a low CHAD with a terrible defensive strategy. Therefore, we attempt to reproduce the CHAD/CHAO method with an additional accounting for the dynamics of the game itself. However, we will still mainly focus on classification.

Experimental Results

We will describe some of clusters and the similarities observed while watching videos of possessions clustered together.

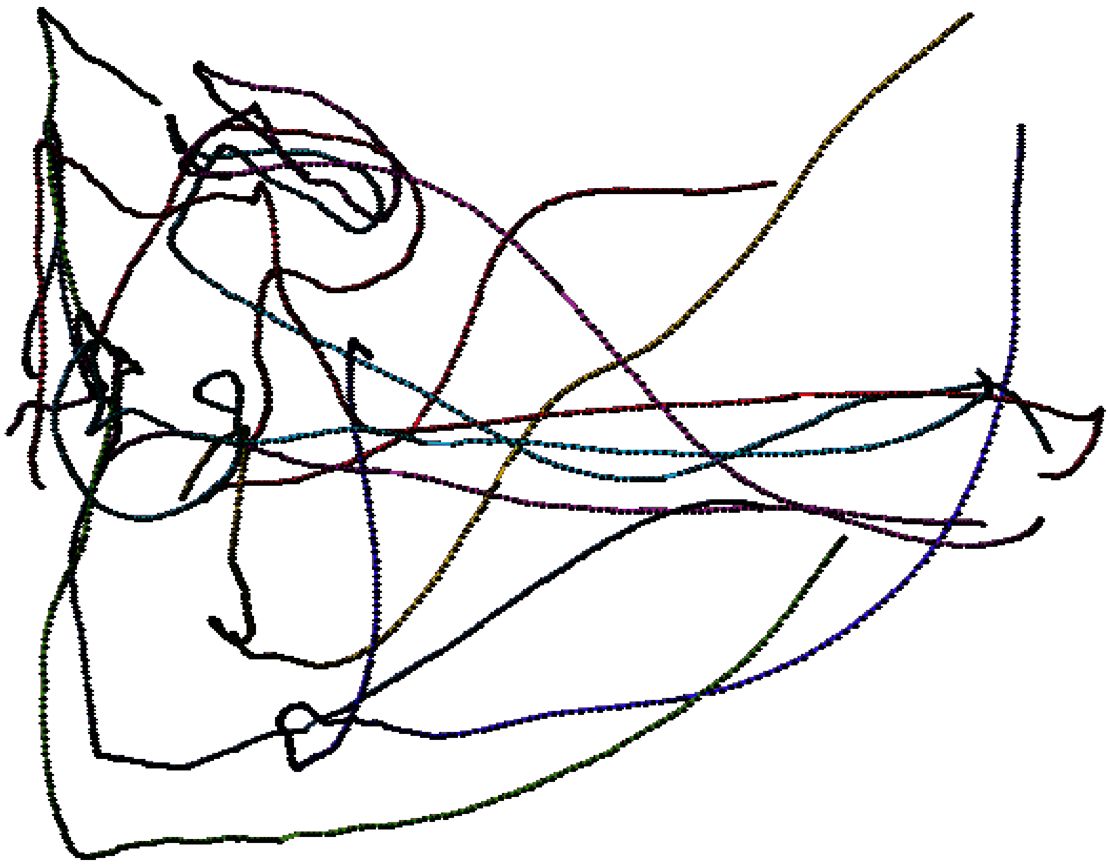

Figure 5.7: The raw trajectory data of the ten basketball players in a particular play within a particular Duke University basketball game.

Raw trajectory data, as seen in Figure 5.7, over a possession tend to look fairly chaotic and incomprehensible. Even with some way to indicate the differences between players, visual inspection of the data itself proved to be unfruitful.

However, watching video clips of the respective games has been the most salient way for identifying a type of play or what makes a cluster similar. Subjective determinations of the type of quality of the play are easy to make and fairly standardized in the basketball community, e.g. there is a refereeing and commentating system.

Observations of Clusters

These clusters are created using spectral clustering and the Delaunay Triangle gridding method.

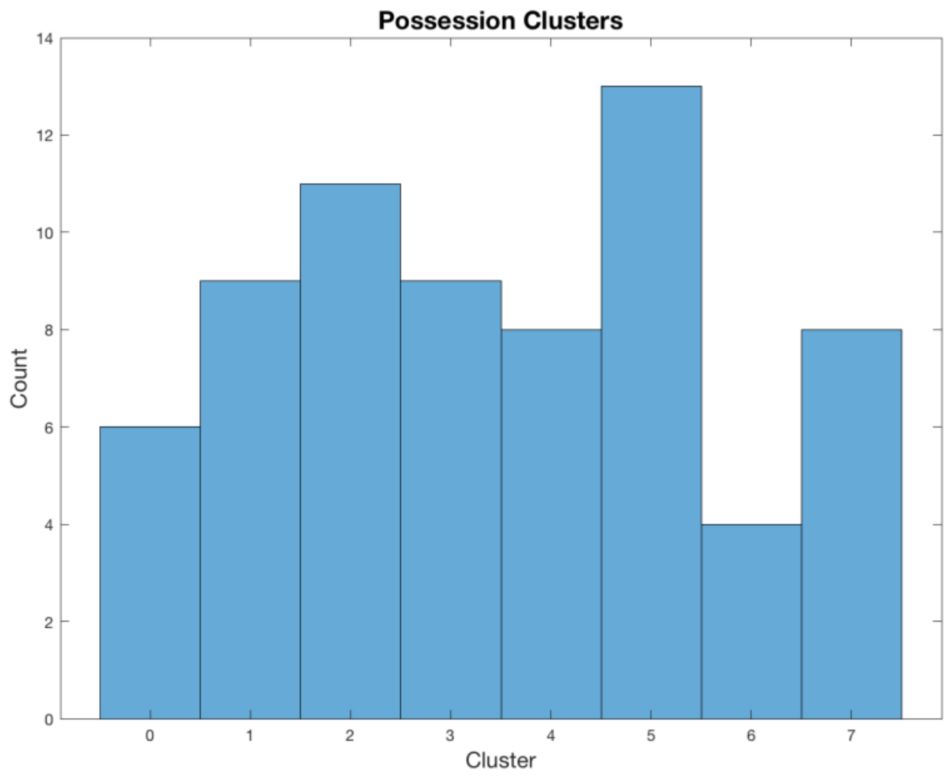

Figure 5.8: A bar graph that displays the size of each of the clusters detected by the spectral clustering method.

Looking at the clusters we found in Figure 5.8, our pipeline works well for short plays where there is a clear defining event. For example, we consistently found a cluster of five plays in which every play was set up as a quick pull up and 3-pointer, these were all executed very quickly. We were also successful in picking up a cluster of mostly drives—in which the defense was set up and one offensive player drove into the paint to score.

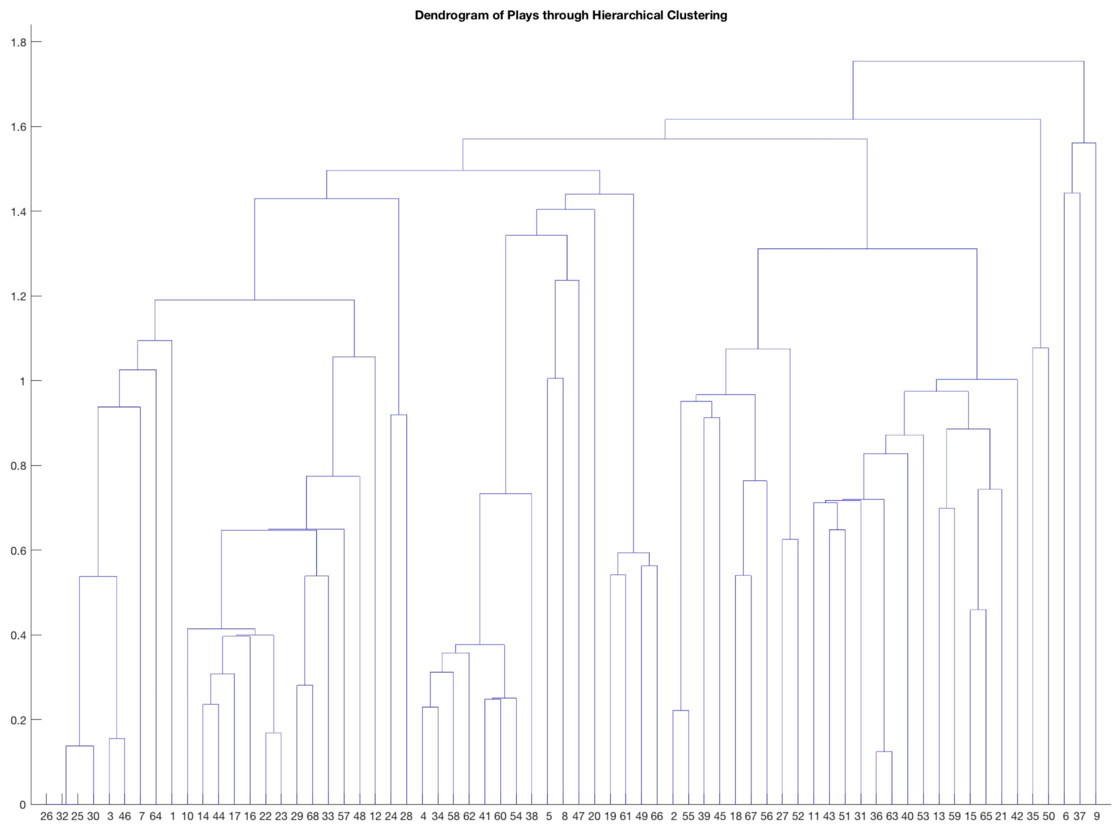

Figure 5.9: Dendrogram of the relationship between plays. Play indices are indicated along the x-axis, while relative similarity is indicated across the y-axis. Clustering is based on Delaunay Triangulation for gridding.

We were less successful differentiating long plays, which all tended to be placed in one miscellaneous cluster. Manipulating the sequence similarity measure could change the overall results. With our setup, any sequence alignment technique could work, which has a rich literature[244]. For example, switching from the LCS-based distance[17] to Smith-Waterman may improve cluster quality.

As seen in Figure 5.9, we were able to capture some interesting clusters. Most of the clusters are around the same size, which we do not necessarily expect to be true all of the time. For example, there could be one successful play (such as a player being "hot" from the 3-point line) that the team runs more frequently, and others which do not get repeated. We suspect that this uniformity in cluster size might be an artifact of our use of Spectral Clustering, but given the size of the dataset, it is difficult to distinguish between certain biases by our techniques and the underlying data itself.

In all, this pipeline provided some interesting ways to validate the intuition on the classification of plays via their inherent geometry. By making dataset more sparse and extractign salient information, the pipeline gave rise to some useful notions of comparing basketball possessions across four different games. The clusters pretty quickly break apart, though, which we thought was expected. Overall our experiments have made us hopeful for the success of this general pipeline, and they have left us with many questions which we hope to answer in the future by running even more experiments and changing pieces of the pipeline.

Cluster Validation

Next, we are interested in how much clusters change across different experiments, as one form of validation. We perform a cross-similarity analysis to show how often plays are clustered together.

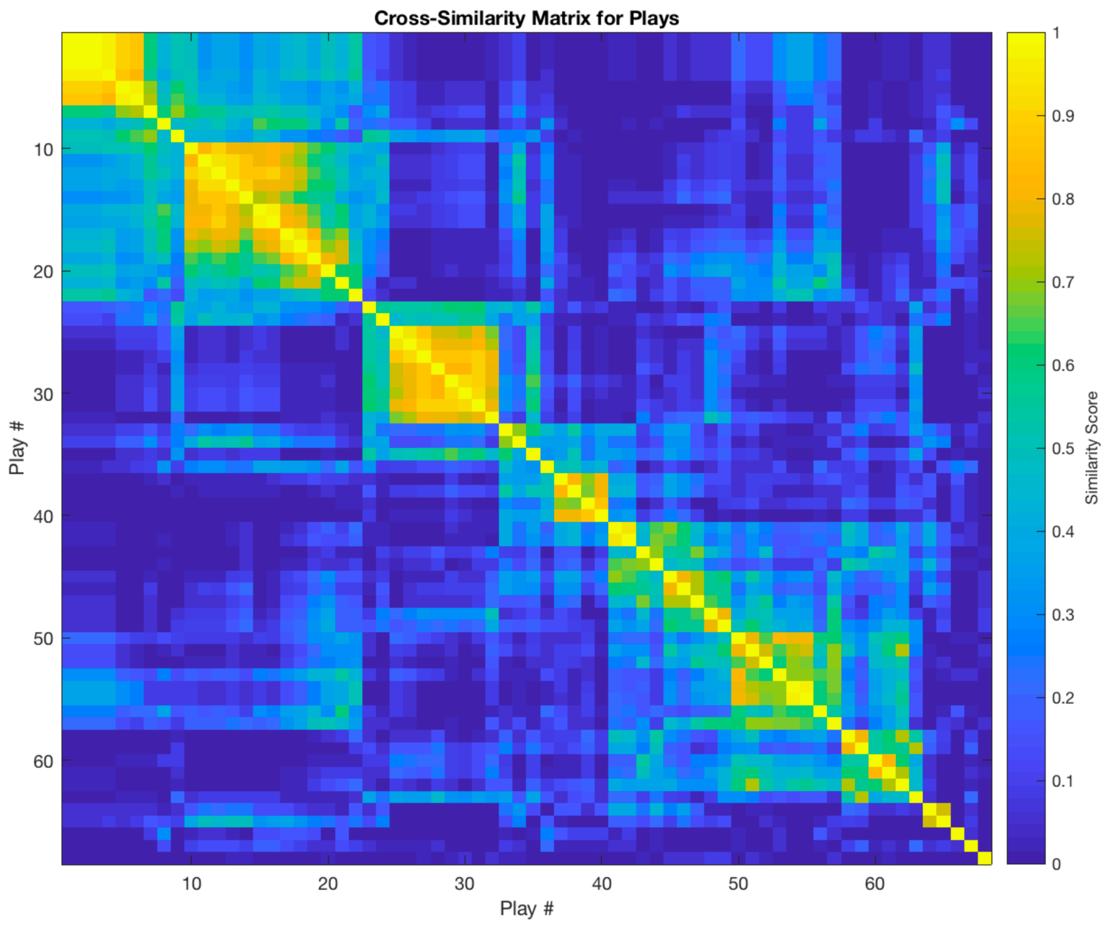

Figure 5.10: Cross-similarity matrix showing aggregate similarity across experiments. Play numbers are reordered to correspond to the order in the dendrogram.

As shown in Figure 5.10, we have some plays which are almost always clustered together, which we take to be a good sign. We originally thought that we couldn't use more typical trajectory similarity tools, such as finding the Fréchet distance, because the trajectories tend to be chaotic, so we hoped converting to sequences would capture some of the relevant position and movement data, but also remove some of the inherent noisiness of the trajectories. A question we have continued to ask ourselves is whether we are looking at this position data at the right resolution.

Finally, our most intriguing validation comes from subjective and qualitative analysis of the plays.

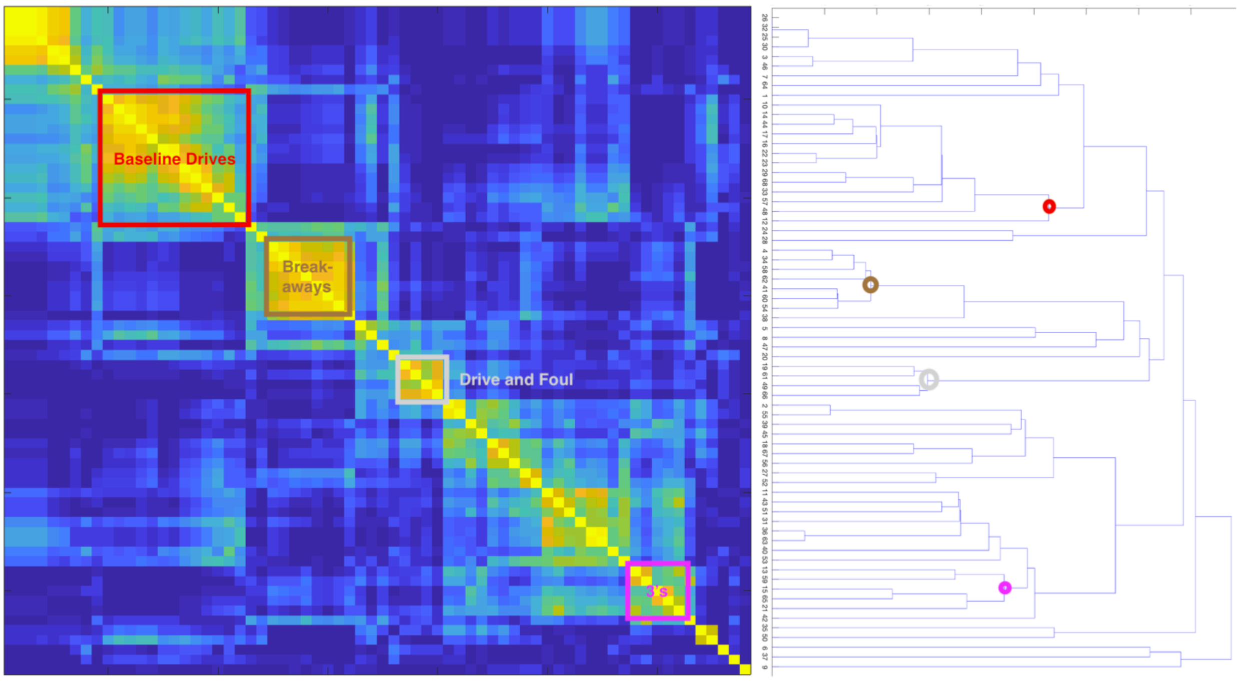

Figure 5.11: Cross-similarity matrix with corresponding dendrogram along with a manual labeling of possession clusters by well-known play types.

Figure 5.11 provides insight into what these clusters represent. In particular, we can see a correspondence between the cross-similarity matrix and the dendrogram generated from this similarity matrix. Moreover, the qualitative assessment provides a classification of the different play types, like "baseline drives," "breakaways," "drive and fouls," and "3's" (a play where the offense focuses outside the three-point line and an offender attempts a three early in the possession).

Gridding Techniques

Upon analyzing the four different gridding techniques, a slight bias towards the Delaunay Triangulation gridding was exhibited qualitatively. However, not enough data (with labels) exists in the dataset (i.e. there are only four games), so it is difficult to present quantitative conclusions on the differences between the gridding techniques. However, it does seem that the similarity scores, overall clustering, and classification component is somewhat sensitive to the underlying gridding technique.

Since small perturbations in the trajectory data can cause dramatic changes in sequence, especially for short sequences, it is possible that more theoretical guarantees need to be enforced on the way in which crossing sequences are generated. For example, a more formal analysis could be computed about the size of perturbation for any one point in the trajectory data and the corresponding change in similarity score of that possession with any other possession.

Convex Hulls

Inspecting the convex hulls provided no additional insight in this context. In particular, the ratios of the CHAD and CHAO did not give rise to any strong or obvious correlation with other information about the play. The CHAD/CHAO ratios did not seem to give insight into each cluster, possibly because each cluster is computed with dynamic data, while the CHAD/CHAO ratio is a decidedely a static measure.

5.1.5. Future Work

We feel that we have a good framework for analysis of basketball data. The most important aspect of future work will be the availability and coherency of the dataset, where many of the techniques presented here could be experimentally validated with more comprehensive inspection.

Otherwise, many of the improvements in this data-driven report lie with the permutation of several different techniques that may satisfy various definitions outlined here. For example, different notions of metrics, clustering, or gridding could be use and cross-validated to demonstrate a richer and more fruitful comparison among techniques. Changes in our sequence similarity technique could also provide another variable within this pipeline.

Moreover, the possession analysis techniques could be used to develop a more granular notion of the "play" in basketball, where the trajectory data could be used to find the demarcation between them. While we have a subjective analysis of a "play," a more quantitative approach to separating plays may provide new insights on their inherent structure.

The CHAD/CHAO ratio could be studied further, like by providing a way of dynamically recomputing this ratio in a possession and studying its behavior as an indicator over time. With a definition of a play, it would be interesting to see the CHAD/CHAO ratio at the level of a play, rather than a possession. Moreover, this ratio could demonstrate some high-level information on the geometry of a play that complements the more local geometry given from the current pipeline.

The incorporation of scoring data would be an exciting innovation, not only to classify possessions, but also to describe the inherent "success" or "failure" of a possession. In particular, the scoring data could be used as a quantitative way of adjudicating if a possession was correctly executed and is consistent with the behavior of possessions that are indeed similar to it. More tractable visualizations and ways to analyze our results would also be useful in this context, as basketball is inherently discoverable and comprehensible. We also are interested in comparing our results to those found under supervised learning, but this will require labeling of datasets, which would be another leap in the quality of the dataset.

5.2. Basketball through Geometry and Machine Learning

This section extends the topological approach in Section 5.1. We present a second perspective on the same type of data given in Figure 5.7. Instead of using a gridding scheme to extract a dynamic network, we explicitly take frame-by-frame networks based on passing opportunities. We then turn to a more sophisticated statistical model (a jump Markov model) and leverage the newer machinery of a Transformer neural network to study the passing dynamics. In Section 5.1, our emphasis was on unsupervised play classification; in this section, we provide experimental validation through a supervised trajectory prediction task.

One of the most salient results from this section is that we can leverage a pre-trained model, namely the famous Transformer model[287]. We do not modify its architecture, nor are we forced to construct our own complicated deep network architecture. Instead, by placing a couple additional layers around Transformer, we can leverage it directly and immediately, which aligns with the current pre-train/fine-tune approach to machine learning. More astoundingly, we repurpose a model in natural language processing for geometric data, which is only possible through dynamic networks. By constructing a proper abstraction around our data, we can harness the power of machine learning without much additional effort.

5.2.1. Introduction

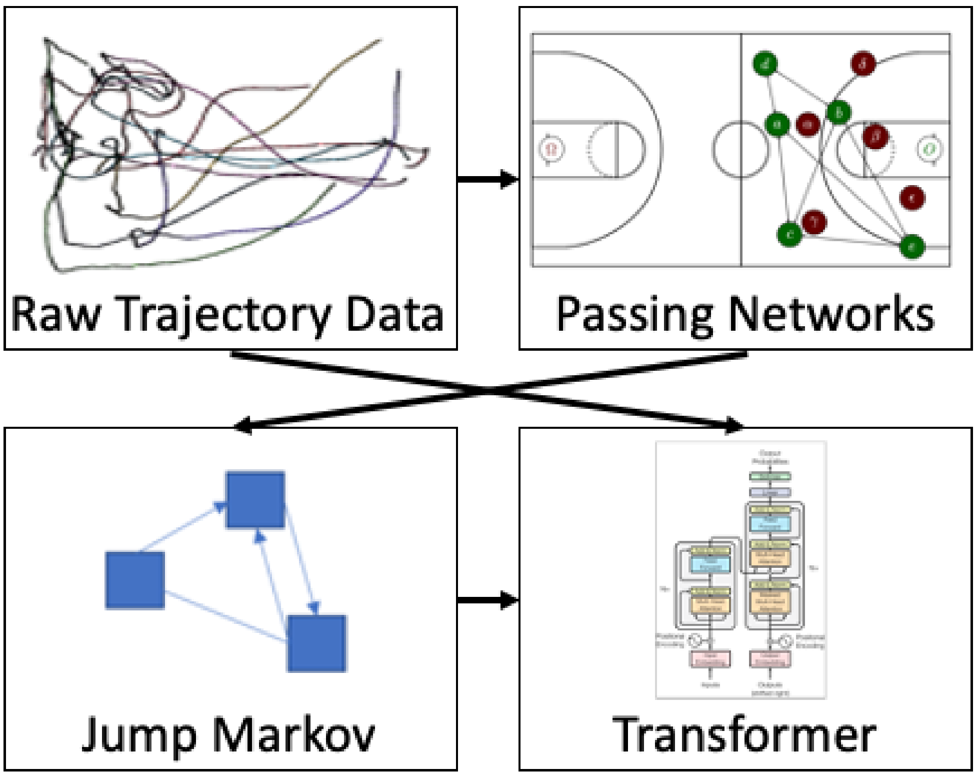

Figure 5.12: Overview of the data analysis pipeline presented in this section. First, the raw trajectory data is converted into interaction networks. Then, by comparing graphs up to isomorphism, we can construct a "library" of possible configurations. We can then construct a jump Markov model by taking the empirical maximum likelihood estimator with graphs as the state space. Finally, we can feed in the raw trajectory data and the graph data from the jump Markov model into a Transformer model for prediction. This article opens the door to future work on inferring game semantics and strategies from actual games.

Multi-agent systems are fascinating both for their geometric properties and for their complex interactions. In a variety of contexts, we would like to understand their underlying dynamics, moving beyond the construction of black-box models that simply replicate their behavior. Therefore, we strive to formulate a domain-specific, "semantic" understanding of a multi-agent system that produces geometric data. In addition, we consider related modeling and prediction problems. One paragon example of a multi-agent geometric system includes fast-paced "invasion" sports like basketball, soccer, and hockey[118].

This chapter focuses on basketball for its relevance, availability of data, and the interesting mathematical setting that it provides. In particular, basketball involves few agents, which makes it more tractable for analysis, but agents are constantly in motion, with sophisticated interactions, which provides a great variety of events to study.

We study the trajectories of players as they move across the court. To develop a rich understanding of the dynamics of basketball players, we develop a model with:

1. Formation discovery: a semantic understanding of the functional roles of players;
2. High compression: an efficient representation of a game, as player trajectory data is large and difficult to interpret;
3. Predictive power: a mechanism for generating synthetic basketball data and predicting trajectories of players.

To achieve these goals, we construct dynamic passing networks, from which we produce a jump Markov model that provides formation discovery, high compression and—when coupled with a Transformer model—predictive power. Our model is also simple and admits a rigorous theoretical analysis that can provide insight into the underlying dynamics of basketball. Our pipeline is summarized in Figure 5.12.

By constructing a jump Markov model, we convert inherently geometric data into sequences of symbols, which are ripe for lexical analysis akin to natural language processing. For this reason, we can use a Transformer to better "parse" the underlying syntactical and semantic features that exist within a basketball game. Thus our pipeline can perform downstream predictive tasks, like trajectory prediction, with both geometric data (which has been previously addressed with varying success) and semantic data.

In summary, the synthesis of semantic and geometric data is the main contribution of the work in this section. We propose a novel pipeline to convert geometric data from a complex, interactive multi-agent system into semantic sequences. These two perspectives on the data provide better insight into the underlying dynamics, as well as stronger results in important applications like trajectory prediction.

5.2.2. Related Work

The analysis of basketball player trajectory began shortly after cutting-edge technology was developed that allowed for comprehensive player tracking [179]. Early models did reasonably well in role discovery and compression, but, over the past decade, predictive power has greatly increased—especially for the most common prediction task, trajectory prediction.

Trajectory Prediction

Trajectory prediction focuses on forecasting the movement of players given their history. Generically, trajectory prediction is a much broader discipline that has historic roots, but we restrict ourselves to those that concretely involve sports[245,199].

One state-of-the-art paper[305] considers a combination of graph neural networks and recurrent neural networks to predict K additional frames of a trajectory, given L initial frames. Interestingly, the generative nature of the model also allows for the testing of counterfactual claims, e.g. predicting player movement given alternative hypothetical histories. This work, however, does not consider the adversarial dynamics, in that the model does not incorporate data from the other team for a given player.

More recently, the Generative Attentional Multi-Agent Network (GAMAN) model, proposed in[167], and Dynamic Neural Relational Inference (dNRI), proposed in[109], seem to attain the state-of-the-art results in predicting trajectory data (though the latter has yet to release full results of the model). Previous work includes a variety of other techniques in deep learning, but have been largely superseded by the aforementioned models. Newer models tend to incorporate both more modern structural elements, e.g. attention, but also have better training mechanisms that render previous work obsolete.

However, whether or not explicit formation or role discovery is important to trajectory prediction is not well-established. Trajectory prediction models must avoid sensitivity to specific player identities, as each player can perform several different functions in a game that will affect their movement patterns. Role discovery can aid in this process by assigning players to specific roles, which may improve trajectory prediction. Certain recent models avoid explicit role discovery, however, and still seem to attain competitive results in trajectory prediction.

Role Discovery

Discovering the actual function that a player is performing can be per se useful. The naive approach of tracking players by their personal identity across plays, games, seasons, and teams may yield a more confused analysis. Therefore, one common approach to analyzing sports data, especially with trajectories, is to develop some kind of "role" categorization, e.g. a point guard. Of course, most sports and indeed basketball come with preconceived notions of what a "role" is, but a first-principles approach to role discovery has yielded interesting results.

Work first done in [179], extended in [307], patented in [26], and updated in [128] proposes an extensive set of efficient methods to classify roles, especially within field hockey. This work relies on the creation of permutation matrices that induce a bijection between players and roles. Through deep learning, the specific permutation matrices are learned given previous trajectory information. The most recent model leverages deep learning to produce "formation templates" that provide standard arrangements of players, along with an assignment of functional roles. In general, these role-based models support "permutation-equivariance" which are less sensitive to the specific identities of players [*].

Role discovery is an important strand of research, as it emphasizes semantics of the game. While trajectory prediction is a compelling problem in its own right, role discovery highlights the underlying structure of a particular game. Through role discovery, we can provide interpretable labels or classifications to particular player formations and movements.

Role discovery has also garnered popular attention [15,23,52,185,320], with a variety of approaches in constructing and classifying roles.

[*]This research, as applied to sports, relies heavily on data from STATS, which is a company that has technology to collect and analyze player position data.

Network Analysis

Network analysis for sports data is comparatively old, with some early efforts in soccer beginning in 1979[107]. The most relevant type of network analysis, however, has been on *passing networks* and investigates the frequency with which players pass the ball[221,106]. This research direction emphasizes the study of aggregate network properties, e.g. the *centrality* of a player on a particular team[118]. Network analysis thus far has considered the network of passing frequency over an entire game rather than the specific dynamics during the game itself.

5.2.3. Semantic Geometric Pipeline

Geometric Data

Our principal dataset contains the position of the offense, the defense, and the ball—expressed as (x, y)-coordinates—across an entire game capture at 25 frames per second (i.e. 40ms between frames). Notably, basketball is divided into possessions, where the teams alternate between the defensive and offensive roles. Consider five indexed points as the offense $O = \{o_1, o_2, o_3, o_4, o_5\} \subset \mathbb{R}^2 : |O| = 5$ and five indexed points as the defense $D = \{d_1, d_2, d_3, d_4, d_5\} \subset \mathbb{R}^2 : |D| = 5$. One frame F of data is the ordered pair (O, D). Each possession P is a sequence of frames of data and a game is a sequence of possessions P_l.

Dynamic Passing Networks

From this dataset, we construct *dynamic passing networks*. These networks are defined over the offense (i.e. for basketball, there are five nodes); two players on the same team have an edge joining them if there is no defense player in-between. Figure 5.13 provides some intuition for this construction.

More precisely, given two sets of five members of the offense $\mathcal{O} \subset \mathbb{R}^2$ and five members of the defense $\mathcal{D} \subset \mathbb{R}^2$, we can define a graph G over vertices $\mathcal{O}$. For $o_i, o_j \in \mathcal{O}$, edge (o_i, o_j) is in the graph if there is no $d \in \mathcal{D}$ such that the *line of sight* $l(o_i, o_j)$ intersects with *occlusion* *field* $F_r(d)$. The line of sight between two points x, y is defined as

$$l(x,y) \triangleq \left\{z \in \mathbb{R}^2 \,|\, z = x + (1-\lambda)v,\ v = y - x,\ \lambda \in [0,1]\right\}$$

and an occlusion field on point p and radius $r = 3$ as

$$F_r(p) = \left\{x \in \mathbb{R}^2 \,|\, \|x - p\|_2 \le r\right\}$$

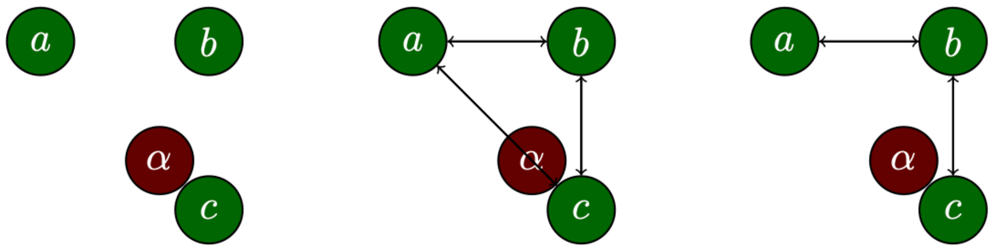

Figure 5.13: A deeper look at a snapshot of a basketball game. Green nodes with Roman characters are offensive players. Red nodes with Greek letters are defensive players. On the left, we see player positions with occlusion fields. In the center, we see all offensive lines of sight. On the right, we see the occlusion network with only the offensive edges.

For each frame, we construct these networks by starting with a complete graph with the offense players as the nodes; we then remove edges from this graph if a defensive player occludes the straight line of sight between a pair of offense players. Figure 5.14 depicts a passing network for one frame.

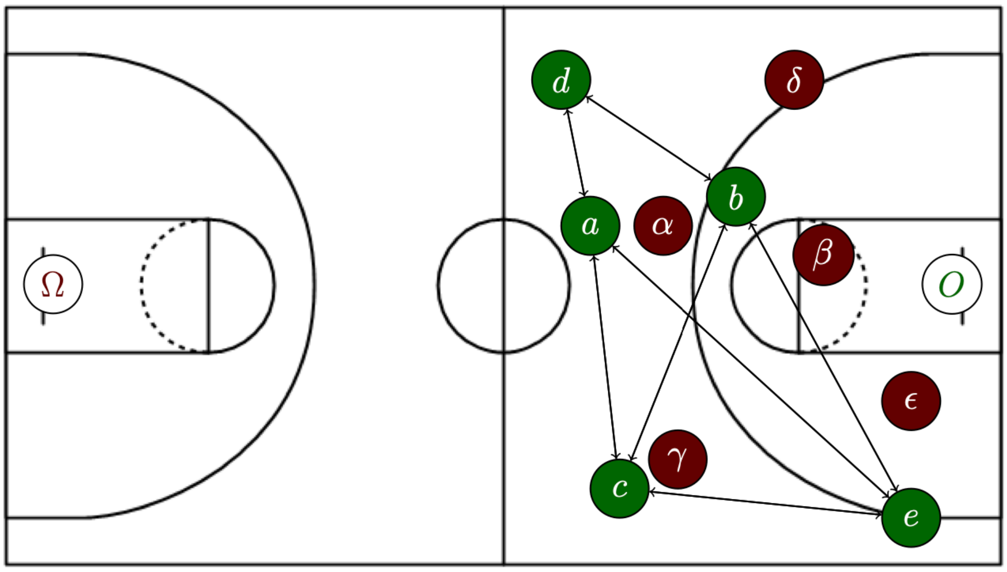

Figure 5.14: A frame of the basketball game with the constructed passing network. Green circles with Latin letters are the offense, while red circles with Greek letters are the defense.

Thus, we can convert each frame of the basketball game from two sets of (x, y)-coordinates to a graph. We further compress this representation by only considering graphs up to isomorphism, which allows us to store a label to a representative graph per frame. This procedure thus converts a sequence of frames of position data into a sequence of labels.

Notably, this representation sheds the direct geometry of the basketball game. However, this sequence of labels provides a purely semantic and highly compressible representation of the game and is justified by three physical assumptions, which are validated by our results:

1. From one frame to the next, there can be at most one edge that changes and this edge change can only occur as a result of well-behaved player trajectories;

2. While it is possible that two very different geometries produce the same graph, a sequence of graphs must come from a real play, and thus provide enough information on the possession;

3. Basketball is "fast-paced" enough that it follows a Markovian property. Mainly, players do not have time to consider the history of the game to factor into a future strategy, and

instead either follow a set strategy or respond nearly instantaneously to their current environment.

Jump Markov Model

We can construct the maximum likelihood estimator of a *jump Markov model* that characterizes this sequence of labels. A Markov model assumes that a sequence of labels exhibits the Markovian property, namely that the label of one entry in the sequence is only influenced by the previous entry, and no others. Such a sequence can be characterized by "transition" probabilities: for every possible pair of labels (x, y), the frequency with which y appears right after x in our sequences.

A twist on such a model is a "jump" Markov model that additionally assigns a *hold time* to each state: namely, since a basketball game is a continuous-time process, we can also capture the average length of time that our sequence of frames does not change.

More precisely, given an indexed state space $\mathbb{S} = \{s_1, \ldots, s_n\}$ of some symbols and an $(n \times n)$-stochastic matrix P, we say that state s_i transitions to state s_j with probability $P_{i,j}$. This state space and transition matrix define a (time-homogeneous, discrete, finite) Markov chain, which is a stochastic process that we write as E_k with $k \in \mathbb{N}$.

Next, consider a Poisson process, which is a continuous-time stochastic process with rate λ and associated counting process $\{N(t)\}$. A Poisson process is a stochastic process that "counts" the number of exponentially distributed events. More formally, given an i.i.d. sequence τ_k of exponential random variables with parameter $\lambda > 0$, we can construct the sequence $T_{k+1} = T_k + \tau_k$ with $T_0 = 0$. In short, these T_k variables give us the "time" that the k-th event has occurred. The counting process $\{N(t)\}$ is defined as $N(t) = \max\{k : T_k \leq t\}$

Overlapping the discrete-time Markov chain on the continuous-time Poisson process (where the "events" that occur are transitions of the Markov chain) yields a jump Markov model, defined as the continuous-time stochastic process $X(t)$ such that

$$X(t) = E_{N(t)}$$

which is our stochastic process of interest[250].

We use our passing networks up to isomorphism to define the state space of the Markov chain. From here, we can construct the maximum likelihood estimator of the jump Markov model that captures the behavior of the changes in state. Each state s in the state space can also be associated with a *jump time* λ_s that defines the average time in each state.

A Transformer

Finally, to validate the empirically constructed jump Markov model, we can use a Transformer. Transformer is a state-of-the-art deep neural network that excels at two tasks: sequence completion and sequence translation. Transformer is generally a staple of natural language processing, but can work in a variety of different sequence-related tasks[287], as detailed in Figure 5.15. For our experiments, we retained all of the standard architectural elements of the out-of-the-box Transformer, only making a mild alterations as necessary (described in the following sections). For our experiments, we divided the frames into short sequences of length 50; with a standard 80-10-10 allocation, these sequences were then split into a training, validation, and test set respectively.

Baseline: Sequence Completion

To establish a baseline, we first used Transformer for sequence completion: given 40 frames of positional data, we used Transformer to predict 10 frames of positional data. For this task, we removed the standard lookup embedding layer for transformer, and instead directly concatenated all player positions to construct a vector in $\mathbb{R}^{10\times 2} = \mathbb{R}^{20}$ (10 players, each with an x and y coordinate).

Then, we used only the Decoder part (i.e. right half) of Transformer to complete the sequence. Though a naive baseline, this setup provides insight into the raw performance of sequence completion using just the positional data. It provides insight into how well we can predict the rest of a sequence from just the positional data, without any data manipulation or enhanced insights.

Comparison: Sequence Translation

For our second task, we leveraged Transformer to "translate" between the sequences produced in the state space of the Markov model to positional data. In other words, we used the full Transformer system to convert graphs into position data. Notably, the output of this sequence translation experiment is the same as the previous sequence completion experiment. Therefore, we can directly compare the performance of the two experiments to measure how accurately we can predict 10 frames of positional data from 40 frames of given data. In the previous completion experiment, the 40 given frames were also positional data; in the sequence translation experiment, the 40 given frames are positional data, as well as graph data.

This setup corresponds directly to Figure 5.12. For inference, we:

1. Convert raw frames of trajectory data into frames of passing networks
2. Convert passing networks into a token, by assigning a label to each equivalence class of graphs
3. Use the jump Markov model to predict the next token in a sequence
4. Feed the predicted tokens into a Transformer to predict frames of position data.

However, for training, we directly allowed Transformer to predict a frame of position data from a token, which helps isolate the problem of translating tokens to positional data. Additionally, we computed the transition matrix and hold times of the jump Markov model during training. During inference (i.e. testing and validation), we use the predictions made from the jump Markov model and then fed these predicted tokens into Transformer. This second translation experiment thus highlights the importance of the jump Markov model in our setup.

Each frame of the positional basketball data was therefore converted into a graph. Then, all graphs were compared with each other up to graph isomorphism and a representative graph was chosen for each equivalence class of graphs. Each representative was given an arbitrary label (token). Thus, each frame of position data was converted into a token. These token sequences, along with the associated position data were treated as the "input" sequence to the Transformer and the positions as "output" data.

Since the input sequences were already tokens, we could use a standard learned embedding. The output data, as raw geometric data, were treated slightly differently. To match the input and output embedding dimension, we used a fully dense linear layer to further embed the output sequence into the appropriate model dimension and then a used another fully dense linear layer to convert the result of the decoders back into vectors in $\mathbb{R}^{20}$, in lieu of a softmax layer. More precisely, our data is expressed as a sequence of vectors in $\mathbb{R}^{20}$, but the model operates over vectors in $\mathbb{R}^{d_{model}}$ where d_{model} is some learned hyperparameter and will not in general be 20. Therefore, we used two dense linear layers at each end of the model to solve this problem: $L_1 : \mathbb{R}^{20} \to \mathbb{R}^{d_{model}}$ and $L_2 : \mathbb{R}^{d_{model}} \to \mathbb{R}^{20}$.

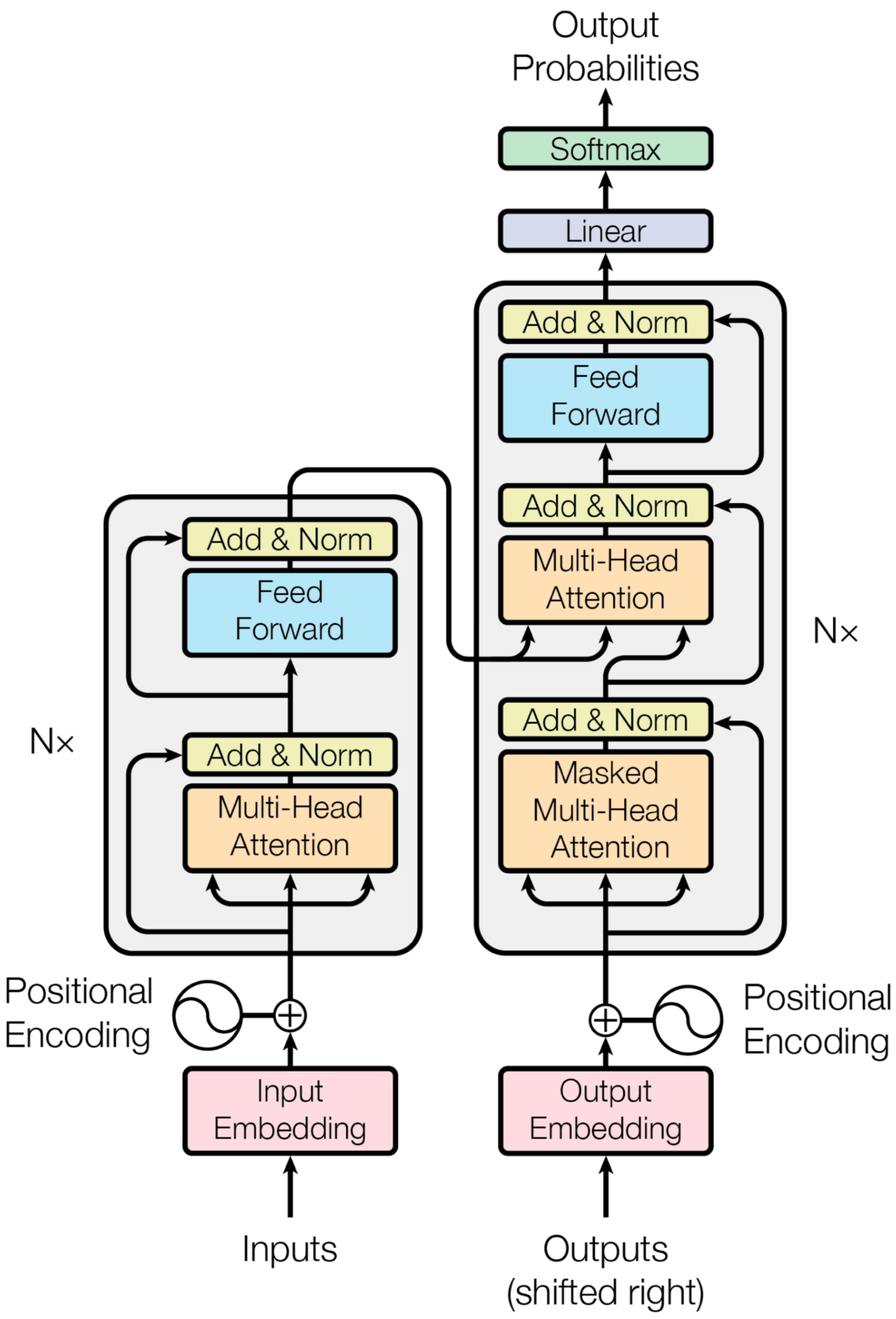

Figure 5.15: The architecture diagram of a Transformer, as presented in[287]. The Transformer has an encoder system (left) and a decoder system (right) and uses self-attention in three ways to accurately capture semantic information from both sequences.

5.2.4. Results

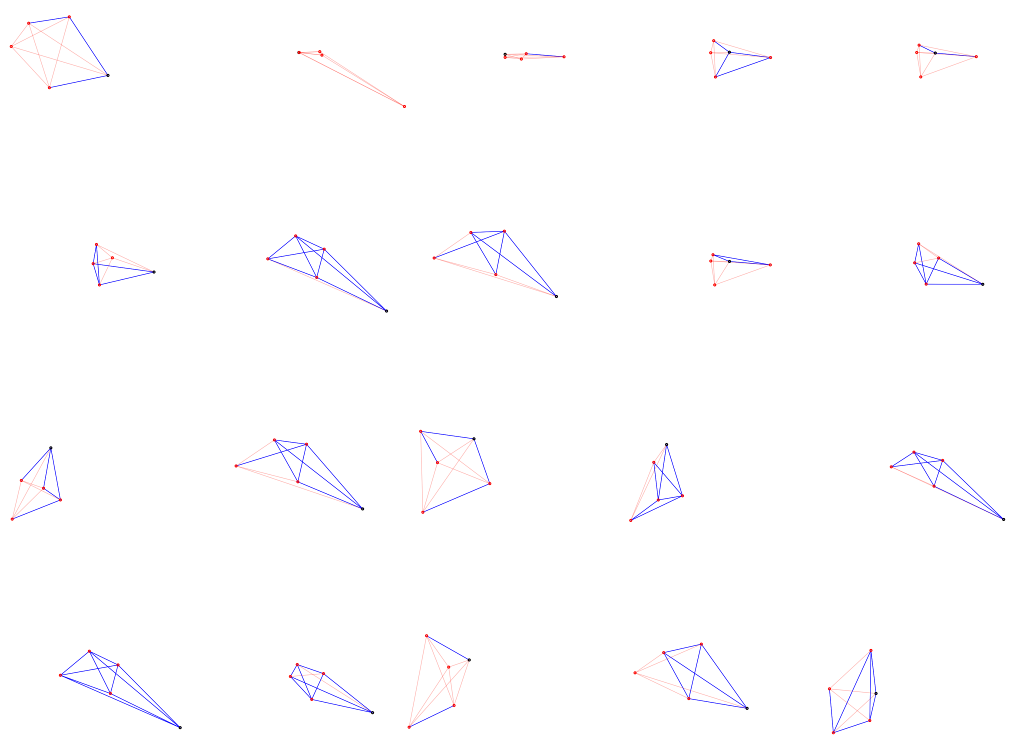

Figure 5.16: Sample library of graphs for a possession, descending in order of frequency from left to right and top to bottom.

Markovian Property

First, we consider the sequence of graphs that we construct from the geometric data. Understanding the dynamics of these graphs provides insight into the overall dynamics of the game. It is paramount to find a model that captures these dynamics. Fortunately, our data seems to express a Markovian property and we can therefore use a Markov model to capture the essential elements of our graph sequences. In this section, we provide empirical validation for this claim.

To begin, we note our assumption that dynamic passing networks can only change by one edge at a time. From a theoretical perspective, we study the case of a disc of radius r that is full intersected by some line segment segment, as in Figure 5.17). If the ball is constrained to move some arbitrarily small distance ϵ, that this ball will continue to be fully intersected by the line segment. This analogy extends to our basketball sequences: r is the radius of occlusion for some defensive player; ϵ is the maximum distance a member of the defense can move within one frame of data. Given that there is some universal constant v that defines the maximum speed a human can move, then we can arbitrarily increase our sampling rate to yield some maximum ϵ distance. In our context, if we sample the basketball player positions often enough, we have some intuition as to why it is unlikely for more than one edge to change at a time. Moreover, we could enact some tie-breaking scheme to enforce that one edge changes at a time. Empirically, 99.3% of changes in number of edges are within one edge, which we could bring to 100% if we could sample more frequently. In fact, given that 99.3% is quite close to saturation, we can conclude that our sampling frequency is nearly correct: neither too often nor too sparse.

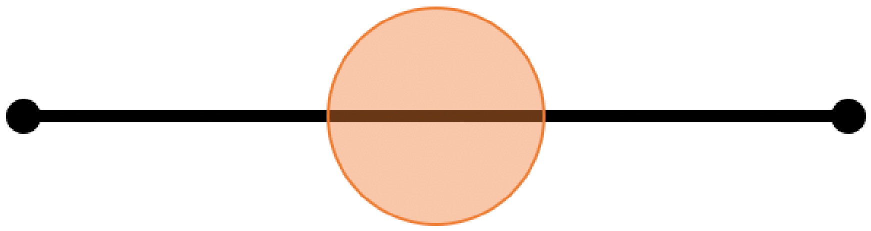
Figure 5.17: Two players on the offense (black points on either side of the line segment) with a member of the defense in between them (large orange ball in the center).

This assumption, that only one edge changes at one time, is convenient, as it allows us to study the change in number of edges of the graphs, which is much simpler to analyze. In other words, we can convert a sequence of graphs into a sequence of number of edges and learn much about our system without having to rely on a full classification of the graphs. It is possible to define a variety of partial and total ordering schemes on graphs and without a "natural" method of comparing one graph to the next, we would require some additional assumptions or constructions. Instead, we look at the number of edges, which is simply an integer, giving us a natural order. We can also state a fundamental assumption: if the number of edges does not change in a graph sequence, then the graph (up to isomorphism) also does not change; this fact follows directly from our assumption that only edge can change at a time. For example, in Figure 5.18, we can see the number of edges in the passing network sequence over time (i.e. per frame).

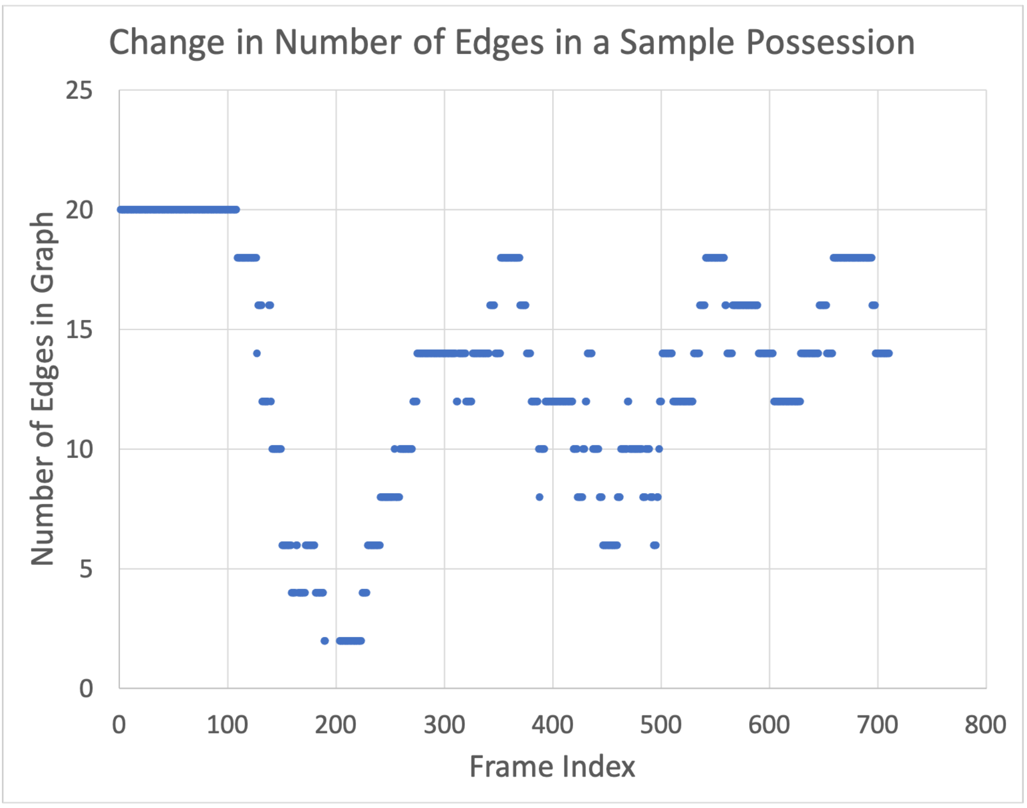

Figure 5.18: The number of edges for a particular possession in the game. Mathematically, the number of edges are a random walk over the frames of data within a possession.

If we treat the number of edges as a Markov chain, we can verify the Markovian property of our edge count data by checking how a change in edges predicts the next change. Table 5.1 gives, for an entire game, this edge change probability. This data for the game accurately reflects the distribution for each possession, as well, which obviates some concerns about variability within possessions.

The data in Table 5.1 thus shows us that the previous change in edges does not influence a future change in edges, in that edge change increments are in fact independent. This data therefore verifies that the number of edges over the course of a possession is in fact Markovian, which is a surprising, but very useful property.

	-	0	+
-	0.07	0.87	0.06
0	0.05	0.89	0.05
+	0.07	0.85	0.08

Table 5.1: Probability table for changes in number of edges. The first row indicates that if the previous change was a decrease in number of edges, then there was a 7%, 87%, and 6% chance respectively that the next change in number of edges was a decrease, no change, or increase. The second row follows the same pattern given the previous change was no change in the number of edges. The third row is the same for an increase in number of edges.

Moreover, this suggests that the graph sequence itself is Markovian because of our underlying assumption that one edge change at a time can occur. Therefore, we can use a Markov model to approximate the sequence of graphs over the course of a possession. Finding the maximum likelihood estimator of a Markov chain is also quite simple, as it is the empirically found transition probabilities.

Jump Markov Model

While interpreting the sequences of graphs as a Markov chain is a big step towards understanding the underlying dynamics, we have some minor adjustments we can make to further refine this model. In particular, we observe from Table 5.1 that the overwhelming behavior is for the number of edges in the graph sequences to not change. Because of our assumption that only one edge change can occur at a time, this suggests that the overwhelming behavior is for the graphs to not change.

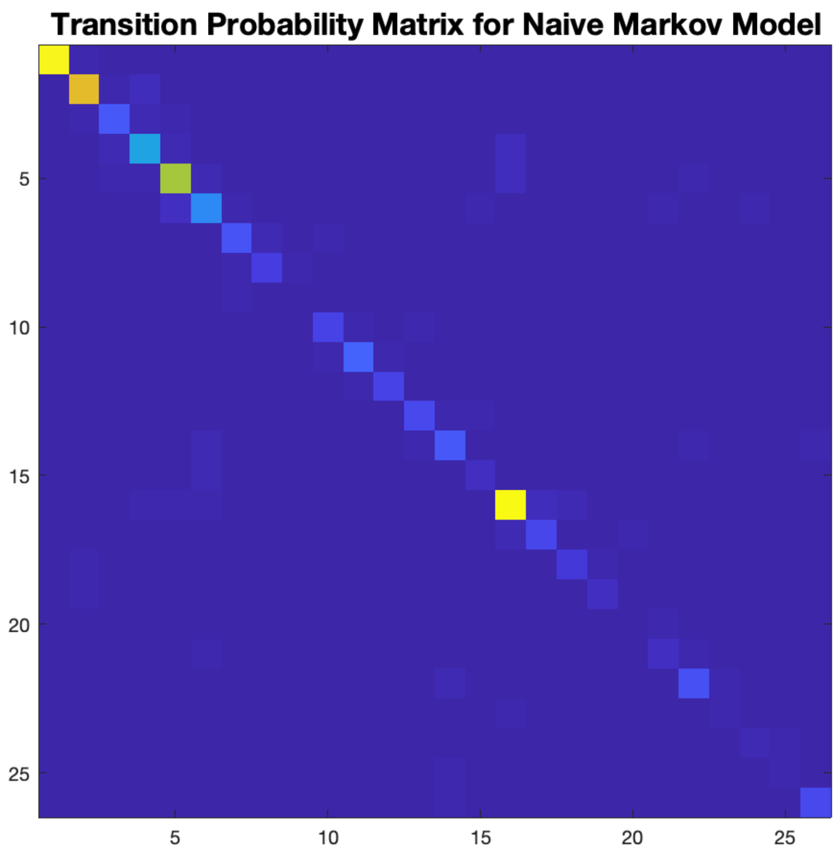

Figure 5.19: Probability transition matrix for a sample possession. This transition matrix assumes a naive Markov chain and therefore has most of the density for each row on the diagonal.

If we label the graph sequences up to isomorphism and construct the transition probabilty matrix, we see that the majority of the density is placed on the diagonal, which indicates a self-transition. Figure 5.19 provides a representative transition matrix for a possession. This behavior warrants some additional investigation.

To wit, a basketball game is obviously continuous, but our Markov chain with a transition matrix is discrete. For this reason, we lift the Markov model into a jump Markov model, which allows for different "hold times" per state, i.e. the distribution of time spent in a particular state before a transition occurs. In the naive Markov chain, these hold times are modeled as self-transitions, whereas in a jump Markov model, these are modeled as occurring along some exponential distribution.

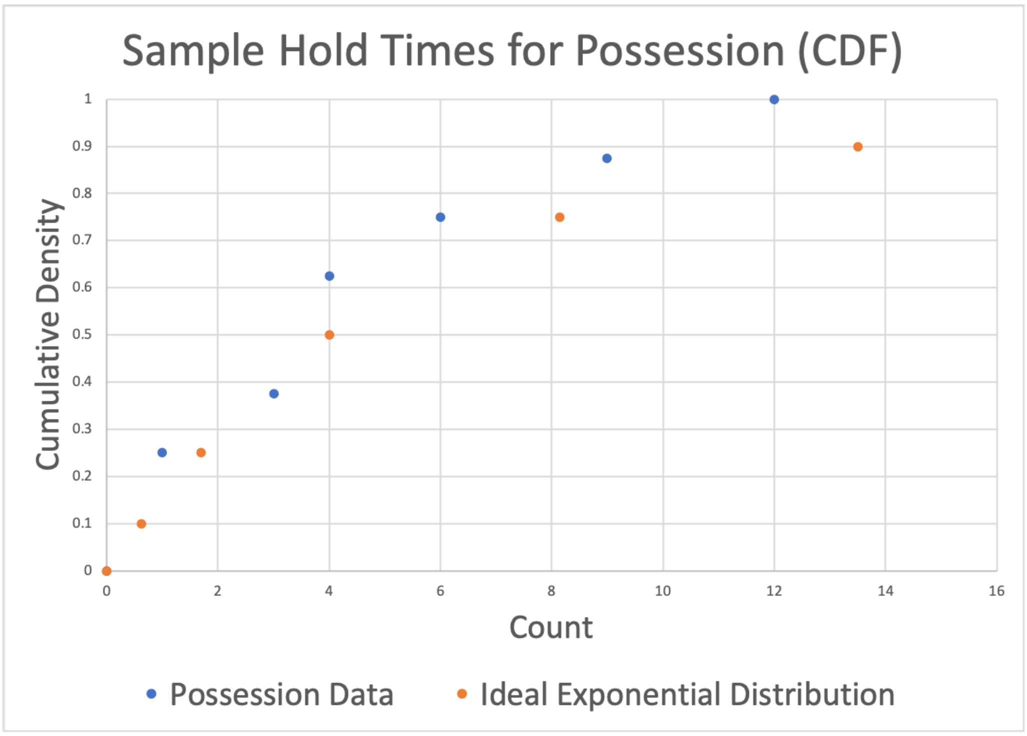

Figure 5.20: Sample hold time distribution as a cumulative density function of a representative graph in a representative possession. The blue dots are the true data, which have an average of 5, giving an exponential distribution with parameter $\frac{1}{5}$ The orange dots represent an ideal exponential distribution with the same parameter.

Figure 5.20 provides a sample hold time distribution for a representative graph in a representative possession. In this case, we see the various hold times for the particular labeled graph, and for reference, an ideal exponential distribution. Therefore, we can see that a jump Markov model with explicit hold times per graph provides a more robust perspective on state changes.

By using a jump Markov model, we can actually see stable configurations of players throughout the game. The probability transition matrix for the model suggests the "semantics" of the dynamics, insofar as we can track likely changes in configurations for the players. The hold times demonstrate the overall "stability" of a configuration, i.e. how much time is spent in a particular conformation.

We also can generate a "library" of graphs that provide insight into which configurations appear most often and what typical sequences of graphs look like. Figure 5.16 is such a representative library of graphs presented in descending order by frequency of occurrence in a possession.

Translating with Transformer

Our final set of experiments comes from using a Transformer model. We attempt a standard task in invasion sports, namely predicting the trajectory of player positions across the court. We use a 40-10 split, predicting 10 frames of data from 40 given frames.

The first model is a naive setup where we attempt to predict trajectory data from the geometric data present in a frame alone. Using the Transformer encoder, the position data is fed as a sequence to just the encoder system (6 encoder layers) and a final linear layer to convert the Transformer memory into a concatenated vector of position data.

The second model is the full Transformer setup as described in Section 5.2.3 where the encoder accepts a full sequence of geometric data and the decoder operates on the geometric data as described.

Holding all factors constant, such as learning rate (0.0001), optimization algorithm (Adam), batch size (64), embedding dimension (200), number of layers (6), number of attention heads (8), and all other hyperparameters, we can see the effects of translating from the sequence of graphs through ablation.

The validation loss is set as the sum mean-square error of the 10 frames to be predicted. Namely, when predicting frames $x_{41}, \ldots, x_{50}$ with model output $\hat{x}_{41}, \ldots, \hat{x}_{50}$, the validation error is

$$\frac{1}{10} \sum_{i=41}^{50} (x_i - \hat{x}_i)^2$$

Importantly, this experimental setup provides us with direct insight into how useful and fruitful the jump Markov model is. By using a naive baseline with Transformer, we essentially are performing an ablation analysis, where we see the effect of our entire graph extraction setup. In this context, the MSE score (i.e. the validation error defined above) provides us with a useful quantitative result on the utility of our entire pipeline.

Over the course of several training runs that were run to convergence, the naive model achieves an MSE score of approximately $\mathbf{280}$, while the translation model achieves a best MSE score of $\mathbf{94}$ (lower MSE is better). This dramatic decrease in MSE through translation represents the effect of including the graph data, instead of attempting to learn position data directly.

5.2.5. Conclusion

This work presents a semantic analysis of geometric data from sports. By converting geometric position data to graph data, we can leverage the gleaned structure to improve the overall accuracy of downstream applications, like trajectory prediction. Additionally, the construction of a jump Markov model provides clarity into the overall structure of a game through two important properties: first, a transition matrix that indicates which configuration of players lead to others and second, hold times, which suggest the stability of particular configurations in context. By constructing this jump Markov model, we can develop insights into our data that goes beyond the geometry.

From our experiments, we can conclude that our pipeline, in incorporating the graph data, does provide practical insight. Through the completion-vs-translation setup, we can quantify exactly how much information is extracted out of this setup; the compelling drop in MSE garnered by the use of graph data underscores the importance of extracting and using the graph data.

Future work could use this structure to refine trajectory prediction, perform more sophisticated "play" (strategy) discovery, predict in-game dynamics with speculative pairings of different players, and much more. Plus, the jump Markov model could be used to create synthetic datasets to either augment data for existing applications or to inspect the dynamics of hypothetical situations.

In all, this work presents a model that discovers formations through the graph states of the jump Markov model; that is highly compressible by requiring only a transition matrix for states and a list of hold times; and has predictive power when used in a downstream application. From geometry to semantics, we highlight the spectrum of perspectives through which we can view our data.

5.3. Lessons on Dynamism

There are four important lessons to be learned from this chapter. First and foremost, we can see how powerful topological abstractions can be. Shedding away unnecessary geometric structure provides a comprehensible and learnable set of features of our application domain. We have philosophically extended Euler's ideas in this way.

Second, we have demonstrated the computational underpinnings of even dense data with high-fidelity sampling. Our data is produced at 24 frames per second, which results in voluminous information, since our system is fast-moving. However, we are still able to compute a theoretically grounded set of features that assist in practical performance.

Third, just as with any data-heavy area, we provide multiple featurizations of the same data. Dynamic networks are not a trivial nor obvious construction and there may not be a unique and dominant way to extract dynamic networks from the same dataset. Instead, we provide two primary different ways of extracting network information, each with further options and variations. In this way, we demonstrate the capacity for dynamic networks to arise within a variety of contexts, even in the same setting.

Finally, we studied the power of contemporary data analysis tools. While graph theory is a historic discipline, it is still highly relevant today and will likely continue to be relevant for the foreseeable future. We can use both topological data analysis and machine learning to study our systems and by using dynamic networks, we improve on existing standard techniques to study this data. In all, our abstraction of dynamic networks prove to be cutting edge tools that can contend well with other methods from the age of Big Data.

Modern science says: The sun is the past, the earth is the present, the moon is the future. From an incandescent mass we have originated, and into a frozen mass we shall turn. Merciless is the law of nature, and rapidly and irresistibly we are drawn to our doom.

Nikola Tesla

6 Final Frontier: Inspired by Space

SPACE has captivated humans and animals alike for our entire living memory. While we have not yet conquered this marvelous, awesome void, we have made tremendous progress in its exploration within the 20th and 21st centuries. We have much further to go, but one of the greatest innovations of the modern era has been to extend the boundaries of usable places well past the atmosphere and into the cosmos. While there have been several tangible milestones in the development of space technology, progress is generally slow and iterative. Fittingly, this chapter covers a set of studies conducted in collaboration with the *National Aeronautics and Space Administration* (NASA).

One particular area of study is our ability to communicate through space, whether it be to the International Space Station or to our probes flying beyond the Solar System. Fundamentally, this setting contains an important dynamic network: the communication network as we and many space objects hurtle throughout the galaxy. This chapter covers contributions to space networking, with our dynamic networks providing a framework to studying the problem of communication in this precarious environment. We will first study the structure and nature of these space networks by analyzing lunar networks, after which we focus on routing problems in more general settings. More precisely, the first section provides a comprehensive theoretical framework for studying dynamic networks, especially in the space-networking context. The second section applies these contributions and provides examples, data, and other empirical results on their utility.

Taken in concert, this chapter strives to provide a thorough and detailed exploration of dynamic networks in the space networking context. By leveraging the theory and praxis of a wide range of mathematical and computational tools, we can better understand long-range communication and design the next generation of space networks.

6.1. Temporal Graphs in Lunar Networks

In this section, we unpack the theory of dynamic networks. In particular, we construct a few useful definitions and provide some foundational theoretical results (along with proofs) of how these objects operate. We then provide an extension of these definitions through a technique called *summary graphs*, which also provides its own theoretical results. Finally, we look at commentary on how this work connects with other important areas of mathematics in this research application area.

6.1.1. Temporal Graph Theory

As we have seen, graphs are one of the fundamental mathematical structures used to study computer networks. As always, we continue to build upon a simple, finite graph; in this section, we distinguish between undirected and directed graphs and we call the former a "graph" and the latter a "digraph." Mathematically, an undirected graph identifies $(i, j) \equiv (j, i)$ for all $(i, j) \in E$, whereas a directed graph does not. This abstract structure allows us to model relationships between objects, such as connectivity. In the setting of computer networks, an edge might be a link between two devices.

In this setting, it is natural to study the importance of a device to the network overall. An example of something that measures this importance is *centrality* which ranks nodes based on their importance to connecting the graph. It is also natural to study how one optimizes in a network from an individual, greedy perspective versus from the perspective of the network overall.

Graphs can also be "decorated" by coloring their vertices, adding directions to the edges, or weights indicating the capacity or cost of a connection, for example. Once graphs are decorated, algorithms and analysis can be applied to solve several problems, such as single source shortest path, max flow, or minimum spanning tree, to name a few. There are several well-established tools for solving these problems*.

One of the implicit assumptions in the construction of many terrestrial networks is that there exists a “backbone” of the system that will largely go unchanged as time passes. Another assumption is that communications can be passed near instantaneously. The way graphs are used to model these networks for engineering purposes often rely on these assumptions.

Both of these assumptions quickly fall apart in the context of space networks as the assets are subject to orbital mechanics and may lose line of sight or have a nontrivial one-way light time. Thus, when designing a networking system in this setting, one needs to consider more general tools. However, many of the most powerful theorems and algorithms that are useful for networking problems such as max-flow min-cut or Dijkstra’s algorithm are built to work with a specific collection of common graph decorations such as directed edges and edge weights. This begs the question: how can one bridge the gap between our system assumptions and the tools which have been successful in past network engineering?

One approach taken by the developers of Contact Graph Routing (CGR) is to construct a graph that represents aspects of our time-evolving network in a way that fits the assumptions of a desired mathematical tool (in this case Dijkstra’s algorithm). This approach can be fruitful but also limited as these representations are not the most natural way to think about time-varying systems. Thus, robust analysis of these networks is limited†. In light of this, we consider a different collection of graph decorations which more naturally model the time-varying nature of our system.

The first tool we consider is a time-varying digraph. These structures are equivalent to dynamic graphs, but we use an alternative definition to Definition 2.1.10 for practical reasons and to simplify the notation within this chapter.

Definition 6.1.1 (Time-varying Digraph). We define a time-varying digraph as a triple $G = (V, E, T)$, where V is a finite vertex set, $E \subseteq V \times V$ is a set of directed edges, and $T = \{T_e : e \in E\}$, where each T_e comes from some *timing set*. In particular, T_e expresses the *availabil-*

*The reader may review [19] for some background in these algorithms.

†The reader may refer to a mathematical analysis of CGR [94].

ity of an edge, namely e is available for all times represented by T_e. Note that we allow loops, that is, edges that connect a vertex to itself.

Definition 6.1.2 (Continuous Time-varying Digraph)**.** A time-varying digraph where T_e is a set of intervals.

Definition 6.1.3 (Discrete Time-varying Digraph)**.** A time-varying digraph where $T_e \subseteq \mathbb{N}$.

Discrete time-varying digraphs, sometimes referred to as time-evolving or temporal digraphs, have been a fruitful subject of study in computer science literature[194 192 42]. However, we found scant literature concerning continuous time-varying digraphs as a subject of study.

Discrete time-varying digraphs

Discrete time-varying digraphs have a discrete set decorating each edge which indicates at which times that edge is available. We first consider a way of describing how a time-varying digraph changes over its discrete time steps.

Definition 6.1.4. Let $G = (V, E, T)$ be a discrete time-varying digraph with times given by T_e for each edge e. Let $\Lambda = \bigcup_{e \in E} T_e$ be ordered so that $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$ where for each i, $\lambda_{i-1} < \lambda_i$. Define the *static expansion* to be the graph H with vertex set V_H and edge set E_H such that:

- If $u \in V$, then $u_i \in V_H$ for each $i \in \Lambda \cup \{\lambda_1 - 1\}$.
- If $e = (u, v) \in E$ and $\lambda_i \in T_e$, then $(u_{\lambda_{i-1}}, v_{\lambda_i}) \in E_H$.

An advantage of this representation is that it allows us to think of the vertices of this expanded digraph as images of our original vertices in time[194].

Dynamic Connectivity

Another way to expand a time-varying digraph into a larger structure is to construct a sequence of graphs. The idea is to represent how the network appears at any interval through discrete steps.

Definition 6.1.5. Let $G = (V, E, T)$ be a discrete time-varying digraph with times given by T_e for each edge e. Let $\Lambda = \bigcup_{e \in E} T_e$ be ordered so that $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$ and $\lambda_{i-1} < \lambda_i$. Define the *digraph sequence* to be the sequence of graphs $\mathcal{G} = \{G_\lambda\}_{\lambda \in \Lambda} = \{(V, E_\lambda)\}_{\lambda \in \Lambda}$ where the edge sets are given by $E_\lambda = \{e \in E \mid \lambda \in T_e\}$.

Note that each digraph in the sequence has the same (identifiable) vertices, but the topology of the digraph is changing. We could decorate each digraph with edge weights, node colorings, etc. However, we do not allow for a change in the number or identity of vertices. The length of this sequence could be finite or infinite depending on the context.

We now will extend the notion of a path, as seen in Definition 2.1.4 For a sequence of digraphs, we can define an s-journey J^s.

Definition 6.1.6 (s-Journey)**.** An s-**journey** is a sequence of vertices, i.e. $J^s = (v_{i_0}, v_{i_1}, \ldots, v_{i_k})$, such that for $1 \leq j \leq k$, $(v_{i_{j-1}}, v_{i_j}) \in E_{s+j-1}$.

With a notion of a journey, we will now define analogs to shortest path length and the graph diameter, as seen in Definitions 2.1.5, 2.1.7. From here, we can define *s-diameter*, which defines a diameter given a particular timestep s.

Definition 6.1.7 (s-diameter)**.** Given a digraph sequence $\mathcal{G}$, let $\mathcal{J}^s$ be the set of all finite s-journeys and $\mathcal{J}^s(i, j)$ be the set of all finite s-journeys where the first vertex is v_i and the last vertex is v_j. We define the shortest journey length $SJ^s(i, j)$ as

$$SJ^s(i, j) = \min_{J^s \in \mathcal{J}^s(i,j)} length(J^s)$$

Therefore, we can now define the s-diameter of $\mathcal{G}$ as

$$diameter^s(\mathcal{G}) = \max_{i,j \in |V|} SJ^s(i, j)$$

Notably, $SJ^s(i, j)$ can be ∞, so the s-diameter can be, too.

We can thus define the following notions of *connectivity*. A graph is *(strongly) connected* if it has finite diameter. In graph theory for directed graphs, it is common to use *strongly connected* if the graph has finite diameter, but we will simply say connected.

A digraph sequence is:

- *δ-disconnected* if the s-diameter is ∞ for all s;
- *δ-weakly connected* if the s-diameter is finite for some s;
- *δ-connected* if the s-diameter is finite for all s;
- and *δ-uniformly connected* if there exists a finite C such that the s-diameter is at most C for all s.

Furthermore, we say a digraph is *non-stranding* if each vertex has at least one outbound edge. We say a digraph is *holding* if each vertex has a loop. We say a digraph sequence is *non-stranding* if all digraphs in the sequence are non-stranding. And we say a digraph sequence is *holding* if all digraphs in the sequence are holding. Finally, we say a digraph sequence is *fixed* if all digraphs in the sequence are the same, i.e. $\mathcal{G} = \{G_k\}$ and $G_k = G$ for all k. We say that G is the *base graph* of a fixed sequence.

First, note that we have harmonized the static and dynamic graph properties.

Proposition 2 (Harmonization of Fixed Sequences). A fixed digraph sequence $\mathcal{G}$ is connected if and only if the base graph G is connected. Moreover, such a connected graph sequence is δ-uniformly connected, and its uniform bound is the diameter d of the base graph. This bound is in fact tight, i.e. the s-diameter is d for all s.

Proof. The main idea is that for a fixed digraph sequence, journeys on the dynamic graph coincide with paths in the base graph, which implies all of our results. To see this, we observe that, for all s, s-journeys in the digraph sequence are paths on the base graph with edge set E. If a sequence of vertices $(v_0, \ldots, v_k)$ is an s_0-journey for arbitrary timestep s_0, then $(v_{i_{j-1}}, v_{i_j}) \in E_{s_0+j-1}$ for all $j \in \{1, \ldots, k\}$ by definition of an s-journey. However, since this is a fixed digraph sequence, we can rewrite this as $(v_{i_{j-1}}, v_{i_j}) \in E$. This is exactly the definition of a path on base graph G. Symmetrically, note that every path on G is a valid s-journey for every timestep s. Because journeys in $\mathcal{G}$ exist for all s if and only if the same path exists in G, then $\mathcal{G}$ is connected if and only if G is connected.

Finally, inspect the s-diameter. Suppose there is a timestep s_0 such that the s_0-diameter is strictly less than d, the diameter of G. Then, there exists an s_0-journey of length less than d between each pair of vertices. But, since each s_0-journey is a path, then there must exist a path between each pair of vertices of length less than d, which would imply that the diameter of G is less than d, which is a contradiction. In the other direction, note that if the diameter of G is d, then there exists a path of length at most d between each pair of vertices. Since every path is an s-journey for every timestep s, then there exists an s-journey of length at most d from each pair of vertices for every s. Therefore, there is a uniform bound d on the s-diameter.

$\square$

Next, we note that holding is an important property that links static and dynamic notions of connectivity.

Proposition 3 (Uniform Connectivity of Connected Holding Sequences). A holding digraph sequence $\mathcal{G}$ where each digraph in the sequence is connected is δ-uniformly connected. In particular, the number of vertices n is a uniform bound and is generically tight.

Proof. Fix a digraph sequence $\mathcal{G}$. We select an arbitrary starting time t_0 and source vertex v_s. We will show that there exists an t_0-journey between v_s and every other vertex in our finite vertex set V and each journey has length at most n where $n = |V|$.

To accomplish this, we will show three properties of so-called "reachability sets" of the vertex: weak monotonicity, complementary inclusion, and then strong monotonicity. To start, note that for a finite journey of length i, we can write the journey as

$$J = (v_{t_0}, v_{t_1}, \dots, v_{t_i})$$

We say this journey "reaches" or "ends at" vertex v_{t_i} and "starts at" vertex v_{t_0}. A *reachability set* is a set $U_i \subseteq V$ of vertices that can be reached in an t_0-journey of exactly length i such that the journey starts at v_s. We let $U_0 = \{v_s\}$.

We will first show that for a holding digraph sequence, the sequence of U_i satisfy weak monotonicity, in that

$$U_0 \subseteq U_1 \subseteq \cdots \subseteq U_n \subseteq \cdots$$

Note that with a holding digraph sequence, U_1 is non-empty, in that it at least includes v_s. Therefore, suppose that vertex $v_u \in U_k$ for some k. Then, $v_u \in U_{k+1}$ since $(v_u, v_u) \in E_{k+s}$ by the holding property. In other words, we can extend a length k journey that reaches v_u to a length $k+1$ journey by appending v_u.

Next, we show that for any digraph sequence of connected digraphs, we have complementary inclusion. In other words, if $V \setminus U_k$ is non-empty, then there exists some vertex $v_c \in U_{k+1}$ such that $v_c \in V \setminus U_k$. First, suppose $V \setminus U_k$ is non-empty. Because we assumed a holding digraph, U_k must be non-empty, so there must be at least one vertex in this set. Finally, note that because our graphs are connected, there must exist some edge $(v_{c-1}, v_c) \in E_k$ where $v_{c-1} \in U_k$ and $v_c \in V \setminus U_k$. If this were not true, then U_k and $V \setminus U_k$ would be a partition of the vertices of the graph G_k without a path between them, which is a contradiction.

Finally, we combine these properties to get strong monotonicity: either $U_k \subset U_{k+1}$ or $U_k = V$. Moreover, if $U_k \subset U_{k+1}$, then $|U_k| + 1 \leq |U_{k+1}|$. In other words, the chain of reachability sets must increase in size by at least 1. Thus, we can conclude that $U_{n-1} = V$, which gives us a uniform bound n. This bound is generically tight, e.g. an unchanging dynamic network of a base graph that is a cycle with all self-loops achieves this bound. □

Finally, we provide a simple result that shows that the dynamic diameter must change incrementally, which establishes another bound on how dynamic networks evolve.

Proposition 4 (Non-Stranding Bound on s-diameter). A non-stranding digraph sequence $\mathcal{G}$ with finite s-diameter has finite t-diameter for all $t \leq s$. Moreover, if the s-diameter is some finite value c, then the $(s-1)$-diameter is at most $c+1$ and this bound is tight. Finally, note that if the s-diameter is infinite, then the t-diameter is infinite for all $t \geq s$.

Proof. Our main observation is that non-stranding sequences never disconnect paths.

Let us first begin by assuming that the s-diameter is some value c. Take two vertices u, v. Starting at time $s - 1$, we know that u must have one outbound edge, as the sequence is non-stranding; call this edge (u, w). By the definition of s-diameter, we know the shortest journey length from w to v starting at time s is at most c. Therefore, the $(s - 1)$-diameter is at most $c + 1$, which is a relatively conservative bound, though we will not examine the tightness claim in detail.

From here, we can prove the rest of our claim through two immediate observations: first, since s is just some finite number, the 1-diameter is bounded from above by $c + s$, which is certainly finite. Finally, we note that the last statement is simply the contrapositive of our claim.

□

6.1.2. Summary Graphs

The time-varying nature of space networks makes it challenging to use traditional graph theory and graph analysis to model delay tolerant networks. Static graphs are simple methods of conceptualizing networks, but often do not fully represent the heterogeneity of delay tolerant networks. Static graphs lack the nuance to convey edges that come in and out, extensive traversal time, and lack of end-to-end connectivity that are characteristic of space networks. On the other hand, while temporal graphs better encapsulate space networks, they are far more complicated to work with and make graph analysis more challenging.

In this subsection, we propose a novel concept of graph summarization. In essence, we convert a complex, dense data structure (that of a digraph sequence) into a compressed representation of a single weighted digraph. Summarization captures essential characteristics of a digraph sequence, without the overwhelming storage cost of maintaining the full sequence. In addition, summarization can extract salient information that may not be so obvious from the full sequence.

We will loosely use the term “summary graph” to refer to a weighted digraph that arises from a "summarization" of a graph sequence. As from before, we assume the definition of a digraph and that of a discrete-time digraph sequence. More precisely, the basic precept of creating a graph summary is to find a map f that maps a graph sequence $\mathcal{G}$ to a single, static graph G.

Edge Weighting and Completeness

We can also define the concept of an "edge weight" that assigns a *weight* to each edge. More precisely, we write a digraph $G = (V, E, w)$, where

$$w : E \to \mathbb{R}_{\geq 0} \cup \{+\infty\}.$$

This weight could represent a variety of things for a static (di)graph. Introductory examples of edge-weighting techniques include, but are not limited to:

1. The amount of time required to traverse the edge
2. The capacity of the edge
3. The cost of traversing the edge
4. The percentage of time in which the edge is disconnected in a given time interval
5. The percentage of time in which the edge is connected in a given time interval

Notably, we have not placed any conditions so far on the weight function, other than that it be a non-negative real number or positive infinity. By convention, though, depending on what the edge weight represents, we may "extend" the domain of the weight function to be defined over all pairings $V \times V$ and assign any pair $(i, j) \notin E$ the weight 0 or $+\infty$ depending on context. In this sense, we will consider each (di)graph in a sequence to be a complete (di)graph to make certain definitions easier to wrangle.

Finally, we must harmonize the definition of the weight function with a digraph sequence, as it is not immediately obvious how this definition would extend to a graph sequence. We define it as follows

$$\mathcal{G} = (G_t = (V, E_t)_{t \in \mathbb{T}}, w)$$

where

$$w : \mathbb{V} \times \mathbb{V} \times \mathbb{T} \to \mathbb{R}_{\geq 0}.$$

In other words, the weight function maps from a pair of vertices and the time-indexing set to a real-number. In our notation, we will write $w_e : \mathbb{T} \to \mathbb{R}_{\geq 0}$ to refer to the weight function of a particular pair of vertices $e = (i, j)$. We also impose one additional condition: that w_e be integrable with respect to the standard measure of the time-indexing set.

Types of Dynamic Networks

Our definitions operate well with different notions of time-indexing and we can define four different types of graph sequences:

1. An infinite, discrete-time graph sequence: $\mathbb{T} = \mathbb{Z}_{\geq 0}$.
2. An infinite, continuous-time graph sequence: $\mathbb{T} = \mathbb{R}_{\geq 0}$.
3. A finite, discrete-time graph sequence: $\mathbb{T} = \{0, \dots, n\} \subset \mathbb{Z}_{\geq 0}$ for some $n \in \mathbb{Z}_{\geq 0}$.
4. A finite, continuous-time graph sequence: $\mathbb{T} = [0, T] \subset \mathbb{R}_{\geq 0}$ for some $T \in \mathbb{R}_{\geq 0}$.

Graph Summarizations from Different Sources

One class of summarization comes from **time-based summary statistics**, which do not focus on the particular weights under a weight function, but rather report the underlying time of occurrence for various events. In other words, the weights of the summary graph from the time indexing set, rather than from the domain of the original weight function. Some examples include:

- percent of time where the connection is present within the discrete time interval

- average length of time that a connection occurs for within the discrete time interval, in this case one day
- longest length of time that a connection occurs for within the discrete time interval

Alternatively, we could perform summarization with **data-based summary statistics**, which draw values from the original weight function of the sequence. Some examples include:

- Maximum amount of data packets that can be transmitted along an edge within the discrete time interval
- Maximum amount of data packets that can be transmitted along one connection within the discrete time interval
- Average amount of data packets that can be transmitted along one instance of a connection

Centrality Measures

In static graphs, there are similar notions of summarization, one popular form being **centrality measures** that characterize the influence of particular nodes or edges within a graph. We provide a brief overview of those notions of centrality here.

- **Katz Centrality**: it measures the relative influence of a node in a network by taking into account the number of walks between pairs of nodes in a network[295]. Katz centrality first measures the number of immediate neighbors of a node, and then the other nodes that can be connected from the immediate neighbors. Each connection is assigned a weight based on an attenuation factor a and a distance associated with the attenuation factor. The Katz centrality of node i:

$$C_i = \sum_{k=1}^{\infty} \sum_{j=1}^{n} a^k (A^k)_{ji} \tag{6.1}$$

where A^k is the adjacency matrix (cf. Definition 2.1.6). By the structure of A, each entry of A_k represents the number of paths of length k between two vertices. A more central node according to Katz centrality is a node that not only has multiple connections, but whose neighboring connections are close by. This technique functions well when we might want to use Katz centrality in the computation of shortest path (or at least the idea of it).

- **Eigenvector Centrality**: it assigns scores to nodes based on the relative scores of their connected nodes. Thus the eigenvector centrality of a node is the weighted average of the centrality values of its connected neighbors[296]. In particular, the weight $\boldsymbol{x} \in \mathbb{R}^{|V|}_{\geq 0}$ is a vector such that

$$x_u = \frac{1}{\lambda} \sum_{v \in N(u)} x_v \tag{6.2}$$

  where $N(v)$ is the neighborhood (cf. Definition 2.1.2) of v and x_v is the relatively centrality of the vertex v. Because of the structure of the adjacency matrix A, we can write

$$x_u = \frac{1}{\lambda} \sum_{v} a_{uv} \cdot x_v \tag{6.3}$$

$$A\boldsymbol{x} = \lambda \boldsymbol{x} \tag{6.4}$$

  This last form gives rise to the term *eigenvector centrality*; for undirected graphs, the adjacency matrix is symmetric, so by the Spectral theorem and because we require $\boldsymbol{x}$ to be non-negative, by the Perron–Frobenius theorem, we are guaranteed the existence and uniqueness of a satisfying eigenvalue λ (i.e. the greatest eigenvalue of A) and desired associated eigenvector $\boldsymbol{x}$, which is exactly our centrality scores.

- **Degree Centrality**: measures the number of connections a node makes with its neighbors.

- **Betweenness Centrality**: a measure of how many times a node lies on the shortest path between any pair of source and sink nodes.

Example: Instant-Graph Path Summarization with Cost

This method is mainly defined for continuous-time graph sequences, though we will also define appropriate notions for the discrete-time case, as well. We call a graph G_t at a particular timestep an *instant graph at instance* t. This method focuses on paths defined on weighted instant graphs where the weights represent the cost of traversing an edge.

We will use the definition of a path introduced as Definition 2.1.4. For the purposes of this subsection, we will assume that each graph in our sequence is complete; if we had wanted to exclude an edge, we simply assign the instant cost of those edges to be $+\infty$. Additionally, we assume that the weights are strictly positive at any instant. The instant weight of a path is the sum of the instant weight of the edges; if any edge has an instant weight of $+\infty$, then the instant weight of the path is $+\infty$.

Therefore, given a vertex set, we can define the set of all possible finite paths from one vertex to the next as $\mathcal{P}(i,j)$ and the set of all possible finite paths as $\mathcal{P}$. From here, given a graph sequence, we can define the instant set of shortest paths as a set $\mathrm{SP}^t(i,j) \subseteq \mathcal{P}(i,j)$ such that a path is an instant shortest path if:

1. its cost is not $+\infty$ at instance t
2. its cost is not greater than the cost of any other path in $\mathcal{P}(i,j)$ at instance t

Finally, we say that the set $\mathrm{SP}^t = \bigcup_{(i,j)\in V\times V} \mathrm{SP}^t(i,j)$

From here, we define *shortest path participation* ω of a path P for a continuous-time sequence as

$$\omega(P) = \frac{1}{T}\int_0^T \mathbf{1}_P(t)\,\mathrm{d}t \qquad \text{for a finite sequence}$$

$$\omega(P) = \lim_{\tau\to\infty}\frac{1}{\tau}\int_0^\tau \mathbf{1}_P(t)\,\mathrm{d}t \qquad \text{for an infinite sequence}$$

and for discrete-time sequence as

$$\omega(P) = \frac{1}{N+1}\sum_{t=0}^{N} \mathbf{1}_P(t) \qquad \text{for a finite sequence}$$

$$\omega(P) = \lim_{N\to\infty} \frac{1}{N+1}\sum_{t=0}^{N} \mathbf{1}_P(t) \qquad \text{for an infinite sequence}$$

where $\mathbf{1}_P(t)$ is an indicator function for $P \in \mathrm{SP}^t$.

One thing to note: each path must have a start edge and end edge; each edge starts and ends at a particular vertex. Therefore, it is well-defined to say that a path starts and ends at a particular vertex. Thus, in our definitions of w, we could have restricted the indicator function to look at the the shortest path set that originates and terminates at the appropriate vertices; this would not change the calculation.

In all four cases, we are suggesting a simple idea: take the proportion of time that each path acts as a shortest path. Each path as a *shortest path participation* between 0 and 1. If the graph becomes disconnected, the shortest path set could be empty, which works well with our definitions. We call this "instant-graph summarization" because we consider traversal for each instant graph separately, without really taking into account traversal across time.

Finally, we can define the summary graph as a complete graph with edge weights to be the *shortest path participation* with weight function $\bar{\omega}$, where

$$\bar{\omega}(e) = \sum_{P\in\mathcal{P}:e\in P} \omega(P)$$

Proposition 5 (Bounds on $\bar{\omega}, \omega$). $\bar{\omega}$ is non-negative and finite. ω is non-negative and at most 1.

Proof. ω is non-negative, as it is either the integral over or the sum over a non-negative function, namely an indicator function. $\bar{\omega}$ is thus non-negative, as it is the finite sum of non-negative values.

Bound from above, we see that ω is at most 1 because the indicator function $\mathbf{1}_P(t) \leq 1$.

For finite-time, we see that

$$\begin{aligned}\frac{1}{N+1}\sum_{t=0}^{N}\mathbf{1}_P(t) &\leq \frac{1}{N+1}\sum_{t=0}^{N}1\\ &= \frac{1}{N+1}(N+1)\\ &= 1\end{aligned}$$

and that

$$\begin{aligned}\frac{1}{T}\int_0^T \mathbf{1}_P(t)\,\mathrm{d}t &\leq \frac{1}{T}\int_0^T 1\,\mathrm{d}t\\ &= \frac{1}{T}(T)\\ &= 1\end{aligned}$$

To show this for our infinite time definitions, we see that

$$\int_0^\tau \mathbf{1}_P(t)dt \leq \int_0^\tau 1dt$$

and

$$\sum_{t=0}^{N}\mathbf{1}_P(t) \leq \sum_{t=0}^{N}1$$

respectively for all τ and all N. Solving for the right-hand side of both inequalities, we see that

$$\int_0^\tau \mathbf{1}dt = \tau$$

and that

$$\sum_{t=0}^{N}\mathbf{1} = N+1$$

respectively. Finally, when taking the limit of

$$\lim_{\tau\to\infty}\frac{1}{\tau}\tau$$

and of

$$\lim_{N\to\infty} \frac{1}{N+1} N + 1$$

we yield 1 in both cases. Therefore, we know that ω for any path is at most 1.

Finally, we must show that $\bar{\omega}$ is finite for edges. However, this follows easily from the fact that the number of finite paths without repeated edges (i.e. without cycles) in a finite, simple graph is itself finite. Therefore, $\bar{\omega}$ is summing a finite number of values that are at most 1 and so is finite. □

Example: Traversal-Time DiGraph Journey Summarization with Traversal Time

We already defined a few simple methods of attributing edge weights above. In this subsection we walk through a more complex example of defining edge weights.

Here, we assume that the weight function on the digraph sequence defines the "traversal time" of a particular edge, i.e. how many units of time are required to traverse the edge. For now, we only consider continuous-time sequences, but our definitions could be easily adapted to the discrete-time case with a bit of care. Here, by convention, traversal time is $+\infty$ if there is no edge; therefore, we can assume our digraph sequence is complete.

We define the *velocity* as $\nu_e(t) = \frac{1}{w_e(t)}$. We denote the velocity to be $+\infty$ if $w = 0$ and $\nu = 0$ if $w = +\infty$. Therefore, we can define the notion of s-traversal time of an edge, ${}^{\mathsf{s}}(\mathsf{e})$. Define the s-traversal completion set[*]:

$$\mathrm{tcs}^s(e) = \left\{ x : \int_s^x \nu_e(t)\,\mathrm{d}t \geq 1 \right\} \tag{6.5}$$

with the convention that $\int_y^y (+\infty)\,\mathrm{d}t = +\infty$. If $\mathrm{tcs}^s(e)$ is empty, then ${}^{\mathsf{s}}(\mathsf{e}) = +\infty$; otherwise,

$$\mathrm{tt}^s(e) = \inf\{\mathrm{tcs}^s(e)\} - s. \tag{6.6}$$

[*]Note that computing this value in practice is tractable, but requires some optimizations to do efficiently. We do not compute the entire set, but instead directly compute tt. See Appendix A.3.2 for more information on how this is computed, specifically Algorithm 3.

What does this definition accomplish? It captures the amount of time (in some time unit) it takes to traverse one whole edge given that the traversal velocity is given by ν. We define it through the $\inf$ of a set to handle some analytic issues with $+\infty$. As a helper, we define the s-traversal end time of an edge $\mathrm{tet}^s(e) = \inf\{\mathrm{tcs}^s(e)\}$, which is defined to be $+\infty$ if $\mathrm{tcs}^s(e)$ is empty. We will, therefore, define the s-traversal time of a path (using all of the previous definitions of a path) with the following recursion:

$$\begin{aligned}
\mathrm{tt}^s(P)_1 &= \mathrm{tt}^s(e_1) \\
\mathrm{tet}^s(P)_1 &= \mathrm{tet}^s(e_1) \\
\mathrm{tt}^s(P)_2 &= \mathrm{tt}^{\mathrm{tet}^s(P)_1}(e_2) \\
\mathrm{tet}^s(P)_2 &= \mathrm{tet}^s(P)_1 + \mathrm{tt}^s(P)_2 \\
&\vdots \\
\mathrm{tt}^s(P)_k &= \mathrm{tt}^{\mathrm{tet}^s(P)_{k-1}}(e_k) \\
\mathrm{tet}^s(P)_k &= \mathrm{tet}^s(P)_{k-1} + \mathrm{tt}^s(P)_k
\end{aligned}$$

with the convention that if tt or tet reaches $+\infty$ at any step, then so do the rest. We can write $\mathrm{tt}^s(P) = \mathrm{tt}^s(P)_k$.

Thus, we can define the notion of a shortest path $\mathrm{SP}^t(i, j)$ as the set of shortest paths minimum cost with respect to s-traversal time and that have s-traversal time. This set could be empty. We can, therefore, use the same definitions as before of ω, i.e. shortest path participation.

As such, we have developed a model for defining summarization functions along edge weights, where edge weights can represent higher-level changes about the network structure and connectivity. In instances where the complexity of time-varying graphs proves too computationally heavy for graph analysis, summary graphs could serve as a bridge from static graphs to temporal graphs.

6.1.3. Algebraic and Applied Topology

Zigzag Persistence

Persistent homology is a powerful tool in applied topology, of which we have seen some constructions in Section 5.1 [81]. Zigzag persistent homology allows us to track network connectivity changes [80,41]. Zigzag persistent homology is based off of standard persistent homology, a tool from algebraic topology that is used to study features of topological spaces. The main differences between zigzag and standard persistence stems from the inclusion maps; in zigzag persistence, the inclusion maps are allowed to go either direction whereas in standard persistence, they are restricted to a single direction. This difference allows us to consider changes in connectivity as time elapses.

Zigzag persistence is one technique that can assist in analyzing a dynamic network. In particular, zigzag persistence allows us to extra multiscale information from dynamic networks, respecting both spatial edges and temporal edges between nodes. Calculating the zigzag persistence summarizes the connectivity information of the network. We introduce zigzag persistence here and then briefly demonstrate an example.

Let X_i be a topological space. We define a *zigzag sequence of topological spaces* as

$$X_1 \leftrightarrow X_2 \leftrightarrow \ldots \leftrightarrow X_n$$

where $\leftrightarrow$ is an inclusion map that either maps $X_i \;\rightarrow\; X_{i+1}$ or $X_i \;\leftarrow\; X_{i+1}$. Recall, the homology functor (with coefficients in a field) is a functor that maps topological spaces to vector spaces. Hence, the *zigzag module* of a zigzag sequence of topological spaces is

$$H_p(X_1) \leftrightarrow H_p(X_2) \leftrightarrow \ldots \leftrightarrow H_p(X_n).$$

We can use zigzag persistence (with $\mathbb{Z}/2\mathbb{Z}$ coefficients) to understand the structure and features of a topological space. For example, $H_1(X_i)$ describes how many 1-dimensional holes are in X_i. In the example below, we see that a 1-dimensional hole is born at $t = 2$ and dies at $t = 3$.

Leveraging our definition of a time-varying network, the most natural way to model a communication network is to treat satellites, ground stations, and other members of the network as the vertices and treat contacts between members of the network as edges. Thus, for any G_t, this forms a simplicial complex. From this simplicial complex, for a given time interval $[1, n]$, let N be the zigzag sequence

$$G_1(V, E_1) \leftrightarrow G_2(V, E_2) \leftrightarrow \ldots \leftrightarrow G_n(V, E_n).$$

Note, each arrow represents the inclusion map between two graphs. This arrow will change direction based on the direction of the inclusion. Then, we can apply the homology functor to our zigzag sequence to get

$$H_p(G_1(V, E_1)) \leftrightarrow H_p(G_2(V, E_2)) \leftrightarrow \ldots \leftrightarrow H_p(G_n(V, E_n)).$$

Therefore, we can calculate the p^{th} homology for our zigzag sequence of graphs. Note, for $p \geq 2$, the p^{th} homology of a graph will be zero, so we are specifically interested in $p = 0, 1$.

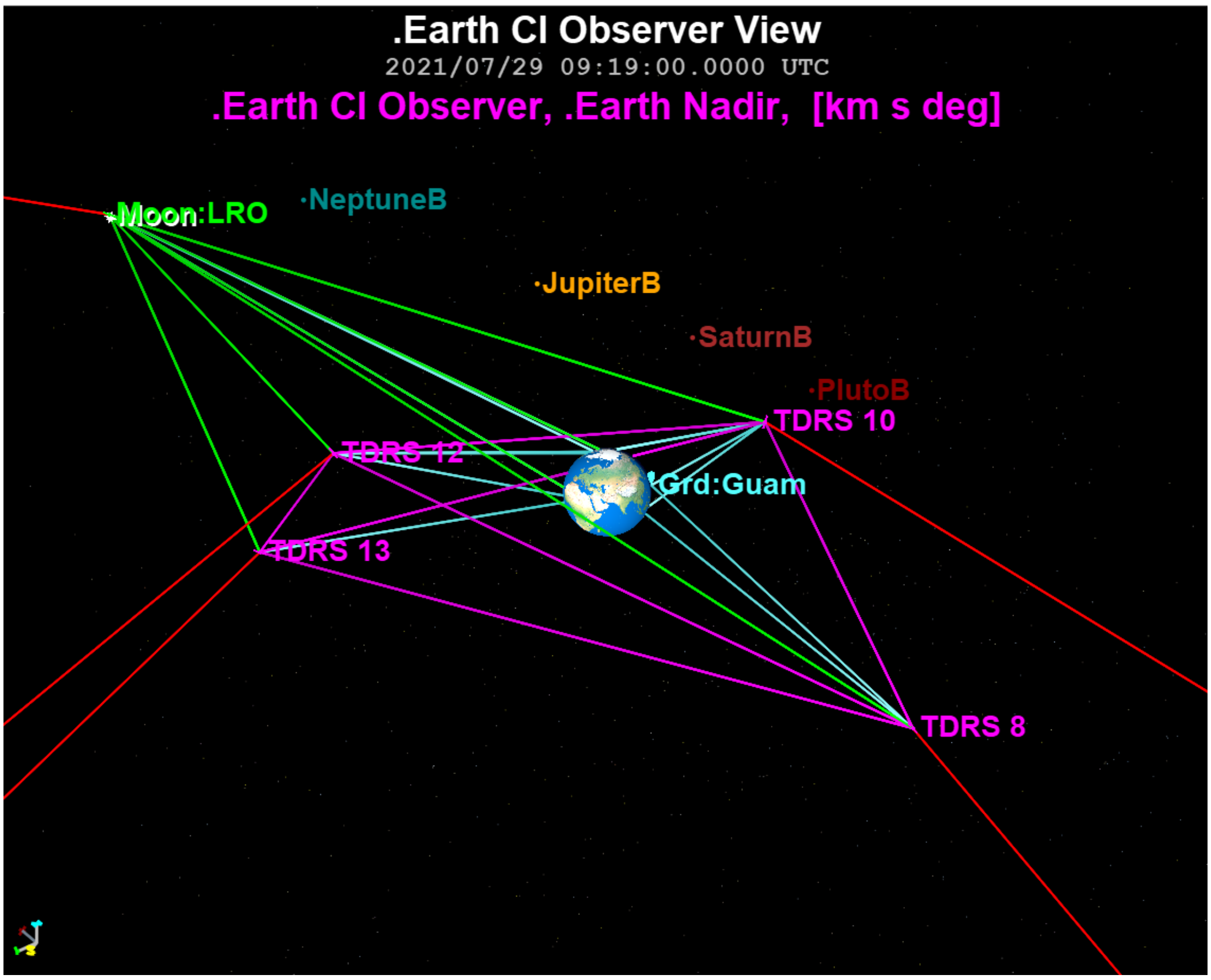

Figure 6.1: A picture of an example satellite network from a simulation tool used at NASA. Note, connections are denoted by a line between two devices.

The 0-dimensional homology tells us how often we have a communication device that is disconnected from the network. Further, the 1-dimensional homology captures when all communication devices are connected. This information can be used to group satellites by either continuous connectivity or by location. We perform this analysis on the system displayed in Figure 6.1, which features the objects and ground stations at MRO, LRO, TDRS 8, TDRS 10, TDRS 12, TDRS 13, Canberra, Madrid, Goldstone, White Sands, and Guam.

Intuitively, it might make sense to group the four TDRS satellites together, as they are always connected. Alternatively, we could group each TDRS with the ground station it is closest to. There might be additional groupings not listed, but zigzag persistence might provide insight as to what those groupings are. Over the course of 1 day, the zigzag persistence diagram has 110 bars which captures the connectivity of the network for that day. Of note, there are eight 1-dimensional holes that live for the entire day. Hence, this tells us that some subsection of satellites are always connected. We can use this information to construct a new network where those satellites are represented by a single node. Further work will be done to test the different subnetworking options by recalculating the zigzag persistence of the new network model and computing the bottleneck distance between the two networks.

6.1.4. Conclusion and Future Work

Here, we have presented several topics that we hope can contribute to a firmer mathematical foundation for Delay Tolerant Networking. Here we summarize the different approaches and how they relate to foundational networking aspects. Then, we dive deeper into how this research may continue with a future works portion for each topic.

Connections across Mathematics

Much like the layers of the Open Systems Interconnection (OSI) model represent different layers of abstraction, our mathematical models represent different layers of abstraction in network understanding. At its core, a network is built on the connections present, and we present two different methods for modeling connectivity in a time-evolving network. However, this work also sits in the milieu of a much broader context in this area, which we also indicate here. All of our interesting approaches come from graph theory, but each presents its own approach with corresponding pros and cons.

- Temporal graph theory provides precise connection information for time-evolving networks with a variety of representations. Different representations can be better or worse as networks grow in size.
- Directed multigraphs are a fruitful data structure for routing algorithms, as demonstrated by the updated pathfinding algorithm for CGR presented elsewhere.

- Summary graphs provide useful statistics for comparing connection availability, and may provide bridges from temporal to more traditional graph theoretic results using centrality measures. However, summary graphs represent a potential for information loss, such as ordering of intervals, that could be extremely relevant for different applications.

Once connectivity is established, being able to analyze connectivity structures and discover strong subnetworking options is our next layer up. While small networks might be better understood directly, part of the goal in introducing algebraic objects is to provide support in returns to scale. Invariants from applied algebraic geometry and topology can be brought to bear on networks of any size.

- Homology algebraizes connectivity information from a graph enabling computer legibility and returns to scale[136].

- Zigzag persistent homology provides a means of tracking continuous chunks of connectivity over time, including joining and separating connections. This provides a strong foundation for automated subnetworking decisions[80].

- Graph varieties provide a different algebraic interpretation of connectivity information which enables more direct relations to more powerful data structures, such as sheaves[63].

Once structures are established and analyzed, optimization is the next layer up. Game theory provides an interesting framework for constructing and comparing network optimization approaches that may be helpful in addressing congestion in delay tolerant networks.

Temporal Graph Theory

Temporal graph theory is a still young field. While we introduce some of our work in formalizing temporal graphs in subsection 6.1.1, there is much more to describe and prove about temporal graphs and graph sequences in forthcoming papers. This includes studying various properties of temporal graphs and determining extensions of network properties to temporal networks. One direction of exploration in temporal graph theory that could be particularly fruitful for space networks is the study of periodic temporal networks. The fact that orbits are often naturally periodic could be leveraged in the study of periodic temporal graphs and leveraged for the benefit of network analysis.

Another key thing to note is a shift in information available in a temporal graph. In the cases we are considering in this section, the graphs are pre-determined and known, so global statistics can be computed. However, in practice we may only know the current configuration of a network without knowing its future configurations – perhaps relying on probability to provide possible futures. A temporal graph construction that views temporal graphs as a dynamical system rather than a fixed object, would be extremely valuable to the field. This would also be remarkably applicable to future space networks.

Another consideration is how local perspectives influence the system. Note that from the vantage point of any given node at any given time, its imagining of the network will differ from every other node. As such, constructing network views locally will have great influence on consistency in routing across different components. It is imperative that tools, such as sheaves, are utilized to synthesize and validate consistency of data across temporal networks. This may be useful in the probabilistic or deterministic settings.

Directed Multigraphs

While not described in detail in this work, one method for modeling time-evolving networks using directed multigraphs, and provided an alternate approach to pathfinding in Contact Graph Routing based on these multigraphs. Upcoming work in[205] will provide more of the details of this alternate algorithm, including a proof that it requires less than or equal to the number of iterations taken by the previous Contact Graph Dijkstra Search. It will also include experiments on simple networks simulated in SOAP that compare the two algorithms.

Future work in this area will likely continue to reformulate CGR in the language of directed multigraphs. As mentioned above, one opportunity for this change in perspective may be in the implementation of Yen's algorithm, which uses Dijkstra's algorithm repeatedly.

While the existing version of CGR uses a version of Yen's algorithm for contact graphs, it is likely this could also be put in the framework of the multigraphs considered here. We hope that reformulating CGR in the language of multigraphs will not only increase the speed of computations but also lead to a more transparent approach to routing and serve as a foundation for future algorithms.

Centrality Measures and Summary Graphs

Summary graphs, which we exposit in Section 6.1.2, represent an interesting way to collect summary statistics for time-varying graphs. Determining which statistics are worth representing in this structure is a significant future project. Since summary graphs represent a kind of lossy compression of time-varying graph structures, knowing which statistics are preserving valuable information from the time-varying graphs is valuable outright. Moreover, if we want to feed this information into a machine learning algorithm for optimization, summary graphs seem primed for supplying concise yet relevant information.

Another interesting application of summary graphs comes in the form of network centrality measures. Typical network centrality measures are better suited to static graphs rather than dynamic graphs. So, applying network centrality measures to the summary graph can yield interesting ways to detect central nodes in a dynamic graph. Also, different summary statistics will likely correspond to different rankings for the same centrality measures. Such a comparison would be of great interest to us.

There are also adaptations of network centrality measures for time-varying networks already. It would be interesting to compare the results of these dynamic centrality measures with the same measures applied to different summary graphs. Certainly, this could detect some of the information loss from compressing to the summary graph.

Zigzag Persistent Homology and Applied Topology

Zigzag persistence is an important topological tool and we provide some foundational information and example in Section 6.1.3. One further extension for exploration would be leveraging the bottleneck distance to compare the persistence diagram of a subnetwork with that of the larger, original network[80]. The bottleneck distance can be described in the following way:

Definition 6.1.8 (Bottleneck Distance)**.** Given two finite diagrams $D_1, D_2 \subset \mathbb{R}^2$, we define a *matching* as a bijection m such that $m : D_1 \cup L \to D_2 \cup L$, where $L \triangleq \{(x_1, x_2) \in \mathbb{R}^2 : x_1 = x_2\}$. (n.b. strictly speaking, diagrams are multisets, but we can assume that all points in our diagrams are distinct, since this does not alter the rest of our definition.) The cost c of a matching is

$$c(m) \triangleq \sup_{x \in D_1} \max\left(|x_1 - m(x)_1|, |x_2 - m(x)_2|\right)$$

Finally, we can say the *bottleneck distance* d between two diagrams is defined as

$$d(D_1, D_2) \triangleq \inf_{m \in M} c(m)$$

where M is the set of all matchings between the two diagrams.

The bottleneck distance is indeed a proper metric[80]. This metric finds the lowest-cost perfect matching between two persistence diagrams using the $\sup$ norm between points. It accounts for the fact that two diagrams may have a different number of points by augmenting the matched sets with all points on the diagonal. There are technical reasons why the points on the diagonal are the "correct" default point to match to, but this information is beyond the scope of this dissertation and is not salient. Equipped with this metric, we can thus compare persistence diagrams of subnetworks with each other and with the persistence diagram for the whole network.

Another way we hope to apply zigzag persistence to space networking is to apply it within the context of CGR. After constructing the subnetworks, one can construct a contact plan with this new structure. This reduces the number of vertices which in turn reduces the computation time. Again, we can use the bottleneck distance to compare the two networks and to see if this small change does not drastically change the underlying structure of the network. We hope to use this as a way to justify subnetwork construction and show the feasibility of the groupings.

The final extension we would like to explore is looking at the clique complex of the network as opposed to the simplicial complex associated to the network. A collection of vertices with all pairwise connections is a simplex in a clique complex; if we have three points connected, we fill in the triangular face, if we have four points connected to each other, we fill in the tetrahedron. This provides more information about the level of connectivity and the interaction between multiple devices.

Graph Varieties and Applied Algebraic Geometry

The theory of graph varieties allows us to model a static network as an algebraic variety. Space networks, however, are not static. Thus, it is necessary to formulate a theory of graph varieties that can be applied to temporal networks. A naïve first approach to such a theory is to view a temporal network as a sequence of graphs and simply compute the corresponding sequence of graph varieties, which as discussed in Section 1.2, does not work in general. One possible solution is to leverage cellular sheaves, which are a powerful tool for analyzing networks[63]. Subsequently, an algebro-geometric analogue of the Dijkstra sheaf would, in theory, give insight into the problem of finding shortest paths in a temporal network[204]. Dijkstra sheaves are a specific type of sheaf that corresponds this algebraic topological structure with Dijkstra's shortest-path algorithm on static networks. It is known that Dijkstra's algorithm already works on temporal networks with sufficient regularity conditions[69]; combining this result with Dijkstra sheaves would potentially deliver a "temporal Dijkstra sheaf" that could provide insight into the structure of dynamic networks.

Game Theoretic Networking

As space networks become increasingly complex, requiring global communication for decision-making is likely impossible given propagation delays that outlast windows of opportunity for real-time feedback. It is clear that new modeling techniques for routing decisions are essential for further space exploration. Game theory provides a well-developed foundation for understanding how decision-making might function in a network with incomplete information.

It is, however, likely that there are subnetworks that can share enough data to be cooperative although between two such subnetworks cooperation is infeasible. Such mixed approaches have been used to study Wi-Fi congestion in apartment buildings and could be generalized and reapplied to large-scale DTNs[286]. It should be investigated if the output could be used for real-time decision making for routing decisions, for example for load balancing.

6.2. Routing Problems and Dynamic Graphs

In this section, we consider direct applications of our foundational dynamic graph theory to Delay Tolerant Networking (DTN). We introduce this important application area, provide its appropriate mathematical context, and then provide a few code-driven applications (e.g. an FPGA-based project in tropical geometry and parameterized graphs). Through the lens of dynamic networks, we see not only how we can solve problems in networking with dynamic graphs.

6.2.1. Background

Delay Tolerant Networking (DTN) is the standard approach to the networking of space systems with the goal of supporting the Solar System Internet (SSI). Current space networks have a small scale and often depend on rigorously scheduled (pre-determined) contact opportunities; this manual approach inhibits scalability. The goal of this section is to recast these scheduling problems in order to apply the optimization machinery of tropical geometry.

Contact opportunities in space are dependent on such factors as orbital mechanics and asset availability, which induce time-varying connectivity; indeed, end-to-end connectivity might never occur. Routing optimization within this structure is classically difficult and typically utilizes Dijkstra's algorithm as applied to contact graphs. Alternatively, we follow the successes of tropical geometry in train schedule optimization, job assignments, and even traditional networking, by extending this approach to this more general (i.e. disconnected) problem space.

These successes imply tropical geometry provides a useful framework in the context of DTNs, starting with applications to queuing theory and long-haul links. Recently, tropical geometry has been applied to parametric path optimization on graphs with variable edge weights. In this work, we extend these advances to account for the problem of routing in a space network, and find that tropical geometry is well-suited to the challenges offered by this new setting, including contact schedules featuring probabilities. Our approach leverages the combinatorial nature of the problem to give feasible shortest path trees in the presence of variable channel conditions and latency, evolving topologies, and uncertainty inherent in space routing.

We discuss our tropical approach to DTN for two Python implementations, a Verilog Tropical ALU implementation, tropical frameworks for other parametric graph problems, and solution stability. Lastly, a program for future work is included to illuminate the path ahead.

6.2.2. Introduction

One major complication associated with space networking is that the links between assets are constantly changing. Both in terms of links coming up and down, but also in terms of channel characteristics. One such characteristic is latency, which is the time it takes for information to leave the transmitter and arrive at the receiver. In space, this latency mainly arises from light travel time, which can be significant (as many as tens of minutes between Earth and Mars).

In addition, the main form of transmissions being electromagnetic waves means that a space network will suffer due to the inverse-square law. This law basically dictates that the power received is inversely proportional to the square of the distance to the transmitter. From an information theoretic perspective, this severely limits the data rate one could hope to achieve based on the physics of light alone, even ignoring any complications with modulations or electronics. Note that both latency and received power depend on distance, a quantity which is constantly changing for assets in space. subsection 6.2.3 details an algorithm and two implementations that account for these limitations by casting them as a parametric shortest path problem.

Below, in Figure 6.2, we see a small example of a space network between the Earth and five satellites. Despite the relatively 'small' size - there are only six assets - the graph is highly dependent on orbital mechanics and hence time. This emphasizes the need for rigorous approaches to parametric graphs in space communications.

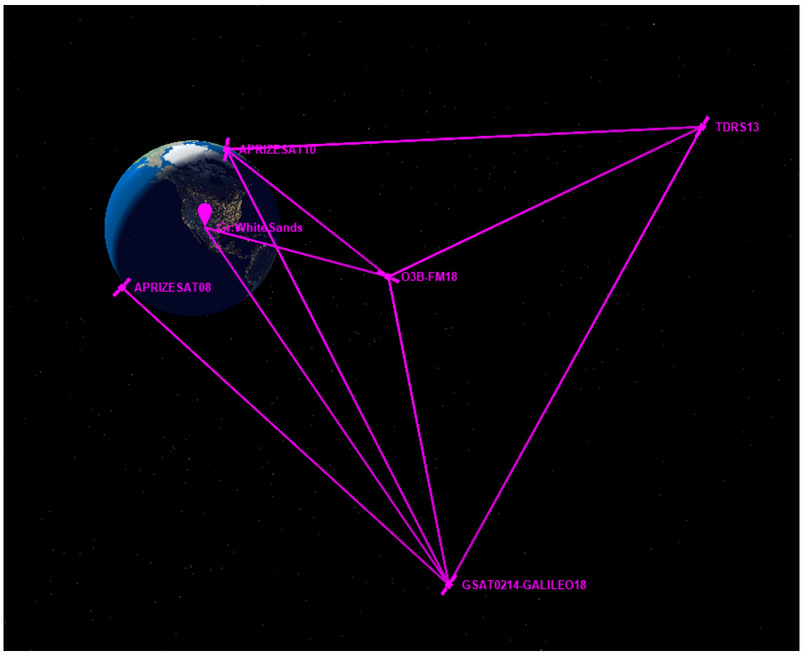

Figure 6.2: An example space network that connects a system of satellites with communication nodes to Earth.

Parametric graphs are modeled as graphs with variable weights, see Definition 6.2.1. These weights could correspond to distance (in light-seconds) or bit rate (in Mbps) for applications to DTN. Regardless of their interpretation, one aims to find optimal paths as a function of these weight parameters x_i, which themselves could depend on time. The complexity of this problem is reflected in a partition of parameter space, which depends on the topology of the network as well as given weights. Over each region in this partition, optimal routing is determined using a shortest path tree in the graph, but different regions may have the same or different trees. Generating these regions and trees is accomplished in the tropical setting using Joswig's algorithm[147], which is reviewed below. For a concrete example of a parametric graph, consider Figure 6.3. Depending on the value of x, different routing decisions will be used and a complete classification of these routing decisions, using tropical geometry, will be provided later; see Figure 6.4.

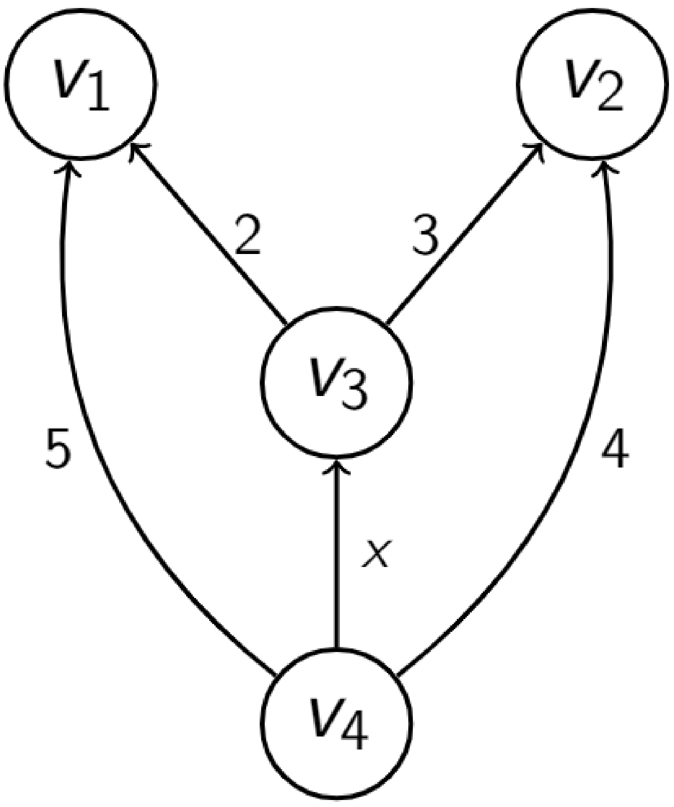

Figure 6.3: An example of a parametric graph.

Another application of these methods comes from a recent analysis of the current standard of DTN routing, contact graph routing[95]. Originally, it was thought that contact graphs were acyclic, but it has recently been shown that they can in fact feature routing loops[258]. Hence, one feature of our analysis is the ability to track how topological features - such as loops - both arise and disappear as the underlying parameters vary. This motivates a more general parametric approach.

To summarize our contributions, we initiate a more general parametric and tropical geometric approach to time-evolving graphs with variable edge characterizations. We provide a slight generalization of the tropically inspired method of Joswig[147] for solving the parametric shortest path problem and prove two novel implementations of this algorithm: one in Python and one using a Verilog 32-bit tropical arithmetic logic unit (ALU). Key to our approach is the extraction of regions of parameter space where certain shortest path trees are optimal. By understanding these, one could design in advance what the routing tables should look like for portions of a Solar System Internet. We then move beyond the tropical setting of Joswig to provide a more general mathematical framework for other parametric graph problems and characterize stability of solutions to these in a novel way. We anticipate that these methods will be highly applicable to DTN as the characteristics of these networks include not only time varying capacities, but also other time varying summaries of note, such as centrality.

Tropical Geometry and Max-Plus Algebra

Before we proceed with the main line of analysis, we offer a quick summary of tropical geometry and its application to graph theory. Max-plus algebra and tropical geometry are rich topics and their application to graph theory is well-studied[183,57,12].

Tropical geometry, in broad strokes, can be thought of as a piecewise-linear version of algebraic geometry, which studies solution sets (i.e. zero sets) of systems of polynomials. The polyhedral view of tropical geometry allows one to phrase things like optimization problems fairly easily, as we will see in subsection 6.2.3 and beyond.

Whereas ordinary geometry occurs over the ring of real numbers $\mathbb{R}$, tropical geometry occurs over the min-plus[*] semiring $\mathbb{T} = \mathbb{R} \cup \{\infty\}$ where the operations of addition and multiplication are redefined as follows:

$$a \oplus b := \min\{a, b\}$$

$$a \otimes b := a + b$$

To illustrate how this works, we offer the following examples:

- $5 \oplus 7 = 5$,
- $5 \otimes 7 = 12$,

[*]One can define the max-plus semiring by instead including $-\infty$ and defining the $\oplus$ operation as max instead. We will use the min-plus semiring because in our optimization setting we wish to minimize path length.

- $a^{\oplus b} = a$,
- $a^{\otimes b} = ba$,
- $5 \oplus \infty = 5$, and
- $5 \otimes \infty = \infty$.

To see why $a^{\oplus b} = a$, note that $a^{\oplus b} = \min\{a, \dots, a\} = a$, as this is a "added" to itself b times. To understand $a^{\otimes b}$, observe that for $b \in \mathbb{Z}_{\geq 0}$ we may view exponentiation as repeated application of the $\otimes$ operation, b times. To extend this definition to all integers, we note that $a^{\otimes -1} = -a$ since this is the additive (tropical multiplication) inverse of a. For $b \in \mathbb{Q}$, we can observe that $(a^{\otimes \frac{1}{n}})^n = a$ so $n(a^{\otimes \frac{1}{n}}) = a$, so $a^{\otimes \frac{1}{n}} = \frac{1}{n}a$. Lastly, we can perform a process known as *completion* to get to our extended real values, but this is beyond the scope of this work, but can be found in the core literature in analysis and related subjects[246], [161], [278], [156], and[126].

Generalizing the rules of tropical arithmetic to tropical matrices casts new light on classical lessons from graph theory. Recall that associated to a (di)graph is the adjacency matrix A, where $A_{i,j} = 1$ if there is an edge from vertex i to vertex j, and $A_{i,j} = 0$ otherwise. It is well known that taking the n^{th} power of the adjacency matrix enumerates walks from vertex i to vertex j of length n.

Analogously, one can perform a similar process for a weighted (di)graph in the tropical setting, where the entries in the matrix are now the weights of the arcs. Taking tropical powers of this weight matrix now calculates the total weight of the shortest walk from vertex i to vertex j. See[57] [207] for a detailed explanation on this process, as well as a proper definition of tropical matrix operations. These observations have yielded fruitful insights in the study of train schedules, robotics, and many other applications; see[285].

6.2.3. Joswig Algorithm Generalization

For this subsection, unless explicitly stated, we are working with weighted, parametric digraphs, i.e. directed graphs where some arcs have constant values associated with them (weights), and some arcs have variables associated with them (parameters). Further note that in [147], they work with parametric digraphs satisfying an additional condition which they call "separability" which means that each parameter only appears as part of one arc's parameter expression. Figure 6.3 satisfies this separability property. For our work, we allow our graphs to not have this quality so that we can eventually substitute functions of time for our parameters. This is a minor distinction, but it does limit the claims we can make about our overall solution spaces at the end of our work. Lastly, we assume that all parameter values and edge weights are restricted to be non-negative, and that our graphs have a single sink (or source). These are typical assumptions of Dijkstra's algorithm, which is employed as a step in the Joswig algorithm.

Definitions

In the subsections that follow, we assume the reader has some familiarity with data structures and algorithms. For the reader who wishes to gain this background knowledge, we refer to [294] and [148].

Let T be a spanning tree in a graph G. Define $P_T(v)$ as the cost of the unique path from the source s to the vertex v in T. Furthermore, define $d(vw)$ as the cost of the arc between v and w, with $d(vw) = \infty$ if there is no arc. For a general path $s = v_1, v_2, \ldots, v_{n-1}, v_n = v$, the cost is given by summation, i.e.

$$P(v) = \sum_{i=1}^{n-1} d(v_i v_{i+1}).$$

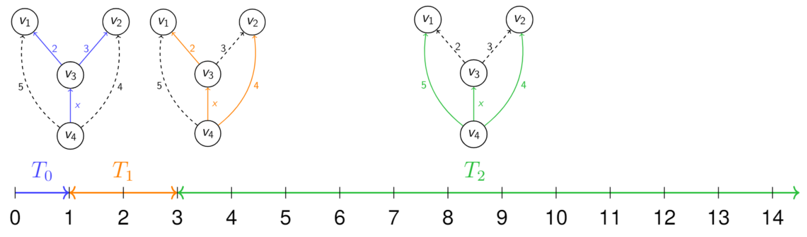

Figure 6.4: An analytical solution of cell decomposition and associated shortest path trees corresponding to Figure 6.3. Results from applying the Joswig algorithm described below. Each interval shows the values of parameter x for which each tree ($T0, T1, T2$) optimal. Note that there is a fourth tree omitted, which is the same as $T0$ except with the path to v_1 going along the leftmost edge with weight 5. It is omitted because there are no values of x that make it optimal.

A *parametric arc* is an edge whose weight is given by a variable. We are interested in graphs with (possibly) multiple parametric arcs and their corresponding parametric shortest path trees. For emphasis, we recall that we restrict ourselves to non-negative edge weights for both constant and parametric arcs.

Now that we've defined some notation, we can describe how one approaches parametric graphs like the one in Figure 6.3. See Definition 6.2.1 for a formal definition of a parametric graph.

Algorithm

The algorithm given in [147] can be summarized as follows:

1. Take a weakly connected digraph G with (variable) edge parameters $x_1, \ldots, x_n$, initialized to any non-negative values $\alpha_1, \ldots, \alpha_n$, fixing a source vertex s. Denote this initialized graph $G(\alpha_1, \ldots, \alpha_n)$.

2. Perform Dijkstra's algorithm on the initialized graph $G(\alpha_1, \ldots, \alpha_n)$ to generate a shortest path tree T rooted at s.

3. Now considering G with original (un-initialized) edge parameters, iterate through each edge vw not in T as follows: consider $P_T(w)$ and $d(vw) + P_T(v)$. If they are incomparable, add $P_T(w) \leq d(vw) + P_T(v)$ to the system of inequalities associated with T. Then generate a new tree T', obtained by removing the path in T to w and replacing it

with the path to w through v via vw. If T' is not already in the list, and is also a shortest path tree, append T' to the list.

4. Repeat step 3 for each tree until no new trees are generated.

The result is a family of shortest path trees, along with systems of inequalities associated to each tree that defines the region of parameter space where that tree is the shortest path tree. A visual representation of this output is shown in Figure 6.4[*]. This process of taking a parametric graph, iterating through its paths, and generating trees along with inequalities is the essence of the Joswig algorithm. This is in fact tropical because the systems of inequalities can be viewed as (systems of) tropical polynomial equations, an observation which is explored in[57]. Each shortest path tree T in the solution family is encoded as a set of monomials which minimize a tropical polynomial for a certain subset of parameter space, which is defined by the inequalities associated to T. Some trees will never be optimal for any value of the parameters, which is reflected by the associated set of inequalities giving an unfeasible solution set (i.e. bounding an empty region).

Implementations

The algorithm above was implemented[†] previously by Ewgenij Gawrilow in Polymake as an optimized extension of Polymake 4.1. In an effort to make the implementation more accessible and applicable to engineering problems, we developed two object oriented implementations written in Python with minimal dependencies. This not only makes it easier to read and understand, but also makes things like field programmable gate array (FPGA) implementations much more feasible. Moreover, the implementation generalizes the algorithm: while previous impementations required edge functions to be linear, the algorithm implemented here was modified to apply to arbitrary functions, such as $\sin(t)$.

[*]Although we include dashed lines to indicate edges from our original graph, technically the shortest path tree only consists of the solid edges.

[†]Link to implementation: `https://polymake.org/extensions/polytropes`

To begin, we should first acknowledge the efficacy of the naive solution: the naive solution here would be to just sample points in the parameter space of the edge weights and keep track of which sample points correspond to which shortest path trees. However, without any good way to pick samples, this severely limits the accuracy to which one can approximate the boundaries between different regions. Moreover, this approach doesn't leverage the fact that the regions in parameter space corresponding to a shortest path tree are all convex: If the points x and y both yield the same tree T, then any point on the line segment between x and y must also correspond to T. This observation gives us a good idea of how to start approximating solutions.

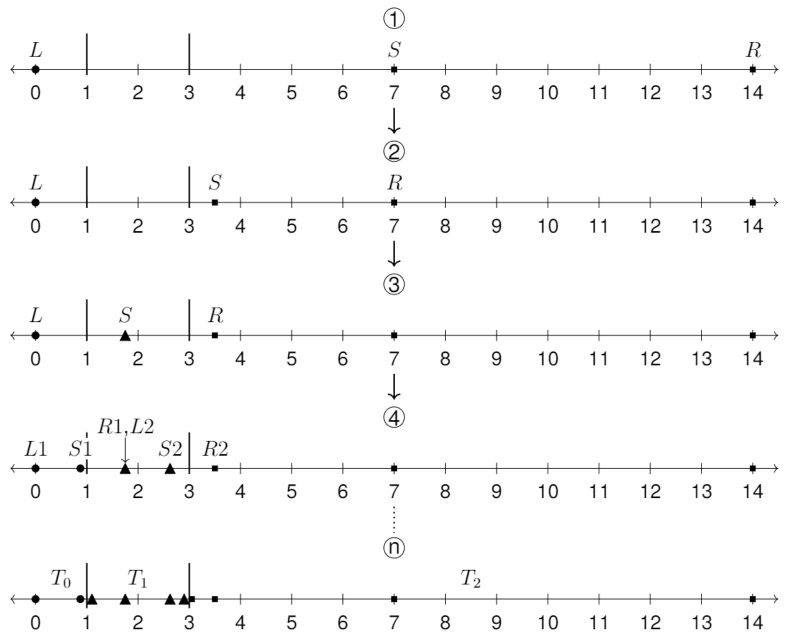

Figure 6.5: An example sequence of steps in the binary search.

Our first implementation is a boundary approximation method based on binary search, and utilizes convexity of these regions in an essential way. The purpose of approximating is to verify our other implementation, but also eventually the Polymake implementation. The problem with this method is that it is inefficient in both memory and time, since it requires sampling a large number of points. The advantage, however, when compared to the naive solution, is that this approximation can be tuned to arbitrary precision, limited by floating point accuracy, among other things. So, while it may not be practical for deployment in a space network, it will at least be accurate enough to determine the validity of our more efficient methods.

This method operates recursively. In one dimension, i.e. only one edge weight is variable, the shortest path tree is found and recorded with the parameter set to the minimum value (e.g. 0). Then another tree is found with the parameter set to the sum of all constant weights (so ignoring parameters) in the graph*. If the two trees are the same, then we only have one shortest path tree feasible and are finished. If they're not, then one recursively performs a binary search between these left and right bounds to find the boundary between the two regions, slowly squeezing these bounds in. In the case that a sample lands on a region with a shortest path tree not yet sampled, the problem is divided into two binary searches – one between the left bound and the sample point, and one between the sample point and the right bound. This process is repeated until the recursion depth reaches a specified limit. This recursion depth determines the accuracy of the approximation, and depths of 5 or 10 seem to suffice for the applications we tested.

Here's an example corresponding to Figure 6.5. The output of this example from our implementation is given in Figure 6.6.

1. Initialize † the search with $L = 0$, $R = 14 = 2 + 3 + 4 + 5$, and $S = (L + R)/2 = 7$.
2. Check containment of L, S, and R from step 1. Generate sample $S = 3.5$ since $L = 0$ corresponds to $T0$ and $R = 7$ corresponds to $T2$.
3. Repeat above to generate sample $S = 1.75$.
4. Checking sample $S = 1.75$ reveals containment in the region for $T1$, unique from $T0$ and $T2$ regions. Generate two new samples for two new searches, one between step 3's L and S, and another between step 3's S and R.

*In one dimension, this is guaranteed to lie in the region extending off to infinity since any path without the parametric arc, assuming such a path exists, will be better than any path including the parametric arc. That is, for any value of the parameter above this sum, you will still get the same shortest path tree.

†In one dimension, we can ignore all parameter values above the sum of all constant weights (i.e. not parameters). There may be tighter bounds, and in some trivial cases, such as no constant weights, this is actually too tight as it is.

⋮

n. After repeated iterations, end up with a reasonable approximation of boundaries at 1 and 3.

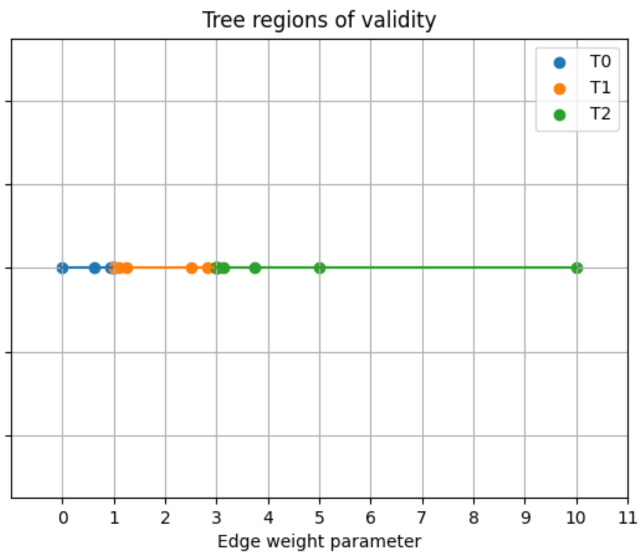

Figure 6.6: Example output of binary (first) implementation applied to Figure 6.3. Note that this agrees with the solution given in Figure 6.4.

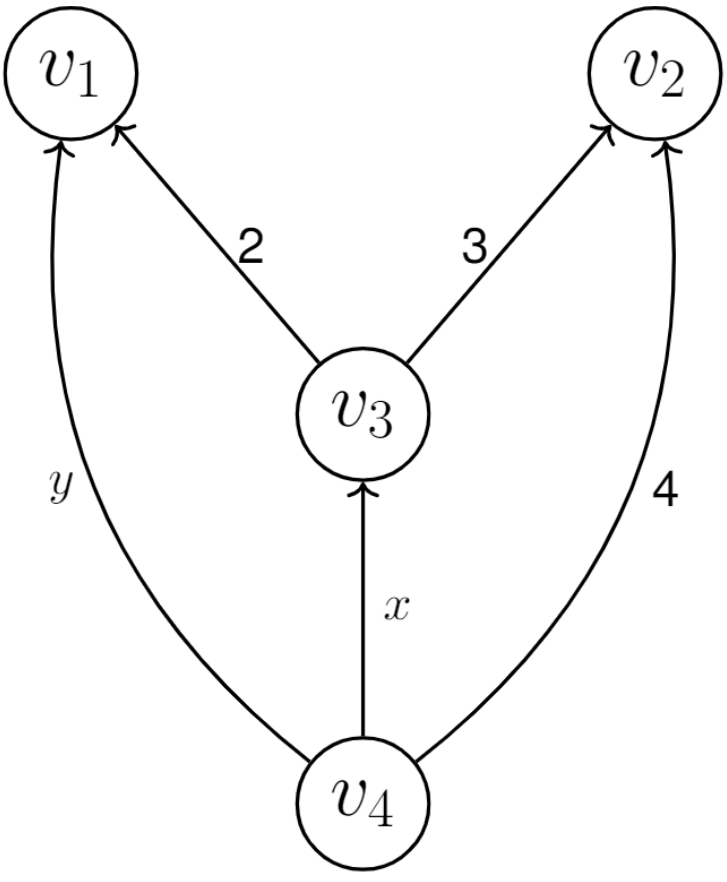

Figure 6.7: Example graph with two parameters. Note the separability.

For two dimensions (i.e. two parametric edges), we essentially repeat the one dimensional process outlined above on a series of lines going through our region of interest. Since we are working in two dimensions, we no longer have the guarantee that regions are constant beyond the total weight of the non-parameterized edges. We provide an example in Figure 6.7 with parameters x and y. For this case, it is easiest to just pick arbitrarily large values, say M, assuming one of the boundaries isn't $y = x$, and search along all lines between $(0, 0)$, $(0, M)$, $(M, 0)$, and (M, M). Then, picking a number of sample points n, e.g. $n = 5$, one draws sets of 5 lines, each set of lines drawn between a vertex and points along one of the opposite edges. An example of this is shown in Figure 6.8. The intuition here is that we are trying to avoid sampling a line perfectly over a boundary, although this may occur in some cases. Generically, our sample lines will go through several regions to help 'detect' as many boundaries as possible, and this is one approach. The result of this two dimensional binary search is shown in Figure

6.9.

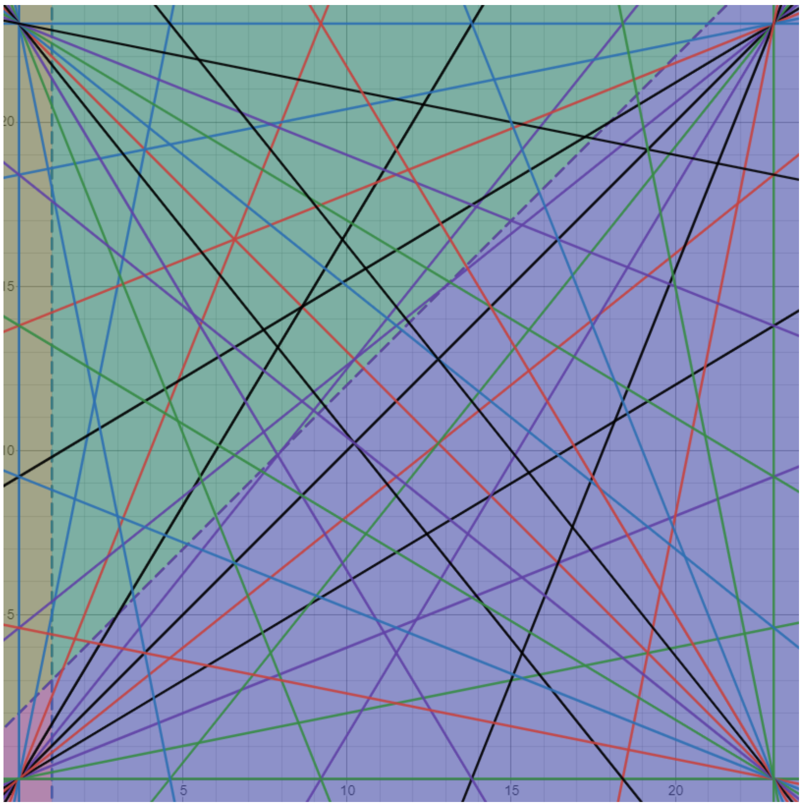

Figure 6.8: Lines for two parameter binary search, $n = 5$ sample lines, corresponding to Figure 6.7

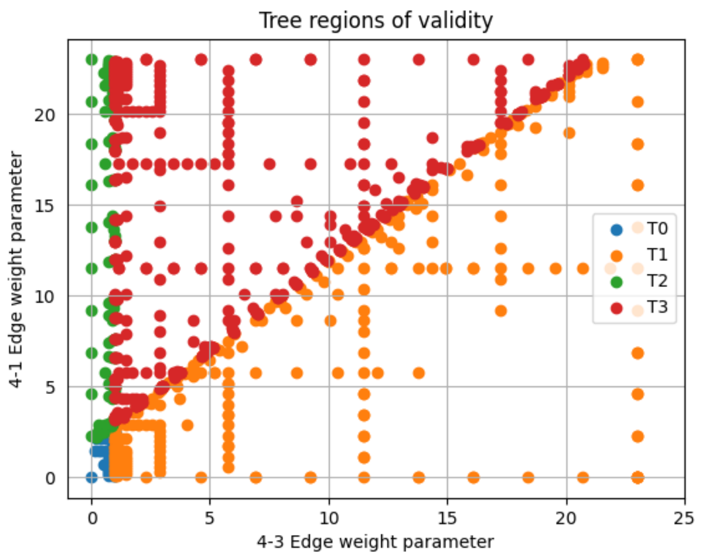

Figure 6.9: Example output of binary (first) implementation applied to Figure 6.7.

Our second implementation[*] is a modified version of the algorithm described above and

[*]Link to implementation: https://github.com/jacleveland/joswig

in [147]. It works by generating a list of all paths possible in the graph, along with the total cumulative weight along each path, as well as whether or not there are variable edges along that path. For each path not on the tree being considered, it solves for the boundary between two regions. One region corresponds to the tree with the original path. The other other region corresponds to the tree with the new path. For example, consider Figure 6.4 which shows the tree T_0. Fix v_4 and v_2 and consider the two paths between them, one going through v_3 with weight $x + 3$, and one directly between v_4 and v_2 with weight 4. We know the boundary between the region for T_0 and the region for T_1 is exactly when $x + 3 = 4$, i.e. $x = 1$, since we are fixing the other edges, namely the path going from v_4 to v_1 through v_3. Lastly, we would find the boundary between T_1 and T_2 by noting that the boundary between them is exactly when $x + 2 = 5$, i.e. $x = 3$. For our example, in Figure 6.4, since there is only one parameter, our boundaries are fixed values. The output of this implementation applied to the graph in Figure 6.3 is shown in Figure 6.10, which shows that our boundaries between regions are at $x = 1$ and $x = 3$. In two parameters, the boundaries between cells are lines. The dashed lines between the regions in Figure 6.8 are such boundaries.

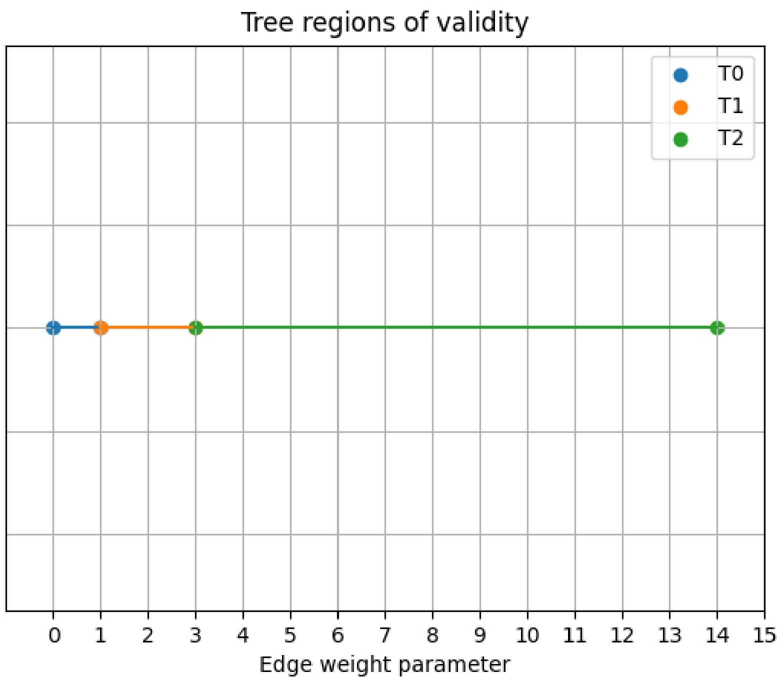

Figure 6.10: Example output of our Joswig (second) implementation applied to Figure 6.3. Note that this agrees with the solution given in Figure 6.4.

More generally, for a graph with d-dimensional parameter space, these boundaries[*] are $d-1$ dimensional affine subspaces of $\mathbb{R}^d$. Assuming separability, i.e. all of the variable edge weights are represented by unique x_i, we know that there will only be coefficients of ± 1 or 0 in front of each parameter, along with a constant representing the shifting of the affine subspace.[†] In essence, we are solving for this constant, and the sign of the coefficient of each variable, which will depend on which path each variable appears on, if at all[‡].

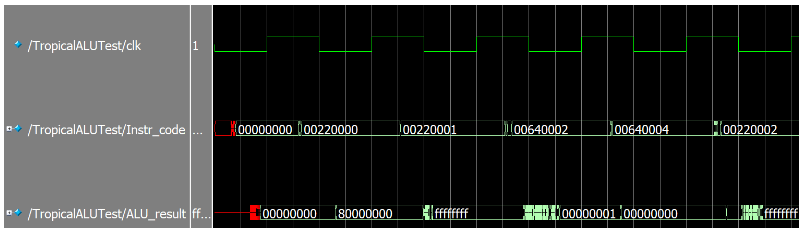

Figure 6.11: Waveform associated with the Verilog Tropical ALU implementation.

[*]Note that points on the boundary represent parameter values associated with multiple trees that are equally optimal. One may convince themselves of this fact by observing that in Figure 6.4, when $x = 1$, both $T0$ and $T1$ are optimal because $x + 3 = 4$ when $x = 1$.

[†]Even if one didn't have separability for the parametric graph, one could force separability by a relabeling process where x on two different arcs would be replaced by y and z, and the process would be performed on this more general graph.

[‡]It is important to note, however, that some of these inequalities will be superfluous, if for example you have two parallel arcs (i.e. same source and sink) with the same exact variables, but the constants of one path add up to a smaller weight than the other. This issue can be solved by pruning the graph to begin before performing the computations, or alternatively, applying separability and treating the variables separately.

```
R0 = 0x00000000; //register file
R1 = 0x80000000; //initialization
R2 = 0xFFFFFFFF;
R3 = 0x7FFFFFFF;
R4 = 0x00000001;

R0 = R1 & R2;
R0 = R1 | R2;
R0 = min(R3,R4); //oplus
R0 = R3 + R4;    //otimes
R0 = min(R1,R2);
```

Figure 6.12: High level representation of program being demonstrated in the waveform of Figure 6.11.

Verilog Implementation

In addition to working on Python implementations, great strides were also made in creating our Verilog implementation* of a 32-bit tropical arithmetic logic unit (ALU). This tropical ALU is capable of executing instructions for the $\oplus$ and $\otimes$ as well as the logical AND and OR operations. The 32nd bit indicates the ∞ of the tropical semiring, i.e.

`32'b1xxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx.`

The arithmetic operations here represent the usual operations as defined in §6.2.2. Note that for $\otimes$, overflowing results in finite values for simplicity, as demonstrated by the waveform in Figure 6.11. An alternative is to have $\otimes$ overflow to ∞ since any value larger than $2^{31} - 1$ could be considered to be ∞. As a compromise, future revisions could include an overflow mode that selects between overflowing normally and overflowing to infinity.

*Verilog implementation available here: https://github.com/jacleveland/tropicalu.

6.2.4. Parameter space decompositions for weighted graphs

In this subsection, we discuss several natural mathematical extensions of the framework introduced in [147]. First, we discuss what happens when one changes the problem under consideration from the single source shortest path problem to other problems of interest for a weighted graph. Second, we consider how the parameter space perspective allows us to characterize some of these problems, even when they don't yield tropical formulations. One upshot of this subsection is the ability to define a principled notion of stability of solutions to these parametric problems. The results of this subsection are outside the purview of traditional tropical geometry, though the perspective taken is heavily inspired by the tropical geometric approach to parametric routing.

Tropical Graph Problems

To formally describe more general parametric problems on graphs, we introduce some terminology:

Definition 6.2.1. Let G be a graph with n nodes V and k edges E. Let $E' \subset E$ be a subset of edges of size $m \leq k$, and suppose that we assign to each $e \in E'$ an affine function in the real variable x_e, and all other edges $s \in E \setminus E'$ a constant weight. We call weighted graphs as defined above *parametric graphs*. We refer to $\mathbb{R}^m$, where $m = |E'|$, as the *parameter space* of G, and identify each $x \in \mathbb{R}^m$ with a fixed choice of parameter values on G.

For a fixed parametric graph G with $m \leq k$ parametric edges, every point x in parameter space $\mathbb{R}^m$ corresponds to a different choice of weights for G.

Definition 6.2.2. A parametric problem P on a weighted graph G with m parametric edges is described by the following:

- A set of discrete objects called the solution set J;
- An assignment $h : \mathbb{R}^m \to \mathcal{P}(J)$, where $\mathcal{P}(J)$ is the power set of J.

Example 1. The parametric all-pairs shortest path problem on a weighted graph $G = (V, E)$ has as its solution the set of all paths U where:

- For each $x, y \in V$, U contains a unique path p_{xy} from x to y;
- If p_{xy} has a subpath from u to w, then p_{uw} is given by this subpath.

The assignment $h : \mathbb{R}^m \to J$ maps each weight vector to the set(s)[*] of lowest weight paths in J.

Example 2. The parameterized max-flow problem on a weighted graph G with source v_{src} and sink v_{sk} has as its solution set all graph cuts separating v_{src} from v_{sk}, i.e. the set of all minimal sets of edges $Q = \{e_1, e_2, \dots, e_l\}$ such that when Q is removed from G, v_{src} and v_{sk} are in different connected components of G. The assignment $h : \mathbb{R}^m \to J$ maps each weight vector to the minimal weight cut(s) in J.

Definition 6.2.3. Let G be a parameterized graph with m parameterized edges. Let P be a problem on G with a discrete solution set J. Let $h : \mathbb{R}^m \to J$ be the assignment function for P. For each $\alpha \in J$, let $U_\alpha := h^{-1}(\alpha)$ be the region in $\mathbb{R}^m$ assigned to α. We say that $\mathcal{U}_J := \{U_\alpha \mid \alpha \in J\}$ be the decomposition of $\mathbb{R}^m$ induced by J.

Following through the definitions, one quickly sees that the decomposition of $\mathbb{R}^m$ induced by the all-pairs shortest path problem on a graph G with m parameterized edges is in fact the same tropical hypersurface introduced in[147]. We observe that many natural parametric problems on graphs may also be phrased in terms of tropical equations, and hence will decompose parameter space into cells given by a tropical hypersurface. We will term such problems as *tropical graph problems*. We now give a (certainly non-exhaustive) list of tropical graph problems:

- Single source shortest path
- All-pairs shortest path

[*]Note that two paths p_{xy} and s_{xy} can be equivalent in terms of having optimal path length and allowing equivalently optimal paths to branch off of them as well. Recall that in subsection 6.2.3, any point on the boundary between two cells had at least two equivalent shortest path trees. Hence some weight vectors will map to multiple equally optimal solutions.

- Minimum spanning tree
- Traveling salesman
- Max-flow

Knowing that a problem is a tropical graph problem immediately provides a lot of information about the parameter space decomposition. This is due to the regularity of tropical hypersurfaces: for instance, we are guaranteed that for any solution $\alpha \in J$ the region assigned to α is a convex, m-dimensional set with piecewise linear boundary, i.e. convex polytopes. Furthermore, we know almost all parameter vectors have a unique solution.[*]

This structure makes such decompositions of parameter space much easier to determine with algorithms, using for instance the algorithm described in the previous subsections, or with substantial modification.

We are also guaranteed that if we take natural "combinations" of such problems, the resulting parameter space decomposition is once again tropical. In particular, consider the natural definition of a product of problems, i.e. a problem with solution set given by a Cartesian product of solution sets for other problems. As natural examples of problems that can be phrased as products of simpler problems, note that shortest path spanning tree may be written as a product of instances of shortest path with fixed endpoints, and that all pairs shortest path may in turn be written as a product of instances of shortest path spanning tree. The following lemma is trivial from a tropical geometry perspective, but is worth mentioning in our new language:

Lemma 6. Suppose that P_I, P_J are problems with assignments given by tropical polynomials $h_I, h_J : \mathbb{R}^m \to \mathbb{T}$. Then the assignment for $P_I \times P_J$ is given by the tropical polynomial $h_I \bigotimes h_J$. $\square$

Though one may always define such a product regardless of whether the problems are tropical, it is comforting to know that tropical problems are closed under this operation.

[*]Note that parameter vectors on the boundary between two cells will correspond to two or more different solutions in J.

More General Problems

From the above discussion, one might be inclined to think that all natural parameterized graph optimization problems are tropical. This is not the case, and a prominent example of problems that do not have this property arise from various centrality measures, which are introduced in Section 6.1.2.

Example 3. A parameterized centrality problem on a weighted graph G has as its solution set all orderings of vertices, where we allow for orderings where two or more vertices to "tie". The assignment $h : \mathbb{R}^m \to J$ maps a parameter vector w to the ordering on vertices induced by a chosen centrality measure computed on G with weights given by w.

Commonly used centrality measures include eigenvector, Katz and betweenness centrality, see[29]. For a given centrality problem with assignment h, we may consider the decomposition induced by orderings on the vertex set. However, we find that the cells in this decomposition are in general no longer convex with piecewise linear boundaries.

Example 4. Consider parameterized eigenvector centrality on a parameterized graph G. Suppose that G has m parameterized edges, and consider the parameterized adjacency matrix $A_G(w)$, which is a function of $w \in \mathbb{R}^m$. Then we see that both the maximum eigenvalue and the corresponding maximal eigenvector

$$A_G(w)v = \lambda_{max}(w)v$$

depends on w. Notice that by definition of the adjacency matrix, the ith entry of the eigenvector v_i may be identified with the ith vertex of G. The eigenvector centrality of a graph is defined to be the ordering imposed on the vertices by the "score function" given by the entries of v, which in the parametric regime is a function of our parameter vector.

We see that solution regions are defined by the (in)equalities of the form

$$\sum_j a_{ij}(w)v_i = \lambda_{max}(w)v_i$$

$$v_i(w) \geq v_k(w), i \neq k.$$

These (in)equalities almost look like the defining (in)equalities of a semi-algebraic set, and indeed if λ_{max} were constant this would be the case. However, λ_{max} depends on the parametric characteristic polynomial of the matrix $A_G(w)$ and hence is (generically) not a polynomial.

Example 5. Despite the difficulties presented by the general case, for certain graph architectures one is able to compute the corresponding eigenvector centrality decomposition without too much difficulty. For instance, consider the star graph with n edges, i.e. the graph with $n+1$ vertices such that there is an edge from the first vertex to all other vertices, and no other vertices are connected by edges. The parameter space of the weight space is $\mathbb{R}^n$. The principal eigenvector is given by $(1, \frac{w_1}{||w||_2}, \frac{w_2}{||w||_2}, \ldots, \frac{w_n}{||w||_2})$ where $||w||_2 = \sqrt{w_1^2 + w_2^2 \ldots + w_n^2}$. Thus we see that every possible permutation $\sigma \in S_{n+1}$ that fixes 1 is a possible solution for this problem, as the ranking is ordered by the ordering on the weights, apart from vertex 1 which is always central. The set of these are in bijection with the symmetric group S_n, and hence we define the set of regions

$$U_\sigma = \{(w_1, w_2, \ldots, w_n) \in \mathbb{R}^n \mid w_{\sigma(i)} \geq w_{\sigma(j)}, \sigma \in S_n\},$$

which define the regions of parameter space for which σ is the ordering induced by the eigenvector.

The situation for Katz centrality is similarly complicated. It is interesting to note that Katz centrality depends on a parameter such that when one takes an appropriate limit, eigenvector centrality is recovered, and hence we expect their respective parameter space decompositions to converge as well. In contrast, the betweenness centrality decomposition yields a description that, while not tropical, can be constructed using convex polytopal regions:

Theorem 7 (Betweenness Centrality is a Polytope Complex). Let β^* be the assignment induced by betweenness centrality. Then for all $\alpha \in J$ the region $h^{-1}(\alpha)$ is a union of convex polytopal regions, glued along (potentially empty) faces. In other words, $h^{-1}(\alpha)$ is a polytope complex.

Proof. We may record the betweenness of N nodes as a vector in $\beta(w) \in \mathbb{R}^N$, which we write this way to emphasize the $w \in \mathbb{R}^m$ dependence. From this, we obtain a solution (for parameter w) to the betweenness centrality problem by considering the ordering of entries of $\beta(w)$, which we have denoted $\beta^*(w)$.

Consider the betweenness of a vertex $v \in V$, which we denote $\beta(w)_v$. Recall that this is given by the formula

$$\sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}.$$

The key observation is that both σ_{st} and $\sigma_{st}(v)$ are determined solutions to the all-pairs-shortest path problem, as both quantities are given by the size of subsets of the set of $\frac{n(n-1)}{2}$ optimal paths for a given weight vector. Hence, we see if $\beta(w)_v \geq \beta(w)_q$ for $v, q \in V$, then this inequality holds for all vectors w in the same region as w in the all-pairs shortest path problem. Thus, $\beta^*(w) : \mathbb{R}^m \to J$ defines a face-wise constant function on the decomposition induced on $\mathbb{R}^m$ by the all-pairs shortest path problem on G. There may be multiple cells of the all-pairs shortest path problem that induce the same ordering α, hence $(\beta^*)^{-1}(\alpha)$ may be a union of polytopes, which are necessarily glued along (possibly empty) faces as they are chosen from a decomposition where all regions have this property. □

Graph Symmetries and Parameter Space Decompositions

A challenge with constructing or even sampling parameter space decompositions is that for any graph of reasonable size one is quickly confronted with a high dimensional problem. However, if the graph topology in question has nontrivial automorphisms, we may be able to leverage the additional symmetry to simplify the decomposition, provided that the particular problem in question respects these symmetries. For simplicity, we will assume that all edges of G are parametric.

Definition 6.2.4. Let $G = (V, E)$ be a graph. A graph automorphism ψ is a bijection from G to itself such that if v and w are adjacent in G then $\psi(v)$ and $\psi(w)$ are also adjacent. The automorphisms of a graph form a group, which we denote $Aut(G)$.

Lemma 8. Let G be a graph with m edges, all of which are parameteric. Then for all such G (outside of 3 cases), $\psi \in Aut(G)$ induces an automorphism on parameter space $\mathbb{R}^m$, where the automorphism is given by permuting the coordinates of $\mathbb{R}^m$.

Proof. For all graphs (excluding 3 cases that are outlined in Corollary 3.3 of[243]), the vertex automorphism group of a graph is isomorphic to the edge automorphism group of the graph (i.e. automorphisms of its line graph). Edge automorphisms are bijections from the edge set to itself satisfying certain conditions, and hence they induce bijections on the edge variables. □

Proposition 9. Let P be a problem in the following set of parametric graph problems: End-to-end shortest path, shortest path tree, traveling salesman tour, all pairs shortest path, betweenness centrality, Katz centrality, eigenvector centrality. Let $h : \mathbb{R}^m \to J$ be the assignment for P. Then there is an action of $Aut(G)$ on J that commutes with the action of $Aut(G)$ on $\mathbb{R}^m$, i.e. such that $\psi(h^{-1}(x)) = h^{-1}(\psi(x))$ for all $\psi \in Aut(G)$.

Proof. We will prove the results for all-pairs shortest path and for eigenvector centrality and comment that the other cases follow similarly.

Suppose that x is a solution to the end-to-end shortest path problem between vertex i and vertex j with respect to parameter vector $w = (w_1, w_2, \ldots, w_m)$. Then x is a path $p_{ij} = v_{i1}v_{i2}\ldots v_{il}$. $\psi \in Aut(G)$ acts on paths in G, by mapping p_{ij} to $\psi(p_{ij})$, where

$$\begin{aligned}\psi(p_{ij}) =& \psi(v_{i1})\psi(v_{i2})\ldots\psi(v_{il}) \\ =& v_{\psi(i1)}v_{\psi(i2)}\ldots v_{\psi(il)},\end{aligned}$$

where we have written the action in a way to emphasize the fact that graph automorphisms can be viewed as certain relabelings of the vertex set. We denote the induced action on labels as ψ. It is clear that p_{ij} is optimal for w iff ψp_{ij} is optimal for $(w_{\psi(1)}, w_{\psi(2)}, \ldots, w_{\psi(n)})$, which shows that $h^{-1}(\psi(x)) = \psi(h^{-1}(x))$ and proves the result for end-to-end shortest path. Now let x be a solution to all-pairs shortest path. Hence x is a union paths p_{ij} for each $i, j \in V$, and by defining the action of ψ to be the diagonal action on the union of these paths, we see that the same result holds for all-pairs shortest path.

For eigenvector centrality, note that the action of $\psi \in Aut(G)$ on G permutes the columns and rows of the adjacency matrix A of G. Hence we see if $Ax = \lambda x$ then $\psi(Ax') = \lambda x'$, where x' is the vector with $x'_i = x_{\psi(i)}$. Thus we may define an induced action ψ on eigenvectors of A, and see that this commutes with the action of ψ on G. $\square$

Corollary 1. Let $v \in V$, and let $h(w) = \sigma_w$ denote the solution to any of the parameterized centrality problems in the above proposition, i.e. σ_w is the induced ordering from V to $[n]$. Then if $\sigma_w(v)$ is constant as a function of w, then v is fixed by all automorphisms of G.

An example demonstrating the above corollary is the central vertex in the n-star graph. A similar result holds for solutions to the various routing problems in the Proposition 3.13, and more sophisticated statements relating the action of $Aut(G)$ to the symmetries of parameter space exist. In theory, incorporating these symmetries into algorithms such as the one described in[147] should result in faster computations, as it will only be necessary to compute a subspace of the total decomposition, and then use the action of $Aut(G)$ on parameter space to reconstruct the rest.

6.2.5. Stability of solution sets

Recall that $h : \mathbb{R}^m \to J$. Considering the geometry of $h^{-1}(x)$[*] gives rise to new perspectives for parametric problems on graphs. In some sense, the size and shape of the set $h^{-1}(x)$ corresponding to a solution $x \in J$ should reflect how "stable" to perturbation this optimal solution is. Consider for instance a convex solution set $h^{-1}(x) = U$ with large volume and "small" boundary measure. Then one can reasonably assume that small perturbations of most vectors in U will stay inside U. In contrast, if U has small volume, or has large volume but with a comparably large boundary (e.g. if U looks like a tree) then we lose such guarantees.

[*]Called the *fiber* of h at x.

To make these ideas precise, we will need to appeal to the language of *measures*. Informally, a measure is a function that assigns a mass to a set. One of their primary applications is in probability theory, where a (normalized) measure on a space of possible events provides a way of assigning a probability to a subset of events occurring. With this in mind, consider the following definitions:

Definition 6.2.5. Let μ be a measure on a measurable space, and let $f : X \to Y$ be a measurable map. The pushforward measure $f_*(\mu)$ is defined by setting $f_*\mu(A) = \mu(f^{-1}(A))$ for all measurable $A \subset Y$.

Definition 6.2.6. Let μ be a compactly supported measure on $\mathbb{R}^m$, let G be a graph with m parameterized edges. Let $h : \mathbb{R}^m \to J$ be the assignment for a problem P with solution set J. Then we say $h_*\mu$ is the measured induced by h relative μ.

Before proceeding, we explain two natural ways one may obtain measures induced on parameter space in a networking setting. For concreteness, suppose our system of interest is modeled by n agents arranged on a graph with fixed topology, but possibly changing edge weights, which may represent for instance distance or available channel capacity.

Example 6. Suppose that our edge weights are subject to uncertainty, due to environmental factors or to account for the possibility transmission errors. This can be modeled by assigning a probability measure μ_i to the edge i, and letting $\mu = \prod_{i=1}^{m} \mu_i$. For instance, in the standard white noise model for communication channels, each μ_i is a one-dimensional Gaussian distribution, and μ is thus an m-dimensional Gaussian. μ induces a measure on each cell $h^{-1}(x)$ in our decomposition, corresponding to how often x is the optimal solution given the presence of uncertainty modeled by μ.

Example 7. Even in a completely deterministic system, time-evolving edge weights induce a measure on parameter space. For instance, suppose that our edge weights evolve periodically in time. We may view then view the evolution of this system as a closed loop $C : \mathbb{R} \to \mathbb{R}^m$ with period T in $\mathbb{R}^m$. This defines a measure μ_C on our solution set J, such that for any $x \in$

J we have

$$\mu_C(h^{-1}(x)) = \int_{h^{-1}(x)} C(t)\,\mathrm{d}t,$$

i.e. the measure induced by the amount of time our system spends in the region $h^{-1}(x)$.

We now offer two different definitions of stability for a solution set $h^{-1}(x)$, relative to an induced measure $h_*\mu$.

Definition 6.2.7. Let $h : \mathbb{R}^m \to J$ be a parameterized problem on G, and let μ be a compactly supported measure on $\mathbb{R}^m$. Then for $x, y \in J$, we say that x is more stable than y if $h_*\mu(x) \geq h_*\mu(y)$. We call $x^* = \arg\max_{x\in J}\{h_*(\mu)(x)\}$ the maximally stable solution.

Definition 6.2.8. Let $h : \mathbb{R}^m \to J$ be a parameterized problem on G, and let μ be a compactly supported measure on $\mathbb{R}^m$. For each $x \in J$ and $\epsilon > 0$, let $W_\epsilon(x)$ be the subset of $h^{-1}(x)$ of all y such that $B_\epsilon(y) \subset h^{-1}(x)$. Then for $x, y \in J$, we say that x is more ϵ-stable than y if $\mu(W_\epsilon(x)) \geq \mu(W_\epsilon(y))$. $x^* = \arg\max_{x\in J}\{W_\epsilon(x)\}$ is the maximally ϵ-stable solution.

Stability describes the optimality of a solution up to uncertainty captured by the probability distribution μ. ϵ-stability also captures this, but additionally accounts for the possibility of a small perturbation of the inputs. This more closely mimics the stability described in the beginning of this subsection, in that sets with relatively small boundary measure are necessarily more stable. Notice that as $\epsilon \to 0$, ϵ-stability recovers stability.

Even for tropical graph problems, determining stability and ϵ-stability is computationally nontrivial. Exact solutions to this problem in full generality is at least as difficult as computing the volume of an n-dimensional polytope, which is known to be computationally challenging[36]. However, there are known to be more efficient approximation schemes (e.g.[79]), and it would be interesting to see how such schemes perform on computing stability or ϵ-stability from systems arising from data.

6.2.6. Conclusion and Future Work

Tropical geometry offers a new potential avenue for approaching delay tolerant networking. The main potential comes from the fact that tropical geometry is a natural setting for studying optimization problems, many of which can be expressed tropically[146]. As demonstrated above, some normally intractable problems become feasible in the tropical setting. Moreover, tropical approaches have been verified to work for scheduling trains[285] which are similar to delay tolerant networks in many ways. One similarity is that trains may have to wait in certain stations, similar to how some bundles have to be buffered at certain nodes. Another similarity is that a train may travel along a path without end to end connectivity, similar to the nature of delay tolerant networks.

The algorithms and framework detailed in this section extend the utility from trains and traditional linear optimization towards the temporal setting of DTNs, and can be used to analyze current and future networks. For example, given a network metric, one may ask for the optimal way to add another communication node.

Because tropical geometry lends itself to computation, including by FPGA, local algorithms that use a tropical-geometric approach to decision-making are feasible, offering a direct path to implementation.

We have also shown how the perspective of studying a parametric problem on a graph through the geometry of its parameter space bears fruit even outside the strict tropical framework. To our knowledge this perspective is underdeveloped in the temporal graph literature.

We end with suggestions for future directions:

Future Work

- Solving temporal networking problems could be made possible through the application of parametric graph optimization. One way to demonstrate this would be to simulate a space network using orbital analysis software, and then attempt to make routing decisions based on the state of the network over time.

- Determine stability of an optimal solution for mild perturbations of the associated graph in parameter space. How much error are we allowed to have in a given network to still meet a threshold quality of service?
- Study the case when edge weights are unpredictable, and instead when edge stability follows a probability distribution with respect to time. What modifications are needed to our approach for this case?
- Study what factors (e.g. bit rate, latency, network demand) are appropriate for analysis as parameters in the context of our approach.
- Finish off Verilog Tropical ALU by adding signed arithmetic, logical and arithmetic shifts, pipeline, flow control (branches, jumps), branch prediction, co-processors, and cache. Implement companion Python matrix multiplication programs for driving an FPGA with several instances of a Tropical ALU (CPU) onboard.
- Investigate the behavior along boundaries of cells in the single source shortest path case. Define a (co)sheaf over the cells and their boundaries that maps trees over these cells.
- Extend the Joswig implementation to higher parameter dimensions, arbitrary graphs, and the ability to solve for trees in time intervals when the parameters are given by arbitrary continuous functions in parameter space.

6.3. Final Notes

The study of space networking is an exciting and crucial field that has captivated the imagination of people across the globe[*]. Over the past century, we have made incredible strides in exploring the vast and mysterious expanse of space. Developing space networks will be instrumental in unlocking the secrets of the cosmos and achieving our goals of exploring the final frontier. One of the critical components of space networking is long-distance communication, which is vital for communicating with the International Space Station and probes that travel beyond our Solar System. Dynamic networks play a significant role in this process, providing a framework for studying communication in this unique and valuable environment.

This chapter has explored the structure and nature of space networks, including lunar networks and routing issues in broader settings. It has provided a comprehensive theoretical framework for studying dynamic networks in the context of space networking and showcased examples, data, and other empirical results of their effectiveness. Moreover, this chapter highlights the critical role that mathematical and computational tools play in space networking. These tools help researchers better understand the dynamics of space networks and design new, innovative solutions to the challenges that arise.

The potential benefits of space networking are vast and far-reaching, from developing new technologies and unlocking the mysteries of the universe to advancing our understanding of the origins of life. As we continue to explore the cosmos, the development of space networks will be instrumental in achieving our goals and taking humanity to new heights.

The study of space networking is essential and fascinating; it has the potential to revolutionize our understanding of the universe. By leveraging the power of dynamic networks and cutting-edge mathematical and computational tools, researchers can continue to advance our understanding of space communication and design the next generation of space networks. With these tools at our disposal, the possibilities for space exploration are truly limitless.

[*]This section was partly written by `ChatGPT`[218], inspired by our study in machine learning in Section 5.2.

Part III

Looking Forward

An equation has no meaning for me unless it expresses a thought of God.

Srinivasa Ramanujan Aiyangar

7 Conclusion

NETWORKS are all around us, all the time. In this dissertation, we have traversed the labyrinthine landscape of dynamic networks, uncovering part of their fascinating nature. Systems of dynamic networks are non-local, yet analyzable; natural, yet abstract; and computationally tractable, yet hard[*]. As we stand on the precipice of this intellectual journey, we are afforded the opportunity to reflect upon the profound implications of our findings and the potential they hold for reshaping our understanding of these remarkable systems.

[*]Though, they are sometimes too hard and sometimes even NP-hard.

7.1. Overview of Results

Dynamic networks, as we have demonstrated, transcend the boundaries of traditional spatial and temporal constraints. Their non-locality lends a certain elegance to the underlying mechanisms of dynamic networks, allowing them to adapt and evolve in response to the ever-changing environments they inhabit. Their modeling power renders them as natural phenomena that seamlessly integrate with the very fabric of our world, reflecting the inherent dynamism and interconnectedness that define the complex tapestry of life.

Our exploration of these networks began in earnest with Chapter 4, where we delved into the realm of viral models of spread. Through our examination of these models, we gained invaluable insights into the intricate dynamics governing the proliferation of ideas, diseases, and information. The lessons we gleaned from these models hold the potential to inform public health strategies, foster the dissemination of beneficial innovations, and catalyze the advancement of social and technological progress.

In this chapter, we also explored the various factors influencing the spread of contagions in both physical and virtual realms. By evaluating the role of network topology, information diffusion, and human behavior, we provided a comprehensive understanding of the mechanisms that drive the spread of viral phenomena. This understanding, in turn, has implications for the design of effective intervention strategies, targeted at reducing the negative impact of undesirable contagions while promoting the propagation of ideas and technologies that can foster global prosperity.

In Chapter 5, we ventured forth into the world of fast-moving networks, examining their unique properties and the powerful implications they hold for the future of communication, logistics, and rapidly-evolving systems. While we focussed on the analysis of sports-based networks, our techniques have promise to generalize beyond their current application areas. As our society continues to evolve at an unprecedented pace, these fast-moving networks will play a pivotal role in fostering rapid adaptation, enhancing global connectivity, and shaping the trajectory of human progress.

From the fluid formations of teams to the rapid transitions in basketball, we explored how the principles of dynamic networks can be applied to enhance our understanding of the strategies and tactics employed by athletes and coaches. By examining the role of network topology, implicit information diffusion, and individual and team behaviors, we provided a comprehensive understanding of the mechanisms that drive success and failure in the context of sports. This understanding, in turn, has implications for the development of innovative training methodologies, performance analysis tools, and game strategies that can elevate the athletic performance of individuals and teams alike. We also provided a framework for leveraging off-the-shelf neural network architectures through the appropriate featurization, which results in quick development time and optimized compute.

Moreover, our exploration of dynamic networks in sports served as a powerful testament to the ubiquity and versatility of these systems, demonstrating their ability to provide valuable insights across a diverse range of domains. The principles gleaned from our investigation of fast-moving networks in sports can be extended to other areas of human endeavor, offering a rich and fertile ground for future research and applications.

Finally, in Chapter 6, we took our inquiry to the stars, collaborating with NASA to investigate the uncharted territory of space networks. As humanity embarks upon the next chapter of its cosmic odyssey, the knowledge we have garnered about the behavior and properties of these networks will be crucial to our success in exploring and harnessing the boundless potential of the universe that lies before us.

The study of dynamic networks has opened a gateway to a deeper understanding of the complex, interconnected systems that underpin our world. As we gaze into the future, we are reminded that our collective journey has only just begun, and that the continued exploration of dynamic networks holds the promise of untold discoveries and unprecedented advancements. It is our hope that this dissertation serves as a beacon, illuminating the path forward and inspiring future generations of scholars to continue unraveling the enigmatic and wondrous nature of dynamic networks, ultimately unlocking the secrets that lie at the heart of the cosmos itself.

7.2. The Bridges of Kaliningrad

The city of Königsberg is now Kaliningrad, located between Lithuania and Poland. It is a peculiar city, as it is part of Russia, but surrounded by non-Russian territory. Königsberg suffered extensive bombing during World War II, and all seven bridges were destroyed by the Allies. There are now five rebuilt bridges and, with this new topology, it is now possible to find an Euler path[266]. The practical relevance of this solution is unclear, but the principles of graph theory continue to function after centuries, world wars, and the rise and fall of nations. It is the inherent marvel of mathematics that our theorems continue to function beautifully long after their discoverers have passed.

We can only see a short distance ahead, but we can see plenty there that needs to be done.

Alan Mathison Turing

8 Future Directions

HOPEFULLY, your interest has been captivated in dynamic graphs. Part of the fascination with these structures is their connections to other areas of study, as well their natural occurrence in a wide range of important applications. In this chapter, we will take a brief tour through some (but far from all) potential future areas of studies. Some of the sections cover unpublished and active areas of my or my collaborators' work, while other sections focus on open questions that would be interesting for the next generation of researchers and students to study.

8.1. Theoretical Considerations

8.1.1. Open Questions

There are a wide range of open theoretical questions that have vexed me throughout my research in this area. One particular issue is the lack of necessary conditions for certain important dynamic connectivity properties of networks.

Linking Static and Dynamic Properties

We would like to link static and dynamic notions of connectivity. So far, the best results we have are Propositions 2, 3, and 4. These Propositions provide only sufficient conditions for dynamic connectivity, which means that we must either find the necessary conditions or we need to provide a laundry list of sufficient ones. From practical experience, it seems unlikely that we will be able to find clear necessary conditions that generally work, but we may be able to restrict ourselves to particular cases or provide results in expectation or probability for stochastic cases. It would also be nice to demonstrate a much more sophisticated link between static graph properties and their dynamic counterparts.

Therefore, we are working towards a theorem that would connect these properties. One observation we have is that it seems as though periodic, repeating structures can be hurtful. We would like to show, then, the following conjecture or some variation thereof:

Conjecture 1 (Dynamic Connectivity from Local Bounding). There exists an $\Omega(n)$ such that a graph sequence that is non-stranding, connected, and non-excessively repeating, with at least $\Omega(n) \times n^2$ edges is δ-connected.

What does this conjecture say? First, it requires several properties of the sequence:

1. Non-stranding: so that the sequence of graphs does not become trivially disconnected
2. Connected: so that there always exists a path from one vertex to another

3. Non-excessively repeating: an edgeset cannot be repeated in the sequence until all other possible edgesets have occurred; this guarantees good "mixing" of the sequence

4. At least $\Omega(n)$ edges: this guarantees that the graph has enough possible paths to fight against pathological behavior.

Notably, these properties are all properties of the individual graphs, with the only global property being "non-excessively repeating." Therefore, this type of conjecture would connect the static graph properties to a dynamic one. To prove this conjecture, there are two foreseeable steps: first, it must be shown that there is non-trivial behavior, i.e. that if there are only n edges, we can always find a disconnected sequence and that if there are n^2 edges, we are always connected. Second, it must be shown that a threshold exists, in that if we know that this property holds with $\tau(n)$ edges, that it holds with $\sigma(n)$ edges, where $\sigma(n) > \tau(n)$.

In the stochastic case, we would like prove something like Conjecture 2.

Conjecture 2 (Bound in Stochastic Setting). We fix T to be our discrete, infinite time-indexing set, e.g. $\mathbb{N}$, and V as some (finite) vertex set. Let μ_i^t be a probability measure indexed over $V \times T$

$$\mu_i^t : \mathcal{P}(V) \to [0, 1]$$

where $\mathcal{P}(V)$ is the powerset of V and we assume the typical axioms of a probability measure. This probability measure indicates the likelihood of a particular set of outgoing edges from edge i at time. In other words, we can define a stochastic graph sequence $G_t = (V, E_t)$ with the probability of a particular edge as

$$\mathbb{P}((u, v) \in E_t) = \sum_{S \in \mathcal{P}(V): v \in S} \mu_u^t(S)$$

We place the additional restriction that $\mu(\emptyset) \neq 1$.

We can now state our conjecture: given a measure such that the graph is almost surely connected except at finite timesteps and the existence of each edge is independent (and somewhat equiprobable) at each timestep, then our dynamic graph is almost surely dynamically connected.

Notably, Conjecture 2 certainly does not provide necessary conditions, only sufficient ones. The ultimate theorem would be to prove a theorem that provides both sufficient and necessary conditions on a measure for almost sure dynamic connectivity. Interestingly, equiprobability is not on its own enough to guarantee dynamic connectivity, but independence is almost certainly too strong of a condition. Almost sure connectivity is also not necessary. However, we need some way to force edges to "shift" over time in a sufficiently uncorrelated manner that our system cannot be "stuck" in some odd state (e.g. the alternating sequence seen in Figure 1.3). With dynamic networks, there are many pitfalls, since all dynamic networks are inherently directed (with respect to time).

Summary Graphs

The notion of graph summarization is introduced in Section 6.1.2, but very little is known about summary graphs. In this work, we have presented a few ideas about certain particular summarization techniques, but we do not have a general operating theoretical framework to construct or analyze summarization and its further connections to linear algebra or analysis. The general idea of graph summarization has perhaps too little structure to generate meaningfully interesting results. Given a particular summary function f, there are many questions to ask:

1. How efficiently can we compute a summary?
2. Can we do summary in an online manner, i.e. as update our summary as we process new graphs?
3. What are the intrinsic invariants that are preserved by a summary function?
4. How sensitive is a summary function to the input sequence?
5. What is the physical or natural interpretation of a summary given the underlying graph system?

Even solving some of these problems with particular classes of summary functions would provide insight into this potentially rich area of study.

Hypergraphs

Our systems are intricately tied to hypergraphs and we did not provide a treatment of these abstract structures (essentially set systems) in this dissertation. However, there are a wide range of open questions about hypergraphs that provide insight into systems of dynamic graphs. For our purposes, we can define a hypergraph as follows:

Definition 8.1.1 (Finite Hypergraph)**.** Let V be some finite vertex set.

A **hypergraph** H is a tuple $H = (V, E)$ such that $E \subseteq \mathcal{P}(V)$, where $\mathcal{P}(V)$ is the powerset of V.

From here, we can equip our hypergraph with attributes to create the attributed hypergraph. This structure, as seen in Definition 8.1.2, provides a more general way of reasoning about temporal structures. We have a very general framework of what constitute vertices, edges, and attributes; perhaps unsurprisingly, it is hard to something in general about these structures. However, there are mainly particular constructions, like line graphs, that are useful[175].

Definition 8.1.2 (Attributed Hypergraph)**.** An **attributed hypergraph** is a system

$$H = (V, E, X, \phi, Y, \epsilon)$$

where (V, E) is a hypergraph, X, Y are the *vertex attribute* and *edge attribute* sets respectively, and $\phi : V \to X, \epsilon : E \to Y$ are *attribute functions*.

The next major area of research in hypergraphs within the context of dynamic networks would be to construct a consistent mechanism for classifying and characterizing the different types of dynamic structures. As seen in Section 6.1.2, there are many types of dynamic networks. Hypergraphs may be a useful mathematical structure to unify the different characterization of dynamic networks[113].

8.1.2. Graph Renormalization

As in other dynamic systems, one interesting strategy of clustering for dynamic networks is through renormalization. Stemming from a sequence of work[47,48,49], I implemented the renormalization algorithm in[49] to ascertain its clustering capability. While it is not a mature practical technique, the results of clustering a dynamic graph are interesting.

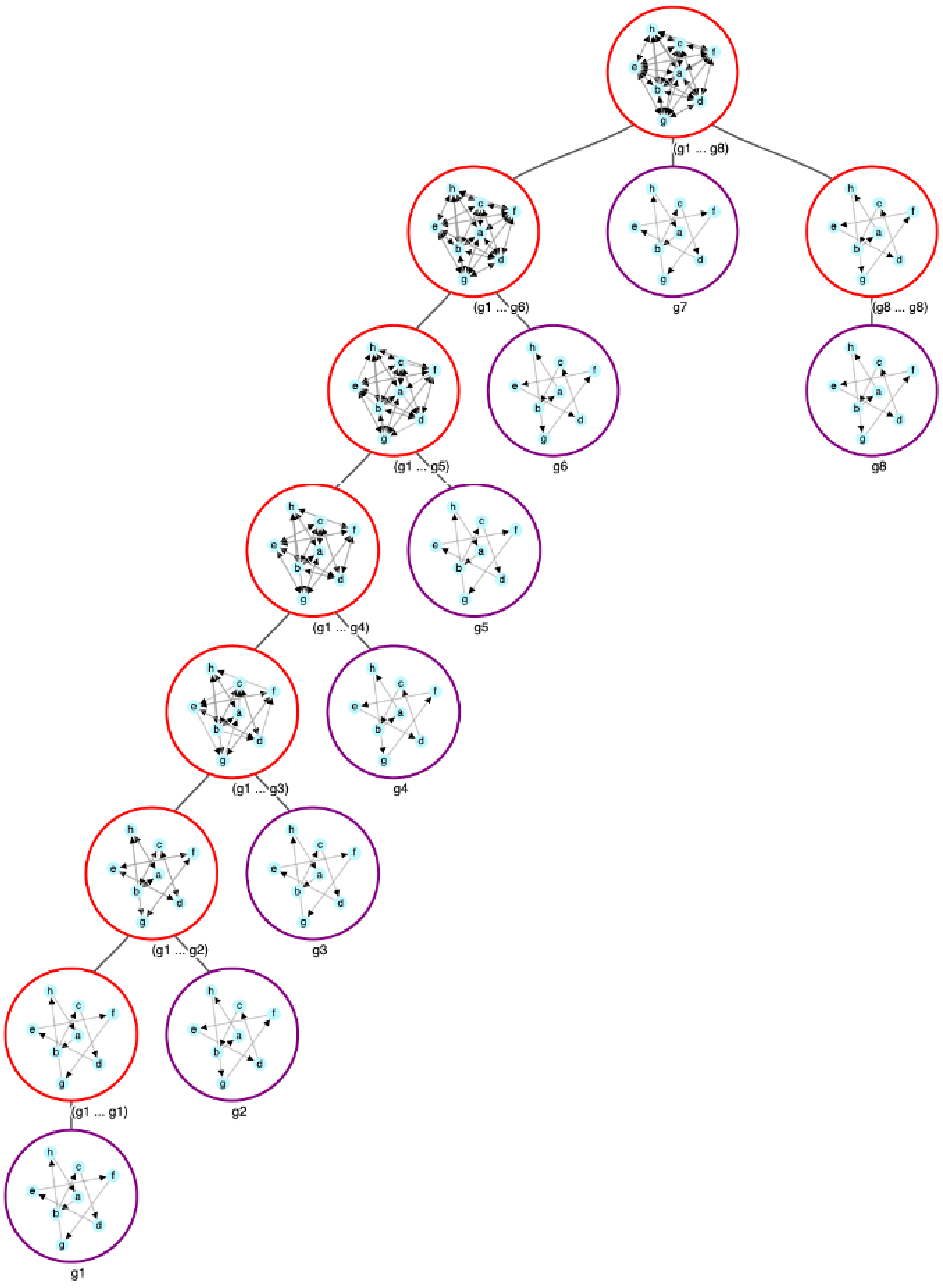

Figure 8.1: A simple graph sequence parsed via renormalization.

We can see in Figure 8.1 that renormalization produces a tree (or graph) of graphs, where each node is itself a filtered version of a parent graph. This technique allows us to see important temporal changes like a phylogenetic tree: the substantial changes are grouped together.

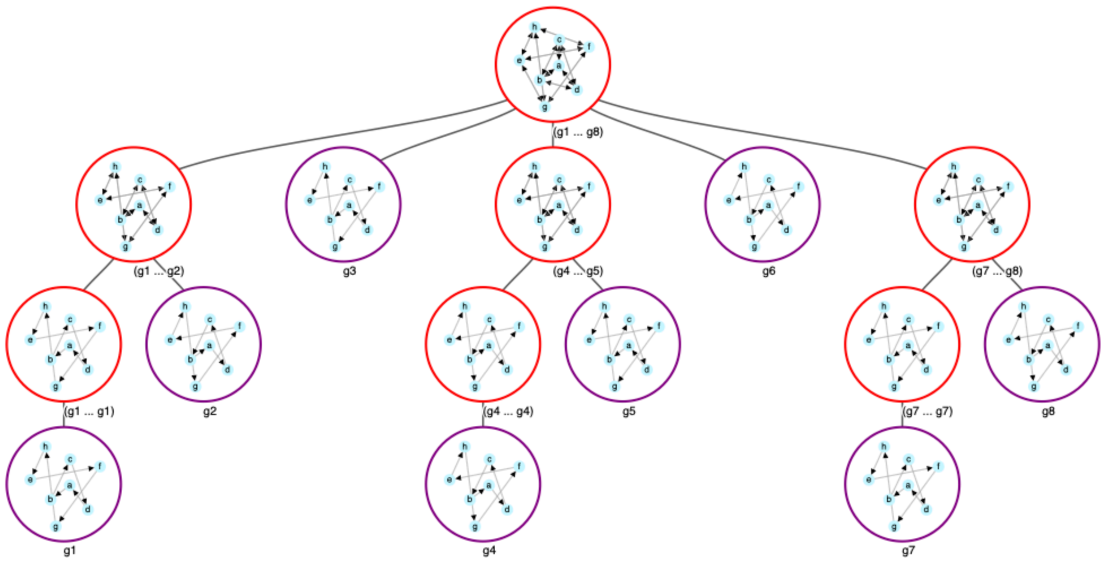

Figure 8.2: The alternating graph sequence with eight nodes (analogous to the graph in Figure 1.3) parsed with renormalization.

The alternating graph produces the renormalization in Figure 8.2, which nicely extracts the periodicity of this graph. In all, this technique may be a way to achieve a coherent and stable clustering of dynamic networks, which would fit into a larger scope of research into spatiotemporal clustering techniques.

One particular technique of interest is *spatiotemporal k-means*[75], which may have a natural extension to dynamic networks. In particular, this technique can cluster spatiotemporal data, but it may have promise for data with an underlying dynamic network structure or where the underlying data itself is a set of dynamic networks.

8.2. Connections to Related Disciplines

8.2.1. Connections to Applied Topology

Spatiotemporal data is fundamental to applied topology, especially with the Topological Data Analysis (TDA) literature[81]. Much work has been done to create both theoretically sound and practically useful algorithms in analyzing, clustering, parsing, and understanding spatiotemporal data. At its core, TDA techniques uncover the homologies of a well-chosen simplicial complex, but there is much ingenuity in the particular implementations. Certain constructions produce triangulations that shift over time (which are inherently dynamic networks) and underpin the techniques in Section 5.1. However, there are many opportunities for further connections to current developments in TDA.

While each is its own exciting research area, the key thematic areas of interest for dynamic networks within TDA are multiparameter persistence, sliding window embeddings, and trajectory analysis. Additionally, these techniques also provide featurizations for downstream technique like machine learning[202].

8.2.2. Connections to Machine Learning

While we demonstrated applications of machine learning to dynamic networks in Section 5.2, we have much work to do in this area. Indeed, graph neural networks are an important class of machine learning models, but are not fully studied, especially for spatiotemporal networks. There is no general featurization system of dynamic networks and even recent approaches are *ad hoc*.

The next step in this area of research would be to focus on the creation of large and general embedding models for graphs, just as exist in *natural language processing (NLP)* and *computer vision (CV)* as disciplines. Indeed, we could imagine some generic and large model that can extract relevant features within a dynamic network system. From there, we could apply this large network to generate embeddings for some data that could then be used in some fine-tuned, zero-shot, or few-shot downstream model.

One approach that we do not explore here is *reinforcement learning (RL)*. It would be interesting to have a study on a dynamical system defined over dynamic networks and to solve some optimization problem within this context. Indeed, RL is a general approach that can be applied to both NLP and CV problems, but its application to dynamic networks may be able to quickly resolve difficult optimization problems over spatiotemporal networks. One particular area of application would be in network design: the objective would be to design a dynamic network that drives some communication or signaling network.

8.3. Application Areas of Interest

There are a wide variety of applications of interest: societally interesting or otherwise popular areas of study that could benefit from their contextualization in dynamic networks. While this dissertation is mute on some of these subjects, there are many areas worth of exploration that could be of interest. Some of these areas have been studied in the course of producing this work, but are not yet explicitly represented.

8.3.1. Applications to Biology

One application area of interest is in embryology. As a mammalian embryo develops, cells split and shift. In particular, we can induce a dynamic network where each cell is a node and, when a cell splits, we generate two new nodes. Then, we can create a heterogenuous set of edges that capture

1. If two cells are physically adjacent or touching
2. Any chemical signals sent from one cell to another
3. The ancestral information from a parent cell to its descendants

A snapshot of this system can be seen in Figure 8.3. Indeed, this setting is quite difficult as there are a variable number of nodes and there are three types of networks. However, this system is usually small and presents a natural example of a dynamic network with heterogenuous data across edges. It is a fascinating application with potentially rich mathematics behind it.

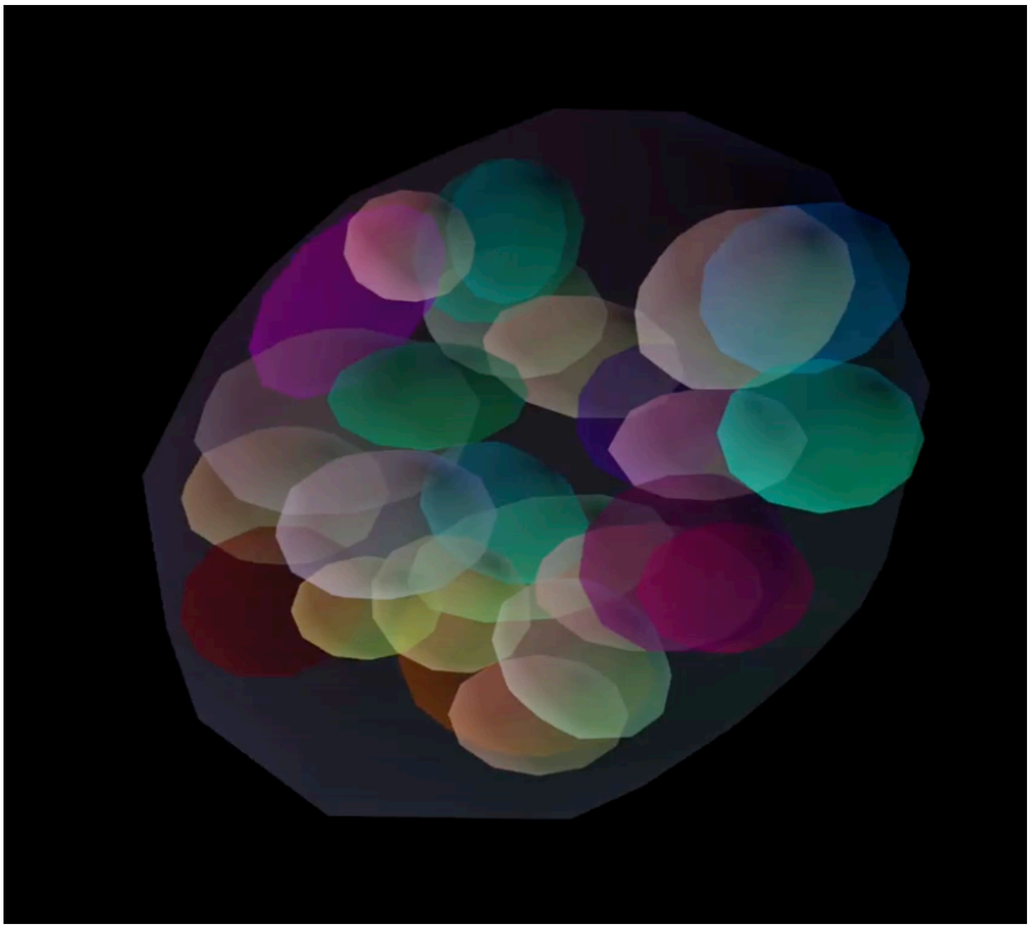

Figure 8.3: A snapshot of a mouse embryo in development. It displays a variety of cells in color and their estimated geometry. Although difficult to discern from this image, it is possible to tell which cells physically adjoin each other and which cells may have inter-cell signaling. Furthermore, it is possible to track cells across time and, as they split, retain ancestry information.

8.3.2. Applications to Opinion Dynamics

Opinion dynamics[260] is a large and ever-growing field in network analysis[122,96,200] and mathematics[51,247]. The work in Section 4.1 fits into this literature. One of the key contributions is that it explicitly models a dynamic network, as opposed to dynamics that sit atop a static network. Currently, it is relatively difficult to provide a full mathematical analysis of the dynamics induced by a dynamic network system. If we could develop techniques to analyze a large class of these systems with more general tools, this type of framework would be a major contribution to the field of opinion dynamics.

8.3.3. Applications to Transit Networks

One of the most interesting types of dynamic network is the transit network. The *General Transit Feed Specification (GTFS)* is a system for reporting consistent transit data and is used by more than 1 800 transit agencies[*]. Indeed, this system is a powerful tool for a wide range of engineering applications, e.g. automated transit-based directions.

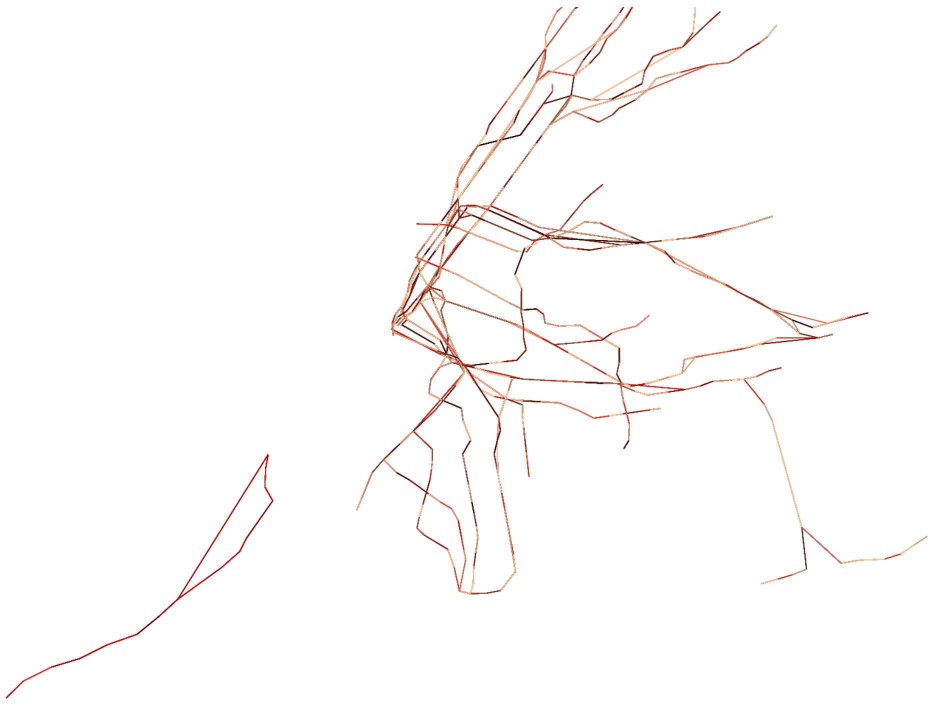

Figure 8.4: A snapshot of the subway system in New York City. Each line represents a subway line and the colors with varying intensity display the congestion across these pathways.

Figure 8.4 gives an example of this data for New York City. Using this type of data, we can develop routing techniques that handle dynamic transit networks. In particular, the dynamics of this network come from:

- Different transit frequencies at different times of the day, different days of the week, and different parts of the year.
- Variable transit frequencies based on traffic and congestion.

[*]See: `https://gtfs.org/about/`

- Closures, exceptional cases, delays, and other practical changes in network frequency.
- Any route changes, updates, or modifications.

While this network may not evolve as rapidly as those seen in Chapter 5, transit networks are still an example of a dynamic network. In particular, transit networks tend to exhibit periodic and quasi-periodic behavior that are certainly an interesting property that arises naturally in this context. Moreover, transit networks have ample data, which may make them tractable to voluminous experimentation and machine learning.

I visualize a time when we will be to robots what dogs are to humans, and I'm rooting for the machines.

Claude Elwood Shannon

A
Code

FORMAL language theory has a distinguished legacy at Princeton University, including contributions from Gödel, Church, and Turing. In contemporary applied mathematics, beyond the written word of proofs, the pre-eminent formal language is code. Much of the content of this dissertation has been generated with code; we would like to represent that work here. In this Appendix, we provide insight into both the function and form of our engineering efforts, as well as links to repositories where all of the code can be found. It would be unhelpful to replicate every line of code in the text of this document, so we instead take this opportunity to demonstrate some concept, idea, or design pattern within each package to illustrate how our code is generally structured. The full code is available within the linked repositories.

As a general matter, we try to maintain best practices for managing software-based projects. We use source control (`git`), Github, automated documentation, testing, and static analysis tools (e.g. linters). With the advent of sophisticated language models, we also heavily relied on `GPT-4`[218] and Github Copilot[101] for code-writing assistance. (In fact, `GPT-4` wrote the BibTeX for citing itself!) Using these tools, we are able to maintain consistent, relatively readable codebases that can be shared and maintained beyond the scope of the original projects.

We also use compute power from a variety of sources. Some of the simulations in this work are performed on computational resources managed and supported by Princeton Research Computing, a consortium of groups including the Princeton Institute for Computational Science and Engineering (PICSciE) and the Office of Information Technology's High Performance Computing Center and Visualization Laboratory at Princeton University. Additionally, we leverage Microsoft Azure and Google Cloud Compute, in addition to local computing resources, for our computing needs.

Note: in the following code snippets, a hyphen - appears as a tilde ~ for readability.

A.1. Viral Networks

A.1.1. `fb-paper`

See: `https://github.com/DbCrWk/fb-paper`

We have produced two packages for analyzing the spread of viral information. The first contains the figures and original paper seen in Section 4.1.

A.1.2. `trasir`

See: `https://github.com/DbCrWk/trasir`

This package leverages current paradigms in `python` development. It is managed via `poetry` and contains the latest (as of this writing) stack of tools, as seen in Listing A.1. It powers the work in Section 4.2.

Listing A.1: `pyproject.toml` file to control the project.

```
[tool.poetry]
name = "trasir"
version = "0.1.0"
description = "TraSIR analysis"
authors = ["Dev Dabke <dev@dabke.com>"]

[tool.poetry.dependencies]
python = "^3.10"
pandas = "^1.5.0"
numpy = "^1.23.3"
jupyterlab = "^3.4.8"
ipykernel = "^6.16.0"
scipy = "^1.9.2"
```

```
torch = "^1.12.1"
dvc = {extras = ["all"], version = "^2.29.0"}
bcrypt = "^4.0.0"
rich = "^12.6.0"
ipywidgets = "^8.0.2"
matplotlib = "^3.6.1"
pydantic = "^1.10.2"
networkx = "^2.8.7"
tqdm = "^4.64.1"
pqdm = "^0.2.0"
wandb = "^0.13.5"
snakeviz = "^2.1.1"
wandb~osh = "^1.0.0"

[tool.poetry.dev~dependencies]
black = "^22.10.0"
pylint = "^2.15.3"
mypy = "^0.982"
pytest = "^7.1.3"
pytest~cov = "^4.0.0"

[build~system]
requires = ["poetry~core>=1.0.0"]
build~backend = "poetry.core.masonry.api"

[tool.pyright]
ignore = ["old"]
```

Our general approach to coding is to use classes to encapsulate data, parameters, and natural objects that reflect the structure of our mathematical tools. We have a declarative API to accessing these mathematical objects. We also leverage parallel and matrix-compute optimized code, as seen in Listing A.2.

Listing A.2: An example of parallel code using our object-oriented, declarative style of coding.

```
"""
A script to run sweeps,
i.e. testing our estimator on a range of parameters.
"""

from typing import Tuple, List

import sys
import os
import traceback
import itertools
import torch
from pqdm.processes import pqdm
import pandas as pd
import wandb

sys.path.append(os.getcwd())

from trasir.model.value.hyper import Hyper
from trasir.model.value.param import Param
from trasir.model.value.initializer
    import SinglePointInitializer
from trasir.model.yoke.runner import RunnerStaticFast
from trasir.model.estim.estimator import LossType
from trasir.model.estim.flexible_grid_estimator
    import FlexibleGridEstimatorStatic
from trasir.model.lossy.masker import OnlyCMasker
from trasir.model.lossy.loser import (
    Loser,
    MseLoser,
    MseLogLoser,
    ScaledMseLoser,
    L1Loss,
    ScaledL1Loss,
    CrossEntropyLoss,
    AbsSubsetMseLoss,
    RelSubsetMseLoss,
    AbsSubsetL1Loss,
    RelSubsetL1Loss,
)


GRID_SIZE = 4
N_JOBS = 38
N_NODES = 100
T_MAX = 1000
GRID_POINTS_EXP = 16

if __name__ == "__main__":
    print("|> EXPERIMENT: Started")
    time_stamp_as_int = int(pd.Timestamp.now().timestamp())
```

```
print(f"|> EXPERIMENT: timestamp {time_stamp_as_int}")
print(
    "|> EXPERIMENT: Config",
    "N_NODES",
    N_NODES,
    "T_MAX",
    T_MAX,
    "GRID_SIZE",
    GRID_SIZE,
    "GRID_POINTS_EXP",
    GRID_POINTS_EXP,
    "N_JOBS",
    N_JOBS,
)

base_hyper = Hyper(
    N=N_NODES,
    t_max=T_MAX,
    alpha=1,
    delta=0.5
)
betas = torch.linspace(0.2, 0.8, GRID_SIZE).tolist()
rhos = torch.linspace(0.2, 0.8, GRID_SIZE).tolist()
gammas = torch.linspace(0.2, 0.8, GRID_SIZE).tolist()
loss_fns = [
    MseLogLoser(),
    L1Loss(),
    MseLoser(),
    RelSubsetMseLoss(0.2),
    AbsSubsetMseLoss(0.02),
]

test_points = [
    (time_stamp_as_int, i, base_hyper, p)
    for i, p in enumerate(
        list(itertools.product(
            betas,
            rhos,
            gammas,
            loss_fns
        ))
    )
    if p[0] > p[2]
    # beta > gamma
]
print(f"""
|> EXPERIMENT: {len(test_points)} total points
""")

print(f"""
|> EXPERIMENT: Running Sweeps
with {N_JOBS} jobs
""")
results_raw = pqdm(
    test_points, run_one_sweep, n_jobs=N_JOBS
)
```

```
print("|> EXPERIMENT: Sweeps Complete")

print(f"""
|> EXPERIMENT: Writing Results to
`results~{time_stamp_as_int}.csv`
""")
results_for_pd = [
    {
        "N": base_hyper.N,
        "t_max": base_hyper.t_max,
        "alpha": base_hyper.alpha,
        "delta": base_hyper.delta,
        "m": base_hyper.m,
        "p": base_hyper.p,
        "beta": p[0],
        "rho": p[1],
        "gamma": p[2],
        "loss_fn": str(p[3]),
        "beta_hat": est.beta,
        "rho_hat": est.rho,
        "gamma_hat": est.gamma,
        "loss": loss,
    }
    for p, est, loss in results_raw
]
df = pd.DataFrame(results_for_pd)
df.to_csv(f"results~{time_stamp_as_int}.csv")
```

A.2. Basketball

We use a few packages for our analysis of basketball data.

A.2.1. bbda

See: `https://github.com/DbCrWk/bbda`

Written in `python3`, this package "**B**asket**b**all **D**ata **A**nalysis" leverages mostly functional programming constructs to conduct the work in Section 5.1. We chain together pure *transforms* on various parts of our datasets, as might be found in contemporary frameworks like Airflow or Prefect. These transforms can be parallelized and fanned out. Listing A.3 gives an example of a transform.

Listing A.3: A pure transform for manipulating data

```python
"""A transform that extracts frames from hfpd
Given a list of hfpd, this transform can generate
a corresponding list of frames, i.e. the position
of all players for every team at a particular time.
The list is structured game clock > frames.
"""


def validate(hfpd):
    """
    The validation method for the FrameExtractionTransform
    Args:
        hfpd (List): some hfpd
    Returns:
        bool: True if data is valid. False otherwise.
    """
    return isinstance(hfpd, list)


def apply(hfpd):
    """
    The application method for the FrameExtractionTransform
    Args:
        hfpd (List): some hfpd
    """
    return _extract_time_changes(hfpd)


def _extract_time_changes(frames):
    """
    Extracts frames by time for a given list of frames
    """
    time_change_mask = [
        _does_time_change(prev, curr)
        for (prev, curr) in zip(frames, frames[1:])
    ]

    # We need to capture the first and last entry,
    # so manually add those in
    first_idx = 0
    last_idx = len(frames)

    new_time_idx = (
        [first_idx]
        + [
            idx
            for idx, value in enumerate(time_change_mask)
            if value
        ]
        + [last_idx]
    )

    time_pairs = zip(
        new_time_idx,
        new_time_idx[1:],
    )

    frame_sets = [
        (
            frames[start_idx:end_idx],
            _get_game_clock(frames[start_idx]),
            _get_game_clock(frames[end_idx ~ 1])
        )

        for (start_idx, end_idx) in time_pairs
        if _is_number_of_frames_valid(start_idx, end_idx)
    ]

    return frame_sets

def _is_number_of_frames_valid(start_idx, end_idx):
    """
    Usually, there should be exactly 14 frames
    in a frame set, which correspond to the number
```

```
        of players on the court (plus some extras).
        Therefore, we should throw away any data that
        does not meet this criterion.
        """

        return (end_idx ~ start_idx) == 14

    def _does_time_change(hfpd_entry_a, hfpd_entry_b):
        """
        Determines, given two hfpd entries
        if the time changes.
        """

        return _get_time(hfpd_entry_a) != _get_time(hfpd_entry_b)

    def _get_time(hfpd_entry):
        """
        Given an hfpd entry
        it extracts the time.
        """

        return int(hfpd_entry[3])

    def _get_game_clock(hfpd_entry):
        """
        Given an hfpd entry, it extracts the game clock.
        """
        return float(hfpd_entry[11])

    FrameExtractionTransform = type(
        'FrameExtractionTransform',
        (),
        {'apply': apply, 'validate': validate}
    )
```

To actually run an experiment, we can apply and map transforms, as seen in Listing A.4.

Listing A.4: An example experiment that applies the map-reduce transform paradigm.

```
"""An experiment"""

def main():
    """Main experiment"""

    from bbda import Transformer
    from bbda import CsvImportTransform
    from bbda import PossessionSegmentationTransform
    from bbda import FrameExtractionTransform
    from bbda import FramePossessionAggregationTransform
    from bbda import PossessionPlayerVelocityTransform

    transformer = Transformer()

    path = './data/events.csv'

    events = transformer.transform_apply(
    path,
    CsvImportTransform
)
possessions = transformer.transform_apply(
    events,
    PossessionSegmentationTransform
)

paths_hfpd = ['./data/Q1.csv', './data/Q2.csv']
hfpd = transformer.transform_map(
    paths_hfpd,
    CsvImportTransform
)
frames = transformer.transform_map(
    hfpd,
    FrameExtractionTransform
)

aggregation = transformer.transform_apply(
    (possessions, frames),
    FramePossessionAggregationTransform
)

transformer.transform_map(
    aggregation,
    PossessionPlayerVelocityTransform
)
```

```
if __name__ == "__main__":
    main()
```

A.2.2. grasket

See: https://github.com/DbCrWk/grasket, https://github.com/DbCrWk/grasket-learn, and https://github.com/DbCrWk/neograsket.

A portmanteau of "graph" and "basket," we have three packages to represent the work done in Section 5.2, which extends the functionality of our basketball data analysis tools. The original `grasket` package is written in Javascript (with `flow`) and uses object-oriented patterns. We have a full set of computational geometry tools, as seen in Listing A.5.

Listing A.5: An example of the computational geometry tools implemented in `grasket`.

```
// @flow
import { errorLib as errorGn } from '../util/logger';
import Point from './Point';
import Circle from './Circle';
import areFloatsEqual from '../util/areFloatsEqual';

const namespace = 'Object > Line';
const error = errorGn(namespace);

class Line {
    slope: number;

    intercept: number;

    areFloatsEqual: (number, number) => boolean;
```

```
static byPoints(a: Point, b: Point): Line {
    if (a.equals(b)) {
        throw error(
            '.byPoints',
            'Points cannot be identical',
            { a, b }
        );
    }

    if (a.x === b.x) {
        return new Line(Infinity, a.x);
    }

    const rise = b.y ~ a.y;
    const run = b.x ~ a.x;
    const slope = rise / run;

    const intercept = b.y ~ (slope * b.x);

    return new Line(slope, intercept);
}

constructor(slope: number, intercept: number) {
    this.slope = slope;
    this.intercept = intercept;
    this.areFloatsEqual = areFloatsEqual(0.001);
}
```

```
equals(line: Line): boolean {
    return (
        this.areFloatsEqual(this.slope, line.slope)
        && this.areFloatsEqual(
            this.intercept,
            line.intercept
        )
    );
}

hasPoint(p: Point): boolean {
    if (this.slope === Infinity) {
        return this.areFloatsEqual(
            p.x,
            this.intercept
        );
    }

    return (
        this.areFloatsEqual(
            p.y,
            this.slope * p.x + this.intercept
        )
    );
}

getPerpendicular(p: Point): Line {
    const getNewSlope = (): number => {
        if (this.slope === Infinity) return 0;
        if (this.areFloatsEqual(this.slope, 0)) {
            return Infinity;
        }
        return ~(1 / this.slope);
    };
    const newSlope = getNewSlope();
    const newIntercept = newSlope === Infinity ?
        p.x : p.y ~ (newSlope * p.x);

    return new Line(newSlope, newIntercept);
}

getLineIntersectionPoint(line: Line): Point {
    // If the lines are the same,
    // then they do not have a unique intersection point
    if (this.equals(line)) return new Point(0, 0);

    // If the lines are parallel, but not the same
    // then they do not have an intersection point
    if (this.areFloatsEqual(this.slope, line.slope)) {
        throw error(
            '.getLineIntersectionPoint',
            'Lines are parallel, but not equal
                and do not have an intersection point',
            { l: this, r: line }
        );
    }
```

```
// If this line is vertical,
// then the intersection is simple; we also
// know that both lines cannot be vertical
// because this condition has
// already been checked
if (this.slope === Infinity) {
    const x = this.intercept;
    const y = line.slope * x + line.intercept;

    return new Point(x, y);
}

// Same case above, but flipped
if (line.slope === Infinity) {
    const x = line.intercept;
    const y = this.slope * x + this.intercept;

    return new Point(x, y);
}

// The difference in slope is well~defined
// and this computation is valid
const x = (
    (line.intercept ~ this.intercept) /
    (this.slope ~ line.slope)
);
const y = this.slope * x + this.intercept;
```

```
        return new Point(x, y);
    }

    getEndPointByTravel(start: Point, travel: number): Point {
        if (!this.hasPoint(start)) {
            throw error(
                '.getEndPointByTravel',
                'Start point not on line',
                { l: this, start, travel }
            );
        }

        if (this.slope === Infinity) {
            return new Point(this.intercept, start.y + travel);
        }

        const xTravel = (
            travel /
            (Math.sqrt(1 + this.slope ** 2)
        );
        const x = start.x + xTravel;
        const y = this.slope * x + this.intercept;

        return new Point(x, y);
    }

    getClosestPoint(p: Point): Point {
```

```
    const r = this.getPerpendicular(p);
    return this.getLineIntersectionPoint(r);
}

getCircleClosestPoint(circle: Circle): Point {
    return this.getClosestPoint(circle.center);
}

getCircleIntersectionPoints(
    circle: Circle
): Array<Point> {
    const closestPoint = this.getCircleClosestPoint(
        circle
    );
    const lengthFromClosestPoint = (
        closestPoint.euclideanDistanceTo(
            circle.center
        )
    );

    if (lengthFromClosestPoint > circle.radius) {
        return [];
    }

    if (this.areFloatsEqual(
        lengthFromClosestPoint, circle.radius)
    ) {
        return [closestPoint];
        }

        const travel = Math.sqrt(
            circle.radius ** 2
            ~ lengthFromClosestPoint ** 2
        );
        return [~travel, travel].map(
            t => this.getEndPointByTravel(
                closestPoint,
                t
            )
        );
    }
}

export default Line;
```

While the code in Listing A.5 represents most of the work in this dissertation, the `neograsket` package is an updated version of this code written in `python3`. It includes a license, code of conduct, and other modern package utilities for open-source code. It similarly uses `poetry`.

Finally, note that we did train a neural network for our results in Section 5.2. We generally leveraged contemporary tools, e.g. `pytorch` and its corresponding `lightning` system. This type of code can be seen in Listing A.6.

Listing A.6: An example of the machine learning tools implemented in `grasket-learn`.

```
"""Lit Tra Transformer
This module contains the lightning implementation
of our tra transformer system.
"""
```

```python
from argparse import ArgumentParser
import torch
import pytorch_lightning as pl
import torch.nn.functional as F
from tramodel.tra_transformer import TraTransformerModel


class LitTraTransformer(pl.LightningModule):

    @staticmethod
    def add_model_specific_args(parent_parser):

        parser = ArgumentParser(
            parents=[parent_parser],
            add_help=False
        )

        parser.add_argument(
            '~~num_tokens',
            type=int,
            default=2000
        )
        parser.add_argument(
            '~~embedding_dimension',
            type=int,
            default=20
        )
```

```
parser.add_argument(
    '~~n_head',
    type=int,
    default=2
)
parser.add_argument(
    '~~n_hidden_dimension',
    type=int,
    default=200
)
parser.add_argument(
    '~~n_layer',
    type=int,
    default=2
)
parser.add_argument(
    '~~dropout',
    type=float,
    default=0.2
)
parser.add_argument(
    '~~learning_rate',
    type=float,
    default=1e~4
)
parser.add_argument(
    '~~hidden_frames',
    type=int,
            default=20
        )

        return parser

    def __init__(self, conf):
        super().__init__()
        self.model = TraTransformerModel(
            conf.num_tokens,
            conf.embedding_dimension,
            conf.n_head,
            conf.n_hidden_dimension,
            conf.n_layer,
            conf.dropout
        )

        self.hparams = conf

    def forward(self, src, tgt):
        return self.model(src, tgt)

    def training_step(self, batch, batch_idx):
        src, tgt = batch

        shift = 1
        tgt_input_start_idx = 0
        tgt_input_end_idx = (
            self.hparams.sequence_length
    ~ shift
)
tgt_output_start_idx = shift
tgt_output_end_idx = self.hparams.sequence_length

cast_tgt = tgt.float()
tgt_input = cast_tgt[
    :,
    tgt_input_start_idx:tgt_input_end_idx,
    :
]
tgt_output = cast_tgt[
    :,
    tgt_output_start_idx:tgt_output_end_idx,
    :
]

output = self(
    src.transpose(0, 1),
    tgt_input.transpose(0, 1)
)
loss = F.mse_loss(
    output.transpose(0, 1),
    tgt_output
)

self.log('train_loss', loss)
return loss
```

```
def validation_step(self, batch, batch_idx):
    src, tgt = batch
    cast_tgt = tgt.float()
    loss = 0

    for i in range(self.hparams.hidden_frames):
        tgt_input = cast_tgt[
            :,
            i:(
                self.hparams.sequence_length
                ~ self.hparams.hidden_frames
                + i ~ 1
            ),
            :
        ].transpose(0, 1)
        tgt_output = cast_tgt[
            :,
            (i + 1):(
                self.hparams.sequence_length
                ~ self.hparams.hidden_frames
                + i
            ),
            :
        ].transpose(0, 1)
        output = self(src.transpose(0, 1), tgt_input)

        frame_loss = F.mse_loss(output, tgt_output)
```

```
            loss += frame_loss

        self.log('val_loss', loss)
        return loss

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(
            self.parameters(),
            lr=self.hparams.learning_rate
        )

        return optimizer
```

A.3. Code in Space

At NASA, we used several codebases to produce the results in Chapter 6.

A.3.1. TropicALU

See: https://github.com/jacleveland/tropicalu

This project implements a tropical geometry ALU, based in `verilog`. The implementation is relatively straightforward, but the key idea is that even exotic mathematical structures can be directly instantiated in hardware. The important lesson with this project is that hardware acceleration could provide a viable path forward for making what would ordinarily expensive in software much faster with a direct hardware implementation.

A.3.2. `sumgraph` and Network Tools

See: https://github.com/DbCrWk/sumgraph and https://github.com/jacleveland/joswig.

The Joswig Algorithm described in Section 6.2.3[147] is implemented in the `joswig` codebase. The `sumgraph` codebase is the reference implementation of the techniques described in Section 6.1.2. The `sumgraph` codebase implements graph summarization in a modern, functional way. It builds upon lessons learned in other codebases referenced in this dissertation and is written entirely in `python3`. It includes modern practices and robust testing, e.g. as seen in Listing A.7.

One important problem we have to solve is finding a particular integration bound to solve Equations 6.5, 6.6. In particular, given a non-negative, integrable function f, a lower bound of integration a such that $\int_a^\infty f(t)\,\mathrm{d}t = \infty$, and a target value $\tau > 0$, we want to find b such that

$$\int_a^b f(t)\,\mathrm{d}t = \tau$$

To do this, we use Algorithm 3.

Algorithm 3 An algorithm to find the upper bound of integration

```
procedure FindUpperBound($f, a, \tau$)
    $F(x) \triangleq \int_a^x f(t)\,\mathrm{d}t$
    $b' \leftarrow a, b \leftarrow a * 2$
    while $F(b) < \tau$ do
        $b' \leftarrow b$
        $b \leftarrow b * 2$
    while $F((b + b')/2) \not\approx \tau$ do
        if $F((b + b')/2) > \tau$ then
            $b \leftarrow (b + b')/2$
        else
            $b' \leftarrow (b + b')/2$
    return $(b + b')/2$
```

We introduce three lemmata about this algorithm.

Lemma 10 (Existence of Bound). Given a non-negative, integrable function f, a lower bound of integration a such that $\int_a^\infty f(t)\,\mathrm{d}t = \infty$, and a target value $\tau > 0$, there exists a b such that

$$\int_a^b f(t)\,\mathrm{d}t = \tau$$

The function $F(x) \triangleq \int_a^x f(t)\,\mathrm{d}t$. If f is positive, then b is unique.

Proof. This result follows from the Fundamental Theorem of Calculus, the definition of an integral, and the Intermediate Value Theorem. □

Lemma 11 (Algorithm 3 Correctness). Algorithm 3 is correct.

Proof. The proof sketch is simple: the integral is monotonically increasing (weakly or strongly), we use exponential backoff to find a maximum upper bound, and then binary search to find the exact bound. Binary search works because F is monotonic from Lemma 10. □

Lemma 12 (Algorithm 3 is $\mathcal{O}(\log(b-a))$). Algorithm 3 runs in $\mathcal{O}(\log(b-a))$ assuming a constant-time integration solver[*].

Proof. Exponential backoff will find an upper bound on b within time proportional to the log of $b-a$. We then perform binary search between some value at most twice b and some value between a and b. □

Listing A.7: An example of the robust testing implemented in `sumgraph`.

```
"""
This module tests the find_integral_bound function.
"""


from math import sqrt, inf
from typing import Callable
import pytest
from sumgraph.helper.find_integral_bound
    import find_integral_bound



def test_basic():
    """
    Perform a simple test
    """


    integrable_function:
        Callable[[float], float] = lambda x: x
```

[*]This assumption is not very reasonable, but we cannot control the time of an integration solver. Therefore, our runtime gives the overhead of just our algorithm. There are many numerical methods to solve an integral with the simplest being the Trapezoid Rule with Tai's Method[272,196] or some much more complicated quadrature technique.

```
    lower_bound = 0
    target_value = 1

    expected_upper_bound = sqrt(2)
    actual_upper_bound = find_integral_bound(
        integrable_function=integrable_function,
        lower_bound=lower_bound,
        target_value=target_value,
        numerical_options={
            "tolerance": 0.0000001
        },
    )

    assert (
        pytest.approx(actual_upper_bound)
            == expected_upper_bound
    )


def test_indicator_fn():
    """
    Check an integrable indicator function
    """

    def integrable_function(
        input_x: float
    ) -> float:
        """
        A basic indicator function
        """
        if 5 <= input_x <= 10:
            return 1

        return 0

    lower_bound = 0
    target_value = 1

    expected_upper_bound = 6
    actual_upper_bound = find_integral_bound(
        integrable_function=integrable_function,
        lower_bound=lower_bound,
        target_value=target_value,
        numerical_options={
            "tolerance": 0.0000001
        },
    )

    assert (
        pytest.approx(actual_upper_bound)
            == expected_upper_bound
    )


def test_infinite_bound():
    """
    Ensure that infinity is returned when
    the bound would exceed the max bound
    """

    integrable_function:
        Callable[[float], float] = lambda x: x

    lower_bound = 0
    target_value = 100

    bound = find_integral_bound(
        integrable_function=integrable_function,
        lower_bound=lower_bound,
        target_value=target_value,
        numerical_options={
            "max_upper_bound": 5
        },
    )

    assert bound == inf


def test_max_iterations():
    """
    Ensure that a proper warning is raised
    when max iterations would be reached
    """
    integrable_function:
        Callable[[float], float] = lambda x: x

    lower_bound = 0
    target_value = 100

    with pytest.raises(RuntimeWarning):
        find_integral_bound(
            integrable_function=integrable_function,
            lower_bound=lower_bound,
            target_value=target_value,
            numerical_options={
                "max_iterations": 1
            },
        )
```

The `sumgraph` package is open-sourced and contains proper licensing information, contributing guidelines, and documentation.

A.4. Exploratory Code

The last two codebases worth highlighting are for exploratory code.

A.4.1. `graph-norm`

See: `https://github.com/DbCrWk/graph-norm`.

Again, this codebase follows the best practices for developing code as seen in other codebases. However, this codebase also defines well-structured JSON-based schemas for annotating data structures and type information. This technique allows for the easy interchange and validation of data, especially in transit. This type of schema can be found in Listing A.8. Modern tools, e.g. VS Code, can automatically detect URLs to these schemas and parse them for powerful automated validation.

Listing A.8: An example of a well-defined JSON schema for defining data types.

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "$id": "https://raw.githubusercontent.com/
        DbCrWk/graph-norm/development/schema/
        parse.tree/1.1.0.json",
    "version": "1.1.0",
    "title": "Grano Parse Tree",
    "description": "A grano file to describe a parse tree",
    "type": "object",
    "properties": {
        "label": {
            "description": "An identifiable text identifier
                for this tree",
            "type": "string"
```

```
},
"version": {
    "description": "The version that the
        parse.tree should be checked against",
    "type": "string",
    "enum": [
        "1.1.0"
    ]
},
"vertices": {
    "description": "A set of vertices to add
        to all entries of the graph sequence",
    "type": "array",
    "items": {
        "$ref": "#/definitions/vertex"
    },
    "minItems": 0,
    "uniqueItems": true
},
"graphs": {
    "description": "A set of graphs that
        can be easily referenced",
    "type": "object",
    "patternProperties": {
        "^[A~Za~z~][A~Za~z0~9_~]*$": {
            "$ref": "#/definitions/graph"
        }
    },
        "uniqueItems": true
    },
    "sequence": {
        "description": "The graph sequence",
        "type": "array",
        "items": {
            "oneOf": [
                {
                    "$ref": "#/definitions/graph_with_label"
                },
                {
                    "$ref": "#/definitions/graph_label"
                }
            ]
        },
        "uniqueItems": false
    },
    "temporal": {
        "$ref": "https://raw.githubusercontent.com/
            DbCrWk/graph~norm/development/schema/
            temporal.tree/1.0.0.json"
    },
    "topological": {
        "$ref": "https://raw.githubusercontent.com/
            DbCrWk/graph~norm/development/schema/
            topological.tree/1.0.0.json"
    },
    "$schema": {
        "type": "string",
        "enum": [
            "https://raw.githubusercontent.com/
                DbCrWk/graph~norm/development/schema/
                parse.tree/1.1.0.json"
        ]
    }
},
"definitions": {
    "vertex": {
        "description": "A single vertex",
        "type": [
            "number",
            "string"
        ]
    },
    "edge": {
        "description": "A single edge",
        "type": "array",
        "items": [
            {
                "$ref": "#/definitions/vertex"
            },
            {
                "$ref": "#/definitions/vertex"
            }
        ],
        "minItems": 2,
        "maxItems": 2,
        "additionalItems": false
    },
    "graph_label": {
        "description": "A string identifier
            for a graph",
        "type": "string"
    },
    "graph": {
        "type": "object",
        "properties": {
            "label": {
                "$ref": "#/definitions/graph_label"
            },
            "edges": {
                "description": "A set of edges for
                    a particular graph",
                "type": "array",
                "items": {
                    "$ref": "#/definitions/edge"
                },
                "minItems": 0,
                "uniqueItems": true
            },
            "vertices": {
                "description": "A set of vertices for
                    this graph",
                "type": "array",
```

```json
                    "items": {
                        "$ref": "#/definitions/vertex"
                    },
                    "minItems": 0,
                    "uniqueItems": true
                }
            }
        },
        "graph_with_label": {
            "$ref": "#/definitions/graph",
            "required": [
                "label"
            ]
        }
    },
    "required": [
        "label",
        "version",
        "temporal",
        "topological",
        "vertices",
        "graphs",
        "sequence"
    ],
    "additionalProperties": false
}
```

A.4.2. Omnibus

See: `https://github.com/DbCrWk/omnibus`

The Omnibus package leverages contemporary techniques in data analysis and our mathematical tools to analyze transit data. What started as a simple set of code tools to analyze MTA bus data for New York City has morphed into a general tool to study data in the *General Transit Feed Specification (GTFS)*. GTFS is a generic system for reporting transit data. The novely of this codebase is that it introduces a helpful `CLI` and includes tools for manipulating large data files.

References

[1] Aggarwal, C. & Subbian, K. (2014). Evolutionary network analysis: A survey. *ACM Comput. Surv.*, 47(1).

[2] Aggarwal, C. C. & Reddy, C. K. (2013). *Data Clustering: Algorithms and Applications*. Chapman and Hall/CRC.

[3] Ajelli, M., Gonçalves, B., Balcan, D., Colizza, V., Hu, H., Ramasco, J. J., Merler, S., & Vespignani, A. (2010). Comparing large-scale computational approaches to epidemic modeling: Agent-based versus structured metapopulation models. *BMC infectious diseases*, 10, 190.

[4] Akrida, E. C., Czyzowicz, J., Gasieniec, L., Kuszner, L., & Spirakis, P. G. (2016). Temporal flows in temporal networks. *arXiv*.

[5] Al Hanbali, A., de Haan, R., Boucherie, R. J., & van Ommeren, J.-K. (2008). A tandem queueing model for delay analysis in disconnected ad hoc networks. In K. Al-Begain, A. Heindl, & M. Telek (Eds.), *Analytical and Stochastic Modeling Techniques and Applications* (pp. 189–205). Berlin, Heidelberg: Springer Berlin Heidelberg.

[6] Alt, H. & Godau, M. (1995). Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 05(01n02), 75–91.

[7] Ameri, K. & Cooper, K. (2019). A network-based compartmental model for the spread of whooping cough in nebraska. *AMIA Joint Summits on Translational Science proceedings. AMIA Joint Summits on Translational Science*, 2019, 388–397.

[8] Angelopoulos, S., Doerr, B., Huber, A., & Panagiotou, K. (2009). Tight bounds for quasirandom rumor spreading. *Electron J Combined*.

[9] Arabi, S., Sabir, E., & Elbiaze, H. (2018). Information-centric networking meets delay tolerant networking: Beyond edge caching. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)* (pp. 1–6).

[10] Araniti, G., Bezirgiannidis, N., Birrane, E., Bisio, I., Burleigh, S., Caini, C., Feldmann, M., Marchese, M., Segui, J., & Suzuki, K. (2015). Contact graph routing in dtn space networks: overview, enhancements and performance. *IEEE Communications Magazine*, 53(3), 38–46.

[11] Artemis orbits (2013). Index of /pub/naif/themis/kernels/spk. https://naif.jpl.nasa.gov/pub/naif/THEMIS/kernels/spk/.

[12] Baccelli, F., Cohen, G., Olsder, G. J., & Quadrat, J.-P. (1992). *Synchronization and Linearity: An Algebra for Discrete Event Systems*. J. Wiley & Sons.

[13] Bai, Y., Yao, L., Wei, T., Tian, F., Jin, D.-Y., Chen, L., & Wang, M. (2020). Presumed asymptomatic carrier transmission of covid-19. *JAMA*, 323.

[14] Bainbridge, G., Hylton, A., & Short, R. (N.D.). Directional cellular sheaves for multicast network routing. unpublished.

[15] Baker, E. (2017). Redefining Basketball Positions with Unsupervised Learning. *towards data science*.

[16] Bakke Botnan, M. & Lesnick, M. (2016). Algebraic Stability of Zigzag Persistence Modules. *arXiv e-prints*, (pp. arXiv:1604.00655).

[17] Bakkelund, D. (2009). An lcs-based string metric.

[18] Ball, F., Sirl, D., & Trapman, P. (2010). Analysis of a stochastic sir epidemic on a random network incorporating household structure. *Mathematical Biosciences*, 242(2), 53–73.

[19] Bang-Jensen, J. & Gutin, G. Z. (2009). *Digraphs: Theory, Algorithms and Applications*. Springer-Verlag London.

[20] Barabási, A.-L. (2013). Network science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1987).

[21] Barab'asi, A.-L. & Oltvai, Z. N. (2004). Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2), 101–113.

[22] Barrat, A., Barthelemy, M., & Vespignani, A. (2008). *Dynamical Processes on Complex Networks*. New York, NY, USA: Cambridge University Press.

[23] Beckham, J. W. (2012). Analytics Reveal 13 New Basketball Positions. *Wired*.

[24] Bendich, P., Chin, S. P., Clark, J., Desena, J., Harer, J., Munch, E., Newman, A., Porter, D., Rouse, D., Strawn, N., & Watkins, A. (2016). Topological and statistical behavior classifiers for tracking applications. *IEEE Transactions on Aerospace and Electronic Systems*, 52(6), 2644–2661.

[25] Bezirgiannidis, N. & Tsaoussidis, V. (2014). Predicting queueing delays in delay tolerant networks with application in space. In A. Mellouk, S. Fowler, S. Hoceini, & B. Daachi (Eds.), *Wired/Wireless Internet Communications* (pp. 228–242). Cham: Springer International Publishing.

[26] Bialkowski, A., Park, S., Carr, G. P., Matthews, I., & Yue, Y. (2018). Analysis Of Team Behaviors Using Role And Formation Information.

[27] Biswas, A., Srinivasan, M., Rogalin, R., Piazzolla, S., Liu, J. Y.-C., Schratz, B. C., Wong, A., Alerstam, E., Wright, M. W., Roberts, W. T., Kovalik, J. M., Ortiz, G., Na-Nakornpanom, A., Shaw, M. D., Okino, C., Andrews, K. S., Peng, M. Y., Orozco, D. S., & Klipstein, W. M. (2017). Status of nasa's deep space optical communication technology demonstration. *2017 IEEE International Conference on Space Optical Systems and Applications (ICSOS)*, (pp. 23–27).

[28] Bjerkevik, H. (2021). On the stability of interval decomposable persistence modules. *Discrete & Computational Geometry*, 66.

[29] Bloch, F., Jackson, M. O., & Tebaldi, P. (2021). Centrality measures in networks. *Arxiv.org*.

[30] Bobrowski, O. & Kahle, M. (2017). The topology of probability distributions on manifolds. *IEEE Trans. Netw. Sci. Eng.*, 4, 215.

[31] Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1), 1–41.

[32] Brauer, F., Driessche, P., & Wu, J. (2008). *Mathematical Epidemiology*. Springer Berlin, Heidelberg, 1 edition.

[33] Burkart, J., II, D. C., Jansen, C., Taylor, A., & O'Keefe, A. (2006). Graphs: graphs and directed graphs (digraphs). Version 0.3.2. A *Macaulay2* package available at https://github.com/Macaulay2/M2/tree/master/M2/Macaulay2/packages.

[34] Burleigh, S. (2007). Interplanetary overlay network: An implementation of the dtn bundle protocol. *2007 4th IEEE Consumer Communications and Networking Conference*, (pp. 222–226).

[35] Byambasuren, O., Cardona, M., Bell, K., Clark, J., Mclaws, M., & Glasziou, P. (2020). Estimating the extent of asymptomatic covid-19 and its potential for community transmission: Systematic review and meta-analysis. *Journal of the Association of Medical Microbiology and Infectious Disease Canada*.

[36] Bárány, I. & Füredi, Z. (1987). Computing the volume is difficult. *Discrete & Computational Geometry*, 2.

[37] Cabacas, R. & Ra, I. (2013). Evaluating mobility models in delay tolerant network. In *2013 International Conference on IT Convergence and Security (ICITCS)* (pp. 1–4).

[38] Caceres, R. S. & Berger-Wolf, T. Y. (2012). The measurement of time-varying networks. In *Temporal Networks* (pp. 147–169).: Springer.

[39] Cahill, L., Haier, R. J., Fallon, J., Alkire, M. T., Tang, C., Keator, D., Wu, J., & McGaugh, J. L. (1996). Amygdala activity at encoding correlated with long-term, free recall of emotional information. *Proceedings of the National Academy of Sciences*, 93(15), 8016–8021.

[40] Carlson, S. C. (2022). Graph theory.

[41] Carlsson, G., de Silva, V., & Morozov, D. (2009). Zigzag persistent homology and real-valued functions. In *Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry*, SCG '09 (pp. 247–256). New York, NY, USA: Association for Computing Machinery.

[42] Casteigts, A., Flocchini, P., Quattrociocchi, W., & Santoro, N. (2010). Time-varying graphs and dynamic networks. *CoRR*, abs/1012.0009.

[43] CelesTrak (2001). Artemis. https://celestrak.com/satcat/tle.php?INTDES=2001-029.

[44] CelesTrak (2021). Search of satcat. https://celestrak.com/satcat/search-results.php.

[45] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., & Weiss, H. (2007). RFC 4838, Delay-Tolerant Networking Architecture. *IETF Network Working Group*.

[46] Chan, J. M., Carlsson, G., & Rabadan, R. (2014). Topology of viral evolution. *Bioinformatics*, 30, 98.

[47] Chazelle, B. (2015). Algorithmic renormalization for network dynamics. *IEEE Transactions on Network Science and Engineering*, 2(1), 1–16.

[48] Chazelle, B. (2019). Some observations on dynamic random walks and network renormalization. In L. A. Gąsieniec, J. Jansson, & C. Levcopoulos (Eds.), *Fundamentals of Computation Theory* (pp. 18–28). Cham: Springer International Publishing.

[49] Chazelle, B. (2020). On the periodicity of random walks in dynamic networks. *IEEE Transactions on Network Science and Engineering*, 7(3), 1337–1343.

[50] Chen, C., Wang, M., Zhang, M., Liu, Y., Wu, Y., & Wu, C. (2020). Dynamic graph collaborative filtering. *arXiv preprint arXiv:2005.02858*.

[51] Chen, M. F. & Rácz, M. Z. (2022). An adversarial model of network disruption: Maximizing disagreement and polarization in social networks. *IEEE Transactions on Network Science and Engineering*, 9(2), 728–739.

[52] Cheng, A. (2017). Using Machine Learning to Find the 8 Types of Players in the NBA. *Fastbreak Data*.

[53] Chierichetti, F., Lattanzi, S., & Panconesi, A. (2010). Almost tight bounds for rumour spreading with conductance. *Proceedings of the forty-second ACM symposium on Theory of computing*, (pp. 399–408).

[54] Choi, P. P. & Hebert, M. (2006). Learning and predicting moving object trajectory: A piecewise trajectory segment approach. *Robotics Institute, Carnegie Mellon University*.

[55] Clare, L., Burleigh, S., & Scott, K. (2010). Endpoint naming for space delay / disruption tolerant networking. In *2010 IEEE Aerospace Conference* (pp. 1–10).

[56] Cleveland, J., Hylton, A., Short, R., Mallery, B., Green, R., Curry, J., Dabke, D. V., & Freides, O. (2022). Introducing tropical geometric approaches to delay tolerant networking optimization. In *2022 IEEE Aerospace Conference (AERO)* (pp. 1–11).

[57] Cleveland, J., Mallery, B., Hylton, A., & Short, R. (Nov 2021). Planting a flag in the tropics the essential tropical geometric background for networking applications.

[58] Cohan, S. (2020). Team LSTM: Player Trajectory Prediction in Basketball Games using Graph-based LSTM Networks. Master's thesis, The University of British Columbia, Vancouver, British Columbia, Canada.

[59] Cohen, J. (1992). Infectious diseases of humans: Dynamics and control. *JAMA: The Journal of the American Medical Association*, 268, 3381.

[60] Colledanchise, M., Parasuraman, R., & Ogren, P. (2018). Learning of Behavior Trees for Autonomous Agents. *IEEE Transactions on Games*, 11(2), 183–189.

[61] Connell, C. (2022). fips2county (dataset). https://github.com/ChuckConnell/articles/blob/master/fips2county.tsv. Accessed October 9, 2022.

[62] Cui, P., Yu, Z., Zhu, S., & Gao, A. (2013). Real-time navigation for mars final approach using x-ray pulsars. *AIAA Guidance, Navigation, and Control (GNC) Conference*.

[63] Curry, J. (2013). Sheaves, cosheaves and applications. *arXiv*.

[64] Dabke, D. V. & Arroyo, E. E. (2016). Rumors with personality: A differential and agent-based model of information spread through networks. *SIAM Undergraduate Research Online*, 9, 453–467.

[65] Dabke, D. V. & Chazelle, B. (2021). Extracting semantic information from dynamic graphs of geometric data. In R. M. Benito, C. Cherifi, H. Cherifi, E. Moro, L. M. Rocha, & M. Sales-Pardo (Eds.), *Complex Networks & Their Applications X - Volume 2, Proceedings of the Tenth International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2021, Madrid, Spain, November 30 - December 2, 2021*, volume 1016 of *Studies in Computational Intelligence* (pp. 474–485).: Springer.

[66] Daganzo, C. F. (1994). The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory. *Transportation Research Part B: Methodological*, 28(4), 269–287.

[67] Daley, D. J. & Kendall, D. G. (1965). Stochastic rumors. *IMA Journal of Applied Mathematics*, 1(1), 42–55.

[68] de Silva, V. & Carlsson, G. (2006). Coordinate-free coverage in sensor networks with controlled boundaries via homology. In *Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing* (pp. 299–308).

[69] Dehne, F., Omran, M. T., & Sack, J.-R. (2012). Shortest paths in time-dependent fifo networks. *Algorithmica*, 62(1), 416–435.

[70] Deligne, P. (1977). Cohomologie etale: Séminaire de géométrie algébrique du bois-marie sga 4 1/2. In *Lecture Notes in Mathematics*, volume 569: Springer.

[71] Demmer, M. & Fall, K. (2007). Dtlsr: Delay tolerant routing for developing regions. In *Proceedings of the 2007 workshop on Networked systems for developing regions*, NSDR '07 New York, NY, USA: Association for Computing Machinery.

[72] Dijkstra, E. W. et al. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1), 269–271.

[73] Ding, X., Huang, S., Leung, A., & Rabbany, R. (2021). Incorporating dynamic flight network in seir to model mobility between populations. *Applied Network Science*, 6.

[74] Doerr, B., Fouz, M., & Friedrich, T. (2012). Why rumors spread so quickly in social networks. *Communications of the ACM*, 55(6), 70–75.

[75] Dorabiala, O., Webster, J., Kutz, N., & Aravkin, A. (2022). Spatiotemporal k-means. https://arxiv.org/abs/2211.05337. pre-print.

[76] Dottori, M. & Fabricius, G. (2014). Sir model on a dynamical network and the endemic state of an infectious disease. *Physica A: Statistical Mechanics and its Applications*, 434.

[77] DSG orbit (2018). Index of /pub/naif/misc/more_projects/dsg. https://naif.jpl.nasa.gov/pub/naif/misc/MORE_PROJECTS/DSG/.

[78] Dunbar, R. (1992). Neocortex size as a constraint on group size in primates. *Journal of Human Evolution*, 22(6), 469 – 493.

[79] Dyer, M., Frieze, A., & Kannan, R. (1988). A random polynomial time algorithm for approximating the volume of convex bodies. *Journal of the ACM*.

[80] Edelsbrunner, H. & Harer, J. (2008). Persistent homology—a survey. *Discrete & Computational Geometry - DCG*, 453.

[81] Edelsbrunner, H. & Harer, J. (2010). *Computational Topology: An Introduction*. American Mathematical Society.

[82] Edelsbrunner, H., Letscher, D., & Zomorodian, A. (2002). Topological persistence and simplification. *Found. Comput. Math.*, 2, 1.

[83] Edmunds, W. J., Kafatos, G., Wallinga, J., & Mossong, J. R. (2006). Mixing patterns and the spread of close-contact infectious diseases. *Emerging Themes in Epidemiology*, 3(1), 10.

[84] Eiter, T. & Mannila, H. (1994). *Computing Discrete Frechet Distance*. Technical report, Technische Universität Wien, Paniglgasse 16, 1040 Wien.

[85] Eletreby, R., Zhuang, Y., Carley, K., Yagan, O., & Poor, H. V. (2020). The effects of evolutionary adaptations on spreading processes in complex networks. *Proceedings of the National Academy of Sciences*, 117, 201918529.

[86] Estrada, E. & Higham, D. J. (2010). Network properties revealed through matrix functions. *SIAM Review*, 52(4), 696–714.

[87] Euler, L. & Newman, J. R. (1960). *The Seven Bridges of Konigsberg*, volume 1, (pp. 573–580). George Allen and Unwin Ltd.

[88] Fewell, J. H., Armbruster, D., Ingraham, J., Petersen, A., & Waters, J. S. (2012). Basketball teams as strategic networks. *PloS one*, 7(11).

[89] Floyd, R. W. (1962). Algorithm 97: Shortest path. *Commun. ACM*, 5(6), 345.

[90] Forman, R. (2001). A user's guide to discrete morse theory. *Sém. Lothar. Combin.*, 48.

[91] Foskey, M. & Booyabazooka (2022). Abstract graph corresponding to bridges of königsberg. https://commons.wikimedia.org/wiki/File:Königsberg_graph.svg.

[92] Fountoulakis, N. & Panagiotou, K. (2010). Rumor spreading on random regular graphs and expanders. *ArXiv e-prints*.

[93] Fout, A., Byrd, J., Shariat, B., & Ben-Hur, A. (2017). Protein interface prediction using graph convolutional networks. *Advances in Neural Information Processing Systems*, 30, 6530–6539.

[94] Fraire, J. A., De Jonckère, O., & Burleigh, S. C. (2021). Routing in the space internet: A contact graph routing tutorial. *Journal of Network and Computer Applications*, 174.

[95] Fraire, J. A., Madoery, P., Burleigh, S., Feldmann, M., Finochietto, J., Charif, A., Zergainoh, N., & Velazco, R. (2017). Assessing Contact Graph Routing Performance and Reliability in Distributed Satellite Constellations.

[96] Gaitonde, J., Kleinberg, J., & Tardos, E. (2020). Adversarial perturbations of opinion dynamics in networks. In *Proceedings of the 21st ACM Conference on Economics and Computation*, EC '20 (pp. 471–472). New York, NY, USA: Association for Computing Machinery.

[97] Ghrist, R. (2008). Data aggregation over networks via integration. The 5th IEEE International Conference on Autonomic Computing. Keynote.

[98] Ghrist, R. (2014). *Elementary Applied Topology*. CreateSpace Independent Publishing Platform.

[99] Ghrist, R. & Hiraoka, Y. (2011). Applications of sheaf cohomology and exact sequences to network coding. *Proc. NOLTA*.

[100] Gillespie, N. A. & Mann, L. (2004). Transformational leadership and shared values: The building blocks of trust. *Journal of Managerial Psychology*, 19(6), 588–607.

[101] GitHub & OpenAI (2021). Github copilot: Your ai pair programmer. https://copilot.github.com.

[102] Giuşcă, B. (2005). The problem of the seven bridges of königsberg. https://en.wikipedia.org/wiki/File:Konigsberg_bridges.png.

[103] Giusti, C., Pastalkova, E., Curto, C., & Itskov, V. (2016). A topological approach to simplifying the dynamics of resting-state networks in the human brain. *PLOS Comput. Biol.*, 12, e1005181.

[104] Goldblatt, R. (2006). *Topoi: The Categorial Analysis of Logic*. Dover Books on Mathematics. Dover Publications.

[105] G'omez, S., D'iaz-Guilera, A., G'omez-Garde nes, J., P'erez-Vicente, C. J., & Moreno, Y. (2013). Diffusion dynamics on multiplex networks. *Physical Review Letters*, 110(2), 028701.

[106] Gonçalves, B., Coutinho, D., Santos, S., Lago-Penas, C., Jiménez, S., & Sampaio, J. (2017). Exploring team passing networks and player movement dynamics in youth association football. *PLoS ONE*, 12(1), 1–13.

[107] Gould, P. & Gatrell, A. (1979). A structural analysis of a game: The Liverpool v Manchester united cup final of 1977. *Social Networks*, 2(3), 253–273.

[108] Goyal, P., Chhetri, S. R., & Canedo, A. (2020). Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. *arXiv preprint arXiv:2001.00596*.

[109] Graber, C. & Schwing, A. (2020). Dynamic Neural Relational Inference for Forecasting Trajectories. *Cvpr*.

[110] Grabowski, A., Kruszewska, N., & Kosinski, R. A. (2008). Properties of on-line social systems. *European Physical Journal B*, 66(1), 107–113.

[111] Grasic, S., Davies, E., Lindgren, A., & Doria, A. (2011). The evolution of a dtn routing protocol - prophetv2. In *Proceedings of the 6th ACM Workshop on Challenged Networks*, CHANTS '11 (pp. 27–30). New York, NY, USA: Association for Computing Machinery.

[112] Grayson, D. R. & Stillman, M. E. (2021). Macaulay2, a software system for research in algebraic geometry. https://math.uiuc.edu/Macaulay2/.

[113] Green, R. (2023). Personal Conversation.

[114] Green, R., Cardona, R., Cleveland, J., Ozbolt, J., Hylton, A., Short, R., & Robinson, M. (2021). Dude where's my stars: A novel topologically justified approach to star tracking. In *2021 IEEE Aerospace Conference*.

[115] Greene, D., Doyle, D., & Cunningham, P. (2010). Tracking the evolving community structure of networks. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems* (pp. 665–673).

[116] Grigor'yan, A., Lin, Y., Muranov, Y., & Yau, S.-T. (2014). Homotopy theory for digraphs. *arXiv e-prints*, (pp. arXiv:1407.0234).

[117] Gu, J., Li, W., & Cai, X. (2008). The effect of forget-remember mechanism on spreading. *The European Physical Journal B*, 62(2), 247–255.

[118] Gudmundsson, J. & Horton, M. (2017). Spatio-temporal analysis of team sports. *ACM Computing Surveys*, 50(2), 1–34.

[119] Guerin, B. & Miyazaki, Y. (2006). Analyzing rumors, gossip, and urban legends through their conversational properties. *The Psychological Record*, 56(1), 23–34.

[120] Gulyàs, A., Bìrò, J. J., Kőrösi, A., Rètvàri, G., & Krioukov, D. (2015). Navigable networks as nash equilibria of navigation games. *Nature Communications*.

[121] Gupta, C. & Wang, D. H. (2014). Evolutionary clustering and analysis of dynamic networks. *ACM Computing Surveys*, 47(2), 1–36.

[122] Hansen, J. & Ghrist, R. (2021). Opinion dynamics on discourse sheaves. *SIAM Journal on Applied Mathematics*, 81(5), 2033–2060.

[123] Har-Peled, S. (2011). *Geometric Approximation Algorithms*. Department of Computer Science, University of Illinois in Urbana- Champaign, Urbana, Illinois 61801: American Mathematical Society.

[124] Hatfield, M. (2020). Time history of events and macroscale interactions during substorms mission: Tracking the space storms responsible for triggering auroras. https://www.nasa.gov/mission_pages/themis/mission/index.html.

[125] Hegselmann, R. & Krause, U. (2002). Opinion dynamics and bounded confidence models, analysis and simulation. *Journal of Artificial Societies and Social Simulation*, 5.

[126] Hewitt, E. & Stromberg, K. (1965). *Real and Abstract Analysis*. Pringer Publishing Co., Inc.

[127] Hillar, C., Krone, R., & Leykin, A. (2021). EquivariantGB: Equivariant Groebner bases and related algorithms. Version 0.2. https://faculty.math.illinois.edu/Macaulay2/doc/Macaulay2-1.17/share/doc/Macaulay2/EquivariantGB/html/index.html.

[128] Hobbs, J., Holbrook, M., Frank, N., Sha, L., & Lucey, P. (2019). Improved Structural Discovery and Representation Learning of Multi-Agent Data. *arXiv*, (pp. 1–16).

[129] Hofbauer, J. & Sigmund, K. (1998). *Evolutionary Games and Population Dynamics*. Cambridge University Press.

[130] Holme, P. & Saram"aki, J. (2012). Temporal networks. *Physics Reports*, 519(3), 97–125.

[131] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8), 2554–2558.

[132] Hylton, A., Raible, D., & Clark, G. (2019). A delay tolerant networking-based approach to a high data rate architecture for spacecraft. In *2019 IEEE Aerospace Conference* (pp. 1–10).

[133] Hylton, A., Raible, D., Clark, G., Dudukovich, R., Tomko, B., & Burke, L. (2019). Rising above the cloud – toward high-rate delay-tolerant networking in low-earth orbit. In *37th AIAA International Communications Satellite Systems Conference* (pp. 1–8).

[134] Hylton, A. & Raible, D. E. (2016). High data rate architecture (hidra). In *34th AIAA International Communications Satellite Systems Conference*.

[135] Hylton, A., Raible, D. E., & Clark, G. (2017). On the development and application of high data rate architecture (hidra) in future space networks. In *35th AIAA International Communications Satellite Systems Conference*.

[136] Hylton, A., Short, R., Cleveland, J., Freides, O., Memon, Z., Cardona, R., Green, R., Curry, J., Gopalakrishnan, S., Dabke, D. V., Story, B., Moy, M., & Mallery, B. (2022). A survey of mathematical structures for lunar networks. In *2022 IEEE Aerospace Conference (AERO)* (pp. 1–17).

[137] Hylton, A., Short, R., Green, R., & Toksoz-Exley, M. (2020). A mathematical analysis of an example delay tolerant network using the theory of sheaves. In *2020 IEEE Aerospace Conference* (pp. 1–11).

[138] Ising, E. (1925). Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik*, 31(1), 253–258.

[139] Israel, D., Edwards, B., Hayes, J., Knopf, W., Robles, A., & Braatz, L. (2019). The benefits of delay/disruption tolerant networking for future nasa science missions. In *70th International Astronautical Congress (IAC)* (pp. 1–12).

[140] Israel, D. J., Edwards, B. L., & Staren, J. W. (2017). Laser communications relay demonstration (lcrd) update and the path towards optical relay operations. In *2017 IEEE Aerospace Conference* (pp. 1–6).

[141] Jackson, M. O. & Zenou, Y. (2014). Games on networks. *Handbook of Game Theory*, 4.

[142] Jain, M. H. & Patra, R. (2003). Implementing delay tolerant networking. *University of California, Berkeley, Computer Science Division, Berkeley, CA*, 94720.

[143] Jain, S., Fall, K., & Patra, R. (2004). Routing in a delay tolerant network. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04 (pp. 145–158). New York, NY, USA: Association for Computing Machinery.

[144] Jia, H., Ren, C., Hu, Y., Chen, Y., Lv, T., Fan, C., Tang, H., & Hao, J. (2020). Mastering Basketball with Deep Reinforcement Learning : An Integrated Curriculum Training Approach □ Extended Abstract. *Adaptive Agents and Multi-Agent Systems*.

[145] Jones, E. P. & Ward, P. A. (2006). Routing strategies for delay-tolerant networks. *Submitted to ACM Computer Communication Review (CCR)*.

[146] Joswig, M. (April 2019). Optimization and tropical geometry: Exercises and problems 1.

[147] Joswig, M. & Schröter, B. (Nov 2020). Parametric shortest-path algorithms via tropical geometry.

[148] Jørgen Bang-Jensen, G. G. (2002). *Digraphs: Theory, Algorithms and Applications*. Springer.

[149] Kamins, M. A., Folkes, V. S., & Perner, L. (1997). Consumer responses to rumors: Good news, bad news. *Journal of Consumer Psychology*, 6(2), 165–187.

[150] Karp, R., Schindelhauer, C., Shenker, S., & V'ocking, B. (2000). Randomized rumor spreading. *Foundations of Computer Science*, Proceedings of the 41st Annual Symposium on Foundations of Computer Science, 565–574.

[151] Kauffman, S. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3), 437–467.

[152] Kermack, W. O. & McKendrick, A. G. (1927). A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 115(772), 700–721.

[153] Kessinger, T. A., Tarnita, C. E., & Plotkin, J. B. (2022). Evolution of social norms for moral judgment.

[154] Kim, H. & Anderson, R. (2012). Temporal node centrality in complex networks. *Physical Review E*, 85(2), 026107.

[155] Knapp, R. H. (1944). A psychology of rumor. *Public Opinion Quarterly*, (pp. 22–27).

[156] Knopp, K. (1928). *Theory and Application of Infinite Series*. Blackie & Son, Ltd.

[157] Krishnan, S. (2014). Flow-cut dualities for sheaves on graphs. *arXiv*.

[158] Kronbichler, A., Kresse, D., Yoon, S., Lee, K., Effenberger, M., & Shin, J. I. (2020). Asymptomatic patients as a source of covid-19 infections: A systematic review and meta-analysis. *International Journal of Infectious Diseases*.

[159] Kurach, K., Raichuk, A., Stańczyk, P., Zając, M., Bachem, O., Espeholt, L., Riquelme, C., Vincent, D., Michalski, M., Bousquet, O., & Gelly, S. (2020). Google research football: A novel reinforcement learning environment. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34, 4501–4510.

[160] Kuramoto, Y. (1975). Self-entrainment of a population of coupled non-linear oscillators. In *International Symposium on Mathematical Problems in Theoretical Physics*.

[161] Landau, E. G. H. (1951). *Foundations of Analysis*. Chelsea Publishing Company.

[162] Landers, J. R. & Duperrouzel, B. (2018). Machine Learning Approaches to Competing in Fantasy Leagues for the NFL. *IEEE Transactions on Games*, 11(2), 159–172.

[163] Lau, M. S., Grenfell, B., Thomas, M., Bryan, M., Nelson, K., & Lopman, B. (2020). Characterizing super-spreading events and age-specific infectiousness of sars-cov-2 transmission in georgia, usa. *medRxiv*.

[164] Layton, A. T. & Sadria, M. (2022). Understanding the dynamics of sars-cov-2 variants of concern in ontario, canada: a modeling study. *Scientific Reports*, 12(1), 2114.

[165] Li, C., Zhu, Y., Qi, C., Liu, L., Zhang, D., Wang, X., She, K., Jia, Y., Liu, T., He, D., Xiong, M., & Li, X. (2021). Estimating the prevalence of asymptomatic covid-19 cases and their contribution in transmission - using henan province, china, as an example. *Frontiers in Medicine*, 8.

[166] Li, L., Cheng, W.-Y., Glicksberg, B. S., Gottesman, O., Tamler, R., Chen, R., Bottinger, E. P., & Dudley, J. T. (2015). Identification of type 2 diabetes subgroups through topological analysis of patient similarity. *Science Translational Medicine*, 7(311), 311ra174–311ra174.

[167] Li, M. G., Jiang, B., Zhu, H., Che, Z., & Liu, Y. (2020a). Generative Attention Networks for Multi-Agent Behavioral Modeling. *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI*, (pp. 7195–7202).

[168] Li, M. Z., Ryerson, M. S., & Balakrishnan, H. (2019). Topological data analysis for aviation applications. *Transportation Research Part E: Logistics and Transportation Review*, 128, 149–174.

[169] Li, R., Pei, S., Chen, B., Song, Y., Zhang, T., Yang, W., & Shaman, J. (2020b). Substantial undocumented infection facilitates the rapid dissemination of novel coronavirus (sars-cov2). *Science*, 368, eabb3221.

[170] Li, Y., Yu, R., Shahabi, C., & Liu, Y. (2018). Geoman: Multi-level attention networks for geo-sensory time series prediction. *IJCAI*, (pp. 3428–3434).

[171] Lin, C.-T. (1974). Structural controllability. *IEEE Transactions on Automatic Control*, 19(3), 201–208.

[172] Lindgren, A., Doria, A., Davies, E., & Grasic, S. (2012). RFC 6693: Probabilistic Routing Protocol for Intermittently Connected Networks. *IETF Network Working Group*.

[173] Liu, D. & Chen, X. (2011). Rumor propagation in online social networks like twitter: A simulation study. *Multimedia Information Networking and Security (MINES) Third International Conference*.

[174] Liu, Q.-H., Ajelli, M., Aleta, A., Merler, S., Moreno, Y., & Vespignani, A. (2018). Measurability of the epidemic reproduction number in data-driven contact networks. *Proceedings of the National Academy of Sciences*, 115, 201811115.

[175] Liu, X. T., Firoz, J., Aksoy, S., Amburg, I., Lumsdaine, A., Joslyn, C., Praggastis, B., & Gebremedhin, A. H. (2022). High-order line graphs of non-uniform hypergraphs: Algorithms, applications, and experimental analysis. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (pp. 784–794). Los Alamitos, CA, USA: IEEE Computer Society.

[176] Liu, Y.-Y., Slotine, J.-J., & Barabási, A.-L. (2011). Controllability of complex networks. *Nature*, 473, 167–173.

[177] Lloyd-Smith, J., Schreiber, S., Kopp, P., & Getz, W. (2005). Superspreading and the effect of individual variation on disease emergence. *Nature*, 438, 355–9.

[178] LRO orbit (2020). Pgda - 50,000 lro orbits. https://pgda.gsfc.nasa.gov/products/77.

[179] Lucey, P., Bialkowski, A., Carr, P., Morgan, S., Matthews, I., & Sheikh, Y. (2013). Representing and discovering adversarial team behaviors using player roles. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, (pp. 2706–2713).

[180] Lyapunov, A. M. (1992). The general problem of the stability of motion. *International Journal of Control*, 55(3), 531–534.

[181] Ma, Q., Liu, J., Liu, Q., Kang, L., Liu, R., Jing, W., Yu, W., & Liu, M. (2021). Global percentage of asymptomatic sars-cov-2 infections among the tested population and individuals with confirmed covid-19 diagnosis: A systematic review and meta-analysis. *JAMA Network Open*, 4, e2137257.

[182] Ma, W., Wang, L., Ruzzo, W. L., & Noble, W. S. (2018). Deepgs: Predicting phenotypes from genotypes using deep learning. *Bioinformatics*, 34(17), i658–i665.

[183] Maclagan, D. & Sturmfels, B. (2015). *Introduction to Tropical Geometry*, volume 161 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI.

[184] Magistretti, E., Kong, J., Lee, U., Gerla, M., Bellavista, P., & Corradi, A. (2007). A mobile delay-tolerant approach to long-term energy-efficient underwater sensor networking. In *2007 IEEE Wireless Communications and Networking Conference* (pp. 2866–2871).: IEEE.

[185] Man, H. (2017). Defining Modern NBA Player Positions - Applying Machine Learning to Uncover Functional Roles in Basketball. *Medium.com*.

[186] Mansourbeigi, S. M. (2018). Sheaf theory as a mathematical foundation for distributed applications involving heterogeneous data sets. In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)* (pp. 28–33).: IEEE.

[187] Mantegna, R. N. (1999). Hierarchical structure in financial markets. *The European Physical Journal B*, 11(1), 193–197.

[188] Mars, K. (2019). Gateway. https://www.nasa.gov/gateway.

[189] Martin, J. L. (2003). Geometry of graph varieties. *Transactions of the American Mathematical Society*, 355, 4151–4169.

[190] Mcauley, J. & Leskovec, J. (2014). Discovering social circles in ego networks. *ACM Trans. Knowl. Discov. Data*, 8(1), 4:1–4:28.

[191] McDowell, J. (2021). Starlink statistics.

[192] Mertzios, G. B., Michail, O., & Spirakis, P. G. (2015). Temporal network optimization subject to connectivity constraints. *CoRR*, abs/1502.04382.

[193] Messaoudi, A., Elkamel, R., Helali, A., & Bouallegue, R. (2017). Cross-layer based routing protocol for wireless sensor networks using a fuzzy logic module. In *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)* (pp. 764–769).

[194] Michail, O. (2015). An introduction to temporal graphs: An algorithmic perspective. *CoRR*, abs/1503.00278.

[195] Mohanta, N. (2021). How many satellites are orbiting the earth in 2021?

[196] Monaco, J. & Anderson, R. (1994). Tai's Formula Is the Trapezoidal Rule. *Diabetes Care*, 17(10), 1224–1225.

[197] Moreno, Y., Nekovee, M., & Pacheco, A. (2004). Dynamics of rumoring spread in complex networks. *Physics Review E*, 69.

[198] Morozov, D. (2017). Dionysus 2. https://mrzv.org/software/dionysus2/.

[199] Morris, B. T. & Trivedi, M. M. (2008). A survey of vision-based trajectory learning and analysis for surveillance. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(8), 1114–1127.

[200] Mossel, E., Neeman, J., & Tamuz, O. (2014). Majority dynamics and aggregation of information in social networks. *Autonomous Agents and Multi-Agent Systems*, 28(3), 408–429.

[201] Mossong, J., Hens, N., Jit, M., Beutels, P., Auranen, K., Mikolajczyk, R., Massari, M., Salmaso, S., Tomba, G. S., Wallinga, J., Heijne, J., Sadkowska-Todys, M., Rosinska, M., & Edmunds, W. J. (2008). Social contacts and mixing patterns relevant to the spread of infectious diseases. *PLoS Med*, 5(3), e74.

[202] Motta, F., Tralie, C., Bedini, R., Bini, F., Bini, G., Eramian, H., Gameiro, M., Haase, S., Haddox, H., Harer, J., Leiby, N., Marinozzi, F., Novotney, S., Rocklin, G., Singer, J., Strickland, D., & Vaughn, M. (2019). Hyperparameter optimization of topological features for machine learning applications. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)* (pp. 1107–1114).

[203] Motter, A. E., Myers, S. A., Anghel, M., & Nishikawa, T. (2013). Spontaneous synchrony in power-grid networks. *Nature Physics*, 9(3), 191–197.

[204] Moy, M., Cardona, R., Green, R., Cleveland, J., Hylton, A., & Short, R. (2020). Path optimization sheaves.

[205] Moy, M., Hylton, A., & Short, R. (2022). An alternate pathfinding algorithm for contact graph routing. in preparation.

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

[207] Muldoon, M. (2020). Tropical arithmetic and shortest paths.

[208] Murphy, D. V., Kansky, J. E., Grein, M. E., Schulein, R. T., Willis, M. M., & Lafon, R. E. (2014). LLCD operations using the Lunar Lasercom Ground Terminal. In H. Hemmati & D. M. Boroson (Eds.), *Free-Space Laser Communication and Atmospheric Propagation XXVI*, volume 8971 (pp. 250 – 256).: International Society for Optics and Photonics SPIE.

[209] NASA (2019). `https://www1.grc.nasa.gov/space/scan/scas/scenic/`.

[210] NASA (2020). Delay/disruption tolerant networking.

[211] Natarajan, S., Kunapuli, G., Judah, K., Tadepalli, P., Kersting, K., & Shavlik, J. (2010). Multi-agent inverse reinforcement learning. *Proceedings - 9th International Conference on Machine Learning and Applications, ICMLA 2010*, 10(1), 395–400.

[212] Newman, M. E. J. (2018). *Networks an introduction*. Oxford University Press.

[213] Nishiura, H., Kobayashi, T., Suzuki, A., Jung, S.-M., Hayashi, K., Kinoshita, R., Yang, Y., Yuan, B., Akhmetzhanov, A., Linton, N., & Miyama, T. (2020). Estimation of the asymptomatic ratio of novel coronavirus infections (covid-19). *International Journal of Infectious Diseases*, 94.

[214] Nogrady, B. (2020). What the data say about asymptomatic covid infections. *Nature*, 587, 534–535.

[215] Ntareme, H., Zennaro, M., & Pehrson, B. (2011). Delay tolerant network on smartphones: Applications for communication challenged areas. In *Proceedings of the 3rd Extreme Conference on Communication: The Amazon Expedition* (pp.1̃4).: ACM.

[216] Oh, O., Agrawal, M., & Rao, H. (2013). Community intelligence and social media services: A rumor theoretic analysis of tweets during social crises. *MIS Quarterly*, 37(2), 407–426.

[217] Olfati-Saber, R. & Murray, R. (2004). Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9), 1520–1533.

[218] OpenAI (2023). GPT-4: A large-scale generative language model. `https://openai.com/research/gpt-4`.

[219] Palsson, B. �. (2015). *Systems Biology: Constraint-based Reconstruction and Analysis*. Cambridge University Press.

[220] Parisi, G. I., Kemény, Z., Fülöp, G., & Erdős, G. (2019). Graph neural networks for the prediction of area-specific properties in neuronal networks. *arXiv preprint arXiv:1903.03242*.

[221] Passos, P., Davids, K., Araújo, D., Paz, N., Minguéns, J., & Mendes, J. (2011). Networks as a novel tool for studying team ball sports as complex social systems. *Journal of Science and Medicine in Sport*, 14(2), 170–176.

[222] Pastor-Satorras, R., Castellano, C., Van Mieghem, P., & Vespignani, A. (2015). Epidemic processes in complex networks. *Rev. Mod. Phys.*, 87, 925–979.

[223] Pastor-Satorras, R. & Vespignani, A. (2001). Epidemic spreading in scale-free networks. *Phys. Rev. Lett.*, 86, 3200–3203.

[224] Patania, A., Petri, G., & Vaccarino, F. (2017). The shape of collaborations. *PLOS ONE*, 12, e0176318.

[225] Pecora, L. M. & Carroll, T. L. (1998). Master stability functions for synchronized coupled systems. *Phys. Rev. Lett.*, 80, 2109–2112.

[226] Peixoto, T. P. (2014). The graph-tool python library. *figshare*.

[227] Pellis, L., Ball, F., & Trapman, P. (2012). Reproduction numbers for epidemic models with households and other social structures. i. definition and calculation of r0. *Mathematical Biosciences*, 235(1), 85–97.

[228] Pellis, L., Spencer, S., & House, T. (2015). Real-time growth rate for general stochastic sir epidemics on unclustered networks. *Mathematical Biosciences*, 265, 65–81.

[229] Petri, G., Expert, P., Turkheimer, F., Carhart-Harris, R., Nutt, D., Hellyer, P. J., & Vaccarino, F. (2013). Homological scaffolds of brain functional networks. *J. R. Soc. Interface*, 10, 20130512.

[230] Pittel, B. (1987). On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1), 213–223.

[231] Poletti, P., Tirani, M., Cereda, D., Trentini, F., Guzzetta, G., Sabatino, G., Marziano, V., Castrofino, A., Grosso, F., Del Castillo, G., Piccarreta, R., Andreassi, A., Melegaro, A., Gramegna, M., Ajelli, M., Merler, S., & Force, A. L. C.-. T. (2021). Association of Age With Likelihood of Developing Symptoms and Critical Disease Among Close Contacts Exposed to Patients With Confirmed SARS-CoV-2 Infection in Italy. *JAMA Network Open*, 4(3), e211085–e211085.

[232] Popa, A., Genger, J.-W., Nicholson, M., Penz, T., Schmid, D., Aberle, S., Agerer, B., Lercher, A., Endler, L., Colaço, H., Smyth, M., Schuster, M., Grau, M., Martinez-Jiménez, F., Pich, O., Borena, W., Pawelka, E., Keszei, Z., Senekowitsch, M., & Bergthaler, A. (2020). Genomic epidemiology of superspreading events in austria reveals mutational dynamics and transmission properties of sars-cov-2. *Science Translational Medicine*, 12, eabe2555.

[233] Ranshous, S., Harenberg, S., Sharma, C., Samatova, N. F., & Govindarajan, N. (2015). *Anomaly Detection in Dynamic Networks: A Survey*. Technical report, North Carolina State University.

[234] Richard, B. (1958). On a routing problem. *Quart. Appl. Math*, 16(1), 87–90.

[235] Rizzi, R., Reimherr, M., & Smirnov, S. (2016). Persistent homology of financial networks. *J. Complex Networks*, 5, 613.

[236] Robinson, M. (2014a). A sheaf-theoretic perspective on sampling.

[237] Robinson, M. (2014b). *Topological Signal Processing*. Mathematical Engineering. Springer Berlin Heidelberg.

[238] Robinson, M. (2016). Modeling wireless network routing using sheaves. *arXiv*.

[239] Robinson, M. (2017). Sheaves are the canonical data structure for sensor integration. *Information Fusion*, 36, 208–224.

[240] Robinson, M. (2018). Assignments to sheaves of pseudometric spaces. *arXiv preprint arXiv:1805.08927*.

[241] Robinson, M. (2019). Hunting for foxes with sheaves. *Notices of the American Mathematical Society*, 66, 661–676.

[242] Robinson, M., Capraro, C., & Praggastis, B. (2016). The pysheaf library.

[243] Rodriguez, L. (2014). Automorphism groups of simple graphs. https://www.whitman.edu/Documents/Academics/Mathematics/2014/rodriglr.pdf.

[244] Rosenberg, M. S. (2009). *Sequence Alignment: Methods, Models, Concepts, and Strategies*. University of California Press, 1 edition.

[245] Rudenko, A., Palmieri, L., Herman, M., Kitani, K. M., Gavrila, D. M., & Arras, K. O. (2020). Human motion trajectory prediction: a survey. *The International Journal of Robotics Research*, 39(8), 895–935.

[246] Rudin, W. (1953). *Principles of Mathematical Analysis*. McGraw-Hill, Inc.

[247] Rácz, M. Z. & Rigobon, D. E. (2022). Towards consensus: Reducing polarization by perturbing social networks.

[248] Sanford, R., Gorji, S., Hafemann, L. G., Pourbabaee, B., & Javan, M. (2020). Group activity detection from trajectory and video data in soccer. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (pp. 3932–3940). Los Alamitos, CA, USA: IEEE Computer Society.

[249] Schildt, S., Morgenroth, J., Pöttner, W.-B., & Wolf, L. (2011). Ibr-dtn: A lightweight, modular and highly portable bundle protocol implementation. *Electronic Communications of the EASST*, 37.

[250] Schuette, C. & Metzner, P. (2009). *Markov Chains and Jump Processes: An Introduction to Markov Chains and Jump Processes on Countable State Spaces*. Freie Universit¨at Berlin.

[251] Scott, K. & Burleigh, S. (2007). RFC 5050, Bundle Protocol Specification. *IETF Network Working Group*.

[252] Scoville, N. A. (2019). *Discrete Morse Theory*. American Mathematical Society.

[253] Segui, J., Jennings, E., & Burleigh, S. (2011). Enhancing contact graph routing for delay tolerant space networking. In *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011* (pp. 1–6).

[254] Sehgal, R. & Peyravi, H. (2015). Delay tolerant networks modeling and analysis. In *Proceedings of the 30th International Conference on Computers and Their Applications, CATA 2015* (pp. 231–236).

[255] Seo, Y., Defferrard, M., Vandergheynst, P., & Bresson, X. (2018). Structured sequence modeling with graph convolutional recurrent networks. In *NeurIPS* (pp. 8468–8478).

[256] Shang, W., Kang, L., Cao, G., Wang, Y., Gao, P., Liu, J., & Liu, M. (2022). Percentage of asymptomatic infections among sars-cov-2 omicron variant-positive individuals: A systematic review and meta-analysis. *Vaccines*, 10(7).

[257] Shea, S. & Baker, C. (2015). Preliminary investigation into defensive stretch. http://www.basketballanalyticsbook.com/2015/09/14/preliminary-investigation-into-defensive-stretch.

[258] Short, R., Green, R., Moy, M., & Story, B. (2021a). *What Type of Graph is a Contact Graph?* Technical Memorandum Upcoming, NASA, Glenn Research Center, Cleveland OH 44135, USA.

[259] Short, R., Hylton, A., Cardona, R., Green, R., Bainbridge, G., Moy, M., & Cleveland, J. (2021b). Towards sheaf theoretic analyses for delay tolerant networking. In *2021 IEEE Aerospace Conference (50100)* (pp. 1–9).

[260] Simonson, D. V. (2022). *Mathematical Theory of Opinion Dynamics with Applications*. PhD thesis, University of California, Irvine.

[261] Singh, G., Mémoli, F., & Carlsson, G. (2007). Topological methods for the analysis of high-dimensional data sets and 3d object recognition. *Int. J. Comput. Geom. Appl.*, 22, 189.

[262] Smith, A. (2014). 6 new facts about facebook. *Pew Research Center*.

[263] Snijders, T. A., van de Bunt, G. G., & Steglich, C. E. (2010a). Introduction to stochastic actor-based models for network dynamics. *Social Networks*, 32(1), 44–60.

[264] Snijders, T. A., van de Bunt, G. G., & Steglich, C. E. (2010b). Introduction to stochastic actor-based models for network dynamics. *Social Networks*, 32(1), 44–60. Dynamics of Social Networks.

[265] Spirent (2016). *TCP Network Latency and Throughput*. Technical report, Spirent.

[266] Stallmann, M. F. (2022). The 7/5 bridges of königsberg/kaliningrad. https://docs.google.com/document/d/1sj-079i802ult7C-CB_YwpX9SRYSmb1ext_q3iEKcKY.

[267] Stodden, D. & Galasso, G. (1995). Space system visualization and analysis using the satellite orbit analysis program (soap). In *1995 IEEE Aerospace Applications Conference. Proceedings*, volume 1 (pp. 369–387 vol.1).

[268] Subramanian, R., He, Q., & Pascual, M. (2021). Quantifying asymptomatic infection and transmission of covid-19 in new york city using observed cases, serology, and testing capacity. *Proceedings of the National Academy of Sciences*, 118, e2019716118.

[269] Sun, K., Wang, W., Gao, L., Wang, Y., Luo, K., Ren, L., Zhan, Z., Chen, X., Zhao, S., Huang, Y., Sun, Q., Liu, Z., Litvinova, M., Vespignani, A., Ajelli, M., Viboud, C., & Yu, H. (2020). Transmission heterogeneities, kinetics, and controllability of sars-cov-2. *Science*, 371, eabe2424.

[270] Swan, R. G. (1964). *The Theory of Sheaves*. University of Chicago Press.

[271] Swinski, J.-P. & et alii (2018). bplib. `https://github.com/nasa/bplib`.

[272] Tai, M. M. (1994). A mathematical model for the determination of total area under glucose tolerance and other metabolic curves. *Diabetes Care*, 17(2), 152––154.

[273] Tang, J. P., Lou, T., & Kleinberg, J. (2010). Analyzing and modeling dynamic processes in complex networks. *Proceedings of the 19th International Conference on World Wide Web*, (pp. 1201–1202).

[274] Tang, R. & Müller, H.-G. (2008). Pairwise curve synchronization for functional data. *Biometrika*, 95(4), 875–889.

[275] Tennison, B. R. (1975). *Sheaf Theory*. London Mathematical Society Lecture Note Series. Cambridge University Press.

[276] The COVID Tracking Project (2021). Daily data on the covid-19 pandemic for the us and individual states (dataset). `https://api.covidtracking.com/v1/states/daily.csv`. Accessed October 9, 2022.

[277] The New York Times (2022). Coronavirus (covid-19) data in the united states (dataset). `https://github.com/nytimes/covid-19-data`. Accessed October 9, 2022.

[278] Thurston, H. A. (1956). *The Number System*. Blackie & Son, Ltd.

[279] Togelius, J. (2018). IEEE Transactions on Games: A Leading Journal for Games Research. *IEEE Transactions on Games*, 10(1), 1–2.

[280] Tralie, C. J., Smith, A., Borggren, N., Hineman, J., Bendich, P., Zulch, P., & Harer, J. (2018). Geometric cross-modal comparison of heterogeneous sensor data. In *2018 IEEE Aerospace Conference* (pp. 1–10).

[281] Tripathy, R., Bagchi, A., & Metha, S. (2010). A study of rumor control stategies on social networks. *CIKM '10 Proceedings of the 19th ACM international conference on information and Knowledge manegment*, (pp. 1817–1820).

[282] Trivedi, R., Dai, H., Wang, Y., & Song, L. (2019). Representation learning over dynamic graphs. In *ICLR*.

[283] United States Census Bureau (2015a). 2011–2015 5-year acs commuting flows (dataset). https://www.census.gov/data/tables/2015/demo/metro-micro/commuting-flows-2015.html. Accessed October 9, 2022.

[284] United States Census Bureau (2015b). 2011–2015 acs 5-year estimates (dataset). https://www.census.gov/programs-surveys/acs/technical-documentation/table-and-geography-changes/2015/5-year.html. Accessed October 9, 2022.

[285] van den Boom, T. (June 2018). Model predictive scheduling of semi-cyclic discrete-even t systems using switching max-plus linear models.

[286] Van Heesch, M., Wissink, P. L. J., Ranji, R., Nobakht, M., & Hartog, F. D. (2020). Combining cooperative with non-cooperative game theory to model wi-fi congestion in apartment blocks. *IEEE Access*, 8, 64603–64616.

[287] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems*, volume 30 (pp. 5998–6008).: Curran Associates, Inc.

[288] Vazintari, A., Vlachou, C., & Cottis, P. G. (2013). Network coding for overhead reduction in delay tolerant networks. *Wireless Personal Communications*, 72(4), 2653–2671.

[289] Švejda, M. & Čechura, T. (2015). Interpolation method for robot trajectory planning. In *2015 20th International Conference on Process Control (PC)* (pp. 406–411).

[290] Wall, M. (2019). Spacex's starlink constellation could swell by 30,000 more satellites.

[291] Wang, T.-E., Lin, C.-Y., King, C.-C., & Lee, W.-C. (2010). Estimating pathogen-specific asymptomatic ratios. *Epidemiology (Cambridge, Mass.)*, 21, 726–8.

[292] Wang, Z., Zhao, C., Di, Z., & Wang, W.-X. (2016). Predicting the dynamics of network connectivity. *Scientific Reports*, 6, 24634.

[293] Wason, P. C. (1960). On the failure to eliminate hypotheses in a conceptual task. *Quarterly Journal of Experimental Psychology*, 12(3), 129–140.

[294] Weiss, M. A. (2011). *Data Structures and Algorithm Analysis in Java*. Pearson, 3 edition.

[295] Wikipedia contributors (2022). Katz centrality — Wikipedia, the free encyclopedia. [Online; accessed 11-April-2023].

[296] Wikipedia contributors (2023). Eigenvector centrality — Wikipedia, the free encyclopedia. [Online; accessed 11-April-2023].

[297] Wolfram, S. (1983). Statistical mechanics of cellular automata. *Rev. Mod. Phys.*, 55, 601–644.

[298] Wood, L., Ivancic, W., Eddy, W., Stewart, D., Northam, J., Jackson, C., Da, A., & Da Silva Curiel, A. (2008). Use of the delay-tolerant networking bundle protocol from space. In *International Astronautical Congress*, volume 5 (pp. 3123–3133).

[299] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2020). A comprehensive survey on graph neural networks. *IEEE Trans. Pattern Anal. Mach. Intell.*

[300] Xia, K. (2015). Persistent homology analysis of protein structure, flexibility, and folding. *Physica A*, 436, 1.

[301] Xu, K., Hu, Y., Leskovec, J., & Jegelka, S. (2019). Temporal graph attention networks for dynamic graphs. In *ICLR*.

[302] Yan, S., Xiong, Y., & Lin, D. (2018). Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI* (pp. 3397–3404).

[303] Yang, F., Chen, L., Zhou, F., Gao, Y., & Cao, W. (2020). Relational State-Space Model for Stochastic Multi-Object Systems. *International Conference on Learning Representations.*

[304] Yang, J., McAuley, J., & Leskovec, J. (2016). Community detection in networks with node attributes. In *Knowledge Discovery in Databases: PKDD 2016* (pp. 243–260).: Springer.

[305] Yeh, R. A., Schwing, A. G., Huang, J., & Murphy, K. (2019). Diverse generation for multi-agent sports games. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June, 4605–4614.

[306] You, J., Ying, R., Ren, X., Hamilton, W. L., & Leskovec, J. (2018). Graphrnn: Generating realistic graphs with deep auto-regressive models. In *NeurIPS* (pp. 5708–5718).

[307] Yue, Y., Lucey, P., Carr, P., Bialkowski, A., & Matthews, I. (2014). Learning Fine-Grained Spatial Models for Dynamic Sports Play Prediction. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 2015-Janua(January), 670–679.

[308] Zhan, X.-X., Liu, C., Zhou, G., Zhang, Z., Sun, G.-Q., Zhu, J., & Jin, Z. (2018). Coupling dynamics of epidemic spreading and information diffusion on complex networks. *Applied Mathematics and Computation*, 332.

[309] Zhang, L. & Fan, S. (2019). Dynamic stock market prediction via graph neural networks. *arXiv preprint arXiv:1912.12220*.

[310] Zhang, M. & Chen, Y. (2018). Link prediction based on graph neural networks. *IEEE Trans. Knowl. Data Eng.*, 30(6), 1212–1225.

[311] Zhang, Q., Jin, Z., Zhang, Z., & Shu, Y. (2009). Network coding for applications in the delay tolerant network (dtn). *2009 Fifth International Conference on Mobile Ad-hoc and Sensor Networks*, (pp. 376–380).

[312] Zhang, X., Sun, G., Zhu, Y., Ma, J., & Jin, Z. (2013). Epidemic dynamics on semi-directed complex networks. *Mathematical Biosciences*, 246(2), 242–251.

[313] Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., & Li, H. (2019a). T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Trans. Intell. Transp. Syst.*, 21(3), 1012–1024.

[314] Zhao, L., Wang, J., Chen, Y., Wang, Q., Cheng, J., & Cui, H. (2012). Sihr rumor spreading model in social networks. *Physica A: Statistical Mechanics and its Applications*, 391(7), 2444–2453.

[315] Zhao, L., Wang, Z., Cheng, J., Chen, Y., Wang, J., & Huang, W. (2011). Rumor spreading model with consideration of forgetting mechanism: A case of online blogging livejournal. *Physica A: Statistical Mechanics and its Applications*, 390(13), 2619–2625.

[316] Zhao, Y., Borovikov, I., Rupert, J., Somers, C., & Beirami, A. (2019b). On Multi-Agent Learning in Team Sports Games. *arXiv*.

[317] Zhou, J., Liu, Z., & Li, B. (2007). Influence of network structure on rumor propagation. *Physics Letters A*, 368, 458–463.

[318] Zhou, L., Yang, Y., Ren, X., Wu, F., & Zhuang, Y. (2020). Dynamic network embedding by modeling triadic closure process. *IEEE Trans. Knowl. Data Eng.*, 32(6), 1212–1225.

[319] Zhu, X., Wang, Y., Chen, L., & Huang, X. (2020). Graph neural networks for temporal recommendations. *arXiv preprint arXiv:2007.06802*.

[320] Zita, C. (2019). Redefining Positions and Players In Todays NBA, Using Machine Learning. *The Sports Scientist*.

[321] Zook, A. & Riedl, M. O. (2018). Learning How Design Choices Impact Gameplay Behavior. *IEEE Transactions on Games*, 11(1), 25–35.

[322] Zweck, J. (2016). Analysis of methods used to reconstruct the flight path of malaysia airlines flight 370. *SIAM Review*, 58(3), 555–574.

THIS WORK WAS TYPESET using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. This dissertation was written according to the requirements outlined by the Seeley G. Mudd Manuscript Library at Princeton University in Princeton, NJ, USA. The above illustration was generated via *stable diffusion* using OpenAI's Dalle·2 model. A template to create a Ph.D. dissertation with this look & feel has been released under the permissive AGPL license and can be found online at https://github.com/suchow/Dissertate or from its lead author, Jordan Suchow.