Graph Signal Processing: Structure and Scalability to Massive Data Sets

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Joya A. Deri

B.S., Electrical Engineering, Stanford University
M.S., Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

December 2016

To my family

Acknowledgments

I would like to thank my advisor Professor José M. F. Moura for his support and guidance. The core of this thesis developed from his suggestions on interesting problems and applications, and his insights led me to the main results. This work would not have been possible without his help.

I would also like to thank my committee members: Professor Franz Franchetti, Professor Soummya Kar, and Professor Hassan Karimi. Their feedback was invaluable in the development of this thesis. Thanks to Professor Kar and Professor Karimi for advice and perspectives. Thanks to Professor Franchetti for providing me with insights into computational challenges that heavily influenced the second half of this thesis.

Many thanks to Dr. Yaniv Althuler of MIT Media Lab and Endor Ltd. for helpful discussions and to the Center for Urban Science and Progress at New York University for providing a venue for exchanging ideas about urban science and introducing me to the New York City taxi data. Professor Pedro Ferreira of the Heinz school's i-Lab and Dr. Pavel Krivitsky were invaluable resources in obtaining data access and starting the project on churn prediction.

To Carol Patterson and Claire Bauerle, thank you for helping everything run smoothly.

I would also like to thank the colleagues and friends who helped me along the way, including group members past and present: Aurora Schmidt, Joel Harley, June Zhang, Kyle Anderson, Nick O'Donoghue, Aliaksei Sandryhaila, Divyanshu Vats, Dusan Jakovetic, Bernardo Pires, Augusto Santos, Liangyan Gui, Subhro Das, Stephen Kruzick, Jonathan Mei, Yuan Chen, Evgeny Toporov, Shanghang Zhang, Satwik Kottur, and Mark Cheung. Thanks to Thom Popovici and Richard Veras, and also Sérgio Pequito, Nipun Popli, Jonathan Donadee, Javad Mohammadi, Anit Sahu, Jon Smereka, Sabina Zejnilovic, Ana Jevtic, Stephen Aday, John Costanzo, and other officemates and friends in Porter Hall B level. Thanks for the discussions and distractions. The memories will stay with me for years to come.

Finally, words cannot express my gratitude to my family. Thank you for your endless encouragement.

This work was supported by a Benjamin Garver Lamme/Westinghouse Graduate Fellowship, AFOSR grant FA 95501010291, NSF grants CCF1011903, CCF1018509, and CCF1513936, and an SYSU-CMU CIRC grant.

Abstract

Large-scale networks are becoming more prevalent, with applications in healthcare systems, financial networks, social networks, and traffic systems. The detection of normal and abnormal behaviors (signals) in these systems presents a challenging problem. State-of-the-art approaches such as principal component analysis and graph signal processing address this problem using signal projections onto a space determined by an eigendecomposition or singular value decomposition. When a graph is directed, however, applying methods based on the graph Laplacian or singular value decomposition causes information from unidirectional edges to be lost. Here we present a novel formulation and graph signal processing framework that addresses this issue and that is well suited for application to extremely large, directed, sparse networks.

In this thesis, we develop and demonstrate a graph Fourier transform for which the spectral components are the Jordan subspaces of the adjacency matrix. In addition to admitting a generalized Parseval's identity, this transform yields graph equivalence classes that can simplify the computation of the graph Fourier transform over certain networks. Exploration of these equivalence classes provides the intuition for an inexact graph Fourier transform method that dramatically reduces computation time over real-world networks with nontrivial Jordan subspaces.

We apply our inexact method to four years of New York City taxi trajectories (61 GB after preprocessing) over the NYC road network (6,400 nodes, 14,000 directed edges). We discuss optimization strategies that reduce the computation time of taxi trajectories from raw data by orders of magnitude: from 3,000 days to less than one day. Our method yields a fine-grained analysis that pinpoints the same locations as the original method while reducing computation time and decreasing energy dispersal among spectral components. This capability to rapidly reduce raw traffic data to meaningful features has important ramifications for city planning and emergency vehicle routing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The primary motivation of this thesis is the development of methods that enhance the utility of graph-based methods for the analysis of real-world network data. Applications include the study of metabolic and energy flows in biological systems [1], traffic flows on transportation grids [2, 3], and information flows on communications and social networks [4, 5]. Graph-based approaches for such analyses are attractive because they exploit information from the underlying network structure to extract a richer and potentially more meaningful set of patterns that describe the flow of data.

Real-world applications are often characterized by two salient features: (1) extremely large data sets and (2) non-ideal network properties that require extensive computation to obtain suitable descriptions of the network. These features can result in extremely long execution times for the analyses of interest. This thesis presents methods that significantly reduce these computation times, for the particular case of graph Fourier transform (GFT) methods [6, 7, 8].

The main body of this work extends the graph signal processing framework proposed by [6, 7, 8] to consider spectral analysis over directed adjacency matrices. References [6, 7] note that the adjacency matrix acts as a shift operator on a graph signal, which, by algebraic signal processing [9, 10, 11], allows a digital signal processing framework over graphs. In particular, the eigenvectors of a graph adjacency matrix represent *spectral components* of the graph corresponding to the eigenvalues, or *graph frequencies*.

The adjacency matrices of real-world large, directed, and sparse networks are frequently *defective*, or non-diagonalizable. Such matrices can be characterized by the Jordan decomposition as in [6]. This decomposition, however, requires determination of Jordan chains, which are expensive to compute, potentially numerically unstable, and non-unique with respect to a set of fixed eigenvectors, resulting in non-unique graph Fourier transforms. In this thesis, we reformulate the graph signal processing framework so that the graph Fourier transform is unique over defective networks and provide a novel method that enables rapid computation of the graph Fourier transform.

Since our objective is to provide methods that can be applied to real-world systems, it is necessary to consider computational costs. Efficient algorithms and fast computation methods are essential, and, with modern computing systems, parallelization and vectorization of software allow decreased computation times [12]. We provide methods here to accelerate computations so that the proposed graph signal processing tools are tractable for massive data sets.

1.1 Previous work

Spectral methods. Principal component analysis (the Karhunen-Loève Transform) was one of the first spectral methods proposed and remains a fundamental tool today. This approach orthogonally transforms data points, often via eigendecomposition or singular value decomposition (SVD) of an empirical covariance matrix, into linearly uncorrelated variables called principal components [13, 14, 15]. The transform is defined so that the first principal components capture the most variance in the data; this allows analysis to be restricted to the first few principal components, thus increasing the efficiency of the data representation.

Other methods determine low-dimensional representations of high-dimensional data by projecting the data onto low-dimensional subspaces generated by subsets of an eigenbasis [16, 17, 18, 19]. References [16, 17] embed high-dimensional vectors onto low-dimensional manifolds determined by a weight matrix with entries corresponding to nearest-neighbor distances. In [18], embedding data in a low-dimensional space is described in terms of the graph Laplacian, where the graph Laplacian is an approximation to the Laplace-Beltrami operator on manifolds. Reference [18] also proves that the algorithm [16] approximates eigenfunctions of a Laplacian-based matrix.

These methods focus on discovering low-dimensional representations for high-dimensional data, capturing relationships between data variables into a matrix for their analysis. In contrast, our problem treats the data as a signal that is an input to a graph-based filter. Our approach emphasizes node-based weights (the signal) instead of edge-based weights that capture data dependencies. Related node-based methods in the graph signal processing framework are discussed next.

Data indexed by graph nodes and Laplacian-based GFTs. The graph signal processing framework developed in this thesis assumes that data is indexed by the nodes of a graph. Studies that analyze data with this framework include [20, 21, 22], which use wavelet transforms to study distributed sensor networks. Other transforms, such as those in [23, 24, 25, 26, 27, 28, 29] represent data in terms of the graph Laplacian and its eigenbasis for localized data processing. In particular, [25, 24] defines a graph Fourier transform

(GFT) as signal projections onto the Laplacian eigenvectors. These eigenvectors form an orthonormal basis since the graph Laplacian is symmetric and positive semidefinite. Graph-based filter banks are constructed with respect to this GFT in [26].

A major issue with analyses based on the graph Laplacian is the loss of first-order network structure of the network, that is, any asymmetry in a digraph. These asymmetries affect network flows, random walks, and other graph properties, as studied, for example, in [30, 31]. Our method captures the influence of directed edges in a graph signal processing framework by projecting onto the eigenbasis of the adjacency matrix.

Adjacency matrix-based GFTs. References [6, 7, 8] define the graph Fourier transform in terms of the eigenvectors of the adjacency matrix $A \in \mathbb{C}^{N \times N}$ of a graph. These references adopt A as the shift operator in digital signal processing. According to the algebraic signal processing theory of [32, 9, 10, 11], the shift generates all linear shift-invariant filters for a class of signals (under certain shift invariance assumptions). By defining the graph Fourier transform in terms of the adjacency matrix, [6] shows that directed network structures can be studied, in contrast to graph Laplacian methods. In addition, [6] develops the concepts of graph filtering and convolution.

The graph Fourier transform of [6] is defined as follows. For a graph $\mathcal{G} = G(A)$ with adjacency matrix $A \in \mathbb{C}^{N \times N}$ and Jordan decomposition $A = VJV^{-1}$, the graph Fourier transform of a signal $s \in \mathbb{C}^N$ over $\mathcal{G}$ is defined as

$$\widetilde{s} = V^{-1}s, \tag{1.1}$$

where V^{-1} is the *Fourier transform matrix* of $\mathcal{G}$. This is essentially a projection of the signal onto the eigenvectors of A. The eigenvectors provide a basis for an orthogonal projection when A is symmetric. When A is normal ($A^H A = AA^H$), the eigenvectors form a unitary basis (i.e., $V^{-1} = V^H$).

This thesis focuses on the case of graph signal processing over *defective*, or non-diagonalizable adjacency matrices. These matrices have at least one eigenvalue with algebraic multiplicity (the exponent in the characteristic polynomial of A) greater than the geometric multiplicity (the kernel dimension of A), which results in an eigenvector matrix that does not span $\mathbb{C}^N$.

The basis can be completed by computing Jordan chains of generalized eigenvectors [33, 34], but this computation introduces degrees of freedom that render these generalized eigenvectors non-unique; in other words, the transform (1.1) may vary greatly depending on the particular generalized eigenvectors that are chosen. To address this issue, our approach defines the GFT in terms of spectral projections onto the Jordan subspaces (i.e., the span of the Jordan chains) of the adjacency matrix.

1.2 Contributions

This thesis makes the following contributions:

- We present a spectral projector-based graph signal processing framework and show that the resulting graph Fourier transform is unique and unambiguous. This formulation addresses the degrees of freedom that arise in the Jordan chain computation and characterize a choice of Jordan basis.

- We define and develop the concepts of isomorphic and Jordan equivalence classes. Isomorphic equivalence classes permit re-orderings of node labels that enable more efficient matrix representations for sparse graph structures. Jordan equivalence classes allow computations of graph Fourier transforms over graphs of simpler topologies.

- Since our spectral projector method still requires a Jordan decomposition step, we propose an inexact but efficient method for computing the graph Fourier transform that simplifies the choice of projection subspaces. This method is motivated by insights based on our graph equivalence classes and dramatically reduces computation on real-world networks. The runtime vs. fidelity trade-off associated with this method is explored.

- We apply our methods to study four years of New York City taxi trip data over the Manhattan road network. We provide computational details of the optimization steps necessary to create a signal extraction framework on a 30-machine cluster with 16-core/16 GB and 8-core/8 GB RAM machines that reduces computation time from 3,000 days to less than a day. We demonstrate empirically that the inexact method disperses less energy over eigenvectors corresponding to low-magnitude eigenvalues.

1.3 Outline

The thesis is organized into four parts as follows:

I. Motivation and background: Chapters 2 and 3;

II. Derivation and properties of the graph Fourier transform (GFT) and inexact method: Chapters 4, 5, and 6;

III. Methods and considerations for applying the graph Fourier transform: Chapters 7 and 8;

IV. Application to New York City taxi data: Chapters 9 and 10.

Chapter 2 describes churn prediction in a 3.7 million mobile caller network in order to motivate the importance of underlying directed graph structures. Chapter 3 provides technical details of the graph signal processing formulation as presented in [6, 7, 8]. We define the subspace projector-based graph Fourier transform in Chapter 4 and discuss a generalized Parseval's identity as well as the relationship of the transform to isomorphic graphs.

This projector allows the definition of Jordan equivalence classes of graphs over which our graph Fourier transform is equal. This concept is explained in detail in Chapter 5 and shown to simplify GFT computations over certain classes. However, these methods still require a full Jordan decomposition. This motivates the inexact approach proposed in Chapter 6, which also discusses in detail the trade-offs associated with using such methods.

Chapter 7 describes the general method of applying the GFT developed in Part II. Chapter 8 presents two example matrices that illustrate potential stability issues in the eigendecompositions of defective and near-defective matrices.

Lastly, in Chapters 9 and 10, we demonstrate the application of the inexact GFT to taxi traffic data on the Manhattan street network. The three computational steps are explained in detail: fast computation of taxi signals, eigendecomposition of the Manhattan road network, and computation of the GFT; some of these results are presented in [35, 36]. Our results demonstrate the speed of the inexact method, and show that it reduces energy dispersal over Jordan subspaces corresponding to low-magnitude eigenvalues.

Part I

Motivation and Background

Part I provides motivation and background for the methods developed in the thesis. Chapter 2 motivates the work by presenting the problem of churn detection for a large caller network. This network consists of 3.7 million subscribers to a mobile service provider. About 2.5% of the subscribers switch providers every month, or *churn*. The service provider would like to predict these churners before the fact in order to design effective targeted advertising strategies. We present a method of extracting features based on localized, directed subgraphs. This method improves churn detection and demonstrates that the underlying network structure plays an important role in caller behaviors. In order to explore the influence of the graph structure, we consider graph signal processing as a general tool and formulate transforms that are applicable to directed networks.

Chapter 3 provides background in two areas: graph signal processing and linear algebra. The graph signal processing framework is first defined in terms of [6, 7, 8]. Graph signals are defined, and the eigendecomposition of the adjacency matrix is presented as the foundation for defining graph filters. These filters transform the signals to the graph Fourier transform domain, which enables spectral analysis of the signals. A key observation is that matrices that are not diagonalizable need to be decomposed to an almost-diagonal form called the Jordan normal form or Jordan canonical form. The concepts in linear algebra that relate to computing Jordan chains and defining Jordan subspaces are presented and used in later chapters to define a spectral projector-based graph Fourier transform that is invariant to the degrees of freedom in the Jordan chain computation.

Chapter 2

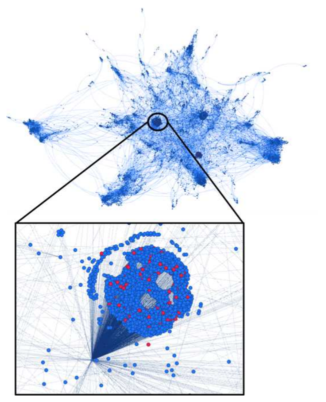
Motivation for Directed Networks

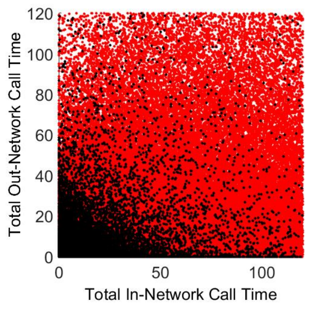
This chapter presents an example application of graph-based analysis to illustrate the power of these methods and to motivate the graph method enhancements presented in this thesis. We consider an anomaly detection problem associated with identification of churners in a caller network, and show how methods based on directed graphs can improve the accuracy of classifiers for this application. Our approach leverages a feature set that utilizes directed subgraphs localized around a node of interest, which increased the separability of the raw features and improved anomaly detection; the details are also provided in [37]. The chapter provides a detailed problem description, an explanation of our graph-based techniques, and our results. In particular, we develop the concept of integral affinity graphs and show their utility in a churn detection application.

2.1 Churn Problem

The goal is to design a classifier that can flag potential churners in a network of 3.7 million prepaid cell phone users; however, raw features from the data are indistinguishable among churners and non-churners. For example, while Figure 2-1b shows that churners are characterized by low activity, there are at least 10 non-churners with low activity for every churner; the non-churner features are not all visible in Figure 2-1b because they overlap with the churner features. While a detector built on these features would have a high false positive rate, service providers require the false positive rate to be as low as possible in order to avoid wasting resources to retain the consumer base. For this problem, a 5% false positive rate is too high, since it corresponds to 185,000 misclassified subscribers. Churn detection is a typical example of the general class of applications involving anomaly detection.

Anomaly detection on large user networks is complicated not only by the computational issues imposed


(a)



(b)

Figure 2-1: (a) Visualization of 3.7 million caller network with inset showing churners (red, 2.5% of network) and non-churners (blue.) (b) Zoomed view of January raw features showing low activity for churners (black) and non-churners (red). In-network call time is plotted versus out-network call time in seconds.

by the size of the data, but also by similarities between anomalous and non-anomalous behavior. For mobile service providers with millions of subscribers, isolating the "churners" (the small percentage of customers who will drop their carriers) is a challenging problem, especially for customers that do not have a fixed period service contract that commits them to the service provider. Carriers would like to identify potential churners before they actually churn; in this way, they can better target advertising and design incentives to prevent or compensate for decreases in their consumer base. In the literature, churn detection has been approached via signal processing techniques such as Kalman filtering [38] as well as machine learning [39], for example. Here, we consider an alternative approach that leverages the underlying network structure of mobile subscribers to train and test classifiers to identify churners. Anomaly detection for networks has been studied in [40, 41], both of which use node neighborhoods to detect anomalies. In [40], neighborhoods in bipartite graphs are explored via random walk-type methods to identify nodes that participate in multiple non-overlapping neighborhoods. In [41], weighted adjacency matrices for localized subgraphs are used to compute outlier statistics corresponding to anomalies in networks with up to 1.6 million nodes. Anomaly detection for identifying faces in images with a cascaded classifier is discussed in [42].

To address the problem of feature separability, we construct a novel feature set that leverages the underlying network structure. While previous studies discussed above also exploit the underlying network, our approach is tailored to finding anomalous subgraphs that highlight churner activity patterns with a method that is fast and efficient. This approach is based on a graph representation of the data set for which nodes represent callers and edges connect callers who call each other. For a node A, a 16-dimensional activity row vector is constructed that collects several usage statistics between a caller and its neighbors. Then, node samples of size M are constructed for each of the neighbors of A and for each of the neighbors of neighbors of A via snowball sampling (see Section 2.2). These M callers form a subgraph called the *affinity graph* of A, with an associated $M \times 16$ activity matrix consisting of the activity vectors of its nodes. These classifiers are tested with $M = 20$, 30, 40, 50, and 100. Next, an expanded set of features is constructed from subgraphs of the affinity graph, and the associated activity vector sums for the callers within each subgraph define the *subgraph activity.* The differences between subgraph activities are defined as features to classify node A. To expedite feature computation, we extend the concept of integral images from [42] to a concept of integral affinity graphs, allowing us to quickly compute subgraph activities and their differences. Details are provided in Section 2.2.

In addition, these features are used to train a cascaded classifier, which reduces false alarm rate to less than 0.1%, with a trade-off in the churn detection. The classifier construction and results are presented in Sections 2.3 and 2.4. The significance of this work in the context of this thesis is discussed in Section 2.5.

2.2 Feature selection

The data set contains eleven consecutive months of cell phone activity for a caller network with over 3.5 million callers in each month [43]. The analysis in this paper is limited to the months of January and February 2009, each with 3.7 million callers. Networks are constructed for these two months of data; the first month is used for training while the second month is used for testing the anomaly classifier. The networks for January and February are denoted by $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ respectively, where V_t is the set of nodes and E_t is the set of undirected edges, $t = 1, 2$. The node set V_t includes all callers of the network at month t that make at least one in-network call in month t. An edge between two callers in V_t exists in E_t if there is at least one call between them in month t.

This section first describes the activity vectors associated with each caller in the network and the computed subgraph-activity features. Then the concept of integral images from [42] is extended to directed graph structures. The extension to trees is first described and then expanded for the case of subnetworks of arbitrary structure. These subgraphs are called *integral affinity graphs*.

Activity vectors and features. Four classes of activity are attributed to every caller $A \in V_t$: calls initiated by A to an in-network user $B \in V_t$; calls received by A from an in-network user $B \in V_t$; calls to an out-of-network user $C \notin V_t$; and, lastly, calls received from an out-of-network user $C \notin V_t$. For each of these classes, the corresponding number of calls, total call time, total number of SMS messages, and the number of callers compose an activity vector of dimension 16 that is recorded for every $A \in V_t$.

In addition, each caller $A \in V_t$ has an associated *affinity graph*, denoted by $G_A = (V_A, E_A)$, $V_A \subset V_t$, $E_A \subset E_t$. The affinity graph for caller A is constructed by performing a snowball sample [44]. First, A is added to node set V_A. Then, snowball sampling is performed to collect its neighbors (the first-wave or 1-hop neighbors). If the target sample size M has not yet been reached, another snowball sample is performed to collect the neighbors of the 1-hop neighbors, which form the second-wave or 2-hop neighbors. Snowball sampling continues until M nodes have been chosen [44]. The corresponding node activity vectors for the collected nodes in V_A form the columns of an $M \times 16$ activity matrix U_A associated with caller A. As discussed in [45, 46, 47, 48, 49], sampled networks can preserve and discover global network properties such as degree distributions. In addition, [49] shows that even biased estimates derived from network samples without re-weighting can reflect global properties. While the primary interest here is in preserving the local structure of a caller A to identify its probability of churn based on its affinity graph, the idea that subsets of the affinity graph reflect global properties motivates our decision to examine subregions of the affinity graph G_A to build our feature set. In particular, the features of interest are the differences of the activity vectors of adjacent subgraphs of G_A.

To compute the features, let $r : V \to \mathbb{R}^{16}$, represent the activity vector sum, and let its ith entry be the sum of the ith entries of the activity vectors of the nodes in a subgraph $G_A = (V_A, E_A)$, i.e.,

$$r_i(G_A) = \sum_{v \in V_A} U_A(v, i), \tag{2.1}$$

where $U_A(v,i)$ denotes the ith entry of the row for caller v in the activity matrix U_A. Suppose $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are two connected subgraphs of G_A and restrict G_2 to the subgraphs of G_1. For pairs of such subgraphs, the feature set F_A for an affinity graph G_A is the set of activity sum differences $F_A = \{r(G_{A,1}) - r(G_{A,2})\}$ over the subgraphs. In Figure 2-2(b), one example feature subset would be $\{r(T_1) - r(T_i) \mid i \in [2, 10]\}$. To efficiently compute these features for affinity graphs, the notion of integral images from [42] is extended to general directed graph structures.

Background on integral images. Consider an image $G = (V, E)$ – i.e., G is a finite two-dimensional lattice. Nodes are labeled in lexicographic order, starting from the top left corner node as node 1 and proceeding sequentially from left to right and top to bottom. With this lexicographic order, the integral image $z(v)$ replaces the image pixel value $U(v)$ at pixel v with the pixel sum of all nodes above and to the left of v. That is,

$$z(v) = \begin{cases} U(v) & \text{if } v = 1, \\ U(v) + \sum\limits_{v' < v} z(v) & \text{otherwise.} \end{cases} \tag{2.2}$$

For example, the pixel sum of window D in Figure 2-2(a) can be computed in four array references: $z(1) + z(4) - (z(2) + z(3))$.

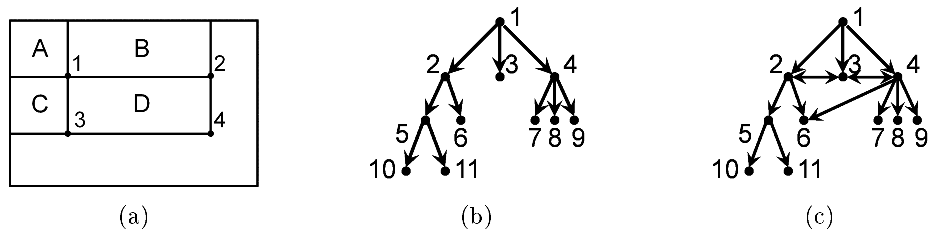


Figure 2-2: Example networks to illustrate integral affinity graphs: (a) a 2-D lattice from [42], (b) a tree, and (c) a general network.

Integral affinity trees. The integral image concept is extended to trees using the following standard definitions for a graph $G = (V, E)$ [50]. A *path* exists between any two nodes $v, w \in V$ if a sequence of edges in E connects them. A *(directed) tree* $T = (V, E)$ is a graph such that any two nodes $v, w \in V$ are connected by exactly one path that has no repeated nodes. A node $w \in V$ is a *descendant* of a node $v \in V$ if there exists a directed path from v to w. Denote by $\mathbb{D}_v$ the set of descendants of v. In addition, the *subtree* T_v of T with root node $v \in V$ is the subgraph containing v and descendants $w \in \mathbb{D}_v$.

For an affinity tree graph G_A associated with node A, the features are extracted as the differences of activity vector sums in adjacent subtrees, i.e., $\{r(T_v) - r(T_{v'})$ for $v' \in \mathbb{D}_v\}$. To compute these features, an *integral affinity graph* is defined as follows:

Definition 2.1. *Consider an affinity tree graph $T_A = (V_A, E_A)$ for a caller A. Then the* integral affinity graph z_A *at a node $v \in V_A$ is defined as the activity vector sum of subtree T_v:*

$$z_A(v) = \sum_{v' \in T_v} U_A(v') = r(T_v), \tag{2.3}$$

where $U_A(v)$ is the activity matrix row corresponding to node v and r is the feature sum function in (2.1).

The integral affinity graph for trees can be expressed by the following recurrence relations:

$$z_A(v) = \begin{cases} U_A(v) & \text{if } |\mathbb{D}_v| = 0 \\ U_A(v) + \sum\limits_{v' \text{ a child of v}} z_A(v') & \text{otherwise} \end{cases}. \tag{2.4}$$

Computing the difference between two subtree activities is equivalent to computing the integral affinity graph difference at the two roots of the subtrees. In Figure 2-2(b), for example, the difference between the subgraph activities for subtrees T_2 and T_5 is $z_1(2) - z_1(5)$.

Extension to general networks. For non-tree affinity graphs $G_A = (V_A, E_A)$, the integral affinity graph is defined in terms of a tree-like structure that accounts for inter-level connections. Breadth first search (BFS) is used to construct spanning trees, which entails exploring all neighbors of caller A before exploring neighbors of neighbors [50]. The BFS tree T_A is constructed from root A and denote the dth *level set* of nodes in G_A with respect to T_A as $L_{T_A,d}$, where $d = 1$ is the root level and d is no more than the maximum depth of T_A. For example, Figure 2-2(b) shows the BFS tree of Figure 2-2(c), so nodes 5 and 6 belong to the level set $L_{T_1,3}$. This representation accounts for connections between levels in the BFS tree since the distance of subtrees from the root is of primary interest – i.e., the integral affinity graph for node 4 in Figure 2-2(c) will include not only the activity sums for its children, but also for the node 6. For simplicity, intralevel edges, such as (2,3) or (3,4), are not accounted for. If properties such as clustering coefficients are features of interest, they can be included in the activity matrix U_A. Note that this formulation exploits the directed nature of influence from the seed node A to its level sets as determined by the underlying graph structure. Subtree differences are used to build features that capture interactions among different waves.

Let Ω_v represent the neighbors of node $v \in V_A$ in the arbitrary network G_A. Define a *region* R_v with root node $v \in V$ at level i in T_A as

$$R_v = \{v \cup (\Omega_v \cap L_{T_A,i+1})\}. \tag{2.5}$$

The resulting definition for general networks is as follows:

Definition 2.2. *Consider an affinity graph $G_A = (V_A, E_A)$ for a caller A. Let T_A denote its corresponding breadth-first-search tree with root A. Then the* integral affinity graph *with respect to T_A for a level-i node $w \in V_A$ is defined as the sum of the activity vectors of w and its neighbors in level set $i+1$:*

$$z_A(v) = \sum_{v' \in R_v} U_A(v') = r(R_v), \tag{2.6}$$

where R_v is the region with root $v \in V_A$, $U_A(v)$ is the activity vector of node v, and $r(R_v)$ is the activity vector sum.

For a node $v \in L_{T_A,i}$, the corresponding recurrence relations are as follows:

$$z_A(v) = \begin{cases} U_A(v) & \text{if } |\mathbb{D}_v| = 0 \\ U_A(v) + \sum\limits_{v' \in R_v} z_A(v') & \text{otherwise} \end{cases}. \tag{2.7}$$

The integral affinity graph allows efficient feature computation. Computing the BFS tree has worst-case time complexity $O(|V| + |E|)$ [50], and computing the integral affinity graph requires traversing the BFS tree from the leaves to the root, which has complexity $O(|V|)$. Given the integral affinity graph, the subgraph activity differences can be computed in two array references. For example, the difference between the regions R_1 and R_6 in Figure 2-2(c) is given by $z_1(1) - z_1(6)$. Since the nodes are in tree levels 1 and 3 respectively, category *31* is assigned to this feature. An affinity graph G_A with a d-level BFS tree T_A will have $\binom{d}{2}$ such feature categories. For 2,000 randomly selected nodes from the month 1 network and fixed snowball sample size $M = 50$, an example feature breakdown is 13,886 *21*-features, 65,407 *32*-features, and 26,692 *42*-features. Note that subgraphs with BFS trees of different depths can be compared by encoding the lack of a level in the associated feature vector. These categories of features are used to train and test a cascaded classifier for churn detection, described in Section 2.3.

2.3 Classifier

Two types of base classifiers from the Python scikit-learn toolbox 0.14 [51] perform well with the constructed features: naive Bayes and decision tree classifiers. Other classifiers that were investigated, including k-nearest neighbor classifiers and stochastic gradient methods [52], did not perform as well. These base classifiers are first described, and then implementation of the cascaded classifier is discussed.

Naive Bayes classifier. The Naive Bayes classifier applies Bayes' theorem with strong independence assumptions [52]. It has a conditional probability model $p(Y \mid X_1, \ldots, X_n)$, where Y is the dependent class that is conditional on the feature variables $X_1, \ldots, X_n$. Applying Bayes' Theorem and independence of the features, the conditional probability can be written as

$$p\left(Y \mid X_1, \ldots X_n\right) = \frac{1}{Z} p\left(Y\right) \prod_{i=1}^{n} p\left(X_i \mid Y\right), \tag{2.8}$$

where Z is a normalization constant. The estimated class $\widehat{y}$ given test data $x_1, \ldots, x_n$ is

$$\widehat{y} = \arg\max_{y} p(y) \prod_{i=1}^{n} p\left(x_i \mid y\right). \tag{2.9}$$

The class priors $p(y)$ are computed empirically from the training data; the independent probability distributions $p(x_i \mid y)$ are Gaussian with mean and variance estimated empirically from the training data. Although the constructed features are not independent and the Gaussian model is not accurate for the problem, naive Bayes is efficient in terms of CPU and memory and is a commonly used testbench classifier in the literature [53].

Decision tree classifier. The decision tree classifier infers simple decision rules from features to build a tree and predict variable classes. Each node in the tree represents a type of feature, and each directed edge from that node refers to a particular instance of that feature. The classifier sorts each instance of the features down a tree from the root to a leaf node that specifies the class of the instance [52]. The C4.5

decision tree algorithm [54] is used in the cascaded classifer.

Cascaded classifier. After tuning, both base classifiers yielded high detection rates but unfortunately also exhibited high false alarm rates, as shown in Table 2.1. For example, given activity vectors of 1000 churners and 1000 non-churners for each month of the 3.7 million caller network, training and testing a naive Bayes classifier with month 1 data yielded a 98% detection rate and a 44% false alarm rate for month 2. To reduce the false alarm rate, a cascaded classifier [42] was built for churner detection.

The cascaded classifier operates in stages. At each stage, a feature category is chosen (see Section 2.2), as well as a sample size M, and a base classifier that together minimize the false alarm rate while ensuring a detection rate above a given threshold. The nodes that are classified as non-churners in the first stage are not considered in later stages and are discarded from the remaining test data. The second and subsequent stages are conceptually equivalent to stage 1. This process continues until the target false alarm rate is reached.

Denote by $p_{d,i}$ the detection rate of the ith stage and by $p_{f,i}$ the false alarm rate of the ith stage, $i \in [1, L]$. Then the detection rate p_d of the cascaded classifier is the product of the L stage detection rates $p_{d,i}$. Likewise, the cascaded false alarm rate p_f is the product of the L stage false alarm rates $p_{f,i}$. Therefore, utilizing these stages drastically reduces the false alarm rate. For example, a 4-stage cascade with a false alarm rate 0.4 at each stage will have an overall false alarm rate of 0.027. While the detection rates also decrease at each stage, they decrease less rapidly than the false alarm rate. Such a classifier is applied to obtain empirical results for the caller data.

2.4 Results

Two months of a 3.7 million caller dataset and their constructed networks are analyzed for churners. Seed nodes are sampled uniformly at random from each month so that 1000 churners and 1000 non-churners are collected. The method outlined in Sections 2.2 and 2.3 is used to compute features and train three cascaded classifiers using month 1 data. Each classifier has an initial stage consisting of a naive Bayes classifier with the activity vectors of the seed nodes as features. The remaining stages use either all naive Bayes classifiers, all decision trees, or a combination of both. After tuning, the decision trees have maximum depth 1 and use all features to find the best split.

Table 2.1 and Figure 2-3 illustrate the results. The mixed-stage classifier has the highest churn detection rate at 71%, while its false alarm rate is 0.8% – that is, 710 of 1000 month 2 churners are correctly identified while 8 of 1000 non-churners are incorrectly classified. In contrast, the initial stage correctly identifies 980 of 1000 churners while incorrectly classifying 440 of 1000 non-churners. Thus, the accuracy increases from

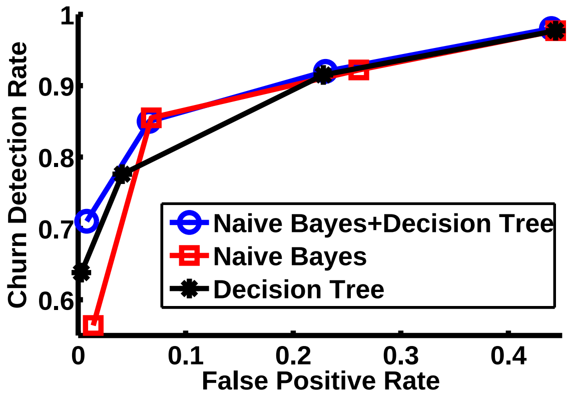


Figure 2-3: Results for three cascaded classifiers. The circles (blue) show results for mixed naive Bayes and decision tree classifiers across stages. The squares (red) are for only naive Bayes and the asterisks (black) are for only decision tree classifiers across stages. We see that the subgraph activity difference features together with the seed vertex vectors and mixed stages allow a false alarm rate of 0.08% with 71% detection.

	Naive Bayes		Decision Tree		Mixed Stages	
Stage	p_d	p_f	p_d	p_f	p_d	p_f
1	0.98	0.44	0.98	0.44	0.98	0.44
2	0.92	0.26	0.92	0.23	0.92	0.23
3	0.86	0.068	0.78	0.041	0.85	0.067
4	0.56	0.014	0.64	0.003	**0.71**	**0.008**

Table 2.1: Detection (p_d) and false alarm (p_f) rates for cascaded classifiers. Mixed stages yield the highest detection rate.

77% to 85%. This improvement is due to exploitation of the local, directed graph topology in the design of the input feature set to the cascaded classifier.

2.5 Significance

This chapter highlights the importance of coupling network structure with node-based features, particularly the use of directed networks to obtain localized node information. We define integral affinity graphs to capture this topology and efficiently build a large feature set. Our results demonstrate the utility of such methods using a real-world application to churn detection in a caller network. In this example, the accuracy of anomaly detection was improved utilizing information from the local graph topology.

These observations motivate further the development of graph-based methods for analysis, including those described in this thesis. Since real-world applications commonly involve large, directed networks that are characterized by sparse, defective matrices, we focus our attention on approaches that extend and accelerate graph signal processing methods to these network topologies.

Chapter 3

Background

This chapter reviews the concepts of graph signal processing and provides a reference for the underlying mathematics. Much of this material is described in greater detail in [6, 7, 8], with additional background on eigendecompositions in [33, 55, 34]. Section 3.1 defines the graph Fourier transform and graph filters. Section 3.2 defines the generalized eigenspaces and Jordan subspaces of a matrix.

3.1 Graph Signal Processing

3.1.1 Graph Signals

Let $\mathcal{G} = \mathcal{G}(A) = (\mathcal{V}, \mathcal{E})$ be the graph corresponding to matrix $A \in \mathbb{C}^{N \times N}$, where $\mathcal{V}$ is the set of N nodes and a nonzero entry $[A]_{ij}$ denotes a directed edge $e_{ij} \in \mathcal{E}$ from node j to node i. In real-world applications, such nodes can be represented by geo-locations of a road network, and the edges can be specified by one-way or two-way roads. Define *graph signal* $s : \mathcal{V} \to \mathcal{S}$ on $\mathcal{G}$, where $\mathcal{S}$ represents the signal space over the nodes of $\mathcal{G}$. We take $\mathcal{S} = \mathbb{C}^N$ such that $s = (s_1, \dots, s_N) \in \mathbb{C}^N$ and s_i represents a measure at node $v_i \in \mathcal{V}$. In real-world applications, such signals can be specified by sensor measurements or datasets.

3.1.2 Graph Shift

As in [6, 8], the graph shift is the graph signal processing counterpart to the shift operator in digital signal processing. The graph shift is defined as the operator that replaces the element s_i of graph signal $s = (s_1, \dots, s_N)$ corresponding to node $v_i \in V$ with the linear combination of the signal elements at its in-neighbors (nodes $v_k \in V$ that participate in an edge $e_{ik} \in \mathcal{E}$), denoted by set $\mathcal{N}_i$; i.e., the shifted signal has

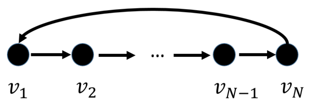


Figure 3-1: Directed cycle graph.

elements $\widetilde{s}_i = \sum_{v_j \in \mathcal{N}_i} [A]_{ij}\, s_j$, or

$$\widetilde{s} = As. \tag{3.1}$$

Consistency with discrete signal processing can be seen by considering the directed cycle graph in Figure 3-1, which represents a finite, periodic time-series signal. The adjacency matrix of the graph is the circulant matrix

$$C = \begin{bmatrix} & & & 1 \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{bmatrix}. \tag{3.2}$$

The shift $\widetilde{s} = Cs$ yields the time delay $\widetilde{s}_i = s_{(i-1) \bmod N}$.

Reference [6] provides more details that show that the graph shift justifies defining a graph Fourier transform as the signal projection onto the eigenvectors of A. Our transform in Chapter 4 builds on this concept to develop a framework to handle defective adjacency matrices.

3.1.3 Graph Filter

The graph shift is a type of graph filter, where a graph filter $\mathbf{H} \in \mathbb{C}^{N\times N}$ represents a (linear) system with output $\mathbf{H}s$ for any graph signal $s \in \mathcal{S}$. As shown in Theorem 1 of [6], graph filter $\mathbf{H}$ is shift-invariant, or

$$A\left(\mathbf{H}\right)s = \mathbf{H}\left(As\right), \tag{3.3}$$

if and only if a polynomial $h(x) = \sum_{i=0}^{L} h_i x^i$ exists for constants $h_0, h_1, \ldots, h_L \in \mathbb{C}$ such that $\mathbf{H} = h(A) = \sum_{i=0}^{L} h_i A^i$. This condition holds whenever the characteristic and minimal polynomials of A are equal [6].

For defective A with unequal characteristic and minimal polynomials such as the examples seen in later chapters, shift-invariance cannot be claimed; however, an *equivalent graph filter* can be designed in terms of a matrix that is the image of a polynomial of A [6]. The properties of such graph filters are established in [6].

3.2 Eigendecomposition

Spectral analysis of graph signals involves the description of signals in terms of their decomposition as a weighted sum of eigenvectors or weighted generalized eigenvectors. Because these eigenvectors reflect network connectivity and interrelationships among nodes, spectral analysis provides an approach for characterizing signal features in terms of the network strucutre. The determination and use of these eigenvectors underly much of the work presented in this thesis. For this reason, a review of the basic concepts and definitions associated with these eigendecompositions is presented here. Additional background may be found in [33, 55, 34, 56].

First, direct sums of subspaces, generalized eigenspaces of a matrix $A \in \mathbb{C}^{N \times N}$, and cyclic Jordan subspaces are presented. The Jordan decomposition is then defined, and examples are provided to clarify their significance. Lastly, the Jordan subspaces are related to their counterparts in algebraic signal processing as in [9, 10, 11].

Let $X_1, \ldots, X_k$ be subspaces of vector space X such that

$$X = X_1 + \cdots + X_k. \tag{3.4}$$

If $X_i \cap X_j = \emptyset$ for all $i \neq j$, then X is the *direct sum* of subspaces $\{X_i\}_{i=1}^k$, denoted as

$$X = \bigoplus_{i=1}^{k} X_i. \tag{3.5}$$

Any $x \in X$ can be written uniquely as $x = x_1 + \cdots + x_k$, where $x_i \in X_i$, $i = 1, \ldots, k$.

Consider matrix $A \in \mathbb{C}^{N \times N}$ with k distinct eigenvalues $\lambda_1, \ldots, \lambda_k$, $k \leq N$. The eigenvalues of A are the roots of the *characteristic polynomial*

$$\varphi_A(\lambda) = \det(A - \lambda I) = \prod_{i=1}^{k} (\lambda - \lambda_i)^{a_i}, \tag{3.6}$$

where I is the identity matrix and exponent a_i represents the *algebraic multiplicity* of eigenvalue λ_i, $i = 1, \ldots, k$. Denote by $\operatorname{Ker}(A)$ the kernel or null space of matrix A, i.e., the span of vectors v satisfying $Av = 0$. The *geometric multiplicity* g_i of eigenvalue λ_i equals the dimension of null space

$$\operatorname{Ker}(A - \lambda_i I). \tag{3.7}$$

The *minimal polynomial* $m_A(\lambda)$ of A has form

$$m_A(\lambda) = \prod_{i=1}^{k} (\lambda - \lambda_i)^{m_i}\,, \tag{3.8}$$

where m_i is the *index* of eigenvalue λ_i. The index m_i represents the maximum Jordan chain length or Jordan subspace dimension, which is discussed in more detail below.

The eigenvectors and generalized eigenvectors of matrix $A \in \mathbb{C}^{N\times N}$ partition $\mathbb{C}^N$ into subspaces, some of which are spans of eigenvectors, eigenspaces, or generalized eigenspaces. The definitions of these subspaces are provided here. Subspace $\mathscr{G}_i = \operatorname{Ker}(A - \lambda_i I)^{m_i}$ is the *generalized eigenspace* or *root subspace* of λ_i. The generalized eigenspaces are A-invariant; that is, for all $x \in \mathscr{G}_i$, $Ax \in \mathscr{G}_i$. The subspace $\mathscr{S}_{ip} = \operatorname{Ker}(A - \lambda_i I)^p$, $p = 0, 1, \ldots, N$, is the *generalized eigenspace of order* p for λ_i. For $p \geq m_i$, $\mathscr{S}_{ip} = \mathscr{G}_i$. The *proper* eigenvectors v of λ_i, or simply eigenvectors of λ_i, are linearly independent vectors in $\mathscr{S}_{i1} = \operatorname{Ker}(A - \lambda_i I)$, the *eigenspace* of λ_i. There are $g_i = \dim \mathscr{S}_{i1} = \dim \operatorname{Ker}(A - \lambda_i I)$ eigenvectors of λ_i. Subspaces $\mathscr{S}_{ip}$ form a *(maximal) chain* of $\mathcal{G}_i$ as depicted in Figure 3-2; that is,

$$\{0\} = \mathscr{S}_{i0} \subset \mathscr{S}_{i1} \subset \cdots \subset \mathscr{S}_{i,m_i} = \mathscr{S}_{i,m_i+1} = \cdots \subset \mathbb{C}^N \tag{3.9}$$

where m_i is the index of λ_i. Vectors $v \in \mathscr{S}_{ip}$ but $v \notin \mathscr{S}_{i,p-1}$ are *generalized eigenvectors of order* p for λ_i.

The generalized eigenspaces $\mathscr{G}_i$ of A corresponding to its k distinct eigenvalues λ_i decompose

$$\mathbb{C}^N = \bigoplus_{i=1}^{k} \mathscr{G}_i \tag{3.10}$$

as depicted in Figure 3-2d.

Let $v_1 \in \mathscr{S}_{i1}$, $v_1 \neq 0$, be one of the g_i proper eigenvectors of A corresponding to the eigenvalue λ_i. It generates by recursion the generalized eigenvectors

$$Av_p = \lambda_i v_p + v_{p-1},\;\; p = 2, \ldots, r \tag{3.11}$$

where r is the minimal positive integer such that $(A - \lambda_i I)^r\, v_r = 0$ and $(A - \lambda_i I)^{r-1}\, v_r \neq 0$. Note that $r \leq m_i$. Such a sequence of vectors $(v_1, \ldots, v_r)$, of maximal length r, that satisfy (3.11) is a *Jordan chain of length* r. The vectors in a Jordan chain are linearly independent and generate the *Jordan subspace*

$$\mathscr{J} = \operatorname{span}(v_1, v_2, \ldots, v_r)\,. \tag{3.12}$$

A Jordan subspace has dimension equal to the length of the Jordan chain from which it is generated. A Jordan subspace is A-invariant. The Jordan subspace $\mathscr{J}$ is also *cyclic* since it can be written by (3.11) as

$$\mathscr{J} = \operatorname{span}\left(v_r, (A-\lambda I)v_r, \ldots, (A-\lambda I)^{r-1}v_r\right) \tag{3.13}$$

for $v_r \in \operatorname{Ker}(A-\lambda I)^r$, $v_r \neq 0$.

The number of Jordan subspaces corresponding to λ_i equals the geometric multiplicity $g_i = \dim \operatorname{Ker}(A-\lambda I)$, since, as noted above, there are g_i eigenvectors of λ_i. Order the Jordan subspaces of λ_i by decreasing dimension and denote by $\mathscr{J}_{ij}$ the jth Jordan subspace of λ_i with dimension $r_{ij} \leq m_i$, where $\{r_{ij}\}_{j=1}^{g_i}$ are called the *partial multiplicities* of λ_i. It can be shown that the Jordan spaces $\{\mathscr{J}_{ij}\}$, $j = 1, \cdots, g_i$ and $i = 1, \cdots, k$, are all disjoint. Then the generalized eigenspace $\mathscr{G}_i = \operatorname{Ker}(A-\lambda I)^{m_i}$ of λ_i can be decomposed as

$$\mathscr{G}_i = \bigoplus_{j=1}^{g_i} \mathscr{J}_{ij} \tag{3.14}$$

as depicted in Figures 3-2a-c. Combining (3.14) and (3.10), we see that $\mathbb{C}^N$ can be expressed as the unique decomposition of Jordan spaces

$$\mathbb{C}^N = \bigoplus_{i=1}^{k} \bigoplus_{j=1}^{g_i} \mathscr{J}_{ij}. \tag{3.15}$$

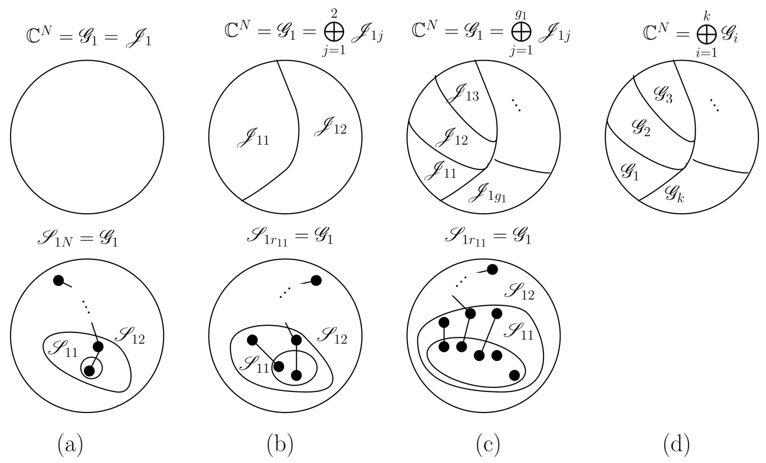
Furthermore, the cyclic Jordan subspaces cannot be represented as direct sums of smaller invariant subspaces; that is, the Jordan subspaces are *irreducible* (see, e.g., p. 318 of [55]); so, (3.15) decomposes $\mathbb{C}^N$ into irreducible components.

Figure 3-2 illustrates possible Jordan subspace structures of A, with the top row showing the tesselation of the base vector space $\mathbb{C}^N$ by the generalized or root eigenspace $\mathscr{G}_i = \operatorname{Ker}\left(A - \lambda_i I\right)^{m_i}$ and by the Jordan spaces $\mathscr{J}_{ij}$, and the bottom row illustrating the telescoping of $\mathbb{C}^N$ by the generalized eigenspaces of order p. Figure 3-2a illustrates $\mathbb{C}^N$ for a matrix with a single Jordan chain, represented by connected points in $\mathbb{C}^N$. The case of a matrix with two Jordan blocks corresponding to the same eigenvalue is shown in Figure 3-2b. Figure 3-2c shows $\mathbb{C}^N$ for a matrix with a single eigenvalue and multiple Jordan blocks, and Figure 3-2d depicts the tesselation of the space in terms of the generalized eigenspaces for the case of multiple eigenvalues.

Jordan decomposition. Let V_{ij} denote the $N \times r_{ij}$ matrix whose columns form a Jordan chain of eigenvalue λ_i of A that spans Jordan subspace $\mathscr{J}_{ij}$. Then the generalized eigenvector matrix V of A is

$$V = \begin{bmatrix} V_{11} \cdots V_{1g_1} & \cdots & V_{k1} \cdots V_{kg_k} \end{bmatrix}, \tag{3.16}$$

where k is the number of distinct eigenvalues. The columns of V are a *Jordan basis* of $\mathbb{C}^N$. Then A has



$\mathbb{C}^N = \mathscr{G}_1 = \mathscr{J}_1$
$\mathbb{C}^N = \mathscr{G}_1 = \bigoplus_{j=1}^{2} \mathscr{J}_{1j}$
$\mathbb{C}^N = \mathscr{G}_1 = \bigoplus_{j=1}^{g_1} \mathscr{J}_{1j}$
$\mathbb{C}^N = \bigoplus_{i=1}^{k} \mathscr{G}_i$
$\mathscr{J}_{11}$
$\mathscr{J}_{12}$
$\mathscr{J}_{13}$
$\mathscr{J}_{1g_1}$
$\mathscr{G}_1$
$\mathscr{G}_2$
$\mathscr{G}_3$
$\mathscr{G}_k$
$\mathscr{S}_{1N} = \mathscr{G}_1$
$\mathscr{S}_{1r_{11}} = \mathscr{G}_1$
$\mathscr{S}_{1r_{11}} = \mathscr{G}_1$
$\mathscr{S}_{11}$
$\mathscr{S}_{12}$
(a)
(b)
(c)
(d)

Figure 3-2: Illustration of generalized eigenspace partitions and Jordan chains of adjacency matrix $A \in \mathbb{C}^{N\times N}$ for (a) a single Jordan block, (b) one eigenvalue and two Jordan blocks, (c) one eigenvalue and multiple Jordan blocks, and (d) multiple eigenvalues. In (a)-(c) (bottom), each point represents a vector in a Jordan chain of A; points connected by lines illustrate a single Jordan chain. The partial multiplicities depicted for λ_1 are (a) N, (b) $r_{11} = N-2$ and 2, and (c) $r_{11} = N-6$, 2, 2, 1, and 1. Each generalized eigenspace $\mathscr{G}_i$ in (d) can be visualized by (a)-(c).

block-diagonal *Jordan normal form* J consisting of Jordan blocks

$$J(\lambda) = \begin{bmatrix} \lambda & 1 & & \\ & \lambda & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{bmatrix}. \tag{3.17}$$

of size r_{ij}; see, for example, [33] or [57, p.125]. The Jordan normal form J of A is unique up to a permutation of the Jordan blocks. The *Jordan decomposition* of A is $A = VJV^{-1}$.

Jordan chains are not unique and not necessarily orthogonal. For example, the 3×3 matrix

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.18}$$

can have distinct eigenvector matrices

$$V_1 = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}, V_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.19}$$

where the Jordan chain vectors are the columns of V_1 and V_2 and so satisfy (3.11). Since Jordan chains are not unique, the Jordan subspace is used in Chapter 4 to characterize the possible generalized eigenvectors belonging to the Jordan chain.

Relation to algebraic signal processing. We briefly relate the Jordan subspaces to the algebraic signal processing theory presented in [9, 10]. This bears mention because the thesis extends [6, 7], which shows that the adjacency matrix provides a shift operator that, by algebraic signal processing theory, allows the definition of filters on graphs. Similarly, algebraic signal processing shows that the Jordan subspaces play an important role in the graph Fourier transform defined in Chapter 4.

In algebraic signal processing [9, 10], the *linear signal processing model* $(\mathcal{A}, \mathcal{M}, \Phi)$ for a vector space V of complex-valued signals generalizes filtering theory, where algebra $\mathcal{A}$ corresponds to a filter space, module $\mathcal{M}$ corresponds to a signal space, and bijective linear mapping $\Phi : V \to \mathcal{M}$ generalizes the z-transform [9]. In this context, the irreducible, $\mathcal{A}$-invariant submodules $\mathcal{M}' \subseteq \mathcal{M}$ are the *spectral components* of (signal space) $\mathcal{M}$.

Since the Jordan subspaces discussed in this section are invariant, irreducible subspaces of $\mathbb{C}^N$ with respect to adjacency matrix A, they can be used to represent spectral components by algebraic signal processing. This motivates our definition of a spectral projector-based graph Fourier transform in Chapter 4.

Part II

Graph Fourier Transform and Inexact Methods

Part II, which includes Chapters 4, 5, and 6, develops the theory behind and properties of the main spectral analysis tools for this thesis: a spectral-projector based graph Fourier transform and an inexact method that reduces computation time. Chapter 4 defines the graph Fourier transform in terms of projections onto Jordan subspaces of adjacency matrix $A \in \mathbb{C}^{N \times N}$. As discussed in Section 3.2 of Chapter 3, the Jordan subspaces of A decompose $\mathbb{C}^N$ and are irreducible. Under this formulation, the spectral representation of a graph signal is unique with respect to a set of fixed proper eigenvectors, which is an important difference from previous methods when the graph adjacency matrix is not diagonalizable (defective). In addition, the graph Fourier transform admits a generalized Parseval's identity, which allows a definition for energy of the signal projections onto the Jordan subspaces. This energy definition is used in Chapter 10 to rank the spectral components of the Manhattan road network. Chapter 4 also shows that the GFT is invariant to permutations of node orderings, which establishes an isomorphic equivalence class of graphs.

Chapter 5 shows that the GFT is equal over more than one defective network structure, which leads to the notion of Jordan equivalence classes of graphs. These graphs are compared to isomorphic equivalence classes and characterized for different types of Jordan normal forms. In addition, a total variation-based ordering of the Jordan subspaces is proposed in terms of the Jordan equivalence class. This is similar to the ordering of frequencies in the time domain and allows the notion of low-pass, high-pass, and pass-band for graph signals.

Chapter 6 builds on Chapter 5 to define efficient, inexact methods for the GFT computations by allowing signal projections on the generalized eigenspaces of A. This approach accelerates computation by relaxing the requirement of computing a full Jordan basis, which is especially useful on sparse real-world networks that have a single eigenvalue corresponding to nontrivial Jordan subspaces.

Chapter 4

Spectral Projector-Based Graph Signal Processing

A main contribution of this thesis is the development of a graph Fourier transform which, in the case of matrices that do not permit diagonalization, allows a more natural representation in terms of the algebraic structure. In particular, spectral components are defined to be the Jordan subspaces of the adjacency matrix. Properties of this definition are explored in this chapter.

This transform admits a generalized Parseval's identity that allows an energy of spectral components to be computed. We also show equivalence of the graph Fourier transform over isomorphic graphs.

4.1 Definition of the Spectral Projector-Based Graph Fourier Transform (GFT)

This section presents a basis-invariant graph Fourier transform with respect to a set of known proper eigenvectors. For graphs with diagonalizable adjacency matrices, the transform resolves to that of [6, 8]. The interpretation of the spectral components is improved in the case of non-diagonalizable, or *defective*, adjacency matrices.

Consider matrix A with distinct eigenvalues $\lambda_1, \ldots, \lambda_k$, $k \leq N$, that has Jordan decomposition

$$A = VJV^{-1}. \tag{4.1}$$

Denote by $\mathscr{J}_{ij}$ the jth Jordan subspace of dimension r_{ij} corresponding to eigenvalue λ_i, $i = 1, \ldots, k$, $j =$

$1, \ldots, g_i$. Each $\mathscr{J}_{ij}$ is A-invariant and irreducible (see Section 3.2). Then, as in algebraic signal processing theory [9, 10], the Jordan subspaces are the spectral components of the signal space $\mathcal{S} = \mathbb{C}^N$ and define the graph Fourier transform of a graph signal $s \in \mathcal{S}$ as the mapping

$$\begin{aligned} \mathcal{F} : \mathcal{S} &\to \bigoplus_{i=1}^{k} \bigoplus_{j=1}^{g_i} \mathscr{J}_{ij} \\ s &\to \left(\widehat{s}_{11}, \ldots, \widehat{s}_{1g_1}, \ldots, \widehat{s}_{k1}, \ldots, \widehat{s}_{kg_k} \right). \end{aligned} \tag{4.2}$$

That is, the Fourier transform of s, is the unique decomposition

$$s = \sum_{i=1}^{k} \sum_{j=1}^{g_i} \widehat{s}_{ij}, \qquad \widehat{s}_{ij} \in \mathscr{J}_{ij}. \tag{4.3}$$

The distinct eigenvalues $\lambda_1, \ldots, \lambda_k$ are the *graph frequencies* of graph $\mathcal{G}(A)$. The *frequency* or *spectral components* of graph frequency λ_i are the Jordan subspaces $\mathscr{J}_{ij}$. The total number of frequency components corresponding to λ_i is its geometric multiplicity g_i. In this way, when $g_i > 1$, frequency λ_i corresponds to more than one (unique) frequency component.

To highlight the significance of (4.2) and (4.3), consider the signal expansion of a graph signal s with respect to graph $\mathcal{G}(A)$:

$$s = \widetilde{s}_1 v_1 + \cdots + \widetilde{s}_N v_N = V \widetilde{s}, \tag{4.4}$$

where v_i is the ith basis vector in a Jordan basis of A, V is the corresponding eigenvector matrix, and $\widetilde{s}_i$ is the ith expansion coefficient. As discussed in Section 3.2, the choice of Jordan basis has degrees of freedom when the dimension of a cyclic Jordan subspace is greater than one. Therefore, if $\dim \mathscr{J}_{ij} \geq 2$, there exists eigenvector submatrix $X_{ij} \neq V_{ij}$ such that $\operatorname{span}\{X_{ij}\} = \operatorname{span}\{V_{ij}\} = \mathscr{J}_{ij}$. Thus, the signal expansion (4.4) is not unique when A has Jordan subspaces of dimension $r > 1$.

In contrast, the Fourier transform given by (4.2) and (4.3) yields a unique signal expansion that is independent of the choice of Jordan basis. Given any Jordan basis $v_1, \ldots, v_N$ with respect to A, the jth spectral component of eigenvalue λ_i is, by (4.3), $\widehat{s}_{ij} = \sum_{k=p}^{p+r_{ij}-1} \widetilde{s}_k v_k$, where $v_p, \ldots, v_{p+r_{ij}-1}$ are a basis of $\mathscr{J}_{ij}$. Under this definition, there is no ambiguity in the interpretation of frequency components even when Jordan subspaces have dimension $r > 1$. The properties of the spectral components are discussed in more detail in the next section.

4.1.1 Spectral Components

The spectral components of the Fourier transform (4.2) are expressed in terms of basis $v_1, \ldots, v_N$ and its dual basis $w_1, \ldots, w_N$ since the chosen Jordan basis may not be orthogonal. Denote the basis and dual basis matrices by $V = [v_1 \cdots v_N]$ and $W = [w_1 \cdots, w_N]$. By definition, $\langle w_i, v_j \rangle = w_i^H v_j = \delta_{ij}$, where δ_{ij} is the Kronecker delta function [56, 58]. Then $W^H V = V^H W = I$, so the dual basis is the inverse Hermitian $W = V^{-H}$.

Partition Jordan basis matrix V as (3.16) so that each $V_{ij} \in \mathbb{C}^{N \times r_{ij}}$ spans Jordan subspace $\mathscr{J}_{ij}$. Similarly, partition the dual basis matrix by rows as $W = [\cdots W_{i1}^H \cdots W_{ig_i}^H \cdots]^T$, with each $W_{ij}^H \in \mathbb{C}^{r_{ij} \times N}$. Suppose $V_{ij} = [v_1 \cdots v_{r_{ij}}]$ with corresponding coefficients $\widetilde{s}_1, \ldots, \widetilde{s}_{r_{ij}}$ in the Jordan basis expansion (4.4). Define an $N \times N$ matrix $V_{ij}^0 = [0 \cdots V_{ij} \cdots 0]$ that is zero except for the columns corresponding to V_{ij}. Then each spectral component corresponding to Jordan subspace $\mathscr{J}_{ij}$ can be written as

$$
\begin{aligned}
\widehat{s}_{ij} &= \widetilde{s}_1 v_1 + \cdots + \widetilde{s}_{r_{ij}} v_{r_{ij}} && (4.5)\\
&= V_{ij}^0 \widetilde{s} && (4.6)\\
&= V_{ij}^0 V^{-1} s && (4.7)\\
&= V \begin{bmatrix} \ddots & & \\ & I_{r_{ij}} & \\ & & \ddots \end{bmatrix} V^{-1} s && (4.8)\\
&= V \begin{bmatrix} \ddots & & \\ & I_{r_{ij}} & \\ & & \ddots \end{bmatrix} W^H s && (4.9)\\
&= V_{ij} W_{ij}^H s, && (4.10)
\end{aligned}
$$

for $i = 1, \ldots, k$ and $j = 1, \ldots, g_i$. If $P_{ij} = V_{ij} W_{ij}^H$, then $P_{ij}^2 = V_{ij} W_{ij}^H V_{ij} W_{ij}^H = V_{ij} W_{ij}^H = P_{ij}$; that is, P_{ij} is the ***projection matrix*** onto $\mathcal{S}_{ij}$ parallel to complementary subspace $\mathcal{S} \setminus \mathcal{S}_{ij}$.

The projection matrices $\{P_{ij}\}_{j=1}^{r_{ij}}$ are related to the ***first component matrix*** Z_{i0} of eigenvalue λ_i. The component matrix is defined as [33, Section 9.5]

$$
Z_{i0} = V \begin{bmatrix} \ddots & & \\ & I_{a_i} & \\ & & \ddots \end{bmatrix} V^{-1} \qquad (4.11)
$$

where $a_i = \sum_{j=1}^{g_i} r_{ij}$ is the algebraic multiplicity of λ_i. This matrix acts as a projection matrix onto the generalized eigenspace, which is important in our formulation of the inexact method in Chapter 6.

Theorem 4.1 provides additional properties of projection matrix P_{ij}.

Theorem 4.1. *For matrix $A \in \mathbb{C}^{N\times N}$ with eigenvalues $\lambda_1, \ldots, \lambda_k$, the projection matrices P_{ij} onto the jth Jordan subspace $\mathscr{J}_{ij}$ corresponding to eigenvalue λ_i, $i = 1, \ldots, k$, $j = 1, \ldots, g_i$, satisfy the following properties:*

(a) $P_{ij}P_{kl} = \delta_{ik}\delta_{jl}P_{ij}$, where δ is the Kronecker delta function;

(b) $\sum_{j=1}^{g_i} P_{ij} = Z_{i0}$, where Z_{i0} is the component matrix *of eigenvalue λ_i;*

Proof. (a) Since $W^H V = I$, the partition of W^H and V that yields (4.10) satisfies $W_{ij}^H V_{kl} = \delta_{ik}\delta_{jl} I_{r_{ij}\times r_{kl}}$, where r_{ij} is the dimension of the Jordan subspace corresponding to P_{ij}, r_{kl} the dimension of Jordan subspace corresponding to P_{kl}, and matrix $I_{r_{ij}\times r_{kl}}$ consists of the first r_{kl} canonical vectors $e_i = (0, \ldots, 1, \ldots, 0)$, where 1 is at the ith index. Then it follows that

$$\begin{aligned} P_{ij}P_{kl} &= V_{ij}W_{ij}^H V_{kl} W_{kl}^H && (4.12)\\ &= V_{ij}\left(\delta_{ik}\delta_{jl} I_{r_{ij}\times r_{kl}}\right) W_{kl}^H. && (4.13)\end{aligned}$$

If $i = k$ and $j = l$, then $P_{ij}P_{kl} = V_{ij} I_{r_{ij}\times r_{ij}} W_{ij}^H = P_{ij}$; otherwise, $P_{ij}P_{kl} = 0$.

(b) Write

$$\begin{aligned} \sum_{j=1}^{g_i} P_{ij} &= \sum_{j=1}^{g_i} V \begin{bmatrix} \ddots & & \\ & I_{r_{ij}} & \\ & & \ddots \end{bmatrix} V^{-1} && (4.14)\\ &= V\left(\sum_{j=1}^{g_i} \begin{bmatrix} \ddots & & \\ & I_{r_{ij}} & \\ & & \ddots \end{bmatrix}\right) V^{-1} && (4.15)\\ &= V \begin{bmatrix} \ddots & & \\ & I_{\sum_{j=1}^{g_i} r_{ij}} & \\ & & \ddots \end{bmatrix} V^{-1} && (4.16)\\ &= Z_{i0}, && (4.17)\end{aligned}$$

or the first component matrix of A for eigenvalue λ_i. □

Theorem 4.1(a) shows that each projection matrix P_{ij} only projects onto Jordan subspace $\mathscr{J}_{ij}$. Theorem 4.1(b) shows that the sum of projection matrices for a given eigenvalue equals the component matrix of that eigenvalue.

While the Jordan basis, or choice of eigenvectors, is not unique, the image of s under P_{ij} is invariant to the choice of Jordan basis. Moreover, the basis of a Jordan subspace does not need to be a Jordan basis of A to satisfy (4.5). Choosing a non-Jordan basis preserves the Jordan normal form of A but may change its elements and the corresponding graph structure. For this reason, we can consider invariance of the graph Fourier transform (4.2) to be an *equivalence relation* on a set of graphs. Equivalence classes with respect to the GFT are explored further in Section 4.3 and Chapter 5. This concept also provides the insight underlying an inexact method described in Chapter 6 that can substantially accelerate the computation of graph Fourier transforms in real-world applications.

4.2 Generalized Parseval's Identity

As discussed above, a chosen Jordan basis for matrix $A \in \mathbb{C}^{N\times N}$, represented by the eigenvector matrix V, may not be orthogonal. Therefore, Parseval's identity may not hold. Nevertheless, a generalized Parseval's identity does exist in terms of the Jordan basis and its dual; see also [58]. For a dual basis matrix $W = V^{-H}$, the following property holds:

Property 4.2 (Generalized Parseval Identity). *Consider graph signals $s_1, s_2 \in \mathbb{C}^N$ over graph $\mathcal{G}(A)$, $A \in \mathbb{C}^{N\times N}$. Let $V = [v_1 \cdots v_N]$ be a Jordan basis for A with dual basis $W = V^{-H}$ partitioned as $[w_1 \cdots w_N]$. Let $s = \sum_{i=1}^{N} \langle s, v_i\rangle v_i = V\widetilde{s}_V$ be the representation of s in basis V and $s = \sum_{i=1}^{N} \langle s, w_i\rangle w_i = W\widetilde{s}_W$ be the representation of s in basis W. Then*

$$\langle s_1, s_2\rangle = \langle \widetilde{s}_{1,V}, \widetilde{s}_{2,W}\rangle. \tag{4.18}$$

By extension,

$$\|s\|^2 = \langle s, s\rangle = \langle \widetilde{s}_V, \widetilde{s}_W\rangle. \tag{4.19}$$

Equations (4.18) and (4.19) hold regardless of the choice of eigenvector basis.

Energy of spectral components. The *energy* of a discrete signal $s \in \mathbb{C}^N$ is defined as [59, 60]

$$E_s = \langle s, s\rangle = \|s\|^2 = \sum_{i=1}^{N} |s|^2. \tag{4.20}$$

Equation (4.19) thus illustrates conservation of signal energy in terms of both a Jordan basis and its dual.

The energy of the signal projections onto the spectral components of $\mathcal{G}(A)$ for the GFT (4.2) is next defined. Write $\widehat{s}_{i,j}$ in terms of the columns of V as $\widehat{s}_{i,j} = \alpha_1 v_{i,j,1} + \ldots \alpha_{r_{ij}} v_{i,j,r_{ij}}$ and in terms of the columns of W as $\widehat{s}_{i,j} = \beta_1 w_{i,j,1} + \ldots \beta_{r_{ij}} w_{i,j,r_{ij}}$. Then the energy of $\widehat{s}_{ij}$ can be defined as

$$\|\widehat{s}_{ij}\|^2 = \langle \alpha, \beta \rangle \tag{4.21}$$

using the notation $\alpha = (\alpha_1, \ldots, \alpha_{r_{ij}})$ and $\beta = (\beta_1, \ldots, \beta_{r_{ij}})$.

The generalized Parseval's identity expresses the energy of the signal in terms of the signal expansion coefficients $\widetilde{s}$, which highlights the importance of choosing a Jordan basis. This emphasizes that both the GFT $\{\widehat{s}_{ij}\}$ and the signal expansion coefficients $\widetilde{s}$ are necessary to fully characterize the graph Fourier domain.

Normal A. When A is normal (i.e., when $AA^H = A^H A$), V can be chosen to have orthonormal columns. Then, $V = W$ so

$$\langle s_1, s_2 \rangle = \langle \widetilde{s}_1, \widetilde{s}_2 \rangle \tag{4.22}$$

and

$$\|s\|^2 = \langle s, s \rangle = \|\widetilde{s}\|^2 . \tag{4.23}$$

Note that (4.22) and (4.23) do not hold in general for diagonalizable A.

4.3 Isomorphic Equivalence Classes

This section demonstrates that the graph Fourier transform (4.2) is invariant up to a permutation of node labels and establishes sets of isomorphic graphs as equivalence classes with respect to invariance of the GFT (4.2). Two graphs $\mathcal{G}(A)$ and $\mathcal{G}(B)$ are isomorphic if their adjacency matrices are similar with respect to a permutation matrix T, or $B = TAT^{-1}$ [61]. The graphs have the same Jordan normal form and the same spectra. Also, if V_A and V_B are eigenvector matrices of A and B, respectively, then $V_B = TV_A$. We prove that the set $\mathbf{G}_A^I$ of all graphs that are isomorphic to $\mathcal{G}(A)$ is an equivalence class over which the GFT is preserved. The next theorem shows that an appropriate permutation can be imposed on the graph signal and GFT to ensure invariance of the GFT over all graphs $\mathcal{G} \in \mathbf{G}_A^I$.

Theorem 4.3. *The graph Fourier transform of a signal s is invariant to the choice of graph $\mathcal{G} \in \mathbf{G}_A^I$ up to a permutation on the graph signal and inverse permutation on the graph Fourier transform.*

Proof. For $\mathcal{G}(A), \mathcal{G}(B) \in \mathbf{G}_A^I$, there exists a permutation matrix T such that $B = TAT^{-1}$. For eigenvector matrices V_A and V_B of A and B, respectively, let $V_{A,ij}$ and $V_{B,ij}$ denote the $N \times r_{ij}$ submatrices of V_A and V_B whose columns span the jth Jordan subspaces $\mathscr{J}_{A,ij}$ and $\mathscr{J}_{B,ij}$ of the ith eigenvalue of A and B, respectively. Let $W_A = V_A^{-H}$ and $W_B = V_B^{-H}$ denote the matrices whose columns form dual bases of V_A and V_B. Since $V_B = TV_A$,

$$W_B = (TV_A)^{-H} \tag{4.24}$$

$$= (V_A^{-1}T^{-1})^H \tag{4.25}$$

$$= T^{-H}V_A^{-H} \tag{4.26}$$

$$= TW_A \tag{4.27}$$

where $T^{-H} = T$ since T is a permutation matrix. Thus, $W_B^H = W_A^H T^H = W_A^H T^{-1}$.

Consider graph signal s. By (4.10), the signal projection onto $\mathscr{J}_{A,ij}$ is

$$\widehat{s}_{A,ij} = V_{A,ij} W_{A,ij}^H s. \tag{4.28}$$

Permit a permutation $\overline{s} = Ts$ on the graph signal. Then the projection of $\overline{s}$ onto $\mathscr{J}_{B,ij}$ is

$$\widehat{\overline{s}}_{B,ij} = TV_{A,ij} W_{A,ij}^H T^{-1} Ts \tag{4.29}$$

$$= TV_{A,ij} W_{A,ij}^H s \tag{4.30}$$

$$= T\widehat{s}_{A,ij} \tag{4.31}$$

by (4.28). Therefore, the graph Fourier transform (4.2) is invariant to a choice among isomorphic graphs up to a permutation on the graph signal and inverse permutation on the Fourier transform. □

Theorem 4.4. *Consider $A \in \mathbb{C}^{N\times N}$. Then the set $\mathbf{G}_A^I$ of graphs isomorphic to $\mathcal{G}(A)$ is an equivalence class with respect to the invariance of the GFT* (4.2) *up to a permutation of the graph signal and inverse permutation of the graph Fourier transform.*

Theorem 4.3 establishes an invariance of the GFT over graphs that only differ up to a node labeling, and Theorem 4.4 follows.

This chapter provides the mathematical foundation for a graph Fourier transform based on projections onto the Jordan subspace of an adjacency matrix. A generalized Parseval's identity is investigated, and the isomorphic equivalence of graphs for this GFT is shown. In the next chapter, the degrees of freedom in graph topology are further explored to define a broader GFT equivalence class.

Chapter 5

Jordan Equivalence Classes

Since the Jordan subspaces of defective adjacency matrices are nontrivial (i.e., they have dimension larger than one), a degree of freedom exists on the graph structure so that the graph Fourier transform of a signal is equal over multiple graphs of different topologies. This chapter defines *Jordan equivalence classes* of graph structures over which the GFT (4.2) is equal for a given graph signal. The chapter proves important properties of this equivalence class that will be used in the applications presented in Chapters 6 and 9.

The intuition behind Jordan equivalence is presented in Section 5.1, and properties of Jordan equivalence are described in Section 5.2. Section 5.3 compares isomorphic and Jordan equivalent graphs. Sections 5.4, 5.5, 5.6, and 5.7 prove properties for Jordan equivalence classes when adjacency matrices have N, one, two, or $2 \leq p < N$ Jordan blocks, respectively. Parseval's identity is revisited in Section 5.8 to establish properties over Jordan equivalence classes. Finally, Section 5.9 derives an ordering of the spectral components of a graph with respect to its Jordan equivalence class.

5.1 Intuition

Consider Figure 5-1, which shows a basis $\{V\} = \{v_1, v_2, v_3\}$ of $\mathbb{R}^3$ such that v_2 and v_3 span a two-dimensional Jordan space $\mathscr{J}$ of adjacency matrix A with Jordan decomposition $A = VJV^{-1}$. The resulting projection of a signal $s \in \mathbb{R}^N$ as in (4.10) is unique.

Note that the definition of the two-dimensional Jordan subspace $\mathscr{J}$ is not basis-dependent because any spanning set $\{w_2, w_3\}$ could be chosen to define $\mathscr{J}$. This can be visualized by rotating v_2 and v_3 on the two-dimensional plane. Any choice $\{w_2, w_3\}$ corresponds to a new basis $\widetilde{V}$. While $\widetilde{A} = \widetilde{V}J\widetilde{V}^{-1}$ does not equal $A = VJV^{-1}$ for all choices of $\{w_2, w_3\}$, their spectral components (the Jordan subspaces) are identical. Consequently, the spectral projections of a signal onto these components are identical; i.e., the GFT (4.2) is

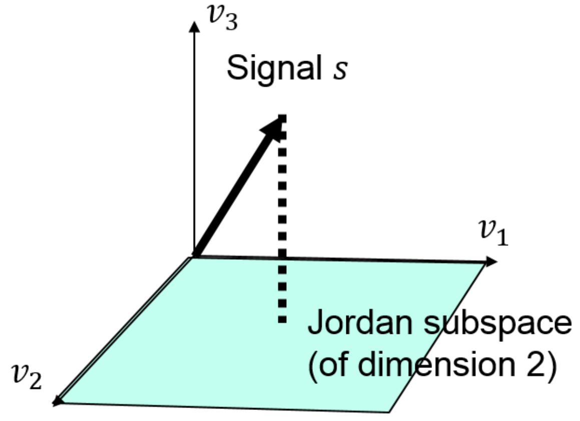


Figure 5-1: Signal projection onto a nontrivial Jordan subspace in $\mathbb{R}^3$.

equivalent over graphs $\mathcal{G}(A)$ and $\mathcal{G}(\widetilde{A})$. This observation leads to the definition of *Jordan equivalence classes* which preserve the GFT (4.2) as well as the underlying structure captured by the Jordan normal form J of A. These classes are formally defined in the next section.

5.2 Definition and Properties

This section defines the *Jordan equivalence class* of graphs, over which the graph Fourier transform (4.2) is invariant. Here we will show that certain Jordan equivalence classes allow the GFT computation to be simplified.

Consider graph $\mathcal{G}(A)$ where A has a Jordan chain that spans Jordan subspace $\mathscr{J}_{ij}$ of dimension $r_{ij} > 1$. Then (4.5), and consequently, (4.10), would hold for a non-Jordan basis of $\mathscr{J}_{ij}$; that is, a basis could be chosen to find spectral component $\widehat{s}_{ij}$ such that the basis vectors do not form a Jordan chain of A. This highlights that the Fourier transform (4.2) is characterized not by the Jordan basis of A but by the set $\mathbf{J}_A = \{\mathscr{J}_{ij}\}_{ij}$ of Jordan subspaces spanned by the Jordan chains of A. Thus, graphs with topologies yielding the same Jordan subspace decomposition of the signal space have the same spectral components. Such graphs are termed *Jordan equivalent* with the following formal definition.

Definition 5.1 (Jordan Equivalent Graphs). *Consider graphs $\mathcal{G}(A)$ and $\mathcal{G}(B)$ with adjacency matrices $A, B \in \mathbb{C}^{N\times N}$. Then $\mathcal{G}(A)$ and $\mathcal{G}(B)$ are* Jordan equivalent graphs *if all of the following are true:*

1. $\mathbf{J}_A = \mathbf{J}_B$*; and*
2. $J_A = J_B$ *(with respect to a fixed permutation of Jordan blocks).*

Let $\mathbf{G}_A^J$ denote the set of graphs that are Jordan equivalent to $\mathcal{G}(A)$. Definition 5.1 and (4.2) establish that $\mathbf{G}_A^J$ is an equivalence class.

Theorem 5.2. *For $A \in \mathbb{C}^{N\times N}$, the set $\mathbf{G}_A^J$ of all graphs that are Jordan equivalent to $\mathcal{G}(A)$ is an equivalence class with respect to invariance of the GFT* (4.2).

Jordan equivalent graphs have adjacency matrices with identical Jordan subspaces and identical Jordan normal forms. This implies equivalence of graph spectra, proven in Theorem 5.3 below.

Theorem 5.3. *Denote by Λ_A and Λ_B the sets of eigenvalues of A and B, respectively. Let $\mathcal{G}(A), \mathcal{G}(B) \in \mathbf{G}_A^J$. Then $\Lambda_A = \Lambda_B$; that is, $\mathcal{G}(A)$ and $\mathcal{G}(B)$ are cospectral.*

Proof. Since $\mathcal{G}(A)$ and $\mathcal{G}(B)$ are Jordan equivalent, their Jordan forms are equal, so their spectra (the unique elements on the diagonal of the Jordan form) are equal. □

Once a Jordan decomposition for an adjacency matrix is found, it could be useful to characterize other graphs in the same Jordan equivalence class. To this end, Theorem 5.4 presents a transformation that preserves the Jordan equivalence class of a graph.

Theorem 5.4. *Consider $A, B \in \mathbb{C}^{N\times N}$ with Jordan decompositions $A = VJV^{-1}$ and $B = XJX^{-1}$ and eigenvector matrices $V = [V_{ij}]$ and $X = [X_{ij}]$, respectively. Then, $\mathcal{G}(B) \in \mathbf{G}_A^J$ if and only if B has eigenvector matrix $X = VY$ for block diagonal Y with invertible submatrices $Y_{ij} \in \mathbb{C}^{r_{ij}\times r_{ij}}$, $i = 1, \dots, k$, $j = 1, \dots, g_i$.*

Proof. The Jordan normal forms of A and B are equal. By Definition 5.1, it remains to show $\mathbf{J}_A = \mathbf{J}_B$ so that $\mathcal{G}(B) \in \mathbf{G}_A^J$. The identity $\mathbf{J}_A = \mathbf{J}_B$ must be true when $\text{span}\{V_{ij}\} = \text{span}\{X_{ij}\} = \mathscr{J}_{ij}$, which implies that X_{ij} represents an invertible linear transformation of the columns of V_{ij}. Thus, $X_{ij} = V_{ij}Y_{ij}$, where Y_{ij} is invertible. Defining $Y = \text{diag}(Y_{11}, \dots, Y_{ij}, \dots, Y_{k,g_k})$ yields $X = VY$. □

5.3 Jordan Equivalent Graphs vs. Isomorphic Graphs

This section proves that isomorphic graphs do not imply Jordan equivalence, and vice versa. The results of this section highlight that the equivalence classes are defined in terms of different equivalence relations. First it is shown that isomorphic graphs have isomorphic Jordan subspaces.

Lemma 5.5. *Consider graphs $\mathcal{G}(A), \mathcal{G}(B) \in \mathbf{G}_A^I$ so that $B = TAT^{-1}$ for a permutation matrix T. Denote by $\mathbf{J}_A$ and $\mathbf{J}_B$ the sets of Jordan subspaces for A and B, respectively. If $\{v_1, \dots, v_r\}$ is a basis of $\mathscr{J}_A \in \mathbf{J}_A$, then there exists $\mathscr{J}_B \in \mathbf{J}_B$ with basis $\{x_1, \dots, x_r\}$ such that $[x_1 \cdots x_r] = T[v_1 \cdots v_r]$; i.e., A and B have isomorphic Jordan subspaces.*

Proof. Consider A with Jordan decomposition $A = VJV^{-1}$. Since $B = TAT^{-1}$, it follows that

$$\begin{aligned} B &= TVJV^{-1}T^{-1} && (5.1) \\ &= XJX^{-1} && (5.2) \end{aligned}$$

where $X = TV$ represents an eigenvector matrix of B that is a permutation of the rows of V. (It is clear that the Jordan forms of A and B are equivalent.) Let columns $v_1, \ldots, v_r$ of V denote a Jordan chain of A that spans Jordan subspace $\mathscr{J}_A$. The corresponding columns in X are $x_1, \ldots, x_r$ and $\text{span}(x_1, \ldots, x_r) = \mathscr{J}_B$. Since $[x_1 \ \cdots \ x_r] = T[v_1 \ \cdots \ v_r]$, $\mathscr{J}_A$ and $\mathscr{J}_B$ are isomorphic subspaces [33]. $\square$

Theorem 5.6. *A graph isomorphism does not imply Jordan equivalence.*

Proof. Consider $\mathcal{G}(A), \mathcal{G}(B) \in \mathbf{G}_A^I$ and $B = TAT^{-1}$ for permutation matrix T. By (5.2), $J_A = J_B$. To show $\mathcal{G}(A), \mathcal{G}(B) \in \mathbf{G}_A^J$, it remains to check whether $\mathbf{J}_A = \mathbf{J}_B$.

By Lemma 5.5, for any $\mathscr{J}_A \in \mathbf{J}_A$, there exists $\mathscr{J}_B \in \mathbf{J}_B$ that is isomorphic to $\mathscr{J}_A$. That is, if $v_1, \ldots, v_r$ and $x_1, \ldots, x_r$ are bases of $\mathscr{J}_A$ and $\mathscr{J}_B$, respectively, then $[x_1 \cdots x_r] = T[v_1 \cdots v_r]$. Checking $\mathscr{J}_A = \mathscr{J}_B$ is equivalent to checking

$$\begin{aligned} \alpha_1 v_1 + \cdots + \alpha_r v_r &= \beta_1 x_1 + \cdots + \beta_r x_r && (5.3) \\ &= \beta_1 T v_1 + \cdots + \beta_r T v_r && (5.4) \end{aligned}$$

for some coefficients α_i and β_i, $i = 1, \ldots, r$. However, (5.4) does not always hold. Consider matrices A and B

$$A = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 2 & 0 \\ -1 & 0 & 2 \end{bmatrix}. \qquad (5.5)$$

These matrices are similar with respect to a permutation matrix and thus correspond to isomorphic graphs. Their Jordan normal forms are both

$$J = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \qquad (5.6)$$

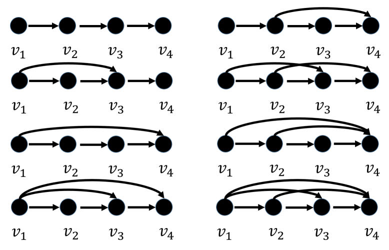


Figure 5-2: Jordan equivalent graph structures with unicellular adjacency matrices.

but they have eigenvector matrices V_A and V_B where

$$V_A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, V_B = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}. \tag{5.7}$$

Equation (5.7) shows that A and B both have Jordan subspaces $\mathscr{J}_1 = \text{span}([1\ 1\ 1]^T)$ for $\lambda_1 = 1$ and $\mathscr{J}_{21} = \text{span}([0\ 1\ 0]^T)$ for one of the Jordan subspaces of $\lambda_2 = 2$. However, the remaining Jordan subspace is span$([1\ 0\ 0]^T)$ for A but span$([0\ 0\ 1]^T)$ for B, so (5.4) fails. Therefore, $\mathcal{G}(A)$ and $\mathcal{G}(B)$ are not Jordan equivalent. □

The next theorem shows that Jordan equivalent graphs may not be isomorphic.

Theorem 5.7. *Jordan equivalence does not imply the existence of a graph isomorphism.*

Proof. A counterexample is provided. The top two graphs in Figure 5-2 correspond to 0/1 adjacency matrices with a single Jordan subspace $\mathscr{J} = \mathbb{C}^N$ and eigenvalue 0; therefore, they are Jordan equivalent. On the other hand, they are not isomorphic since the graph on the right has more edges then the graph on the left. □

Theorem 5.6 shows that changing the graph node labels may change the Jordan subspaces and the Jordan equivalence class of the graph, while Theorem 5.7 shows that a Jordan equivalence class may include graphs with different topologies. Thus, graph isomorphism and Jordan equivalence are not identical concepts. Nevertheless, the isomorphic and Jordan equivalence classes both imply invariance of the graph Fourier transform with respect to different equivalence relations as stated in Theorems 4.3 and 5.2.

The next theorem establishes an isomorphism between Jordan equivalence classes.

Theorem 5.8. *If $A, B \in \mathbb{C}^{N\times N}$ and $\mathcal{G}(A)$ and $\mathcal{G}(B)$ are isomorphic, then their respective Jordan equivalence classes $\mathbf{G}_A^J$ and $\mathbf{G}_B^J$ are isomorphic; i.e., any graph $\mathcal{G}(A') \in \mathbf{G}_A^J$ is isomorphic to a graph $\mathcal{G}(B') \in \mathbf{G}_B^J$.*

Proof. Let $\mathcal{G}(A)$ and $\mathcal{G}(B)$ be isomorphic by permutation matrix T such that $B = TAT^{-1}$. Consider $\mathcal{G}(A') \in \mathbf{G}_A^J$, which implies that Jordan normal forms $J_{A'} = J_A$ and sets of Jordan subspaces $\mathbf{J}_{A'} = \mathbf{J}_A$ by Definition 5.1. Denote by $A' = V_{A'} J_{A'} V_{A'}$ the Jordan decomposition of A'. Define $B' = TA'T^{-1}$. It suffices to show $\mathcal{G}(B') \in \mathbf{G}_B^J$. First simplify:

$$\begin{aligned}
B' &= TA'T^{-1} && (5.8)\\
&= TV_{A'} J_{A'} V_{A'}^{-1} T^{-1} && (5.9)\\
&= TV_{A'} J_A V_{A'}^{-1} T^{-1} \qquad \text{(since } \mathcal{G}(A') \in \mathbf{G}_A^J\text{)} && (5.10)\\
&= TV_{A'} J_B V_{A'}^{-1} T^{-1} \qquad \text{(since } \mathcal{G}(A) \in \mathbf{G}_B^I\text{)}. && (5.11)
\end{aligned}$$

From (5.11), it follows that $J_{B'} = J_B$. It remains to show that $\mathbf{J}_{B'} = \mathbf{J}_B$. Choose arbitrary Jordan subspace $\mathscr{J}_{A,ij} = \operatorname{span}\{V_{A,ij}\}$ of A. Then $\mathscr{J}_{A',ij} = \operatorname{span}\{V_{A',ij}\} = \mathscr{J}_{A,ij}$ since $\mathcal{G}(A') \in \mathbf{G}_A^J$. Then the jth Jordan subspace of eigenvalue λ_i for B is

$$\begin{aligned}
\mathscr{J}_{B,ij} &= \operatorname{span}\{TV_{A,ij}\} && (5.12)\\
&= T\operatorname{span}\{V_{A,ij}\}. && (5.13)
\end{aligned}$$

For the jth Jordan subspace of eigenvalue λ_i for B', it follows from (5.11) that

$$\begin{aligned}
\mathscr{J}_{B',ij} &= \operatorname{span}\{TV_{A',ij}\} && (5.14)\\
&= T\operatorname{span}\{V_{A',ij}\} && (5.15)\\
&= T\operatorname{span}\{V_{A,ij}\} \qquad \text{(since } \mathcal{G}(A') \in \mathbf{G}_A^J\text{)} && (5.16)\\
&= \mathscr{J}_{B,ij}. \qquad \text{(by (5.13))} && (5.17)
\end{aligned}$$

Since (5.17) holds for all i and j, the sets of Jordan subspaces $\mathbf{J}_{B'} = \mathbf{J}_B$. Therefore, $\mathcal{G}(B')$ and $\mathcal{G}(B)$ are Jordan equivalent, which proves the theorem. □

Theorem 5.8 shows that the Jordan equivalence classes of two isomorphic graphs are also isomorphic. This result permits an ordering on the frequency components of a matrix A that is invariant to both the choice of graph in $\mathbf{G}_A^J$ and the choice of node labels, as demonstrated in Section 5.9.

Relation to matrices with the same set of invariant subspaces. Let $\mathbf{G}_A^{\text{Inv}}$ denote the set of all matrices with the same set of invariant subspaces of A; i.e., $\mathcal{G}(B) \in \mathbf{G}_A^{\text{Inv}}$ if and only if $\operatorname{Inv}(A) = \operatorname{Inv}(B)$. The next theorem shows that $\mathbf{G}_A^{\text{Inv}}$ is a proper subset of the Jordan equivalence class $\mathbf{G}_A^J$ of A.

Theorem 5.9. *For $A \in \mathbb{C}^{N\times N}$, $\mathbf{G}_A^{\mathrm{Inv}} \subset \mathbf{G}_A^J$.*

Proof. If $\mathcal{G}(B) \in \mathbf{G}_A^{\mathrm{Inv}}$, then the set of Jordan subspaces are equal, or $\mathbf{J}_A = \mathbf{J}_B$. □

Theorem 5.9 sets the results of this chapter apart from analyses such as those in Chapter 10 of [55], which describes structures for matrices with the same invariant spaces, and [62], which describes the eigendecomposition of the discrete Fourier transform matrix in terms of projections onto invariant spaces. The Jordan equivalence class relaxes the assumption that *all* invariant subspaces of two adjacency matrices have to be equal. This translates to more degrees of freedom in the graph topology. The following sections describe particular cases for adjacency matrices with diagonal Jordan forms, one Jordan block, and multiple Jordan blocks.

5.4 Diagonalizable Matrices

If the canonical Jordan form J of A is diagonal (A is diagonalizable), then there are no Jordan chains and the set of Jordan subspaces $\mathbf{J}_A = \{\mathscr{J}_p\}_{p=1}^N$ where $\mathscr{J}_p = \operatorname{span}(v_p)$ and v_p is the pth eigenvector of A. Graphs with diagonalizable adjacency matrices include undirected graphs, directed cycles, and other digraphs with normal adjacency matrices such as normally regular digraphs [63]. A graph with a diagonalizable adjacency matrix is Jordan equivalent only to itself, which is proven here.

Theorem 5.10. *A graph $\mathcal{G}(A)$ with diagonalizable adjacency matrix $A \in \mathbb{C}^{N\times N}$ belongs to a Jordan equivalence class of size one.*

Proof. Since the Jordan subspaces of a diagonalizable matrix are one-dimensional, the possible choices of Jordan basis are limited to nonzero scalar multiples of the eigenvectors. Then, given eigenvector matrix V of A, all possible eigenvector matrices of A are given by $X = VU$, where U is a diagonal matrix with nonzero diagonal entries. Let $B = XJX^{-1}$, where J is the diagonal canonical Jordan form of A. Since U and J are both diagonal, they commute, yielding

$$B = XJX^{-1} \tag{5.18}$$
$$= VUJU^{-1}V^{-1} \tag{5.19}$$
$$= VJUU^{-1}V^{-1} \tag{5.20}$$
$$= VJV^{-1} \tag{5.21}$$
$$= A. \tag{5.22}$$

Therefore, a graph with a diagonalizable adjacency matrix is the one and only element in its Jordan equivalence class. □

When a matrix has nondefective but repeated eigenvalues, there are infinitely many choices of eigenvectors [34]. An illustrative example is the identity matrix, which has a single eigenvalue but is diagonalizable. Since it has infinitely many choices of eigenvectors, the identity matrix corresponds to infinitely many Jordan equivalence classes. By Theorem 5.10, each of these equivalence classes have size one. This observation highlights that the definition of a Jordan equivalence class requires a choice of basis.

5.5 One Jordan Block

Consider matrix A with Jordan decomposition $A = VJV^{-1}$ where J is a single Jordan block and $V = [v_1 \cdots v_N]$ is an eigenvector matrix. Then A is a representation of a *unicellular transformation* $T : \mathbb{C}^N \to \mathbb{C}^N$ with respect to Jordan basis $v_1, \dots v_N$ (see [55] Section 2.5). In this case the set of Jordan subspaces has one element $\mathscr{J} = \mathbb{C}^N$. Properties of the unicellular Jordan equivalence classes are demonstrated in the following theorems.

Theorem 5.11. *Let $\mathcal{G}(A)$ be an element of the unicellular Jordan equivalence class $\mathbf{G}_A^J$. Then all graph filters $H \in \mathbf{G}_A^J$ are all-pass.*

Proof. Since A is unicellular, it has a single Jordan chain $v_1, \dots, v_N$ of length N. By (4.4), the spectral decomposition of signal s over graph $\mathcal{G}(A)$ yields

$$s = \widetilde{s}_1 v_1 + \cdots \widetilde{s}_N v_N = \widehat{s}; \tag{5.23}$$

that is, the unique projection of s onto the spectral component $\mathscr{J} = \mathbb{C}^N$ is itself. Therefore, $\mathcal{G}(A)$ acts as an all-pass filter. Moreover, (5.23) holds for all graphs in $\mathbf{G}_A^J$. □

In addition to the all-pass property of unicellular graph filters, unicellular isomorphic graphs are also Jordan equivalent, as proven next.

Theorem 5.12. *Let $\mathcal{G}(A), \mathcal{G}(B) \in \mathbf{G}_A^I$ where A is a unicellular matrix. Then $\mathcal{G}(A), \mathcal{G}(B) \in \mathbf{G}_A^J$.*

Proof. Since $\mathcal{G}(A)$ and $\mathcal{G}(B)$ are isomorphic, Jordan normal forms $J_A = J_B$. Therefore, B is also unicellular, so $\mathbf{J}_A = \mathbf{J}_B = \{\mathbb{C}^N\}$. By Definition 5.1, $\mathcal{G}(A), \mathcal{G}(B) \in \mathbf{G}_A^J$. □

The dual basis of V can also be used to construct graphs in the Jordan equivalence class of unicellular A.

Theorem 5.13. *Denote by V an eigenvector matrix of unicellular $A \in \mathbb{C}^{N\times N}$ and $W = V^{-H}$ is the dual basis. Consider decompositions $A = VJV^{-1}$ and $A_W = WJW^{-1}$. Then $\mathcal{G}(A_W) \in \mathbf{G}_A^J$.*

Proof. Matrices A and A_W have the same Jordan normal form by definition. Since there is only one Jordan block, both matrices have a single Jordan subspace $\mathbb{C}^N$. By Definition 5.1, $\mathcal{G}(A_W)$ and $\mathcal{G}(A)$ are Jordan equivalent. $\square$

The next theorem characterizes the special case of graphs in the Jordan equivalence class that contains $\mathcal{G}(J)$ with adjacency matrix equal to the Jordan block $J = J(\lambda)$.

Theorem 5.14. *Denote by $J = J(\lambda)$ is the $N \times N$ Jordan block* (3.17) *for eigenvalue λ. Then $\mathcal{G}(A) \in \mathbf{G}_J^J$ if $A \in \mathbb{C}^{N\times N}$ is upper triangular with diagonal entries λ and nonzero entries on the first off-diagonal.*

Proof. Consider upper triangular matrix $A = [a_{ij}]$ with diagonal entries $a_{11} = \cdots = a_{NN}$ and nonzero elements on the first off-diagonal. By [55, Example 10.2.1], A has the same invariant subspaces as $J = J(\lambda)$, which implies $\mathbf{J}_J = \mathbf{J}_A = \{\mathbb{C}^N\}$. Therefore, the Jordan normal form of A is the Jordan block $J_A = J(a_{11})$. Restrict the diagonal entries of A to λ so $J_A = J$. Then, $\mathcal{G}(J), \mathcal{G}(A) \in \mathbf{G}_J^J$ by Definition (5.1). $\square$

Figure 5-2 shows graph structures that are in the same unicellular Jordan equivalence class by Theorem 5.14. In addition, the theorem implies that it is sufficient to determine the GFT of unicellular A by replacing $\mathcal{G}(A) \in \mathbf{G}_J^J$ with $\mathcal{G}(J)$, where J is a single $N \times N$ Jordan block. That is, without loss of generality, $\mathcal{G}(A)$ can be replaced with a directed chain graph with possible self-edges and the eigenvector matrix $V = I$ chosen to compute the GFT of a graph signal.

Remark on invariant spaces. Example 10.2.1 of [55] shows that a matrix $A \in \mathbb{C}^{N\times N}$ having upper triangular entries with constant diagonal entries a and nonzero entries on the first off-diagonal is both necessary and sufficient for A to have the same invariant subspaces as $N \times N$ Jordan block $J = J(\lambda)$ (i.e., $\mathrm{Inv}(J) = \mathrm{Inv}(A)$, where $\mathrm{Inv}(\cdot)$ represents the set of invariant spaces of a matrix). If $a = \lambda$, Definition 5.1 can be applied, which yields $\mathcal{G}(A) \in \mathbf{G}_J^J$.

On the other hand, consider a unicellular matrix B such that its eigenvector is not in the span of a canonical vector, e.g.,

$$B = \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \tag{5.24}$$

with Jordan normal form $J(0)$. Since the span of the eigenvectors of $J(0)$ and B are not identical, $\text{Inv}(J(0)) \neq \text{Inv}(B)$. However, by Definition 5.1, $\mathcal{G}(B)$ is in the same class of unicellular Jordan equivalent graphs as those of Figure 5-2, i.e., $\mathcal{G}(B) \in \mathbf{G}_J^J$. In other words, for matrices A and B with the same Jordan normal forms ($J_A = J_B$), Jordan equivalence, i.e., $\mathbf{J}_A = \mathbf{J}_B$, is a more general condition than $\text{Inv}(A) = \text{Inv}(B)$. This illustrates that graphs having adjacency matrices with equal Jordan normal forms and the same sets of invariant spaces form a proper subset of a Jordan equivalence class, as shown above in Theorem 5.9.

Remark on topology. Note that replacing each nonzero element of (5.24) with a unit entry results in a matrix that is not unicellular. Therefore, its corresponding graph is not in a unicellular Jordan equivalence class. This observation demonstrates that topology does not determine the Jordan equivalence class of a graph.

5.6 Two Jordan Blocks

Consider $N \times N$ matrix A with Jordan normal form consisting of two Jordan subspaces $\mathscr{J}_1 = \text{span}(v_1, \dots, v_{r_1})$ and $\mathscr{J}_2 = \text{span}(v_{r_1+1}, \dots, v_{r_2})$ of dimensions $r_1 > 1$ and $r_2 = N - r_1$ and corresponding eigenvalues λ_1 and λ_2, respectively. The spectral decomposition of signal s over $\mathcal{G}(A)$ yields

$$s = \underbrace{\widetilde{s}_1 v_1 + \cdots + \widetilde{s}_{r_1} v_{r_1}}_{\widehat{s}_1} + \underbrace{\widetilde{s}_{r_1+1} v_{r_1+1} + \cdots + \widetilde{s}_N v_N}_{\widehat{s}_2} \tag{5.25}$$

$$= \widehat{s}_1 + \widehat{s}_2. \tag{5.26}$$

Spectral components $\widehat{s}_1$ and $\widehat{s}_2$ are the unique projections of s onto the respective Jordan subspaces. By Example 6.5.4 in [33], a Jordan basis matrix X can be chosen for $A = VJV^{-1}$ such that $X = VU$, where U commutes with J and has a particular form as follows.

If $\lambda_1 \neq \lambda_2$, then $U = \text{diag}(U_1, U_2)$, where U_i, $i = 1, 2$, is an $r_i \times r_i$ upper triangular Toeplitz matrix; otherwise, U has form

$$U = \text{diag}(U_1, U_2) + \begin{bmatrix} \underline{0} & U_{12} \\ & \\ U_{21} & \underline{0} \end{bmatrix} \tag{5.27}$$

where U_i is an $r_i \times r_i$ upper triangular Toeplitz matrix and U_{12} and U_{21} are extended upper triangular Toeplitz matrices as in Theorem 12.4.1 in [33]. Thus, all Jordan bases of A can be obtained by transforming eigenvector matrix V as $X = VU$.

A corresponding theorem to Theorem 5.14 is proved here to characterize Jordan equivalent classes when the Jordan form consists of two Jordan blocks. The reader is directed to Sections 10.2 and 10.3 in [55] for more details. The following definitions are needed. Denote $p \times p$ upper triangular Toeplitz matrices $T_{r_2}(b_1, \ldots, b_{r_2})$ of form

$$T_p(b_1, \ldots, b_p) = \begin{bmatrix} b_1 & b_2 & \cdots & b_{p-1} & b_p \\ 0 & b_1 & \ddots & b_{p-2} & b_{p-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & b_1 & b_2 \\ 0 & 0 & \cdots & 0 & b_1 \end{bmatrix}, \tag{5.28}$$

and define $q \times q$ upper triangular matrix for some $q > p$

$$R_q(b_1, \ldots, b_p; F) = \begin{bmatrix} b_1 & b_2 & \cdots & b_p & f_{11} & f_{12} & \cdots f_{1,q-p-1} & f_{1,q-p} \\ 0 & b_1 & b_2 & \cdots & b_p & f_{22} & \cdots f_{2,q-p-1} & f_{2,q-p} \\ \vdots & \vdots & \vdots & & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & & & \cdots & b_p & f_{q-p,q-p} \\ 0 & 0 & 0 & & & \cdots & b_{p-1} & b_p \\ \vdots & \vdots & \vdots & & & \vdots & \vdots & \\ 0 & 0 & 0 & \cdots & & 0 & b_1 & b_2 \\ 0 & 0 & 0 & \cdots & & 0 & 0 & b_1 \end{bmatrix} \tag{5.29}$$

where $F = [f_{ij}]$ is a $(q-p) \times (q-p)$ upper triangular matrix and $b_i \in \mathbb{C}$, $i = 1, \ldots, p$. The theorems are presented below.

Theorem 5.15. *Consider $A = \operatorname{diag}(A_1, A_2)$ where each matrix A_i, $i = 1, 2$, is upper triangular with diagonal elements λ_i and nonzero elements on the first off-diagonal. Let $\lambda_1 \neq \lambda_2$. Then $\mathcal{G}(A)$ is Jordan equivalent to the graph with adjacency matrix $J = \operatorname{diag}(J_{r_1}(\lambda_1), J_{r_2}(\lambda_2))$ where $J_{r_i}(\lambda_i)$ is the $r_i \times r_i$ Jordan block for* *eigenvalue λ_i.*

Proof. By Theorem 5.14, $\mathcal{G}(A_i)$ and $\mathcal{G}(J_{r_i}(\lambda_i))$ are Jordan equivalent for $i = 1, 2$ and A_i upper triangular with nonzero elements on the first off-diagonal. Therefore, the Jordan normal forms of J and A are the same. Moreover, the set of irreducible subspaces of J is the union of the irreducible subspaces of $[J_1\ \underline{0}]^T$ and $[\underline{0}\ J_2]^T$, which are the same as the irreducible subspaces of $[A_1\ \underline{0}]^T$ and $[\underline{0}\ A_2]^T$, respectively. Therefore, $\mathbf{J}_A = \mathbf{J}_J$, so $\mathcal{G}(A)$ and $\mathcal{G}(J)$ are Jordan equivalent. $\square$

Theorem 5.16. *Consider $A = \operatorname{diag}(A_1, A_2)$ where $A_1 = U_{r_1}(\lambda, b_1, \ldots, b_{r_2}, F)$ and $A_2 = T_{r_2}(\lambda, b_1, \ldots, b_{r_2})$, $r_1 \geq r_2$. Then $\mathcal{G}(A)$ is Jordan equivalent to the graph with adjacency matrix $J = \operatorname{diag}(J_{r_1}(\lambda), J_{r_2}(\lambda))$ where $J_{r_i}(\lambda)$ is the $r_i \times r_i$ Jordan block for eigenvalue λ.*

Proof. By Lemma 10.3.3 in [55], A with structure as described in the theorem have the same invariant subspaces as $J = \operatorname{diag}(J_{r_1}(\lambda), J_{r_2}(\lambda))$. Therefore, A and J has the same Jordan normal form and Jordan subspaces and so are Jordan equivalent. $\square$

Theorems 5.15 and 5.16 demonstrate two types of Jordan equivalences that arise from block diagonal matrices with submatrices of form (5.28) and (5.29). These theorems imply that computing the GFT (4.2) over the block diagonal matrices can be simplified to computing the transform over a union of directed chain graphs. That is, the canonical basis can be chosen for V without loss of generality.

As for the case of unicellular transformations, it is possible to pick bases of $\mathscr{J}_1$ and $\mathscr{J}_2$ that do not form a Jordan basis of A. Any two such choices of bases are related by Theorem 5.4. Concretely, if V is the eigenvector matrix of A and X is the matrix corresponding to another choice of basis, then Theorem 5.4 states that a transformation matrix Y can be found such that $X = VY$, where Y is partitioned as $Y = \operatorname{diag}(Y_1, Y_2)$ with full-rank submatrices $Y_i \in \mathbb{C}^{r_i \times r_i}$, $i = 1, 2$.

5.7 Multiple Jordan Blocks

This section briefly describes a special case of Jordan equivalence classes whose graphs have adjacency matrices $A \in \mathbb{C}^{N \times N}$ with p Jordan blocks, $1 < p < N$.

Consider matrix A with Jordan normal form J comprised of p Jordan blocks and eigenvalues $\lambda_1, \ldots, \lambda_k$. By Theorem 10.2.1 in [55], there exists an upper triangular A with Jordan decomposition $A = VJV^{-1}$ such that $\mathcal{G}(A) \in \mathbf{G}_J^J$. Note that the elements in the Jordan equivalence class $\mathbf{G}_J^J$ of $\mathcal{G}(J)$ are useful since signals over a graph in this class can be computed with respect to the canonical basis with eigenvector matrix $V = I$. Theorem 5.17 characterizes the possible eigenvector matrices V such that $A = VJV^{-1}$ allows $\mathcal{G}(A) \in \mathbf{G}_J^J$.

Theorem 5.17. *Let $A = VJV^{-1}$ be the Jordan decomposition of $A \in \mathbb{C}^{N\times N}$ and $\mathcal{G}(A) \in \mathbf{G}_J^J$. Then V must be an invertible block diagonal matrix.*

Proof. Consider $\mathcal{G}(J)$ with eigenvector matrix I. By Theorem 5.4,$\mathcal{G}(A) \in \mathbf{G}_J^J$ implies

$$V = IY = Y \tag{5.30}$$

where Y is an invertible block diagonal matrix. □

The structure of V given in Theorem 5.17 allows a characterization of graphs in the Jordan equivalence class $\mathbf{G}_J^J$ with the dual basis of V as proved in Theorem 5.18.

Theorem 5.18. *Let $\mathcal{G}(A) \in \mathbf{G}_J^J$, where A has Jordan decomposition $A = VJV^{-1}$ and $W = V^{-H}$ is the dual basis of V. If $A_W = WJW^{-1}$, then $\mathcal{G}(A_W) \in \mathbf{G}_J^J$.*

Proof. By Theorem 5.17, V is block diagonal with invertible submatrices V_i. Thus, $W = V^{-H}$ is block diagonal with submatrices $W_i = V_i^{-H}$. By Theorem 5.4, W is an appropriate eigenvector matrix such that, for $A_W = WJW^{-1}$, $\mathcal{G}(A_W) \in \mathbf{G}_J^J$. □

Relation to graph topology. Certain types of matrices have Jordan forms that can be deduced from their graph structure. For example, [64] and [65] relate the Jordan blocks of certain adjacency matrices to a decomposition of their graphs structures into unions of cycles and chains. Applications where such graphs are in use would allow a practitioner to determine the Jordan equivalence classes (assuming the eigenvalues can be computed) and potentially choose a different matrix in the class for which the GFT can be computed more easily. Sections 5.5 and 5.7 show that working with unicellular matrices and matrices in Jordan normal form permits the choice of the canonical basis. In this way, for matrices with Jordan blocks of size greater than one, finding a spanning set for each Jordan subspace may be more efficient than attempting to compute the Jordan chains. Nevertheless, relying on graph topology is not always possible. Such an example was presented in Section 5.5 with adjacency matrix (5.24).

Relation to algebraic signal processing. The emergence of Jordan equivalence from the graph Fourier transform (4.2) is related to algebraic signal processing (see Section 3.2 and [9, 10]). We emphasize that the GFT (4.2) is tied to a basis. This is most readily seen by considering diagonal adjacency matrix $A = \lambda I$, where any basis that spans $\mathbb{C}^N$ defines the eigenvectors (the Jordan subspaces and spectral components) of a graph signal; that is, a matrix, even a diagonalizable matrix, may not have distinct spectral components. Similarly, the signal model $(\mathcal{A}, \mathcal{M}, \Phi)$ requires a choice of basis for the signal space $\mathcal{M}$ in order to define the frequency response (irreducible representation) of a signal [9]. On the other hand, equivalence of the

GFT (4.2) over graphs in Jordan equivalence classes was demonstrated and defined in this chapter, which implies an equivalence of certain bases. This observation suggests the concept of *equivalent signal models* in the algebraic signal processing framework. Just as working with graphs that are Jordan equivalent to those with adjacency matrices in Jordan normal form simplifies GFT computation, we expect similar classes of equivalent signal models for which the canonical basis can be chosen without loss of generality.

This section has shown how the graph Fourier transform via spectral decomposition of Chapter 4 results in Jordan equivalent classes. We next remark on the conservation of energy in the signal and Fourier domains. Then in Section 5.9 the total variation of graph spectral components is defined.

5.8 Parseval's Identity, Revisited

This section characterizes the generalized Parseval's identity described in Section 4.2 in terms of particular Jordan equivalence classes. Denote by V and W the eigenvector and biorthogonal eigenvector matrices of A, respectively.

One Jordan block. By (5.23), a graph signal s equals its GFT $\widehat{s}$ for unicellular A, which leads to the following result.

Theorem 5.19. *For unicellular $A \in \mathbb{C}^{N\times N}$, the GFTs $\widehat{s}_1, \widehat{s}_2$ of signals s_1 and s_2 over $\mathcal{G}(A)$ satisfy*

$$\langle s_1, s_2\rangle = \langle \widehat{s}_1, \widehat{s}_2\rangle \tag{5.31}$$

and

$$\|s_1\|^2 = \langle s_1, s_1\rangle = \|\widehat{s}_1\|^2 . \tag{5.32}$$

Multiple Jordan blocks. Consider A with Jordan normal form J such that $\mathcal{G}(A)$ is Jordan equivalent to $\mathcal{G}(J)$.

Theorem 5.20. *Consider $A \in \mathbb{C}^{N\times N}$ with Jordan normal form J and $\mathcal{G}(A) \in \mathbf{G}_J^J$. Then for graphs signals s_1 and s_2 and their respective signal expansion coefficients $\widetilde{s}_1$ and $\widetilde{s}_2$ over $\mathcal{G}(A)$,*

$$\langle s_1, s_2\rangle = \langle \widetilde{s}_1, \widetilde{s}_2\rangle \tag{5.33}$$

and

$$\|s\|^2 = \langle s, s\rangle = \|\widetilde{s}\|^2 . \tag{5.34}$$

Proof. Eigenvector matrix $V = I$ can be chosen since $\mathcal{G}(A) \in \mathbf{G}_J^J$. Then the dual basis $W = I$, so the result follows. $\square$

Unicellular adjacency matrices are a special case where the energy of the signal can be expressed in terms of the spectral component as shown in Theorem 5.19. The cases discussed here illustrate defective matrices over which Jordan equivalence allows the GFT (4.2) to be readily applied.

5.9 Total Variation Ordering

This section defines a mapping of spectral components to the real line to achieve an ordering of the spectral components. This ordering can be used to distinguish generalized low and high frequencies as in [8]. An upper bound for a total-variation based mapping of a spectral component (Jordan subspace) is derived and generalized to Jordan equivalence classes.

As in [8, 66], the *total variation* for finite discrete-valued (periodic) time series s is defined as

$$\mathrm{TV}\left(s\right) = \sum_{n=1}^{N} \left|s_n - s_{(n-1)\mathrm{mod}N}\right| = \left\|s - Cs\right\|_1, \tag{5.35}$$

where C is the circulant matrix (3.2) that represents the DSP shift operator. As in [8], (5.35) is generalized to the graph shift A to define the *graph total variation*

$$\mathrm{TV}_G\left(s\right) = \left\|s - As\right\|_1. \tag{5.36}$$

Matrix A can be replaced by $A^{\mathrm{norm}} = \frac{1}{|\lambda_{\max}|}A$ when the maximum eigenvalue satisfies $|\lambda_{\max}| > 0$.

Equation (5.36) can be applied to define the total variation of a spectral component. These components are the cyclic Jordan subspaces of the graph shift A as described in Section 3.2. Choose a Jordan basis of A so that V is the eigenvector matrix of A, i.e., $A = VJV^{-1}$, where J is the Jordan form of A. Partition V into $N \times r_{ij}$ submatrices V_{ij} whose columns are a Jordan chain of (and thus span) the jth Jordan subspace $\mathscr{J}_{ij}$ of eigenvalue λ_i, $i = 1, \dots, k \leq N$, $j = 1, \dots, g_i$. Define the (graph) total variation of V_{ij} as

$$\mathrm{TV}_G\left(V_{ij}\right) = \left\|V_{ij} - AV_{ij}\right\|_1, \tag{5.37}$$

where $\|\cdot\|_1$ represents the induced L1 matrix norm (equal to the maximum absolute column sum).

Theorem 5.21 shows that the graph total variation of a Jordan chain is invariant to a relabeling of the graph nodes.

Theorem 5.21. *Let $A, B \in \mathbb{C}^{N\times N}$ and $\mathcal{G}(B) \in \mathbf{G}_A^I$, i.e., $\mathcal{G}(B)$ is isomorphic to $\mathcal{G}(A)$. Let $V_{A,ij} \in \mathbb{C}^{N\times r_{ij}}$*

be a Jordan chain of matrix A and $V_{B,ij} \in \mathbb{C}^{N \times r_{ij}}$ the corresponding Jordan chain of B. Then $\mathrm{TV}_G(V_{A,ij}) = \mathrm{TV}_G(V_{B,ij})$.

Proof. Since $\mathcal{G}(A)$ and $\mathcal{G}(B)$ are isomorphic, there exists a permutation matrix T such that $B = TAT^{-1}$ and the eigenvector matrices V_A and V_B of A and B, respectively, are related by $V_B = TV_A$. This implies that the Jordan chains are related by $V_{B,ij} = TV_{A,ij}$. Then by (5.37),

$$\begin{aligned}
\mathrm{TV}\left(V_B\right) &= \left\|V_{B,ij} - BV_{B,ij}\right\|_1 && (5.38)\\
&= \left\|TV_{A,ij} - \left(TAT^{-1}\right)TV_{A,ij}\right\|_1 && (5.39)\\
&= \left\|TV_{A,ij} - TAV_{A,ij}\right\|_1 && (5.40)\\
&= \left\|T\left(V_{A,ij} - AV_{A,ij}\right)\right\|_1 && (5.41)\\
&= \left\|V_{A,ij} - AV_{A,ij}\right\|_1 && (5.42)\\
&= \mathrm{TV}\left(V_A\right), && (5.43)
\end{aligned}$$

where (5.42) holds because the maximum absolute column sum of a matrix is invariant to a permutation on its rows. □

Theorem 5.21 shows that the graph total variation is invariant to a node relabeling, which implies that an ordering of the total variations of the frequency components is also invariant.

The next theorem shows equivalent formulations for the graph total variation (5.37).

Theorem 5.22. *The graph total variation with respect to GFT (4.2) can be written as*

$$\begin{aligned}
\mathrm{TV}_G\left(V_{ij}\right) &= \left\|V_{ij}\left(I_{r_{ij}} - J_{ij}\right)\right\|_1 && (5.44)\\
&= \max_{i=2,\ldots,r_{ij}} \left\{\left|1-\lambda\right|\left\|v_1\right\|_1, \left\|\left(1-\lambda\right)v_i - v_{i-1}\right\|_1\right\}. && (5.45)
\end{aligned}$$

Proof. Simplify (5.37) to obtain

$$\begin{aligned}
\mathrm{TV}_G\left(V_{ij}\right) &= \left\|V_{ij} - VJV^{-1}V_{ij}\right\|_1 && (5.46)\\
&= \left\| V_{ij} - VJ \begin{bmatrix} \underline{0} \\ I_{r_{ij}} \\ \underline{0} \end{bmatrix} \right\|_1 && (5.47)
\end{aligned}$$

$$= \left\| V_{ij} - V \begin{bmatrix} \underline{0} & & \\ & J_{ij} & \\ & & \underline{0} \end{bmatrix} \right\|_1 \tag{5.48}$$

$$= \left\| V_{ij} - V_{ij} J_{ij} \right\|_1 \tag{5.49}$$

$$= \left\| V_{ij} \left(I_{r_{ij}} - J_{ij} \right) \right\|_1 . \tag{5.50}$$

Let λ denote the ith eigenvalue and the columns $v_1, \ldots, v_{r_{ij}}$ of V_{ij} comprise its jth Jordan chain. Then (5.44) can be expressed in terms of the Jordan chain:

$$\mathrm{TV}_G \left(V_{ij} \right) = \left\| \begin{bmatrix} v_1 \ldots v_{r_{ij}} \end{bmatrix} \begin{bmatrix} 1-\lambda & -1 & & \\ & 1-\lambda & \ddots & \\ & & \ddots & -1 \\ & & & 1-\lambda \end{bmatrix} \right\|_1 \tag{5.51}$$

$$= \left\| \begin{bmatrix} (1-\lambda) v_1 & (1-\lambda) v_2 - v_1 & \cdots \end{bmatrix} \right\|_1 \tag{5.52}$$

$$= \max_{i=2,\ldots,r_{ij}} \left\{ |1-\lambda| \left\| v_1 \right\|_1 , \left\| (1-\lambda) v_i - v_{i-1} \right\|_1 \right\} . \tag{5.53}$$

□

Theorem 5.23 shows that V can be chosen such that $\left\| V_{ij} \right\|_1 = 1$ without loss of generality.

Theorem 5.23. *The eigenvector matrix V of adjacency matrix $A \in \mathbb{C}^{N \times N}$ can be chosen so that each Jordan chain represented by the eigenvector submatrix $V_{ij} \in \mathbb{C}^{N \times r_{ij}}$ satisfies $\left\| V_{ij} \right\|_1 = 1$; i.e., $\left\| V \right\|_1 = 1$ without loss of generality.*

Proof. Let V represent an eigenvector matrix of A with partitions V_{ij} as described above, and let J_{ij} represent the corresponding Jordan block. Let D be a block diagonal matrix with $r_{ij} \times r_{ij}$ diagonal blocks $D_{ij} = (1/\left\| V_{ij} \right\|_1) I_{r_{ij}}$. Since D_{ij} commutes with J_{ij}, D commutes with J. Note that D is a special case of the upper triangular Toeplitz matrices discussed in Section 5.6 of this thesis and [33, Example 6.5.4, Theorem 12.4.1].

Let $X = VD$ and $B = XJX^{-1}$. Then

$$\begin{aligned} B &= XJX^{-1} && (5.54)\\ &= VDJD^{-1}V^{-1} && (5.55)\\ &= VDD^{-1}JV^{-1} && (5.56)\\ &= VJV^{-1} && (5.57)\\ &= A. && (5.58) \end{aligned}$$

Therefore, both V and X are eigenvector matrices of A. □

In the following, it is assumed that V satisfies Theorem 5.23. Theorem 5.24 presents an upper bound of (5.44).

Theorem 5.24. *Consider matrix A with k distinct eigenvalues and $N \times r_{ij}$ matrices V_{ij} with columns comprising the jth Jordan chain of λ_i, $i = 1, \dots, k$, $j = 1, \dots, g_i$. Then the graph total variation* $\mathrm{TV}_G(V_{ij}) \leq |1 - \lambda_i| + 1$.

Proof. Let $\|V_{ij}\|_1 = 1$ and rewrite (5.44):

$$\begin{aligned} \mathrm{TV}_G\left(V_{ij}\right) &\leq \|V_{ij}\|_1 \left\|I_{r_{ij}} - J_{ij}\right\|_1 && (5.59)\\ &= \left\|I_{r_{ij}} - J_{ij}\right\|_1 && (5.60)\\ &= |1 - \lambda_i| + 1. && (5.61) \end{aligned}$$

□

Equations (5.44), (5.45), and (5.61) characterize the (graph) total variation of a Jordan chain by quantifying the change in a set of vectors that spans the Jordan subspace $\mathscr{J}_{ij}$ when they are transformed by the graph shift A. While this total variation bound may not capture the true total variation of a spectral component, it can be generalized as an upper bound for all spectral components associated with a Jordan equivalence class. As seen in Sections 5.5, 5.6, and 5.7, defective graph shift matrices belong to Jordan equivalence classes that contain more than one element, and the GFT of a signal is the same over any graph in a given Jordan equivalence class. Furthermore, for any two graphs $\mathcal{G}(A), \mathcal{G}(B) \in \mathbf{G}_A^J$, A and B have Jordan bases for the same Jordan subspaces, but the respective total variations of the spanning Jordan chains as computed by (5.44) may be different. Since it is desirable to be able to order spectral components in a manner that is invariant to the choice of Jordan basis, we derive here a definition of the total variation of a spectral component of A in relation to the Jordan equivalence class $\mathbf{G}_A^J$.

Let $\mathcal{G}(B)$ be an element in $\mathbf{G}_A^J$ where B has Jordan decomposition $B = VJV^{-1}$. Let the columns of eigenvector submatrix V_{ij} span the Jordan subspace $\mathscr{J}_{ij}$ of A. Then the *class total variation* of spectral component $\mathscr{J}_{ij}$ is defined as the supremum of the graph total variation of V_{ij} over the Jordan equivalence class (for all $\mathcal{G}(B) \in \mathbf{G}_A^J$):

$$\mathrm{TV}_{\mathbf{G}_A^J}\left(\mathscr{J}_{ij}\right) = \sup_{\substack{\mathcal{G}(B)\in\mathbf{G}_A^J \\ B=VJV^{-1} \\ \operatorname{span}\{V_{ij}\}=\mathscr{J}_{ij} \\ \|V_{ij}\|_1=1}} \mathrm{TV}_G\left(V_{ij}\right). \tag{5.62}$$

Theorem 5.25. *Let $A, B \in \mathbb{C}^{N\times N}$ and $\mathcal{G}(B) \in \mathbf{G}_A^I$. Let V_A and V_B be the respective eigenvector matrices with Jordan subspaces $\mathscr{J}_{A,ij} = \operatorname{span}\{V_{A,ij}\}$ and $\mathscr{J}_{B,ij} = \operatorname{span}\{V_{B,ij}\}$ spanned by the jth Jordan chain of eigenvalue λ_i. Then $\mathrm{TV}_{\mathbf{G}_A^J}(\mathscr{J}_{A,ij}) = \mathrm{TV}_{\mathbf{G}_B^J}(\mathscr{J}_{B,ij})$.*

Proof. Let V_A^* denote the eigenvector matrix corresponding to $\mathcal{G}(A^*) \in \mathbf{G}_A^J$ that maximizes the class total variation of Jordan subspace $\mathscr{J}_{A,ij}$; i.e.,

$$\mathrm{TV}_{\mathbf{G}_A^J}\left(\mathscr{J}_{A,ij}\right) = \mathrm{TV}_G\left(V_{A,ij}^*\right). \tag{5.63}$$

Similarly, let V_B^* denote the eigenvector matrix corresponding to $\mathcal{G}(B^*) \in \mathbf{G}_B^J$ that maximizes the class total variation of Jordan subspace $\mathscr{J}_{B,ij}$, or

$$\mathrm{TV}_{\mathbf{G}_B^J}\left(\mathscr{J}_{B,ij}\right) = \mathrm{TV}_G\left(V_{B,ij}^*\right). \tag{5.64}$$

Since $\mathcal{G}(A)$ and $\mathcal{G}(B)$ are isomorphic, Theorem 5.8 implies that there exists $\mathcal{G}(B') \in \mathbf{G}_B^J$ such that $B' = TA^*T^{-1}$; i.e., $V_{B'} = TV_A^*$ where $V_{B'}$ is an eigenvector matrix of B'. By the definition of class total variation (5.62), $\mathrm{TV}_G(V_{B',ij}) \leq \mathrm{TV}_G(V_{B,ij}^*)$. Then, Theorem 5.21 to isomorphic graphs $\mathcal{G}(A^*)$ and $\mathcal{G}(B')$ yields

$$\mathrm{TV}_G\left(V_{A,ij}^*\right) = \mathrm{TV}_G\left(V_{B',ij}\right) \leq \mathrm{TV}_G\left(V_{B,ij}^*\right). \tag{5.65}$$

Similarly, by Theorem 5.8, there exists $\mathcal{G}(A') \in \mathbf{G}_A^J$ such that $B^* = TA'T^{-1}$, or $V_B^* = TV_{A'}$ where $V_{A'}$ is an eigenvector matrix of A'. Apply (5.62) and Theorem 5.21 again to obtain

$$\mathrm{TV}_G\left(V_{A,ij}^*\right) \geq \mathrm{TV}_G\left(V_{A',ij}\right) = \mathrm{TV}_G\left(V_{B,ij}^*\right). \tag{5.66}$$

Equations (5.65) and (5.66) imply that $\mathrm{TV}_G\left(V^*_{A,ij}\right) = \mathrm{TV}_G\left(V^*_{B,ij}\right)$, or

$$\mathrm{TV}_{\mathbf{G}^J_A}\left(\mathscr{J}_{A,ij}\right) = \mathrm{TV}_{\mathbf{G}^J_B}\left(\mathscr{J}_{B,ij}\right). \tag{5.67}$$

. $\square$

Theorem 5.25 shows that the class total variation of a spectral component is invariant to a relabeling of the nodes. This is significant because it means that an ordering of the spectral components by their class total variations is invariant to node labels.

Next, the significance of the class total variation (5.62) is illustrated for adjacency matrices with diagonal Jordan form, one Jordan block, and multiple Jordan blocks.

Diagonal Jordan Form. As shown in Section 5.4, a graph shift A with diagonal Jordan form is the single element of its Jordan equivalence class $\mathbf{G}^J_A$. This yields the following result.

Theorem 5.26. *Let $\mathcal{G}(A)$ have diagonalizable adjacency matrix A with eigenvectors $v_1, \dots, v_N$. Then the class total variation of the spectral component $\mathscr{J}_i$, $i = 1, \dots, N$, of A satisfies (for $\|v_i\| = 1$)*

$$\mathrm{TV}_{\mathbf{G}^J_A}\left(\mathscr{J}_i\right) = \left|1 - \lambda_i\right|. \tag{5.68}$$

Proof. Each spectral component $\mathscr{J}_i$ of A is the span of eigenvector v_i corresponding to eigenvalue λ_i. The class total variation of $\mathscr{J}_i$ is then

$$\mathrm{TV}_{\mathbf{G}^J_A}\left(\mathscr{J}_i\right) = \sup_{\substack{\mathcal{G}(B)\in\mathbf{G}^J_A \\ B=VJV^{-1} \\ \mathrm{span}\{v_i\}=\mathscr{J}_i \\ \|v_i\|_1=1}} \mathrm{TV}_G\left(v_i\right) \tag{5.69}$$

$$= \mathrm{TV}_G\left(v_i\right) \tag{5.70}$$

$$= \left\|v_i - Bv_i\right\|_1 \qquad \text{(by (5.37))} \tag{5.71}$$

$$= \left\|v_i - \lambda_i v_i\right\|_1 \tag{5.72}$$

$$= \left|1 - \lambda_i\right| \left\|v_i\right\|_1 \tag{5.73}$$

$$= \left|1 - \lambda_i\right|. \tag{5.74}$$

$\square$

The result of Theorem 5.26 is consistent with the total variation result for diagonalizable graph shifts in [8]. Next, the class total variation for defective graph shifts is characterized.

One Jordan block. Consider the graph shift A with a single spectral component $\mathscr{J} = \mathbb{C}^N$ and Jordan form $J = J(\lambda)$. The next theorem proves that the total variation of $\mathscr{J}$ attains the upper bound (5.61).

Theorem 5.27. *Consider unicellular $A \in \mathbb{C}^{N \times N}$ with Jordan normal form $J = J(\lambda)$. Then the class total variation of $\mathbf{G}_A^J$ is $|1-\lambda|+1$.*

Proof. Graph $\mathcal{G}(A)$ is Jordan equivalent to $\mathcal{G}(J)$ since A is unicellular. Therefore, the GFT of a graph signal can be computed over $\mathcal{G}(J)$ by choosing the canonical vectors ($V = I$) as the Jordan basis, as shown in (5.23). By (5.45), the maximum of $|1-\lambda| \left\|v_1\right\|_1$ and $\left\| |1-\lambda| \, v_i - v_{i-1} \right\|_1$ for $i = 2, \dots, N$ needs to be computed. The former term equals $|1-\lambda|$ since v_1 is the first canonical vector. The latter term has form

$$\left\| \, |1-\lambda| \, v_i - v_{i-1} \right\|_1 = \left\| \begin{bmatrix} \underline{0} \\ -1 \\ |1-\lambda| \\ \underline{0} \end{bmatrix} \right\|_1 \tag{5.75}$$

$$= 1 + |1-\lambda| \, , \tag{5.76}$$

Since $|1-\lambda| + 1 > |1-\lambda|$, $\mathrm{TV}_G(I) = 1 + |1-\lambda|$. Therefore, (5.61) holds with equality, so the class total variation of $\mathscr{J} = \mathbb{C}^N$ satisfies

$$\mathrm{TV}_{\mathbf{G}_A^J}\left(\mathscr{J}_i\right) = \sup_{\substack{\mathcal{G}(B) \in \mathbf{G}_A^J \\ B = VJV^{-1} \\ \operatorname{span}\{V\} = \mathscr{J} = \mathbb{C}^N \\ \|V\|_1 = 1}} \mathrm{TV}_G\left(V\right) \tag{5.77}$$

$$= \mathrm{TV}_G\left(I\right) \tag{5.78}$$

$$= |1-\lambda| + 1. \tag{5.79}$$

□

Multiple Jordan blocks. Theorem 5.28 proves that the Jordan equivalence class $\mathcal{G}(J)$ where J is in Jordan normal form attains the bound (5.61).

Theorem 5.28. *Let $\mathcal{G}(A) \in \mathbf{G}_J^J$ where J is the Jordan normal form of A and $\mathbf{J}_A = \{\mathscr{J}_{ij}\}_{ij}$ for $i = 1, \dots, k$, $j = 1, \dots, g_i$. Then the class total variation of $\mathscr{J}_{ij}$ is $|1-\lambda_i|+1$.*

Proof. Since $\mathcal{G}(A) \in \mathbf{G}_J^J$, the GFT can be computed over $\mathcal{G}(J)$ with eigenvector matrix $V = I$. Then each $V_{ij} = I_{r_{ij}}$ that spans $\mathscr{J}_{ij}$ has total variation

$$\mathrm{TV}_G\left(I_{ij}\right) = \left\|I_{r_{ij}} - J_{ij}\right\|_1 \tag{5.80}$$

$$= |1-\lambda_i| + 1 \qquad \text{(by (5.61))}. \tag{5.81}$$

Therefore,

$$\mathrm{TV}_{\mathbf{G}_J^J}\left(\mathscr{J}_i\right) = \sup_{\substack{\mathcal{G}(B)\in\mathbf{G}_A^J \\ B=VJV^{-1} \\ \mathrm{span}\{V_{ij}\}=\mathscr{J}_{ij} \\ \|V_{ij}\|_1=1}} \mathrm{TV}_G\left(V_{ij}\right) \tag{5.82}$$

$$= \mathrm{TV}_G\left(I_{r_{ij}}\right) \tag{5.83}$$

$$= |1-\lambda_i| + 1. \tag{5.84}$$

□

Although the total variation upper bound may not be attained for a general graph shift A, choosing this bound as the ordering function provides a useful standard for the comparison of spectral components for all graphs in a Jordan equivalence class. The ordering proceeds as follows:

1. Order the eigenvalues $\lambda_1, \dots, \lambda_k$ of A by increasing $|1-\lambda_i| + 1$ (low total variation to high total variation).

2. Permute the submatrices V_{ij} of eigenvector matrix V to respect the total variation ordering.

Since the ordering is based on the class total variation (5.62), it is invariant to the particular choice of Jordan basis for each nontrivial Jordan subspace.

Such an ordering allows a visualization of the spectral components of a graph signal that can be used to study low frequency and high frequency behaviors of graph signals; see also [8]. Properties that can be studied in this way include the energy (4.21) of signal projections onto the spectral components, which provides a measure for relative component expression. Chapter 7 provides more details on analyzing and applying the GFT to real-world networks.

The Jordan equivalence classes discussed in this section show that there are degrees of freedom over graph topologies with defective adjacency matrices that enable the GFT to be equivalent over multiple graph structures. Finding these classes still requires the eigendecomposition of an adjacency matrix in its class, however.

To avoid issues associated with computing Jordan chains, we present an inexact method in Chapter 6 that reduces the computation time of the eigendecomposition. Particular considerations for applying the GFT are later presented in Chapter 7, and numerical issues associated with eigendecompositions of defective and nearly defective matrices are illustrated in Chapter 8.

Chapter 6

Agile Inexact Method for Accelerating Graph Fourier Transform Computations

This chapter presents the Agile Inexact Method (AIM) for computing the graph Fourier transform (4.2) in order to address the limitations of the spectral projector-based method of Chapter 4. The *inexactness* of the AIM results from the abstraction of the Jordan subspaces that characterize (4.2). The *agility* of the AIM is related to the Jordan equivalence classes discussed in Chapter 5, which demonstrate that the degrees of freedom allowed by defective matrices yield GFT equivalence across networks of various topologies. We emphasize that reducing computation time is a major objective for formulating the AIM.

To obtain the inexact method, the spectral components are redefined as the generalized eigenspaces of adjacency matrix $A \in \mathbb{C}^{N \times N}$, and the resulting properties are discussed here. Section 6.2 establishes the generalized Parseval's identity with respect to the inexact method. An equivalence class with respect to the AIM is discussed in Section 6.3, which leads to a total variation ordering of the new spectral components as discussed in Section 6.4. Section 6.5 discusses the utility of the AIM for real-world network analysis, and Section 6.6 describes the trade-offs associated with its use.

The contributions of this chapter with respect to those of Chapters 4 and 5 are highlighted in the blue boxes of Figure 6-1. The figure illustrates the uniqueness that is gained by projecting onto known eigenvectors (columns of V_{known}) and the Jordan subspaces of (4.2) or generalized eigenspaces of the adjacency matrix (denoted by J and G in Figure 6-1).

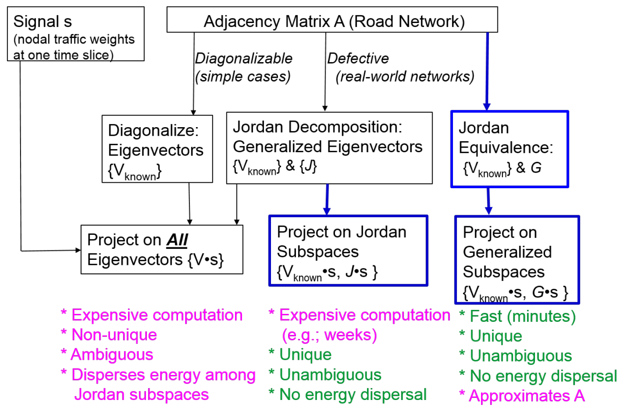

Signal s
(nodal traffic weights at one time slice)
Adjacency Matrix A (Road Network)
Diagonalizable (simple cases)
Defective (real-world networks)
Diagonalize: Eigenvectors {V_known}
Jordan Decomposition: Generalized Eigenvectors {V_known} & {J}
Jordan Equivalence: {V_known} & G
Project on All Eigenvectors {V•s}
Project on Jordan Subspaces {V_known•s, J•s}
Project on Generalized Subspaces {V_known•s, G•s}
* Expensive computation
* Non-unique
* Ambiguous
* Disperses energy among Jordan subspaces
* Expensive computation (e.g.; weeks)
* Unique
* Unambiguous
* No energy dispersal
* Fast (minutes)
* Unique
* Unambiguous
* No energy dispersal
* Approximates A

Figure 6-1: Summary of contributions for GFT analysis over defective network. The columns of V_{known} represent known eigenvectors. The GFT involves projections of a signal s onto the columns of V_{known} as well as the the Jordan subspaces J or generalized eigenspaces G of the adjacency matrix. The execution times mentioned in the figure refer to the NYC taxi data example of Chapter 10 when analyzed by parallel processing on a 30-machine cluster with 16- and 8-core machines.

6.1 Definition

The Agile Inexact Method (AIM) for computing the graph Fourier transform of a signal is defined as the signal projections onto the generalized eigenspaces of the adjacency matrix A. When A has k distinct eigenvalues, the definition of a generalized eigenspace $\mathscr{G}_i$ (see Section 3.2) is

$$\mathscr{G}_i = \operatorname{Ker}\left(A - \lambda_i I\right)^{m_i}, \; i = 1 \ldots, k \tag{6.1}$$

for corresponding eigenvalue λ_i and eigenvalue index m_i. Equation (3.14) shows that each $\mathscr{G}_i$ is the direct sum of the Jordan subspaces of λ_i.

Then, the *Agile Inexact graph Fourier transform* of a graph signal $s \in \mathcal{S}$ is defined as

$$\begin{aligned} \mathcal{H} : \mathcal{S} &\to \bigoplus_{i=1}^{k} \mathscr{G}_i \\ s &\to \left(\breve{s}_1, \ldots, \breve{s}_k\right). \end{aligned} \tag{6.2}$$

That is, the method applied to signal s yields the unique decomposition

$$s = \sum_{i=1}^{k} \breve{s}_i, \qquad \breve{s}_i \in \mathscr{G}_i. \tag{6.3}$$

We define the projection operator for the inexact method. Denote eigenvector matrix

$$V = [v_{1,1} \cdots v_{1,a_1} \cdots v_{k,1} \cdots v_{k,a_k}], \tag{6.4}$$

where $v_{i,j}$ represents an eigenvector or generalized eigenvector of λ_i and a_i is the algebraic multiplicity of λ_i, $i = 1, \ldots, k$, $j = 1, \ldots, a_i$. Let V_i^0 represent the $N \times N$ matrix that consists of zero entries except for columns containing $v_{i,1}$ through v_{i,a_i}. Following (4.4), the signal expansion components are $\widetilde{s} = Vs$ for signal $s \in \mathbb{C}^N$, where

$$\widetilde{s} = [\widetilde{s}_{1,1} \cdots \widetilde{s}_{1,a_1} \cdots \widetilde{s}_{k,1} \cdots \widetilde{s}_{k,a_k}]^T. \tag{6.5}$$

Therefore,

$$\begin{aligned}
\breve{s}_i &= \widetilde{s}_{i,1} v_{i,1} + \cdots \widetilde{s}_{i,a_i} v_{i,a_i} && (6.6)\\
&= V_i^0 \widetilde{s} && (6.7)\\
&= V_i^0 V^{-1} s && (6.8)\\
&= V \begin{bmatrix} \ddots & & \\ & I_{a_i} & \\ & & \ddots \end{bmatrix} V^{-1} s && (6.9)\\
&= Z_{i0} s, && (6.10)
\end{aligned}$$

where Z_{i0} is the *first component matrix* of the ith generalized eigenspace [33]. It is a projection matrix and has an image equal to its generalized eigenspace; the reader is directed to Theorems 9.5.2 and 9.5.4 in [33] for the proofs.

The AIM provides a unique, Fourier transform-like projection space for a signal that is analogous to the definition (4.2) based on Jordan subspaces because the signal space is a direct sum of the generalized eigenspaces. The generalized eigenspaces are not necessarily irreducible components of the signal space, however, as explained in Chapter 3.2. Therefore, (6.2) is inexact, and is not strictly a formulation of a true graph Fourier transform according to the definitions utilized in algebraic signal processing [9, 10].

Note that the AIM (6.2) resolves to the Jordan subspace-based GFT (4.2) whenever the maximum number of Jordan subspaces per distinct eigenvalue is one (i.e., the characteristic and minimal polynomials of A are equal). Otherwise, Jordan subspace information is lost when the AIM is applied.

Next, key properties of (6.2) are discussed, with particular emphasis on the generalized Parseval's identity, graph equivalence classes with respect to (6.2), and total variation ordering.

6.2 Generalized Parseval's Identity

Since the full graph Fourier basis (comprising all eigenvectors and generalized eigenvectors) exists, regardless of whether we compute it or not, the generalized Parseval's identity of Chapter 4.2 still holds for the inexact method. Therefore, the biorthogonal basis set can be used to define signal energy.

To define the energy of a signal projection onto an AIM GFT component $\breve{s}_i$ for signal s, let V and W represent the eigenvector and biorthogonal eigenvector matrices, respectively. Write $\breve{s}_i$ in terms of the columns of V as $\breve{s}_i = \alpha_i v_{i,1} + \dots \alpha_{a_1} v_{i,a_i}$ and in terms of the columns of W as $\breve{s}_i = \beta_i w_{i,1} + \dots \beta_{a_i} w_{i,a_i}$. Then the energy of $\breve{s}_i$ can be defined as

$$\|\breve{s}_i\|^2 = \langle \alpha, \beta \rangle \tag{6.11}$$

using the notation $\alpha = (\alpha_1, \dots, \alpha_{a_i})$ and $\beta = (\beta_1, \dots, \beta_{b_i})$. Equation (6.11) is used in Chapter 10 to compare the inexact method and original GFT formulations.

6.3 Graph Equivalence under the Agile Inexact Method

Just as different choices of Jordan subspace bases yield the same GFT over Jordan equivalence classes of graph topologies, as explained in Chapter 5, different choices of bases for the generalized eigenspaces of the adjacency matrix yield the same inexact GFT. We define a $\mathscr{G}$-equivalence class as follows:

Definition 6.1 ($\mathscr{G}$-Equivalent Graphs). *Consider graphs $\mathcal{G}(A)$ and $\mathcal{G}(B)$ with adjacency matrices $A, B \in \mathbb{C}^{N\times N}$. Then $\mathcal{G}(A)$ and $\mathcal{G}(B)$ are* $\mathscr{G}$-equivalent graphs *if all of the following are true:*

1. *$\mathcal{G}_A$ and $\mathcal{G}_B$ are cospectral; and*

2. *$\{\mathscr{G}_{A,i}\}_{i=1}^k = \{\mathscr{G}_{B,i}\}_{i=1}^k$, where k is the number of distinct eigenvalues, $\mathscr{G}_{A,i}$ is the generalized eigenspace of A corresponding to eigenvalue λ_i, and $\mathscr{G}_{B,i}$ is the generalized eigenspace of B corresponding to eigenvalue λ_i.*

Denote by $\mathbf{G}_A^H$ the $\mathscr{G}$-equivalence class of $\mathcal{G}(A)$.

Graph equivalence under the AIM is independent of the Jordan subspaces and their dimensions (the Segre characteristic), in contrast to Jordan equivalence classes in Chapter 5. As a result, the inexact GFT is simpler and faster to compute, as discussed in greater detail in Section 6.6.

6.4 Total Variation Ordering

In Section 5.9, we defined an ordering on the GFT spectral components based on the total variation of a signal. In this section, similar results are proven for the GFT spectral components obtained via the AIM.

Denote by V_i the submatrix of columns of eigenvector matrix V that span $\mathscr{G}_i$, the ith generalized eigenspace. The (graph) total variation of V_i is defined as

$$\mathrm{TV}_G\left(V_i\right) = \left\|V_i - AV_i\right\|_1. \tag{6.12}$$

Then the following theorems follow immediately from the proofs of Theorems 5.21, 5.23, and 5.24.

Theorem 6.2. *Let $A, B \in \mathbb{C}^{N\times N}$ and $\mathcal{G}(B) \in \mathbf{G}_A^I$, i.e., $\mathcal{G}(B)$ is isomorphic to $\mathcal{G}(A)$. Let $V_{A,i} \in \mathbb{C}^{N\times a_i}$ be a union of Jordan chains that span $\mathscr{G}_i$ of matrix A and $V_{B,i} \in \mathbb{C}^{N\times a_i}$ the corresponding union of Jordan chains of B. Then $\mathrm{TV}_G(V_{A,i}) = \mathrm{TV}_G(V_{B,i})$.*

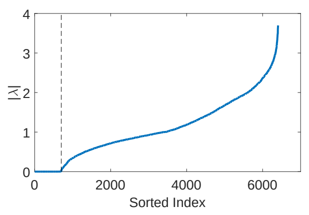
Theorem 6.3. *The eigenvector matrix V of adjacency matrix $A \in \mathbb{C}^{N\times N}$ can be chosen so that each union of Jordan chains that span the ith generalized eigenspace, represented by the eigenvector submatrix $V_i \in \mathbb{C}^{N\times a_i}$, satisfies $\left\|V_i\right\|_1 = 1$; i.e., $\|V\|_1 = 1$ without loss of generality.*

Theorem 6.4. *Consider matrix A with k distinct eigenvalues and $N \times a_i$ eigenvector submatrices V_i with columns corresponding to the union of the Jordan chains of λ_i, $i = 1, \dots, k$. Then the graph total variation is $\mathrm{TV}_G(V_i) \leq |1 - \lambda_i| + 1$.*

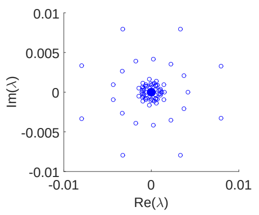
As in Chapter 5.9, we generalize the total variation to a class total variation defined over the $\mathscr{G}$-equivalence class associated with the graph of adjacency matrix A. We define the *class total variation* of spectral component $\mathscr{G}_i$ as the supremum of the graph total variation of V_i over the $\mathscr{G}$-equivalence class (for all $\mathcal{G}(B) \in \mathbf{G}_A^H$):

$$\mathrm{TV}_{\mathbf{G}_A^H}\left(\mathscr{G}_i\right) = \sup_{\substack{\mathcal{G}(B)\in\mathbf{G}_A^H \\ B = VJV^{-1} \\ \mathrm{span}\{V_i\} = \mathscr{G}_i \\ \|V_i\|_1 = 1}} \mathrm{TV}_G\left(V_i\right). \tag{6.13}$$

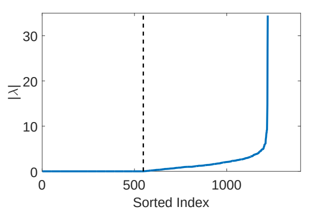
The remaining results of Chapter 5.9 follow in the same way. In particular, the low-to-high variation sorting is achieved via the function $f(\lambda_i) = |1 - \lambda_i| + 1$.



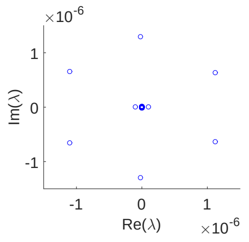
(a) Eigenvalue magnitudes for road network



(b) Small eigenvalues for road network



(c) Eigenvalue magnitudes for political blog



(d) Small eigenvalues for political blog.

Figure 6-2: (a) Eigenvalue magnitudes of a directed New York City road network adjacency matrix. The magnitudes are small in the index range 1 to 699 (up to the dotted line). (b) Road network eigenvalues (699 total) in the sorted index range 1 to 699 plotted on the complex plane. (c) Eigenvalue magnitudes of the largest weakly connected component of a political blog network. The magnitudes are small in the index range 1 to 548 (up to the dotted line). (d) Blog eigenvalues (548 total) in the sorted index range 1 to 548.

6.5 Applicability to Real-World Networks

This section discusses how the AIM can be applied on real-world large, sparse, and directed networks. In particular, we highlight the ease of computing the AIM in the case of a single generalized eigenspace of dimension greater than one.

6.5.1 Large, Directed, and Sparse Networks

In practice, sparsity in directed networks yields a zero eigenvalue of high algebraic and geometric multiplicities that may not be equal. This behavior can be demonstrated using two examples. The first is a directed, strongly connected road network $\mathcal{G}_{\text{rd}} = \mathcal{G}(A_{\text{rd}}) = (\mathcal{V}_{\text{rd}}, \mathcal{E}_{\text{rd}})$ of Manhattan, New York [35]; $\mathcal{G}_{\text{rd}}$ contains 6,408 nodes that represent the latitude and longitude coordinates of intersections and mid-intersection locations (available from [67]), as well as 14,418 directed edges, where edge $e = (v_1, v_2)$ represents a road along which traffic is legally allowed to move from v_1 to v_2 as determined from Google Maps [35]. The second network example is the largest weakly connected component $\mathcal{G}_{\text{bl}} = \mathcal{G}(A_{\text{bl}}) = (\mathcal{V}_{\text{bl}}, \mathcal{E}_{\text{bl}})$ of a political blog network, with 1,222 blogs as nodes and 19,024 directed edges, where edge $e = (v_1, v_2)$ exists between blogs $v_1, v_2 \in \mathcal{V}_{\text{bl}}$ if v_1 contains a link to v_2 [68].

Figures 6-2b and 6-2d show that the numerical eigenvalues of these real-world networks occur in clusters on the complex plane. These eigenvalues have small magnitude (lying left of the dashed lines in Figures 6-2a and 6-2c). On first inspection, it is not obvious whether the numerical eigenvalues are spurious eigenvalues clustered around a true eigenvalue of zero, or whether the numerical eigenvalues are indeed the true eigenvalues.[1] This phenomenon is typically observed in systems with multiple eigenvalues [69, 70, 71]. In practice, we can verify that numerical zero is an eigenvalue using either pseudospectra [72] or techniques that explore the numerical stability of the singular value decomposition [73, 74]. For the examples explored here, the singular value decomposition demonstrated that the clusters of low-magnitude eigenvalues represented a numerical zero eigenvalue. This method involves comparing small singular values to machine precision and is discussed in more detail in Section 10.2.

Furthermore, the adjacency matrices A_{rd} and A_{bl} have null spaces of dimension 446 and 439, respectively, confirming that eigenvalue $\lambda = 0$ has high algebraic (and geometric) multiplicity. In addition, eigenvector matrices V_{rd} and V_{bl} computed with MATLAB's eigensolver have numerical rank $6363 < 6408$ and $910 < 1222$, respectively, implying the existence of nontrivial Jordan subspaces (Jordan chains of length greater than one) for $\lambda = 0$. While these general eigenspace properties can be readily determined, a substantial amount of additional computation is required to determine the dimensions of each Jordan subspace for eigenvalues $\lambda = 0$. For example, the Jordan subspaces for zero can be deduced by computing a generalized null space as in [75], but this computation takes $O(N^3)$ or $O(N^4)$ for an $N \times N$ matrix.

On the other hand, the eigenvectors corresponding to eigenvalues of higher magnitude in these networks are full rank. In other words, from a numerical perspective, the Jordan subspaces for these eigenvalues are all one-dimensional.

In such networks, for which sparsity yields a high-multiplicity zero eigenvalue while the other eigenvalues correspond to a full-rank eigenvector submatrix, the inexact method (6.2) can be applied in the following way. Let V_{known} be the full-rank eigenvector submatrix corresponding to nonzero eigenvalues. The remaining submatrix $\widetilde{V}$ must have columns that span the generalized eigenspace $\mathcal{G}_0$ corresponding to eigenvalue zero.

A simple way to find a spanning set of columns for $\widetilde{V}$ is to find the kernel of V_{known}^T. The resulting inexact graph Fourier basis is

$$\widehat{V} = \begin{bmatrix} V_{\mathrm{known}} & \mathrm{Ker}\left(V_{\mathrm{known}}^T\right) \end{bmatrix}. \tag{6.14}$$

If the original network has adjacency matrix $A = VJV^{-1}$, then (6.14) implies that the network with adjacency matrix $\widehat{A} = \widehat{V}J\widehat{V}^{-1}$ is in the same $\mathscr{G}$-equivalence class as the original network. As shown in Chapter 6.3, this implies that the AIM GFT is equal over both networks.

In this way, for large, sparse, and directed real-world networks that exhibit a single zero eigenvalue of high multiplicity, the AIM can simplify the computation of a graph Fourier basis. Instead of computing

[1]The eigenvalues in Figure 6-2 were computed in MATLAB. Similar clusters appear using `eig` without balancing and `schur`. They also appear when analyzing row-stochastic versions of the asymmetric adjacency matrices.

Jordan chains or generalized null spaces to deduce the structure of the Jordan subspaces, a single kernel computation (with SVD, for example) is needed to compute the spanning basis. This idea is explored further in Section 6.6.

6.5.2 Multiple Nontrivial Jordan Subspaces

In the case of multiple distinct eigenvalues with large but unequal algebraic and geometric multiplicities, it may be possible to compute a few Jordan chain vectors to differentiate the nontrivial generalized eigenspaces. This approach is only computationally efficient if the number of Jordan chain vectors to compute is relatively small, as discussed in [34, 76].

When this is not feasible, a coarser inexact method can be obtained by reformulating (6.14) so that V_{known} consists of all known eigenvectors and is full rank. Then a basis of the kernel of V_{known}^T can be computed. These vectors can be allocated at random to the nontrivial generalized eigenspaces.

The dimensions of each generalized eigenspace must be known so that the correct number of vectors is allocated to each eigenspace. The dimensions can be determined by recursively computing for eigenvalue λ_i

$$f(l) = \dim \operatorname{Ker}(A - \lambda_i I)^l - \dim \operatorname{Ker}(A - \lambda_i I)^{l-1}, \qquad l = 2, \dots, N. \tag{6.15}$$

This equation provides the number of Jordan chains of length at least l [33]. The value of $\dim \operatorname{Ker}(A-\lambda_i I)^{l-1}$ for $l > 1$ at which $f(l) = 0$ is the dimension of generalized eigenspace $\mathscr{G}_i$. If $|\lambda_{\max}| > 1$, the condition $f(l) = 0$ may not be attained; instead, $f(l)$ becomes a monotonically increasing function for large l. In this case, the value of $\dim \operatorname{Ker}(A - \lambda_i I)^{l-1}$ for l at which this occurs is the approximate generalized eigenspace dimension.

The random assignment of missing eigenvectors to the nontrivial generalized eigenspaces would be a coarser inexact method compared to the AIM method (6.2); in particular, it is unknown which eigenvector assignment, if any, corresponds to a graph in the same $\mathscr{G}$-equivalence class as the original graph. On the other hand, the matrices built from a series of random assignments could be used to construct a filter bank such that each assignment corresponds to a different graph filter. An optimal or near-optimal weighted combination of such filters could be learned by using time-series of filtered signals as inputs to train a classifier. While the resulting Fourier basis is only an approximation of the one required for the AIM method (6.2), such a learned combination of filters would provide an interpretable tool to analyze graph signals while also ranking each filter (representing a random assignment of vectors to generalized eigenspaces) in terms of its utility for typical graph signals.

6.6 Runtime vs. Fidelity Trade-off

While maximizing the fidelity of the network representation is desirable in general, obtaining as much knowledge of the Jordan chains as possible is computationally expensive. As this section will demonstrate, execution time for the Jordan decomposition is, to first order, linear with respect to the number of Jordan chain vectors to compute. In practice, however, computation of increasing numbers of chains requires allocating memory for matrices of increasing size. This memory allocation time can increase nonlinearly once specific limits imposed by the computing hardware are exceeded. Since practical applications of GFT analysis may be constrained by execution time requirements and available computing hardware, methods that allow fidelity vs. speedup tradeoffs will prove useful.

Our inexact GFT approach provides precisely such a method for trading between higher fidelity analysis (a greater number of Jordan chains computed) and execution time, as discussed in Section 6.5. In this section, the details of such trade-offs are discussed in detail.

To illustrate the cost of computing the Jordan chain vectors of $A \in \mathbb{C}^{N\times N}$ that complete the generalized eigenspace of eigenvalue λ_i, consider the general algorithm given by:

```
function COMPUTE_CHAINS
    V = FULL_RANK_EIGENVECTORS($A$)
    R = PSEUDOINVERSE($A - \lambda_i I$)
    N = KER($A - \lambda_i I$)
    Nnew=[ ]
    for c in 1..maximum chain length do
        for v in N do
            if [$(A - \lambda_i I)$  v] full rank then
                v2 = R*v
                Append v2 to V.
                Append v2 to Nnew.
            end if
        end for
        N = Nnew
    end for
end function
```

The eigenvector and pseudoinverse computations are pre-processing steps that can be executed just once for a network with stationary topology. Efficient methods for these computations include power iterations, QR algorithms, and, for large, sparse matrices, Lanczos algorithms and Krylov subspace methods [34]. The key bottlenecks of concern reside in the `for` loop. This loop consists of a rank-checking step to verify that the current eigenvector is in the range space of $A - \lambda_i I$. This step can be implemented using the singular value decomposition, which has $O(kMN^2 + k'M^3)$ operations [34] on an $M \times N$ matrix, $N < M$; constants k and k' could be 4 and 22, respectively, as in the R-SVD algorithm of [34]. In addition, the matrix-vector product in the `for` loop takes about $2MN$ operations. The original dimension of V is $M \times N(j)$, where the number of columns $N(j)$ approaches M from below as the number j of vectors traversed increases. The resulting time complexity for a single iteration is

$$O(kMN(j)^2 + k'M^3) + O(2MN(j)). \tag{6.16}$$

The first `for` loop will run for m_i iterations, where m_i is the maximum chain length corresponding to λ_i. This can be estimated as demonstrated in Section 10.3. The second `for` loop will first run for g_i iterations, where g_i is the dimension of the null space of $A - \lambda_i I$ (the geometric multiplicity of λ_i). On the lth run of the outer loop, the number of iterations of the inner loop equals the difference of kernel dimensions

$$f(l) = \dim \operatorname{Ker}(A - \lambda_i)^l - \dim \operatorname{Ker}(A - \lambda_i)^{l-1}, \tag{6.17}$$

a monotonically decreasing function of l [33]. Therefore, the total number of iterations is $m_i \cdot \sum_{l=1}^{m_i} f(l)$, or

$$m_i \cdot (\dim \operatorname{Ker}(A - \lambda_i I)^{m_i} - \dim \operatorname{Ker}(A - \lambda_i I)) = m_i \cdot (a_i - g_i), \tag{6.18}$$

where a_i and g_i are the algebraic and geometric multiplicities of λ_i, respectively. Since this total depends on the adjacency matrix, we will denote it as b_A, which has an implicit dependence on the $M \times N(j)$ dimensions of A. The total time complexity of the `for` loops is then

$$\sum_{j=1}^{b_A} O(kMN(j)^2 + k'M^3) + O(2MN(j)). \tag{6.19}$$

In addition, the time to allocate memory for the matrix $[(A - \lambda I)\ \ \mathrm{v}]$ is $O(MN(j)) \cdot m(MN(j))$, where $m(\cdot)$ is the platform-dependent time per unit of memory allocation as a function of the matrix size. Since each `for` loop allocates memory in this way, the total memory allocation time is

$$\sum_{j=1}^{b_A} O(MN(j)) \cdot m(MN(j)). \tag{6.20}$$

Assuming c is time per floating-point operation that is also platform-dependent and SVD constants $k = 4$ and $k' = 22$, the total expected runtime for the Jordan chain computation can be approximated as

$$\sum_{j=1}^{b_A} c\left(4MN(j)^2 + 22M^3 + 2MN(j)\right) + MN(j)m\left(MN(j)\right), \tag{6.21}$$

where c is the platform-dependent time per floating-point operation and the full SVD complexity coefficients are set to $k = 4$ and $k' = 22$.

Equation (6.21) shows that the runtime for the Jordan chain computations is linear with the number of missing vectors one needs to compute. However, in practice, the time to allocate memory can scale nonlinearly with the number of nodes as the size of the eigenvector matrix approaches system capacity. For networks at massive scale, this can present a significant problem.

In contrast, the algorithm to compute an orthogonal set of missing vectors is very fast. The algorithm is the following, where Vt denotes the transpose of V:

```
function COMPUTE_MISSING
    V = FULL_RANK_EIGENVECTORS($A$)
    Vt = TRANSPOSE($V$)
    Vnew = KER(Vt)
    V = [V  Vnew]
end function
```

The slowest step is computing the null space of V^T, which has time complexity $O(kMN(j)^2+k'M^3)$ for a SVD-based null space computation. In addition, memory allocation for V^T and V each takes $O(MN(j))$ time. Notably, this allocation is only performed once in the AIM implementation, as compared to the repeated allocations required in the `for` loop of the above Jordan chain decompositon.

This section demonstrates how our inexact method can be employed to enable GFT analysis to meet execution time constraints by trading runtime for fidelity, by combining the inexact method with projections onto Jordan subspaces. The analysis of execution times in these two cases provides the mathematical basis for quantifying execution times.

In summary, this chapter defined the AIM (6.2) for applying the graph Fourier transform (4.2). This formulation does not require the computation of Jordan chains but loses fidelity to the original graph eigenstructure. We also show that the generalized Parseval's identity and total variation ordering of spectral components from the Jordan subspace-based transform apply to the inexact case.

Sections 6.5 and 6.6 demonstrate considerations that need to be addressed when applying the graph Fourier transform to real-world problems. The general method for applying graph signal processing is discussed in the next chapter.

Part III

Methods and Considerations for Applying the Graph Fourier Transform

Part III summarizes the methodology and pitfalls of applying the graph Fourier transform to real-world networks. Chapter 7 presents the general method, which consists of (1) signal extraction, (2) eigendecomposition of the adjacency matrix, and (3) computation of the GFT. Particular issues regarding the choice of platforms and pre-processing steps, as well as techniques for handling large-scale networks and massive data sets, are discussed at a high level.

Chapter 7

Applying Graph Signal Processing to Real-World Networks

The graph Fourier transform identifies highly expressed spectral components that reflect localized co-behavior of nodes in a network. For example, this framework is well suited for finding high-density traffic locations on a road network or influential nodes in a social network. As the available data and underlying networks increase in size, graph signal processing provides an especially attractive method for finding localized behavior that may not be evident from raw signals or other methods. This chapter presents a general approach for computing graph Fourier transforms of rich, batch signals on large, real-world networks.

The application of the graph Fourier transform (4.2) involves three distinct steps: signal extraction, eigendecomposition, and computation of the graph Fourier transform itself. This chapter discusses these steps in addition to computational and numerical properties that influence the design of platform-specific implementations. Figure 7-1 illustrates that the GFT and inexact methods combine the signal extraction and eigendecomposition outputs by projecting graph signals onto the known eigenvectors and the nontrivial Jordan subspaces or generalized eigenspaces of the adjacency matrix. Projection onto generalized eigenspaces is preferable when the Jordan chain computation is too expensive; see Section 6.6.

7.1 Signal Extraction

The first step in applying graph signal processing is construction of graph signals from the data. This section briefly describes frameworks for real-world applications and related considerations.

Scientific computing languages such as Python [77] and MATLAB [78] as well as new computing

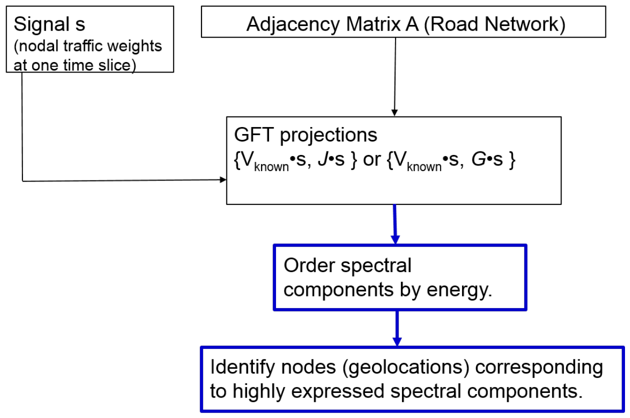

Signal s
(nodal traffic weights at one time slice)
Adjacency Matrix A (Road Network)
GFT projections
{V_known•s, J•s } or {V_known•s, G•s }
Order spectral components by energy.
Identify nodes (geolocations) corresponding to highly expressed spectral components.

Figure 7-1: Method for applying graph signal processing. The columns of V_{known} represent known eigenvectors. The GFT involves projections of a signal s onto the columns of V_{known} as well as the the Jordan subspaces J or generalized eigenspaces G of the adjacency matrix. The blue boxes highlight an example analysis that can be performed with the GFT results.

languages such as Julia [79] provide platforms for computing statistics and modeling graphs and other complex data structures. While these languages offer convenient features for rapid prototyping, they may require substantial amounts of memory that limit their use on very large datasets. In memory-constrained systems, signal extraction may require array-based implementations in the C programming language for efficiency.

Additional speedups can be achieved by decomposing the signal extraction into jobs that can run simultaneously on high-throughput computing platforms such as Hadoop [80] and Spark [81] Spark uses resilient distributed dataset (RDD) structures to provide faster computation times than the Hadoop framework for iterative methods such as training machine learning algorithms. When the signal computation can be decomposed into small jobs with a low memory footprint, an appropriate platform is HTCondor, an open-source high throughput computing environment that runs on a machine cluster [82].

Platform selection depends on the data format, the required data pre-processing, and the available hardware. If the pre-processing requires in-memory storage of complex data structures at runtime, e.g., for computing graph-based statistics, it is essential to choose platforms with minimal overhead and to utilize or design memory-efficient data representations so that the problem is feasible. In addition, implementing efficient algorithms facilitates parallelization of the signal extraction. We illustrate a design process for signal extraction in Chapter 9.

7.2 Eigendecomposition

Obtaining the eigendecomposition of the graph adjacency matrix $A \in \mathbb{C}^{N \times N}$ is essential for applying graph signal processing. Numerically stable methods such as the power method, QR algorithms, and Jacobi methods for diagonalizable matrices are applicable methods for eigendecomposition; see [34] and the references therein for more details. Furthermore, eigendecompositions of large, sparse matrices can be computed with iterative Lanczos or Krylov subspace methods [34, Chapter 10], [83].

On the other hand, for defective or nearly defective matrices, small perturbations in network structure due to numerical round-off errors can significantly perturb the numerical eigenvalues and eigenvectors; for example, for a defective eigenvalue of A corresponding to a p-dimensional Jordan block, $O(\epsilon)$ perturbations in A can result in $O(\epsilon^{1/p})$ perturbations in λ [34, 84]. In addition, while computing the Jordan normal form can be numerically unstable [73, 74, 76, 34], forward stability was shown for the Jordan decomposition algorithm in [85] and forms the basis for an SVD-based implementation discussed in Chapter 10.

7.3 GFT Computation

The final step is the computation of the graph Fourier transform as in (4.2) or (6.2). The computation is essentially a sparse matrix-vector or matrix-matrix computation since many real-world networks have highly sparse connections. As a result, very large multiplications can be computed by vectorizing or optimizing hardware; see, for example, [86].

The output of these multiplications is the projection of the signal onto invariant subspaces (either Jordan subspaces or generalized eigenspaces) of the graph adjacency matrix A. The energy of the signal's projection onto each spectral component can be computed by (4.21) or (6.11) and ordered by total variation as described in Sections 5.9 and 6.4. Such an ordering provides insight into the low-frequency versus high-frequency nature of a signal; see [8] for more details.

In order to identify highly expressed spectral components, the following procedure can be applied. First, the spectral components are ranked by decreasing energy. Then, a threshold is applied to extract the components corresponding to a certain amount of signal energy. These components correspond to node weights that highlight influential behaviors.

This chapter summarizes the steps of computing the GFT (4.2) or the AIM (6.2). One potential issue of this method is ill-conditioning of the eigendecomposition step. The following chapter provides illustrative examples to examine the numerical behavior of example defective and nearly defective matrices.

Chapter 8

Numerical Issues for Perturbed Eigendecompositions

The sensitivity of defective or nearly defective matrices to small perturbations is a source of concern in the application of graph signal processing. For example, introducing an edge that has a weight equal to a small $\epsilon > 0$ worsens the conditioning of a binary $(0/1)$ matrix and creates numerical instabilities even for simple graph structures. This is an important and disconcerting issue when faced with the problem of computing Jordan chains.

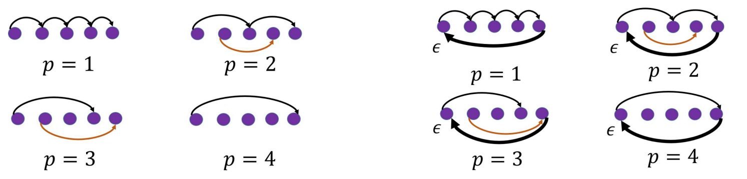
Our objective in this chapter is to illustrate particular cases where numerical eigendecompositions behave erratically. Sensitivity of eigenvalues and eigenvectors to matrix perturbations has also been studied in, for example, [76, 83, 87, 88, 89]. These references characterize properties of general perturbations to a matrix A; i.e., they study behavior for matrices $\widetilde{A}$ that satisfy $\|A - \widetilde{A}\| < \epsilon$ for small $\epsilon > 0$. This chapter does not perform such analysis; instead, it considers a particular perturbed structure for A and presents the analysis in terms of the matrix eigendecomposition.

In order to understand the problem of numerical instability in more detail, we analyze two example matrices: the Jordan block $H_N = J(0)$ and its powers in Section 8.1, and the adjacency matrix of a directed, acyclic feed-forward network in Section 8.2. The corresponding digraphs are shown in Figures 8-1a and 8-2a. Each section is ordered in the following way:

1. Eigendecomposition analysis of the adjacency matrix;

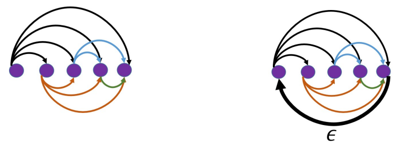
2. Numerical eigendecomposition;

3. Eigendecomposition analysis of a perturbed adjacency matrix; and



(a) $\mathcal{G}\left(H_5^p\right)$ (b) $\mathcal{G}\left(H_5^p|_{\epsilon,j}\right)$

Figure 8-1: Example digraphs for (a) Jordan block powers and (b) perturbed powers.



(a) $\mathcal{G}\left(A_5\right)$ (b) $\mathcal{G}\left(A_5|_{\epsilon,5}\right)$

Figure 8-2: Digraphs for (a) A_5 and (b) $A_5|_{\epsilon,5}$. Edges that originate from the same node are shown with the same color.

4. Numerical eigendecomposition for the perturbed case.

Omitted proofs for the eigendecompositions can be found in Appendices A and B.

8.1 Eigendecomposition for Jordan Blocks and Their Powers

This section presents results on the eigendecomposition of the $N \times N$ Jordan block $H_N = J(0)$ and its powers. Section 8.1.1 constructs the Jordan normal form and eigenvector matrices for these matrices. Section 8.1.2 discusses the ability to extract the eigendecomposition using MATLAB [78] as a numerical tool. Perturbations that correspond to the digraphs in Figure 8-1b are also studied by exploiting underlying graph structure to find the spectral decompositions in Section 8.1.3; also see [90, 91, 92]. The numerical behavior of the perturbed matrices is illustrated in 8.1.4. Omitted proofs can be found in Appendix A.

8.1.1 Analysis for Eigendecomposition of H_N^p

A recursive formulation for the Jordan canonical form is found. In addition, eigenvector matrices for the matrix are computed.

Jordan normal form for H_N^p. A recursion for the structure of Jordan form of H_N^p, $p = 1, 2, \ldots N$, is shown here. Please see Appendix A for the proofs. Figure 8-1a shows that H_N^p corresponds to a digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consisting of p disjoint connected components that are either *isolated nodes*, which have no out- or in-edges in $\mathcal{E}$, or *directed paths*, where a directed path from v_i to v_j, $i \neq j$, is a sequence $(v_i, w_1, \ldots, w_k, v_j)$ of distinct nodes in V such that $(v_i, w_1), (w_k, v_j), (w_l, w_{l+1}) \in \mathcal{E}$ for $l = 1, \ldots, k-1$. For $p \geq N$, the digraph consists of N isolated nodes since $H_N^p = 0_{N \times N}$.

Since H_N and its powers are upper triangular with zero diagonal entries, they have a zero eigenvalue with algebraic multiplicity N. Define the Jordan decomposition $H_N^p = V_{N,p}J_{N,p}V_{N,p}^{-1}$. The next theorem describes the form of $J_{N,p}$.

Theorem 8.1. *The following statements characterize the Jordan canonical form $J_{N,p}$ of H_N^p:*

(i) $J_{N,p}$ consists of p Jordan blocks.

(ii) The maximum block size of $J_{N,p}$ is $\left\lceil \frac{N}{p} \right\rceil$, or the minimum integer greater than or equal to $\frac{N}{p}$.

(iii) The minimum block size of $J_{N,p}$ is $\left\lfloor \frac{N}{p} \right\rfloor$, or the maximum integer less than or equal to $\frac{N}{p}$.

The properties of Theorem 8.1 lead to a recursion for the Jordan blocks of H_N^p that is dependent on the *Segre characteristic* of the Jordan normal form, which is defined as the list of Jordan block sizes in decreasing order for a given eigenvalue [33]. Together with the associated eigenvalues, the Segre characteristic uniquely characterizes the Jordan form [33] and is provided for H_N^p in the following lemma.

Lemma 8.2. *The* Segre characteristic $(e_1, \dots, e_p)$ *of H_N^p is*

$$e_i = \begin{cases} \left\lceil \frac{N}{p} \right\rceil, & \text{if } i \in [1, N - pl + p] \\ \left\lfloor \frac{N}{p} \right\rfloor, & \text{if } i \in [N - pl + p + 1, p] \end{cases} \tag{8.1}$$

where l is the maximum Jordan chain length of A.

Lemma (8.2) leads to a recursion for the Jordan form of H_N^p in the next theorem.

Theorem 8.3. *The Jordan form $J_{N,p}$ of H_N^p can be decomposed as the following (up to a permutation on the order of blocks) for all integers $p = 1, \dots, N-1$ and $N \geq 3$:*

$$J_{N,p} = \begin{bmatrix} H_{\lceil \frac{N}{p} \rceil} & \\ & J_{N - \lceil \frac{N}{p} \rceil, p-1} \end{bmatrix}. \tag{8.2}$$

Theorems 8.1 and 8.3 show the dependence of Jordan block counts and sizes on N and p. As p increases for fixed N, the maximum Jordan block size decreases. As seen in Figure 8-1a, the decrease in Jordan block size is reflected in the corresponding digraph as an increase in the number of weakly connected components. The proper and generalized eigenvectors are characterized in the next section.

Eigenvector matrices of H_N^p. This section finds proper and generalized eigenvectors of H_N^p. Let the ith column of $E_{N,k} \in \mathbb{R}^{N \times k}$ be the ith canonical vector.

Theorem 8.4. *One possible set of proper eigenvectors for the matrix H_N^p, $p = 1, \ldots, N$ is the set of columns of $E_{N,p}$.*

The eigenvectors from Theorem 8.4 can be used to generate Jordan chains corresponding to the Jordan blocks of Theorem 8.3. Lemma 8.5 shows the number of nontrivial Jordan chains.

Lemma 8.5. *Of the p eigenvectors of H_N^p, $\min(N-p,p)$ of them correspond to Jordan blocks of size $r > 1$.*

Proof. Choose the eigenvectors of H_N^p to be the columns of $E_{N,p}$ as in Theorem 8.4. For a Jordan block of size $r > 1$ and $\lambda = 0$, the generalized eigenvectors for the Jordan chain satisfy the recurrence equation [33]

$$H_N^p v_i = \lambda v_i + v_{i-1} = v_{i-1}, \qquad i = 2, \ldots, r, \tag{8.3}$$

where v_1 is a proper eigenvector. If v_1 is in the column space $R(H_N^p)$ of H_N^p, (8.3) is consistent. Since the column space of H_N^p is the span of the first $N-p$ canonical vectors of $\mathbb{R}^N$, the matrix of proper eigenvectors corresponding to Jordan blocks of size $r > 1$, denoted here by W, satisfies

$$\begin{aligned} \operatorname{span}\{W\} &= \operatorname{span}\{E_{N,p}\} \cap R\left(H_N^p\right) && (8.4)\\ &= \operatorname{span}\{E_{N,p}\} \cap \operatorname{span}\{E_{N,N-p}\} && (8.5)\\ &= \operatorname{span}\{E_{N,\min(p,N-p)}\} && (8.6) \end{aligned}$$

where $\operatorname{span}\{\cdot\}$ represents the span of the columns of a matrix. Therefore, there are $\min(N-p,p)$ proper eigenvectors that correspond to Jordan blocks of size $r > 1$. □

Assuming the proper eigenvectors provided by Theorem 8.4, the nontrivial Jordan chains of H_N^p can be characterized in terms of rectangular stride permutation matrices of order p as described in the next theorem

.

Theorem 8.6. *For Jordan blocks of H_N^p with size $r > 1$, the proper and generalized eigenvectors can be given by an $N \times r$ stride permutation matrix $P_{N,r}^{(k)}$ of stride p:*

$$\left[P_{N,r}^{(k)}\right]_{ij} = \begin{cases} 1, & \text{if } j = k + (ip \bmod N) + \lfloor ip/N \rfloor \\ 0, & \text{otherwise.} \end{cases} \tag{8.7}$$

where $0 \leq i, j \leq N-1$ and $1 \leq k \leq \min(N-p,p)$.

Proof. By (8.6) in the proof of Lemma 8.5, any column of $E_{N,\min(p,N-p)}$ corresponds to a proper eigenvector that generates a nontrivial Jordan chain. Set v_1 to the kth column of $E_{N,\min(p,N-p)}$ and note that it corresponds to the first column of matrix $P^{(k)}_{N,r}$ from (8.7).

Since $r > 1$, the generalized eigenvectors $v_2, \ldots, v_r$ in the Jordan chain for v_1 can be found by solving the recurrence equation (8.3). We solve for the first generalized eigenvector v_2. Let $w = v_2 = (w_1, \ldots, w_N)$. There are $N-p$ unit elements along the pth diagonal of H^p_N, so equation (8.3) constrains $w_{p+k} = 1$ in addition to $w_{p+n} = 0$ for $n = 1, \ldots, N-p$, $n \neq k$. The other elements of w are arbitrary, so they can be set to zero to get the second column of $P^{(k)}_{N,r}$.

Solving the second generalized eigenvector $w = v_3$ yields the constraints $w_{2p+k} = 1$ and $w_{p+n} = 0$ for $n = 1, 2, \ldots N-p$, $n \neq p+k$. The other elements of w are arbitrary, so they can again be set to zero. The resulting vector $w = v_3$ is the third column of $P^{(k)}_{N,r}$. By induction, the rth generalized eigenvector $w = v_r$ for proper eigenvector v_1 is constrained by $w_{(r-1)p+k} = 1$ and $w_{p+n} = 0$ for $n = 1, 2, \ldots N-p$, $n \neq (r-2)p+k$. Setting the other elements of w to zero yields the rth column of matrix $P^{(k)}_{N,r}$. □

The next theorem combines the results of Theorem 8.4, Theorem 8.6 and Lemma 8.5 to define an eigenvector matrix $V_{N,p}$ of H^p_N.

Theorem 8.7. *Let $r_1 \geq r_2 \geq \cdots \geq r_k$ denote sizes of Jordan blocks of H^p_N that are greater than one. Then an eigenvector matrix $V_{N,p}$ of H^p_N corresponding to Jordan form* (8.2) *is*

$$V_{N,p} = \begin{bmatrix} P^{(1)}_{N,r_1} \; P^{(2)}_{N,r_2} & \ldots & P^{(k)}_{N,r_k} & E_{N-p}\left(k+1 : N-p\right) \end{bmatrix}, \tag{8.8}$$

where $E_{N-p}\left(k+1 : N-p\right)$ denotes the last $N-p-k$ columns of proper eigenvector matrix E_{N-p}, and $P^{(i)}_{N,r_i}$ denotes the $N \times r_i$ stride permutation matrix (8.7) *of stride p.*

Proof. By Lemma 8.5, an eigenvector matrix $V_{N,p}$ of H^p_N has $k = \min(N-p, p)$ proper eigenvectors corresponding to Jordan blocks of size greater than one. By Theorem 8.6, the Jordan chains for Jordan blocks of size $r_i > 1$ can be chosen to be $P^{(i)}_{N,r_i}$ with stride p. The remaining proper eigenvectors corresponding to Jordan blocks of size one can be set to the last $N-p-k$ columns of E_{N-p}. □

Theorem 8.1, Theorem 8.3, and Theorem 8.7 provide the complete eigendecomposition of H^p_N. The next section discusses the numerical evaluation of this decomposition in MATLAB.

8.1.2 Numerical Eigendecomposition of H^p_N

The Jordan normal form and eigenvector matrix of Theorem 8.1 and Theorem 8.7 can be found numerically by MATLAB `jordan` [78], which computes the Jordan decomposition symbolically. This was tested for matrix dimensions $N \in [10, 100]$. The computation is very slow, however, so testing for N on the order of 1000 was not feasible on a single 16-core, 16-GB machine. In contrast, computing the eigendecomposition of H_N with MATLAB `eig` can be done for very large N. The output consists of only the proper eigenvectors as specified by Theorem 8.4. Therefore, the resulting eigenvector matrices are not full rank. In fact, the eigenvector matrices found in this way for H_N^p have rank p. In this way, multiplicity information for the zero eigenvalue can be deduced.

The next section studies perturbations on H_N^p in order to examine the behavior of the numerical decomposition in the presence of perturbed data or numerical sensitivities.

8.1.3 Perturbed H_N^p

Eigendecompositions of perturbations on H_N^p are presented in this section. The convergence of the eigenvalues of the perturbed matrix to the multiple zero eigenvalue of H_N^p is analyzed. Proofs for this section are provided in Appendix A.

Perturb H_N^p to $H_N^p|_{\epsilon,j} = H_N^p + \epsilon H'_{N,j}$, where $\epsilon > 0$ and $H'_{N,j} \in \mathbb{R}^{N\times N}$ is zero except for $[H'_{N,j}]_{j1} = 1$ for $2 \leq j \leq N$. For $j = N$, the matrix has characteristic polynomial

$$\varphi_{N,1}|_{\epsilon,N}(\lambda) = (-1)^N \left(\lambda^N - \epsilon\right) \tag{8.9}$$

for $p = 1$ and $j = N$, so the eigenvalues of $H_N|_{\epsilon,N}$ are

$$\lambda_r = \epsilon^{\frac{1}{N}} \exp\left(\frac{2\pi i r}{N}\right), r = 0, 1, \ldots, N-1. \tag{8.10}$$

The corresponding Jordan chains have length one because the eigenvalues are distinct (see also [83, p.15]). Note that $H_N|_{\epsilon,N}$ corresponds to a weighted directed cycle as in Figure 8-1b and is related to the shift matrix discussed in Section 3.1.2 as $\epsilon \to 1$.

For $2 \leq j < N$, the characteristic polynomial is given by

$$\varphi_{N,1}|_{\epsilon,j}(\lambda) = (-1)^N \lambda^{N-j}\left(\lambda^j - \epsilon\right). \tag{8.11}$$

Therefore, the eigenvalues of $H_N|_{\epsilon,j}$ are

$$\lambda_r = \begin{cases} \epsilon^{\frac{1}{j}} \exp\left(\frac{2\pi i r}{j}\right), & r = 0, 1, \ldots, j-1, \\ 0, & r = j, \ldots, N. \end{cases} \tag{8.12}$$

Figure 8-1b shows that digraph $\mathcal{G}(H_N^p|_{\epsilon,j})$ may become disconnected for $p > 1$ or lose strongly connectedness for $j < N$. As in [90, 91, 92], this type of digraph component structure allows the derivation of the spectrum. For this reason, Theorem 8.8 characterizes the structure of $H_N^p|_{\epsilon,j}$ in terms of isolated nodes, directed paths, and *directed cycles*, which are directed paths with identical start and end nodes.

Theorem 8.8. *The following properties characterize the component structure of the digraph $\mathcal{G}_{N,p}|_{\epsilon,j}$ of $H_N^p|_{\epsilon,j}$:*

(i) The digraph contains at most one directed cycle;

(ii) A directed cycle of length $\lceil \frac{j}{p} \rceil$ exists in $\mathcal{G}_{N,p}|_{\epsilon,j}$ if $j-1$ is a multiple of p.

(iii) The weakly connected components are directed chains or isolated nodes unless a directed cycle exists, in which case one weakly connected component is either a directed cycle or the union of a directed cycle and a directed path;

(iv) If a directed cycle exists, it is the sole strongly connected component of size greater than one. Otherwise, $\mathcal{G}_{N,p}|_{\epsilon,j}$ consists of N strongly connected components of size one.

Theorem 8.8 implies the eigenvalues of $H_N^p|_{\epsilon,j}$ as shown in the next theorem.

Theorem 8.9. *If $j-1$ is a multiple of p, then $H_N^p|_{\epsilon,j}$, $2 \le j \le N$, has $l = \lceil \frac{j}{p} \rceil$ eigenvalues λ_r that satisfy*

$$\lambda_r = \epsilon^{\frac{1}{l}} \exp\left(\frac{2\pi i r}{l}\right) \qquad r = 0, 1, \ldots, l-1 \tag{8.13}$$

and a zero eigenvalue of algebraic multiplicity $N - l$. Otherwise, $H_N^p|_{\epsilon,j}$ has a zero eigenvalue of algebraic multiplicity N.

The convergence of the eigenvalues λ_r to the multiple zero eigenvalue of H_N^p is analyzed next. Note that $l = \lceil \frac{j}{p} \rceil$ becomes large for small p and large j. Then, as $p \to 1$ and $j \to \infty$, $\log|\lambda_r| \to 0$, which implies that $|\lambda_r| \to 1$.

In the other direction, $l \to 1$ as j decreases and p grows. This implies $\log|\lambda_r| \to \log|\epsilon|$, or $|\lambda_r| \to \epsilon$.

Define $\epsilon = \alpha^N$, $0 < \alpha < 1$. Then $\log|\lambda_r| = \frac{N}{l} \log \alpha$. This implies that eigenvalue magnitudes are on the order of α even for perturbations exponentially decaying with N.

The number of Jordan blocks corresponding to the zero eigenvalue of $H_N^p|_{\epsilon,j}$ is found next.

Theorem 8.10. *The number of Jordan blocks k corresponding to the zero eigenvalue of $H_N^p|_{\epsilon,j}$, $2 \le j \le N$, is*

$$k = p - 1 + \delta\left(j \in [2, N-p]\right), \tag{8.14}$$

where $\delta(\cdot)$ is the Kronecker delta function.

Proof. The number of Jordan blocks for eigenvalue zero equals the number of free variables in the system $H_N^p|_{\epsilon,j}x = b$, where $x, b \in \mathbb{R}^N$. Since a zero column of $H_N^p|_{\epsilon,j}$ corresponds to a free variable and there are $p-1$ such columns, there must be $p-1$ Jordan blocks. An additional free variable is present if ϵ is not the sole nonzero element in the jth row. Since rows 1 to $N-p$ of $H_N^p|_{\epsilon,j}$ contain unit elements and $j \geq 2$, there is an additional Jordan block if $2 \leq j \leq N-p$. □

The next theorem provides the Jordan canonical form of $H_N|_{\epsilon,j}$.

Theorem 8.11. *The Jordan canonical form $J_{N,1}|_{\epsilon,j}$ of $H_N|_{\epsilon,j}$, $2 \leq j \leq N$, has form*

$$J_{N,1}|_{\epsilon,j} = \begin{bmatrix} H_{N-j} & 0_{(N-j)\times j} \\ 0_{j\times(N-j)} & \operatorname{diag}\left(\Lambda_j|_{\epsilon,j}\right) \end{bmatrix}, \tag{8.15}$$

where H_{N-j} is the Jordan block for $\lambda = 0$, and $\operatorname{diag}\left(\Lambda_j|_{\epsilon,j}\right)$ *is the $j \times j$ diagonal matrix with entries equal to the nonzero eigenvalues of Theorem 8.9.*

Proof. By Theorem 8.9, $H_N|_{\epsilon,j}$ has j distinct nonzero eigenvalues if $\mathcal{G}_N|_{\epsilon,j}$ contains a directed cycle. Let $\operatorname{diag}(\Lambda_j|_{\epsilon,j})$ denote a $j \times j$ diagonal matrix with the distinct eigenvalues on its diagonal. This provides the lower-right matrix in (8.15).

For the zero eigenvalue, the algebraic multiplicity is $N-j$, and the geometric multiplicity g_0 equals the dimension of $\operatorname{Ker}(H_N|_{\epsilon,j})$. If $p=1$ and $j=N$, there are no zero eigenvalues; otherwise, for ϵ in the jth row of the first column of A (i.e., $[A]_{j1} = \epsilon$, $2 \leq j < N$), there is a single free variable corresponding to row j in system $H_N^p|_{\epsilon,j}x = b$, where $x, b \in \mathbb{R}^N$. Thus, $g_0 = 1$, the number of Jordan blocks for $\lambda = 0$. □

The eigenvectors of $H_N|_{\epsilon,j}$ are next characterized. Theorem 8.12 finds eigenvectors for nonzero eigenvalues, and Theorem 8.13 finds eigenvectors for the zero eigenvalues.

Theorem 8.12. *The elements of eigenvector $v = (v_1, \ldots, v_N)$ corresponding to a nonzero eigenvalue of $H_N|_{\epsilon,j}$ satisfy*

$$v_i = \begin{cases} \lambda^{i-1}v_1 & \text{if } 1 \leq i \leq j, \\ \left(\lambda^j - \epsilon\right)v_1 & \text{if } i = j+1, \\ \lambda^{(i-j-1)}\left(\lambda^{j-1} - \epsilon\right)v_1 & \text{if } j+1 < i \leq N. \end{cases} \tag{8.16}$$

Proof. Solving $H_N|_{\epsilon,j}v = \lambda v$ yields

$$v_i = \lambda v_{i-1}, \qquad i \neq j+1 \tag{8.17}$$

$$\epsilon v_1 + v_i = \lambda v_{i-1}, \qquad i = j+1. \tag{8.18}$$

For $1 < i \leq j$, it can be shown by recursion that $v_i = \lambda^{i-1}v_1$. Combining $v_{j+1} = \lambda v_j - \epsilon v_1$ and $v_j = \lambda^{j-1}v_1$ yields $v_{j+1} = (\lambda^j - \epsilon)v_1$. This implies that $v_{j+2} = \lambda v_{j+1} = \lambda(\lambda^j - \epsilon)v_1$ for $i > j+1$. It can be shown recursively that $v_i = \lambda^{i-j-1}(\lambda^j - \epsilon)v_1$. □

Since $|\lambda| = \epsilon^{1/N} < 1$ for $\epsilon \in (0,1)$ by (8.10), Theorem 8.12 implies that eigenvectors corresponding to the nonzero eigenvalues have exponentially decaying elements.

The next theorem defines a set of proper eigenvectors for eigenvalue $\lambda = 0$.

Theorem 8.13. *The proper eigenvector of $H_N|_{\epsilon,j}$ corresponding to the Jordan block for the zero eigenvalue has two nonzero components v_{j+1} and $v_1 = -\frac{1}{\epsilon}v_{j+1}$.*

Proof. Solving $H_N|_{\epsilon,j}v = 0$ yields $v_i = 0$ for $i \notin \{1, j+1\}$ and $\epsilon v_1 + v_{j+1} = 0$. The latter equation shows the relation between the only nonzero elements. □

The $1/\epsilon$ factor in the first element of the eigenvector in Theorem 8.13 propagates through the Jordan chain by (8.3). Therefore, it is expected to observe instability in the eigenvector components as $\epsilon \to 0$. Furthermore, as the length of the Jordan chain increases (e.g., as $N \to \infty$ and $j \to 1$), the instability should be more noticeable in the generalized eigenvectors. The numerical behavior of the eigendecomposition is described in the next section.

8.1.4 Numerical Eigendecomposition of Perturbed H_N

The eigendecomposition of perturbed H_N is computed with MATLAB `eig` and shown in Figure 8-3. As the previous section discussed, the eigenvalue magnitudes of $H_N^p|_{\epsilon,j}$ are on the order of the perturbation for small $l = \lceil \frac{j}{p} \rceil$. Figure 8-3 confirms this behavior; for example, eigenvalue magnitudes for $N = 1000$ decrease from 0.9441 for $\epsilon = 10^{-5}$ to 0.8414 for $\epsilon = 10^{-15}$. Thus, convergence to the multiple zero eigenvalue of H_N^p is slow even for small ϵ.

Theorem 8.12 shows that the eigenvectors for nonzero eigenvalues have exponentially decaying components. To observe this phenomenon, the matrix decomposition is computed with MATLAB `eig` for $N = j = 20$ and $\epsilon = 10^{-15}$, which yields an eigenvector matrix $V_{N,1}|_{\epsilon,j}$ with minimum elements on the order of ϵ.

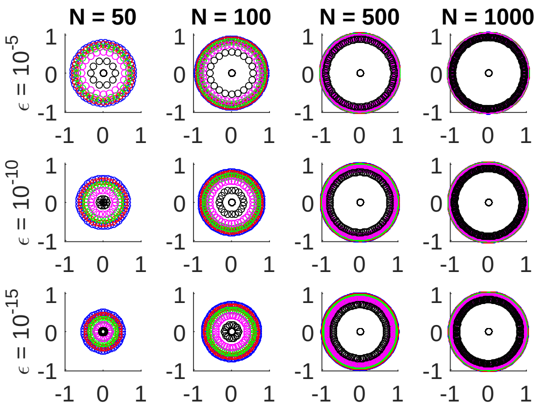


Figure 8-3: Eigenvalues of $H_N|_{\epsilon,j}$ in the complex plane for $j \in \{\frac{N}{5}, \frac{2N}{5}, \frac{3N}{5}, \frac{4N}{5}, N\}$ denoted by black, magenta, green, red, and blue, respectively. The magnitudes increase with N, ϵ, or j.

Therefore, a numerical rank computation is highly dependent on the choice of threshold. Computing the rank of $V_{N,1}|_{\epsilon,j}$ with threshold on the order of ϵ shows that the matrix has full numerical rank.

On the other hand, when the zero eigenvalue corresponds to nontrivial Jordan chains, Theorem 8.13 shows a $1/\epsilon$ factor in the proper eigenvectors that propagates through the Jordan chains. For an eigenvector matrix computed with MATLAB `jordan` for $N = 20$, $\epsilon = 10^{-15}$, and $j = 5$, each proper eigenvector v_1 for the zero eigenvalue has first element $v_{11} \sim 10^{15}$, and the kth generalized eigenvector in the chain has $v_{k1} \sim 10^{15k}$. This shows that the eigenvector instability increases in the presence of Jordan blocks of high dimension. These results show that the effect of ill-conditioning is heavily pronounced when nontrivial Jordan chains characterize the decomposition of A.

The effect of ill-conditioning on the proper eigenvectors is less pronounced when computing the eigenvector matrix with MATLAB `eig`. The numerical rank of the matrix in the previous paragraph assuming threshold ϵ is $j + 1$ as expected from Theorem 8.9 and Theorem 8.10. Although the generalized eigenvectors are missing, the algebraic multiplicity of eigenvalue zero can be deduced and used to compute the vectors through the recurrence equation (8.3).

This section has illustrated numerical behaviors of powers of the Jordan block H_N and a perturbed form. The next section presents a similar analysis for a denser directed network.

8.2 Eigendecomposition for Feed-Forward Network

This section studies the $N \times N$ matrix $A_N = \sum_{i=0}^{N-1} H_N^i$ where $H_N = J(0)$, or

$$A_N = \begin{bmatrix} 0 & 1 & 1 & \dots & 1 \\ & 0 & 1 & \ddots & \vdots \\ & & 0 & \ddots & 1 \\ & \underline{0} & & \ddots & 1 \\ & & & & 0 \end{bmatrix}. \tag{8.19}$$

Since A_N is upper triangular with diagonal entries of zero, all eigenvalues are zero.. Unlike the powers of H_N studied in Section 8.1, A_N represents a denser digraph where every node connects to its rightward neighbors as depicted in Figure 8-2a. The rightward neighbors are “leaders” in the sense that other network members are influenced by their behavior; this situation occurs in real-world graphs such as those representing blog networks [68] and diffusion networks [93, 94]. The leftmost node can be interpreted as the node of maximum influence in a diffusion network.

Section 8.2.1 provides analysis for an eigendecomposition of A_N with numerical comparisons given in Section 8.2.2. Section 8.2.3 analyzes a perturbed form of A_N. Section 8.2.4 compares the perturbed eigendecomposition to numerical computations.

8.2.1 Analysis for Eigendecomposition of A_N

This section presents the full eigendecomposition of (8.19); certain proofs are provided in Appendix B. The Jordan canonical form J_N and eigenvector matrix V_N are found for the Jordan decomposition $A_N = V_N J_N V_N^{-1}$.

Theorem 8.14. *The Jordan canonical form J_N of A_N is H_N.*

Proof. It suffices to show that A_N has a single Jordan block of size N. Perform elementary row operations to put A_N into row-reduced echelon form rref(A_N). Since rref$(A_N) = H_N$, there is one free variable in system of equations $A_N x = b$ for $x, b \in \mathbb{R}^N$. The geometric multiplicity of $\lambda = 0$ equals the number of free variables [33, 56], so there is a single Jordan block of size N. □

By Theorem 8.14, an eigenvector matrix V_N of A_N will have $N-1$ generalized eigenvectors corresponding to Jordan block J_N. Theorem 8.15 shows an eigenvector matrix V_N based on the upper-triangular Pascal matrix P_N defined as [95]

$$P_N = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & \dots & 1 \\ & 1 & 2 & 3 & 4 & \dots & N-1 \\ & & 1 & 3 & 6 & \dots & \binom{N-1}{2} \\ & & & 1 & 4 & \dots & \binom{N-1}{3} \\ & & & & \ddots & \ddots & \vdots \\ & \underline{0} & & & & 1 & N-1 \\ & & & & & & 1 \end{bmatrix}. \tag{8.20}$$

Theorem 8.15. *The inverse of the Pascal matrix* (8.20) *is an eigenvector matrix V_N of A_N.*

Theorem 8.15 proves one possible eigenvector matrix for A_N; please see Appendix B.1 for the proof. Since solving for the Jordan chains of a matrix that is not full rank introduces degrees of freedom, the generalized eigenvectors of A_N are not unique. The next theorem proves another possible eigenvector matrix V_N.

Theorem 8.16. *The matrix*

$$V_N = \begin{bmatrix} 1 & 0_{N-1}^T \\ 0_{N-1} & P_{N-1}^{-1} \end{bmatrix} \tag{8.21}$$

with inverse

$$V_N^{-1} = \begin{bmatrix} 1 & 0_{N-1}^T \\ 0_{N-1} & P_{N-1} \end{bmatrix} \tag{8.22}$$

is an eigenvector matrix for the matrix A_N of (8.19), *where P_N denotes the $N \times N$ Pascal matrix defined in* (8.20).

The behavior of the numerical eigendecomposition of A_N is discussed next.

8.2.2 Numerical Eigendecomposition of A_N

Theorems 8.14, 8.15, and 8.16 are compared to the numerical decomposition of A_N via the MATLAB `eig` and `jordan` functions. For `eig`, the eigendecompositions yield the zero eigenvalue with multiplicity N; however, the generalized eigenvectors are not computed, so the resulting eigenvector matrix V_N has numerical rank 1 corresponding to the single proper eigenvector of A_N. The Jordan form is then computed with MATLAB's `jordan` for $N \in [10, 100]$. The algebraic and geometric multiplicities of eigenvalue zero as well as the generalized eigenvectors are all found, since the Jordan canonical form matches Theorem 8.14 and the eigenvectors match (8.21). However, the eigenvector matrices are ill-conditioned; for example, V_{10} and V_{100} have condition numbers 1.0717×10^{27} and 1.5233×10^{47}, respectively. In this way, V_N may not have full numerical rank even though it is full rank in actuality.

8.2.3 Analysis for Eigendecomposition of Perturbed A_N

Perturb A_N to $A_N|_{\epsilon,j} = A_N + \epsilon A'_{N,j}$, $2 \leq j \leq N$, where $\epsilon > 0$ and matrix $A'_{N,j}$ is zero except for $[A'_{N,j}]_{j1} = 1$. When $j = N$, $A_N|_{\epsilon,j}$ corresponds to a strongly connected digraph as in Figure 8-2b, where a larger ϵ represents a greater degree of strongly connectedness in the network. For $j < N$, the digraph consists of a strongly connected subgraph of j nodes. We will see that $A_N|_{\epsilon,j}$ exhibits a multiple zero eigenvalue and distinct nonzero eigenvalues without spurious numerical computations; however, numerical instability is shown in the eigenvectors.

Eigenvalues of $A_N|_{\epsilon,j}$. Lemma 8.17 below is needed to find the eigenvalues of $A_N|_{\epsilon,j}$; the proof is provided in Appendix B.3.

Lemma 8.17. *Define $N \times N$ matrix M_N, $N \geq 2$, of form*

$$M_N = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ -\lambda & 1 & 1 & \dots & 1 \\ & -\lambda & 1 & \dots & 1 \\ & & \ddots & \ddots & \\ & & & -\lambda & 1 \end{bmatrix}, \tag{8.23}$$

where the elements below the diagonal of $-\lambda$ elements are all zero (M_N is upper Hessenberg). The determi-

nant of M_N is

$$\det(M_N) = (\lambda+1)^{N-2}. \tag{8.24}$$

The next theorem follows from Lemma 8.17 and provides the eigenvalues of $A_N|_{\epsilon,j}$, $2 \le j \le N$.

Theorem 8.18. *The characteristic polynomial of $A_N|_{\epsilon,j}$ is*

$$\varphi_N|_{\epsilon,N}(\lambda) = (-1)^N \left(\lambda^N - \epsilon(\lambda+1)^{N-2}\right). \tag{8.25}$$

for $j = N$; for $2 \le j < N$,

$$\varphi_N|_{\epsilon,j} = (-\lambda)^{N-j}\varphi_N|_{\epsilon,N}(\lambda). \tag{8.26}$$

Equivalently, substituting $j = N-k$,

$$\varphi_N|_{\epsilon,N-k} = (-\lambda)^k \varphi_{N-k}|_{\epsilon,N}(\lambda). \tag{8.27}$$

Theorem 8.18 shows that $A_N|_{\epsilon,j}$ for $j < N$ has a zero eigenvalue (zero root of (8.26)) of multiplicity $N-j$ and j distinct eigenvalues characterized by the roots of (8.25). This shows that changing the location of ϵ induces multiple (non-spurious) zero eigenvalues.

Theorem 8.18 also gives the characteristic polynomial (8.25) of $A_N|_{\epsilon,N}$, whose roots characterize the non-zero, complex eigenvalues of $A_N|_{\epsilon,j}$. The asymptotic behavior is studied to characterize convergence to the multiple zero eigenvalue of A_N. From (8.25), the eigenvalues of $A_N|_{\epsilon,N}$ satisfy

$$\lambda^N = \epsilon(\lambda+1)^{N-2}\text{, or,} \tag{8.28}$$

$$\left(\frac{\lambda}{\lambda+1}\right)^N = \frac{\epsilon}{(\lambda+1)^2}. \tag{8.29}$$

Let $\Lambda_N|_{\epsilon,N} = \{\lambda_1, \ldots, \lambda_N\}$ denote the eigenvalues satisfying (8.28) and (8.29), ordered by decreasing magnitude. The *spectral radius* $\rho_{N,\epsilon} = \max_{1\le i\le N}|\lambda_i|$ is found for for $\lambda_i \in \Lambda_N|_{\epsilon,N}$. Asymptotic behavior for λ is proven to satisfy $\operatorname{Re} z \le -\frac{1}{2}$. Lastly, convergence to zero is demonstrated for small ϵ by applying following lemma.

Lemma 8.19. *For all $z \in \mathbb{C}$,*

$$\left|\frac{z}{z+1}\right| = \frac{|z|}{|z+1|}\begin{cases} < 1 & \text{if } \operatorname{Re} z > -\frac{1}{2} \\ \ge 1 & \text{if } \operatorname{Re} z \le -\frac{1}{2}. \end{cases} \tag{8.30}$$

Proof. Note $|zw| = |z|\,|w|$ for all $z, w \in \mathbb{C}$ (see [96]). The result follows by substituting $z = \operatorname{Re} z + j\operatorname{Im} z$. $\square$

The next theorem shows the asymptotic behavior of the spectral radius $\rho_{N,\epsilon}$ (the eigenvalue of maximum magnitude).

Theorem 8.20. *For $\epsilon \in (0,1)$, the spectral radius $\rho_{N,\epsilon}$ of $A_N|_{\epsilon,N}$ has the following asymptotic properties:*

(i) $\lim_{N\to\infty} \rho_{N,\epsilon} = \infty$*;*

(ii) $\lim_{N\to 0} \rho_{N,\epsilon} = 0$*;*

(iii) Let $\epsilon = \alpha^N$, $0 < \alpha < 1$. Then

$$\lim_{N\to\infty,\alpha\to 1} \rho_{N,\epsilon} = \infty \quad \textit{and} \quad \lim_{N\to\infty,\alpha\to 0} \rho_{N,\epsilon} = 0.$$

For certain epidemic models in diffusion networks, the spectral radius is inversely proportional to the critical epidemic threshold [93]. By Theorem 8.20, $\rho_{N,\epsilon}$ increases for large N and large ϵ, so digraph $\mathcal{G}(A_N|_{\epsilon,N})$ is more robust to viral propagation as the number of nodes and the degree of strongly connectedness increase.

Theorem 8.21 shows the eigenvalue asymptotic behavior for $\operatorname{Re}\lambda > -\frac{1}{2}$.

Theorem 8.21. *Let $0 < \epsilon < 1$. Eigenvalues λ of $A_N|_{\epsilon,N}$ satisfying $\operatorname{Re}\lambda > -\frac{1}{2}$ have the following asymptotic properties:*

(i) For $\operatorname{Re}\lambda > 0$, $\lim_{N\to\infty} |\lambda| = \infty$, *and, for* $\operatorname{Re}\lambda < 0$, $\lim_{N\to\infty} \operatorname{Re}\lambda = -\frac{1}{2}^{+}$;

(ii) $\lim_{N\to 0} |\lambda| = 0$*;*

(iii) Let $\epsilon = \alpha^N$, $0 < \alpha < 1$. Then $\lim_{N\to\infty,\alpha\to 0} |\lambda| = 0$. Moreover,

$$\lim_{N\to\infty,\alpha\to 1} |\lambda| = \infty \textit{ for } \operatorname{Re}\lambda > 0, \textit{ and}$$
$$\lim_{N\to\infty,\alpha\to 1} \operatorname{Re}\lambda = -\frac{1}{2}^{+} \textit{ for } \operatorname{Re}\lambda < 0.$$

Theorem 8.21 describes the asymptotic behavior of the eigenvalues of $A_N|_{\epsilon,N}$ with the restriction $\operatorname{Re}\lambda > -\frac{1}{2}$. In particular, the eigenvalue magnitudes increase in this region for large N and large ϵ. Furthermore, the eigenvalue magnitudes approach 0 sublinearly as $N \to 0$ and as ϵ decays exponentially with N. Theorem 8.22 shows that most eigenvalues lie in the region $\operatorname{Re}\lambda > -\frac{1}{2}$ as $\epsilon \to 0$.

Theorem 8.22. *There exists $\epsilon' > 0$ such that, for all $\epsilon < \epsilon'$, the eigenvalues $\Lambda_N|_{\epsilon,N}$ of $A_N|_{\epsilon,N}$ all converge to the multiple zero eigenvalue of A_N.*

By Theorem 8.22, the eigenvalues of $A_N|_{\epsilon,N}$ converge to the multiple zero eigenvalue of A_N as $\epsilon \to 0$; on the other hand, the numerical results of Section 8.2.4 show that this convergence is slow.

By the characteristic polynomial (8.26), moving ϵ up the first column (i.e., reducing the size j of the strongly connected component in digraph $\mathcal{G}(A_N|_{\epsilon,j})$) decreases the number of non-zero eigenvalues. Thus, even for $N \to \infty$, placing ϵ close to the main diagonal, such as at entry (2,1), results in a zero eigenvalue of multiplicity $m_0 \approx N$. Choosing small enough ϵ with this placement ensures the few nonzero eigenvalues converge to zero.

The next theorem characterizes the Jordan form of $A_N|_{\epsilon,j}$, $2 \le j < N$.

Theorem 8.23. *The Jordan form $J_N|_{\epsilon,j}$ of $A_N|_{\epsilon,j}$, $2 \le j < N$, (unique up to a permutation of the Jordan blocks) is*

$$J_N|_{\epsilon,j} = \begin{bmatrix} H_{N-j} & 0_{N-j\times j} \\ 0_{j\times N-j} & \operatorname{diag}(\Lambda_j|_{\epsilon,j}) \end{bmatrix}, \tag{8.31}$$

where H_{N-j} is the $(N-j)\times(N-j)$ Jordan block for eigenvalue zero and $\operatorname{diag}(\Lambda_j|_{\epsilon,j})$ *is the diagonal matrix with entries equal to the eigenvalues $\lambda \in \Lambda_j|_{\epsilon,j}$ of $A_j|_{\epsilon,j}$.*

Eigenvectors of perturbed A_N. The eigenvectors of $A_N|_{\epsilon,N}$ are characterized here. Solve $A_N|_{\epsilon,N}v = \lambda v$ for $v \in \mathbb{C}^N$:

$$v_{N-k} = \begin{cases} \frac{\epsilon}{\lambda} v_1, & k = 0, \\ \frac{1}{\lambda}\sum_{l=N-k+1}^{N} v_l, & 1 \le k \le N-1, \end{cases} \tag{8.32}$$

or,
$$v_i = \begin{cases} \frac{\epsilon}{\lambda} v_1, & i = N, \\ \frac{1}{\lambda}\sum_{l=i+1}^{N} v_l, & 1 \le i \le N-1. \end{cases} \tag{8.33}$$

An expression is developed for the eigenvector elements v_i in Theorem 8.25. This requires the following lemma.

Lemma 8.24.
$$\sum_{l=N-k}^{N} v_l = \left(1 + \frac{1}{\lambda}\right)^k v_N. \tag{8.34}$$

Theorem 8.25. *The elements v_i, $1 \le i \le N$, of eigenvector $v = (v_1, \ldots, v_N)$ of $A_N|_{\epsilon,N}$ can be expressed as*

$$v_i = \frac{1}{\lambda}\left(1 + \frac{1}{\lambda}\right)^{N-i-1} v_N. \tag{8.35}$$

Proof. Combine (8.32) and Lemma 8.24 to get (8.35) in terms of $i = N - k$. □

Substituting $v_N = \lambda(1 + \frac{1}{\lambda})^{-(N-1)} v_1$ in (8.35) yields

$$v_i = \frac{1}{\lambda}\left(1 + \frac{1}{\lambda}\right)^{N-i-1} \cdot \lambda \left(1 + \frac{1}{\lambda}\right)^{-(N-1)} v_1 \tag{8.36}$$

$$= \left(1 + \frac{1}{\lambda}\right)^{1-i} v_1 = \left(\frac{\lambda}{\lambda+1}\right)^{i-1} v_1. \tag{8.37}$$

The next theorem shows that the elements v_i of eigenvectors v exponentially decay as index i increases for large N and small ϵ.

Theorem 8.26. *Eigenvectors $v = (v_1, \ldots, v_N)$ of $A_N|_{\epsilon,N}$ have exponentially decaying elements for small $\epsilon > 0$.*

Proof. By Theorem 8.22, there exists ϵ' such that $\operatorname{Re}\lambda > -\frac{1}{2}$ for all $\epsilon < \epsilon'$. Then, by Lemma 8.19, $\left|\frac{\lambda}{\lambda+1}\right| < 1$. for all λ satisfying (8.28). Therefore, the geometric series (8.37) converges to zero as index $i \to \infty$ – i.e., as $N \to \infty$. □

Theorem 8.26 and the corresponding effect on the numerical rank of the eigenvector matrix is discussed in more detail in Section 8.2.4. The eigenvectors of the distinct $\Lambda_N|_{\epsilon,j}$ in (8.31) are found next.

Theorem 8.27. *An eigenvector $v = (v_1, \ldots, v_N)$ corresponding to a nonzero eigenvalue of $A_N|_{\epsilon,(N-k)}$, $2 \le k < N$, has the following properties:*

(i) Elements v_i, $N - k < i < N$, satisfy

$$v_i = \frac{1}{\lambda}\sum_{j=i}^{N} v_j = \frac{1}{\lambda}\left(1 + \frac{1}{\lambda}\right)^{N-i};$$

(ii)

$$v_1 = \frac{1}{\epsilon}\left(\lambda v_{N-k} - \left(1 + \frac{1}{\lambda}\right)^{k-1} v_N\right);$$

(iii) For $2 \le i \le N - k - 1$,

$$v_i = \left(\frac{\lambda}{\lambda+1}\right)^{i-1} v_1.$$

Proof. *(i)* Solving $A_N|_{\epsilon,(N-k)} v = \lambda v$ yields

$$\begin{cases} \sum_{l=i+1}^{N} v_l = \lambda v_i, & i \ne N - k \\ \epsilon v_1 + \sum_{l=i+1}^{N} v_l = \lambda v_i, & i = N - k. \end{cases} \tag{8.38}$$

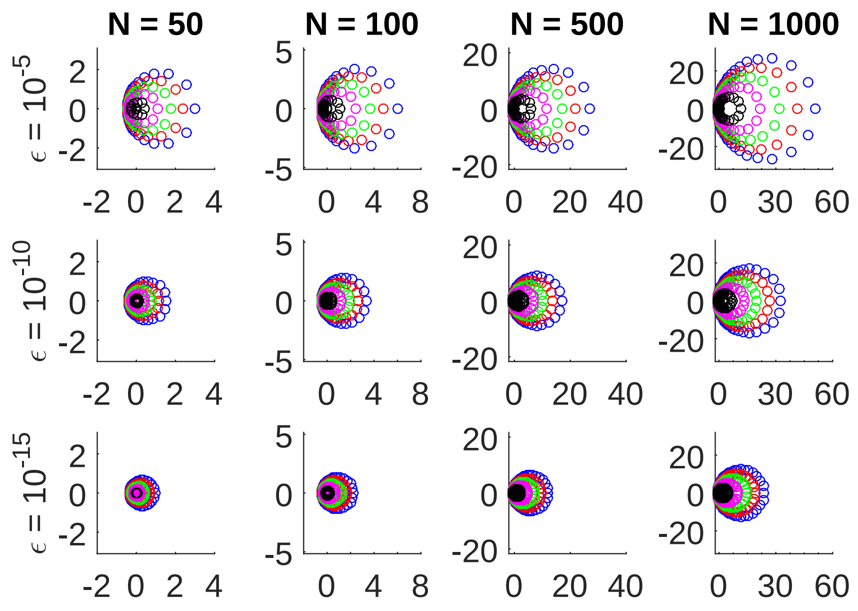


Figure 8-4: Eigenvalues of $A_N|_{\epsilon,j}$ for $j \in \{\frac{N}{5}, \frac{2N}{5}, \frac{3N}{5}, \frac{4N}{5}, N\}$, denoted by black, magenta, green, red, and blue, respectively. The spectral radii increase with N, ϵ, or j.

Rows $N-k+1$ to $N-1$ have form (8.32), so $v_{N-k+1}, \ldots, v_{N-1}$ satisfy Lemma 8.24. Substitution gives the result.

(ii) The result follows from row $N-k$ and Lemma 8.24.

(iii) Rows 1 to $N-k-1$ can be written as

$$v_i = \frac{\lambda}{\lambda+1} v_{i-1}, \qquad 2 \le i \le N-k-1. \tag{8.39}$$

Recursively writing (8.39) in terms of v_1 yields (iii). □

Theorem 8.27 shows that the first element of an eigenvector v for a nonzero eigenvalue is unstable as $\epsilon \to 0$. Since the first $N-k-1$ elements of v are multiples of v_1, these elements are also unstable. We next find the eigenvectors for $\lambda = 0$.

Theorem 8.28. *The proper eigenvector $v = (v_1, \ldots, v_N) \in \mathbb{C}^N$ corresponding to the Jordan block $H_{N-j\times N-j}$ in* (8.31) *has three non-zero elements v_1, v_{N-k} and v_{N-k+1} such that $v_1 = -\frac{1}{\epsilon} v_{N-k+1}$ and $v_{N-k} = -v_{N-k+1}$.*

Proof. The result follows by inspecting rows $N-k-1$ and $N-k$ of the system of equations (8.38) for $\lambda = 0$. □

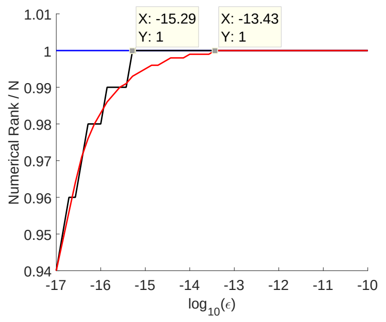


Figure 8-5: Ratio of numerical rank (with threshold 10^{-15}) to matrix dimension N versus $\log_{10}(\epsilon)$ for $N = 10$ (blue), $N = 100$ (black), and $N = 1000$ (red). The eigenvector matrix $V_{N,\epsilon}$ does not lose rank for $N = 10$. The smallest ϵ at which $V_{N,\epsilon}$ is full rank is $\epsilon = 10^{-15.29}$ for $N = 100$, and $\epsilon = 10^{-13.43}$ for $N = 1000$.

8.2.4 Numerical Eigendecomposition of Perturbed A_N

Numerical eigenvalues. The eigenvalues of $A_N|_{\epsilon,j}$ are computed with MATLAB `eig` and shown in Figure 8-4. The properties proven in the previous section are compared here to the numerical eigenvalues.

Theorem 8.18 implies that changing the location of ϵ induces multiple zero eigenvalues. This is true in Figure 8-4, which shows that $A_{50}|_{\epsilon,10}$ has 40 zero eigenvalues and 10 that are the roots of $\varphi_{50}|_{\epsilon,10}(\lambda)$, while $A_{50}|_{\epsilon,40}$ has 10 zero eigenvalues with the remaining 40 described by the roots of $\varphi_{50}|_{\epsilon,40}(\lambda)$.

The convergence properties shown in Theorem 8.20 are observed in Figure 8-4 as well. Decreasing N from 1000 to 50 causes the spectral radius to decrease from $\rho_{1000,\epsilon} = 51.01$ to $\rho_{50,\epsilon} = 3.02$ for $\epsilon = 10^{-5}$. This convergence of $\rho_{N,\epsilon}$ is sublinear with N for fixed ϵ.

In addition, the eigenvalues converge sublinearly to zero for fixed N and decreasing ϵ as seen Figure 8-4. For $N = 50$ the resulting spectral radii are $\rho_{N,10^{-5}} = 3.02$, $\rho_{N,10^{-10}} = 1.55$, and $\rho_{N,10^{-15}} = 0.95$. Even for a perturbation as small as 10^{-15} the spectral radius is significantly greater than zero. This verifies Theorem 8.20(iii); i.e., the spectral radius does not approach zero if the perturbation does not exponentially decay with N. This means that the (multiple) zero eigenvalue in the unperturbed graph is not attained, and knowledge of the unperturbed eigendecomposition does not influence the behavior of the perturbed case. This is clearly seen in terms of the GFT (4.2), since the unperturbed case has a single spectral component while the perturbed case has N spectral components.

In addition, Figure 8-4 shows that the eigenvalue magnitudes increase in the region $\operatorname{Re}\lambda > -\frac{1}{2}$ for large N and large ϵ. The magnitudes approach 0 sublinearly as $N \to 0$ and as ϵ decays exponentially with N. These observations hold with the results of Theorem 8.21. Figure 8-4 also shows that most eigenvalues lie in the region $\operatorname{Re}\lambda > -\frac{1}{2}$ as $\epsilon \to 0$, which is formalized in Theorem 8.22.

The final result for the eigenvalues of $A_N|_{\epsilon,N}$ is that they converge to the multiple zero eigenvalue of A_N as $\epsilon \to 0$ by Theorem 8.22. This behavior is observed in Figure 8-4, although this convergence is slow.

Numerical eigenvectors. The numerical rank of the eigenvector matrices are computed by the MATLAB `rank` function with a threshold of 10^{-15}. Figure 8-5 shows that the ϵ at which $V_N|_{\epsilon,N}$ is no longer full rank decreases as N increases; for example, $A_N|_{10^{-15},N}$ is full rank for $N = 100$ but loses rank for $N = 1000$. These behaviors are expected since Theorem 8.26 shows that, for large i and small ϵ, the eigenvector elements v_i become small enough that the numerical rank of the eigenvector matrix $V_N|_{\epsilon,N}$ decreases. While the eigenvalue computation of $A_N|_{\epsilon,j}$ does not result in stability issues as N and ϵ vary, Figure 8-5 shows that the eigenvector matrix $V_N|_{\epsilon,j}$ loses numerical rank as $N \to \infty$ and $\epsilon \to 0$, and that the threshold ϵ at which this occurs increases with N.

The first element of the eigenvector for $\lambda = 0$ is unstable for small ϵ by Theorem 8.28, and this instability propagates to the generalized eigenvectors via (8.3). This is evident with MATLAB `jordan`. For example, the proper eigenvector v_1 for $\lambda = 0$ has first element v_{11} on the order of 10^{15} for $N = 20$ and $\epsilon = 10^{-15}$; the kth generalized eigenvector in the Jordan chain has first element $v_{k1} \sim 10^{15k}$, $2 \leq k \leq N - j$. As a result, the eigenvector matrix $V_N|_{\epsilon,j}$ loses rank.

Despite the effects of ill-conditioning that are illustrated in this chapter, our results show that the conditioning of the proper eigenvectors as found by MATLAB `eig` is reasonable as long as the threshold for the numerical rank computation is on the order of the chosen ϵ. In other words, a numerical (non-symbolic) eigendecomposition can be used to estimate the eigenvalues and a set of proper eigenvectors for a general adjacency matrix regardless of its defective or nearly defective properties. This is important for applications in which a set of linearly independent eigenvectors is needed, such as computing the graph Fourier transform (4.2) or inexact transform (6.2).

Consequently, the methods described in Chapter 7 can still be applied. This is demonstrated in the following chapters, which detail the steps we took to apply the Agile Inexact Method (AIM) for the GFT (6.2) to New York City taxi data. Chapter 9 describes the signal extraction step for the taxi data. Chapter 10 addresses the eigendecomposition of the Manhattan road network and shows results of the AIM.

Part IV

Application to New York City Taxi Data

In Part IV, our method for applying graph signal processing to four years (2010-2013) of New York City taxi data is presented. Chapter 9 describes the signal extraction step, which requires converting the static start-/end-location framework provided by the data to an estimate of a taxi's trajectory. This specification requires significant pre-processing. The chapter details the platforms and design trade-offs that were necessary to obtain a solution that can compute the trajectories and statistics in less than a day.

Chapter 10 describes steps that are necessary to compute Jordan chains for the zero eigenvalue of the Manhattan road network. This allows us to obtain a Jordan basis with which signal projections onto eigenvectors can be obtained. This method is compared to the Agile Inexact Method (AIM) described in Chapter 6, which is much faster to compute. The eigenvector and AIM results express the same eigenvector locations, while the AIM disperses less energy among the Jordan subspaces.

Chapter 9

Signal Extraction for New York City Taxi Data

Understanding traffic flow in cities has significant ramifications, from optimizing daily commutes to facilitating evacuations in the case of emergencies and natural disasters [97, 98]. In particular, the tempo and pattern of traffic flow must be known in order to implement policies that can handle traffic congestion or unforeseen events. With the advent of smart city initiatives such as New York City's Open Data project [99], information obtained from cameras [100] and taxi GPS records [101] provide new data sources for examining the movement of city traffic, opening new research in areas such as big urban data visualization and analytics [97, 98, 102, 35].

This chapter and Chapter 10 demonstrate an application of the GFT(4.2) to four years (2010-2013) of New York City taxi data. The data set consists of 700 million taxi trips for 13,000 medallions (taxi cabs) with data fields including pick-up and drop-off locations and times [101]. Since the data does not include taxi trajectories, computation of the taxi paths is an essential pre-processing step for our application. Other studies based on tracking taxi GPS data over a road network include [2, 103, 104, 105]. In [2], GPS traces for taxi trips in China are used to discover anomalous trajectories. In [106], optimal locations of traffic monitoring units are determined using betweenness centrality measures based on origin and destination data as well as a transportation network.

Our objective here is to apply the graph signal processing framework developed in this thesis to identify sites of localized co-behavior in Manhattan which reflect traffic activity patterns that may not be obvious from the raw data. In order to do this, the methods for analysis must be efficient in both memory utilization and execution time so that the large scale of the available data can be fully exploited.

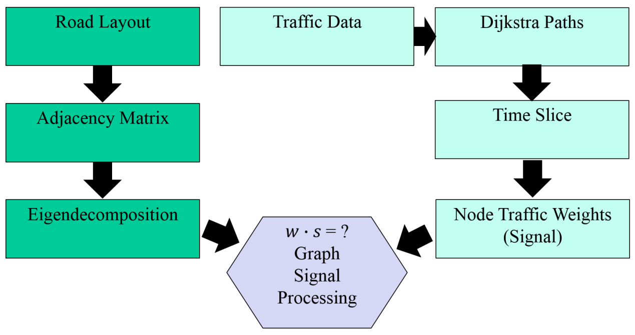


Figure 9-1: Method for applying graph signal processing to NYC taxi data.

In this chapter, the computation behind extracting key spatial and temporal behaviors using four years of New York City taxi data as traffic indicators is analyzed and discussed. In particular, we design a low-latency, memory-efficient solution to compute statistics on taxi movement in New York City. Our overall method is shown in Figure 9-1.

9.1 Data Set Descriptions

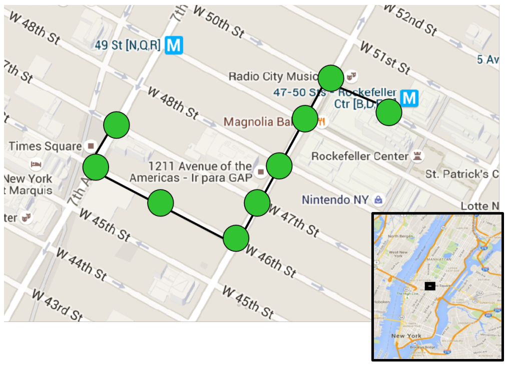
Our goal is to extract behaviors over space and time that characterize taxi movement through New York City based on four years (2010-2013) of New York City taxi data [101]. Since the path of each taxi trip is unknown, an additional processing step is required to estimate taxi trajectories before extracting statistics of interest. For example, if a taxi picks up passengers at Times Square and drops them off at the Rockefeller Center, it is desirable to have statistics that capture not just trip data at the landmarks, but also at the intermediate locations as depicted in Figure 9-2a.

Estimating tax trajectories requires overlaying the taxi data on the New York City road network. We describe the taxi data and define the network as follows.

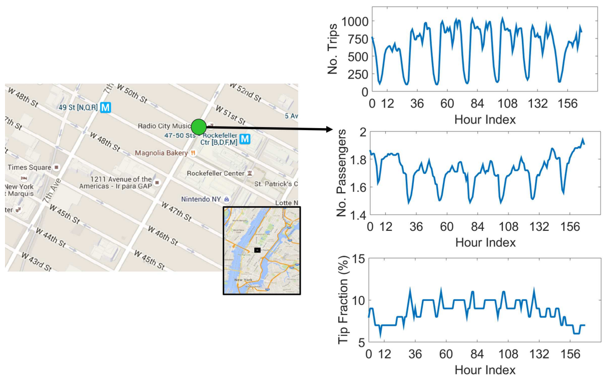
9.1.1 NYC Taxi Data

The 2010-2013 taxi data we work with consists of 700 million trips for 13,000 medallions [101]. Each trip has about 20 descriptors including pick up and drop off timestamps, latitude, and longitude, as well as the passenger count, trip duration, trip distance, fare, tip, and tax paid. The data is available as 16.7 GB of compressed CSV files.

Dynamic representation. Since the available data provides only static (start and end) information for each trip, an estimate of the taxi trajectory is needed in order to capture the taxi locations interspersed throughout the city at a given time slice. Section 9.2 explains our method for estimating these trajectories,



(a) Example Dijkstra shortest path.



(b) Example four-year average taxi statistics.

Figure 9-2: (a) Dijkstra shortest path of a single taxi trip from Times Square to the Rockefeller Center. (Map data: Google [107]). (b) Examples of four-year average statistics computed at an intersection close to Rockefeller Center: average number of trips (top right), average number of passengers (middle right), and average tip fraction (bottom right). The hour index corresponds to an hour of the week with index 0 corresponding to Sunday 12am, index 12 to Sunday 12pm, index 24 to Monday 12pm, etc.

which is based on Dijkstra's algorithm.

Once taxi paths are estimated, they are used to extract traffic behavior. Example statistics of interest include the average number of trips that pass through a given location at a given time of day, the average number of passengers of these trips, and the average tip paid. Figure 9-2b illustrates such statistics.

9.1.2 NYC and Manhattan Road Networks

The road network $G = (V, E)$ consists of a set V of $|V| = 79,234$ nodes and a set E of $|E| = 223,966$ edges which we represent as a $|V| \times |V|$ adjacency matrix A. The nodes in V represent intersections and points along a road based on geo-data from [67]. Each edge $(v_i, v_j) \in E$, $v_i, v_j \in V$, corresponds to a road segment on which traffic may flow from geo-location v_i to geo-location v_j as determined by Google Maps [107]. An edge of length $d_{ij} > 0$ is represented by a nonzero entry in the adjacency matrix so that $[A]_{ij} = d_{ij}$. A zero entry corresponds to the absence of an edge; i.e., no edges of length zero are present in the network.

The network G is *directed* since the allowed traffic directions along a road segment may be asymmetric. In addition, G is *strongly connected* since a path (trajectory) exists from any geo-location to any other geo-location.

Our analysis focuses on the Manhattan grid, which is a subgraph of G with 6,408 nodes and 14,418 edges. This subgraph is also strongly connected. The eigendecomposition in Chapter 10.1 is based on this network.

9.2 Summary of Method

Initially, computing taxi behaviors along each trajectory appears to be a single-pass problem. However, as discussed in this chapter, better performance is attained by separating the trajectory estimation and behavior extraction steps. This chapter motivates and explains the method taken to efficiently obtain paths and statistics from 700 million taxi trips on a memory-constrained computer cluster of 32 16-core/16 GB and 8-core/8 GB machines.

For our problem, the shortest path computation becomes a bottleneck. To illustrate this with an extreme example, a Python implementation required one hour to compute the Dijkstra paths for 10,000 taxi trips when running on a 16 GB RAM/16-core machine. Scaling this result suggests that 3,000 days of computing would be required to compute all 700 million trips. This chapter shows how the computation time can be reduced to a few weeks by parallelizing the computation on a 30-machine cluster with an appropriate high-throughput computing platform such as HTCondor [82, 108]. Nonetheless, this solution still involves lengthy execution time, and does not provide the preferred time-to-solution of less than one day. The solution described here has the following properties:

- It reduces computation from 3,000 days to less than a day.
- The problem is recast as a two-pass problem.
- The solution is presented for a memory-efficient, portable C implementation.
- It is parallelizable for HTCondor.

Since memory is one of our primary runtime constraints; this requires the implementation to eschew data structures in favor of simple array-based representations. For this reason, our solution is implemented in the C programming language. In addition, We chose to work with HTCondor, an open-source high throughput computing environment [82], since this platform was installed on the available cluster of machines. HTCondor works well when each job is designed to have a low memory footprint, which influences the design we present in this chapter.

Related path-planning algorithms are described in Section 9.3. The problem formulation and the engineering constraints are described in detail in Sections 9.4, 9.5, and 9.6. Appendix D provides further details about the implementation. Results and depictions of particular signals are shown in Section 9.7.

9.3 Motivation for Dijkstra's algorithm

Dijkstra's algorithm [109] is related to shortest path algorithms such as breadth-first search, the Bellman-Ford algorithm, and the Floyd-Warshall algorithm [50, 110]. Breadth-first search can be used to find a shortest path on an unweighted, directed graph. The Bellman-Ford algorithm finds shortest paths from a single source on graphs that are both weighted and directed, but it computes distances to multiple destination nodes. The Floyd-Warshall algorithm computes all-pairs shortest paths, takes $O(|V|^3)$ time, and can be optimized as in [110]; it can be modified to find a single path but is usually used for dense graphs.

This chapter focuses on Dijkstra's algorithm, which performs single-source single-destination path-finding with non-negative (distance-based) edge weights. Dijkstra's algorithm can run in $O(|E| \log |V|)$ time for sparse, strongly connected networks; more details are provided in Section 9.5. Variations on Dijkstra's algorithm include the A* algorithm, which partially computes a tree of paths to guide the search to the destination node [111, 112, 113]. In addition, methods such as pre-computing distance oracles [113], creating a hierarchical representation of the road network with contraction hierarchies [114, 115], or predicting sub-networks that contain the desired path by pre-computing cluster distances [111] have been shown to improve performance with a space trade-off.

We selected Dijkstra's algorithm [109, 50] over the alternatives mentioned above because the benefits of storing pre-computed paths to accelerate computation were outweighed by the utility of implementing an algorithm with a low memory footprint. While Dijkstra's algorithm may not reflect a true taxi trajectory, we use it as an approximation to demonstrate our method. More details on improving the path computation are in Section 9.8. Related evaluations of Dijkstra's algorithm include [116], which provides a probabilistic analysis to compare priority queue implementations, and [117], which compares serial and parallel implementations of Dijkstra's algorithm.

The following sections present the design considerations for implementing Dijkstra's algorithm in a high-throughput environment that requires a low memory footprint.

9.4 Memory-Efficient Design

This section presents the design decisions required for a memory efficient implementation on the available computer cluster, which comprised 30 machines with 16 GB/16 cores or 8 GB/8 cores. Code snippets for this section are provided in Appendix C.

Road network. Our method allows the entire road network to be available in memory at runtime to compute shortest paths. While alternative methods such as contraction hierarchies or pre-computing cluster

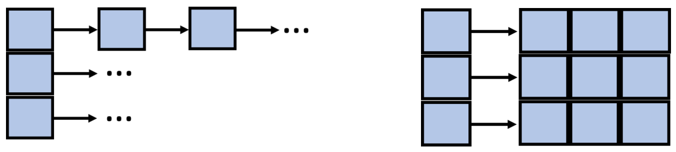

(a) Array of linked lists. (b) Array of arrays.

Figure 9-3: Adjacency list implementations.

distances to predict subnetworks that contain the shortest path do not require the entire network to be in memory for each computation [111, 114], the additional memory cost of storing pre-computed trajectories outweighs the benefit of faster path computations in terms of parallelization on HTCondor on the provided cluster.

Similarly, using data abstraction to represent the road network is not ideal since it introduces unnecessary overhead while jobs run on HTCondor require a low memory footprint. For these reasons, our implementation is developed in the C programming language and uses memory-efficient representations of the road network with C structs and arrays.

The road network is represented as an adjacency list instead of an adjacency matrix since the network is highly sparse. An adjacency matrix takes $O(|V|^2)$ memory, while an adjacency list takes $O(|V| + |E|)$ memory [50]. The adjacency matrix is sparse enough ($|E| << |V|^2$) that an adjacency list representation is ideal.

A common implementation of an adjacency list is an array of linked lists as in Figure 9-3a. However, memory must be allocated multiple times during the computation when using this approach. In addition, storing pointers between the linked list entries each takes 8 bytes assuming IEEE 754 standard double-precision floating point (64 bit). For our problem, we can compute the maximum degree of the network a priori and implement the adjacency list as a two-dimensional C array as in Figure 9-3b. In this way, pointers are not needed and memory is allocated only once.

Structures `point_t` and `edge_t` are defined to represent the full road network as shown in Figure C-1. The full representation consists of four elements: an array of `point_t` structs to store the nodes, and an array of `edge_t` structs to store the edges, an `int` array to store the out-degree of each node, and a two-dimensional array to store the adjacency list. The node and edge arrays allow for constant-time lookup of node and road segment properties, which is necessary for both shortest path computation and behavior extraction.

In order to represent the adjacency list in a memory-efficient way, the second dimension of the adjacency list array (the width of the array in Figure 9-3b, e.g.) is extended by a factor of two. This enables encoding of both the node index and the edge index for each neighbor so that Dijkstra's algorithm can traverse the

Struct name	Memory	Array name	Memory
`point_t`	19B	`nodearr`	1.5MB
`edge_t`	51B	`edgearr`	11MB
		`degarr`	320KB
		`adjlist`	6MB
		Total	**20MB**

Table 9.1: Memory required for road network representation.

road network with constant-time lookup of node and road segment properties. The number of neighbors of each node in the adjacency list is encoded in the degree array.

Assuming `double`= 8B, `int`=4B, and 3B compiler padding of C structs, a `point_t` struct requires 19B while an `edge_t` struct requires 51B. As a result, the node array takes 1.5MB, the edge array is 11MB, the degree array is 320KB, and the adjacency list is 6MB as shown in Table 9.1, for a total of 20MB. These values reflect the memory required to represent the complete network of 79K nodes and 220K edges. Although the network is too large to be stored as a local variable, it is small enough to be stored as a global variable.

Shortest path computation. For the shortest path computation, the full road network in Figure C-1 is in memory at runtime as well as an additional fixed-length array to store the current path. The pick-up and drop-off coordinates from the taxi data are also required for the path computation. Although the data takes 16.7 GB of disk space, which is small enough to be stored on a desktop hard drive, it is infeasible to load it at runtime on the available cluster machines, which have 8GB or 16GB RAM. For this reason, the data is opened as a filestream and loaded a single line at a time to compute each shortest path.

Behavior extraction. To calculate taxi movement statistics at each node over time, counters are pre-allocated to track the behavior of each node. For this purpose, a `node_t` struct is defined as in Figure C-2. Assuming `double`=8B and `int`=4B as well as 3B padding, this struct takes 10MB, and a full array takes about 800MB to store. In practice, more statistics can be added in order to track multiple behaviors simultaneously; the full array would then consume about 2 or 3GB. In this way, the behavior extraction has a much larger memory footprint than the shortest path computation, which only has the 20MB road network as a major memory requirement. This difference in memory footprints is one reason to separate the computations into pre- and post-processing jobs to submit to HTCondor.

In the next section, the algorithm choices and expected runtimes are analyzed.

9.5 Algorithm Design

This section presents the algorithm design. The first step is to estimate the taxi trajectory, defined as the path between a trip's start and end coordinates that minimizes the distance traveled. Statistics along each trajectory are then computed.

Shortest paths. The min-priority queue implementation of Dijkstra's algorithm can be implemented in $O(|V|^2 + |E|) = O(|V|^2)$ time [50]. Figure C-3 shows the inner loop of the algorithm. The runtime depends on the implementation of the min-priority queue and has been shown to be faster with binary heaps or Fibonacci heaps [50, 116]. Since the road network is sparse, binary heap implementations can run in $O(|E| \log |V|)$ time on a strongly connected graph, assuming $|E| = o(|V|^2 / \log(|V|))$ [50]. A Fibonacci heap implementation runs in $O(|E| + |V| \log |V|)$ time [50]. Our solution implements binary heaps since they have been shown to perform better in practice [116, 118].

An analysis of the shortest path algorithm performance is calculated here in terms of floating point operations per second (FLOPS). Since the NYC road network is sparse, Dijkstra's algorithm runs in $O(|E| \log |V|)$ time, which requires about $300\text{K} \cdot \log(79\text{K})$ or 1.5M floating-point operations. For a single core of an Intel Core i5-6500T processor using x87 scalar code (2 flops per cycle at 2.5 GHz, or 5 GFLOPS), the corresponding runtime is about 300μs. Running 700 million computations (for all 700 million trips in our data set) requires about 10^{15} floating point operations for a runtime of about 57 hours or 2.4 days. Assuming 20% of peak performance (1 GFLOPS), the 700 million computations would run in about 290 hours, or 12 days on a single CPU.

Behavior extraction. On the other hand, extracting statistics from a taxi trajectory is approximately linear as $O(|V|)$ since the computation is, to first order, a constant-time operation at each node of the shortest path. Scaling to the 700 million trips, the theoretical runtime on the same Core i5-6500T 2.5 GHz processor using x87 scalar code is about 3 hours at peak performance, or 15 hours assuming 20% peak performance.

Compared to the 12 days needed to run the Dijkstra algorithm, the computation time of the statistics is relatively short. In this way, splitting the computation into a pre-processing step consisting of the Dijkstra computations and a post-processing step for the statistics computations is appropriate.

To summarize, the design decisions for the path computation and signal extraction algorithms were presented and analyzed. Reasons for separating the signal extraction into pre- and post-processing steps were presented. Pre-processing requires loading the 20 MB road network and computing Dijkstra's algorithm with expected runtime of 12 days for 700 million computations on a single CPU. While post-processing requires a 2 or 3 GB struct array in addition to the road network, it can complete in 15 hours on a single CPU.

The next section describes how the high-throughput nature of the problem was leveraged to shorten the 12-day shortest path computation and 15-hour behavior extraction.

9.6 High-Throughput Computing

The signal extraction problem fits a high throughput paradigm since small pieces of code need to be run for millions of iterations. As seen in Section 9.5, the Dijkstra algorithm for a single taxi path can be computed in 300μs with reasonable memory resources. As a result, these computations can be parallelized on HTCondor.

HTCondor is open-source and provides a high throughput computing environment [82]. A user submits a series of jobs to HTCondor, which waits until a machine on the dedicated cluster is idle to start a job. The available cluster has 32 machines that are either 16-core, 16 GB RAM or 8-core, 8 GB RAM. Assuming 8-core machines with the same Core i5 processors in Section 9.5, the 57 CPU-hour estimate can be reduced to about 0.2 wall clock hours; at 20% peak performance, an HTCondor implementation is expected to reduce the runtime from 290 CPU-hours (12 days) to 1 or 2 wall clock hours. Since these projections do not account for overhead such as the time to load the road network into RAM, high-throughput computing is even more essential for reducing the time to solution. This analysis shows the impact of exploiting the high-throughput nature of the problem.

The design decisions described in this section are crucial to attain a workable solution because of the large scale of the problem. Section 9.7 describes the achieved speedup and illustrates some generated signals.

9.7 Signal Extraction Results

This section compares our design over several frameworks in terms of relative speedup. Results for the shortest path computations and the behavior extraction step are presented separately.

Pre-processing. Table 9.2 and Figure 9-4 summarize the speedup results for the pre-processing step. The initial run of this algorithm was implemented in Python on a 64-bit machine with an Intel Core i5-4300U CPU at 2.50 GHz. The time to solution for the first 10,000 shortest paths was about one hour. Each month contains about 15 million trips, which would take about 1,500 hours to compute. There are 48 months in the entire data set, so the total number of trips to compute is on the order of $48 \cdot 1,500 = 72,000$ hours, or 3,000 days. The current implementation in C returns shortest paths for 500,000 trips in 1 hour, which corresponds to 30 hours for 1 month, or $30 \cdot 48 = 1440$ hours or 60 days to get shortest paths for the entire data set, which is 50x speedup compared to the Python implementation.

Using HTCondor to submit simultaneous jobs to a cluster of 32 machines that are either 16-core, 16 GB RAM or 8-core, 8 GB RAM, shortest paths for the 700 million rides were computed in 6.5 hours, which is a speedup

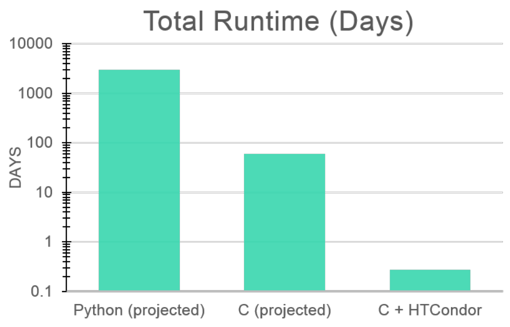


Figure 9-4: Dijkstra speedup results.

Platform	No. trips per hour	Total runtime (in hours)	Speedup vs. Python	Speedup vs. C
Python	10,000	72,000		
C	500,000	1,440		50x
C + HTCondor	500,000 (1 job)	6.5 (1,584 jobs)	11,000x	220x

Table 9.2: Speedup across platforms for shortest path computations (pre-processing).

of about 220 compared to non-simultaneous performance and a speedup of about 11,000 compared to the original Python implementation.

Post-processing. The `node_t` struct in Figure C-2 was designed to compute taxi counts, trip counts, tip, passenger counts, return frequencies and return times for the paths as well as pickups and dropoffs. The time resolution was chosen to be each hour of the week and the associated struct took about 2 GB in memory.

The time to solution for behavior extraction with the C implementation was one hour for 60 million trips (1 year of taxi data), or about four hours for the 700 million trips. Each year's data was processed as a separate job on HTCondor so that the total statistics were computed in 1 hour (4x speedup). Averaging required an additional 4 hours, for a total runtime of 5 hours for post-processing. Figure 9-5 shows some example results.

In total, pre- and post-processing together have a runtime of 11.5 hours, or about half a day. Compared to the single-machine Python implementation with projected runtime of 3,000 days, we achieve a workable solution that can be run and rerun to study and evaluate taxi statistics on New York City.

Using the design principles discussed here, statistics on the New York City road network that characterize taxi movement can be extracted. Weekly averages are computed and stored in 168-element arrays such that index 0 corresponds to Sunday 12am-1am, index 1 corresponds to Sunday 1am-2am, index 24 corresponds to Monday 12am-1am, etc. For instance, the average number of trips and the average number of passengers per trip are two example statistics shown in Figure 9-5. Figure 9-5a shows that the average number of taxis passing through Manhattan is much higher 8am–9am on Mondays compared to 8am–9am on Sundays, reflecting the expected weekday rush hour congestion. The number of passengers can also be studied; for example, Figure 9-5b shows that taxis that pick up more passengers tend to have trajectories around the perimeter via Hudson River Parkway or FDR Drive instead of through the heart of Manhattan. The average number of passengers per trip also decreases on Mondays for trips that travel through or near the Brooklyn Bridge.

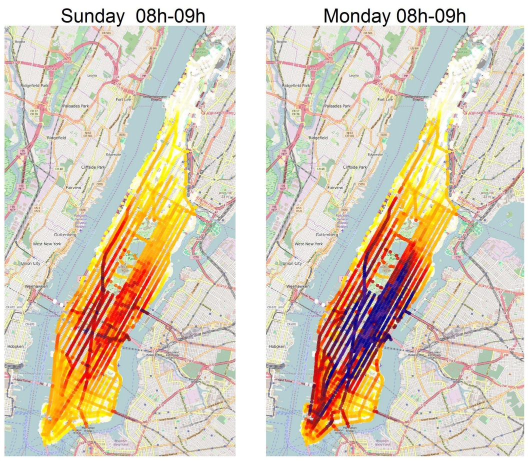
9.8 Note on Choice of Weights for Dijkstra's Algorithm

The signals and subsequent analysis depend on the route-finding algorithm that was implemented, which is Dijkstra's algorithm with distances between geolocations on the road network as the weights (see Chapter 9.5). This is a fast approximation of the true taxi trajectories that is feasible on the provided cluster of machines. Since historical trajectory information is not provided, such information is not incorporated into the Dijkstra weights.

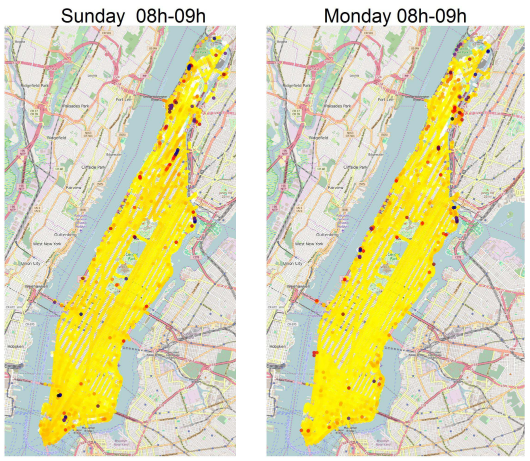
Our particular choice of Dijkstra's weights explains certain features in Figure 9-5, such as the high number of trips on Broadway in Figure 9-5a. A possible modification could be a weighting scheme that disperses the concentration of taxi trips. For example, the weight $w_{ij} = \alpha d_{ij} + (1-\alpha) n_{ij}$ can be defined for each edge $(v_i, v_j) \in E$, where $\alpha \in (0,1)$, d_{ij} is the (normalized) Euclidean distance between v_i and v_j, n_{ij} is the (normalized) average number of trips from v_i to v_j. Since n_{ij} is an output of a single computation, multiple iterations can be run such that the output $n_{ij}^{(p)}$ of iteration p is the input to the weights of iteration $p+1$. The iterations continue until the process converges (e.g., $\left\| n_{ij}^{(p+1)} - n_{ij}^{(p)} \right\| < \epsilon$ for all i,j and threshold $\epsilon > 0$). The applicability of such an iterative method provides additional motivation for the development of fast, efficient solutions to compute statistics.

In addition, different Dijkstra cost functions must be implemented to address different questions. For example, historical congestion information for each NYC road, capacity of each road measured in terms of the number of lanes, and the expected number of pedestrians that stop traffic at each intersection at a particular time of day would provide invaluable information as to the optimal trajectory for a taxi to take through the city. In addition, if the problem constraints change, such as if optimality is defined as the route that minimizes air pollution as addressed in [119], the Dijkstra cost function should change to address the problem.

The preceding discussion illustrates how incorporating additional constraints and data can improve the accuracy of estimated trajectories. Use of such data, however, will also increase the computational scale of the problem. In this case as well, the design considerations addressed in this section are crucial for attaining



(a) Average number of trips.



(b) Average number of passengers per trip.

Figure 9-5: Four-year average June-August statistics on Manhattan, NYC, for Sundays and Mondays 8am to 9am. Colors denote $\log_{10}$ bins of (a) the average number of trips (699 log bins; white to yellow: 0–12, orange: 12–92, red: 92–320, blue: 320–280, purple: 280–880, black: 880–1,430) and (b) the average number of passengers per trip (499 log bins; white to yellow: 0–1.6, orange: 1.6–2.1, red: 2.1–3.4, blue:3.4–4.2, purple:4.2–4.9, black: 4.9–6). The plots were generated with ggmap [120] and OpenStreetMap [121].

a feasible solution on memory-bound systems.

This chapter explains the design of a signal extraction implementation for the NYC taxi data. This implementation is memory-efficient so that the taxi path computations can be parallelized with HTCondor, and the computation completes in less than a day. The resulting signals are analyzed with the Agile Inexact Method for the graph Fourier transform in the next chapter. As explained in Chapter 7, the next steps for applying the graph Fourier transform are the eigendecomposition step and the transform computation. Chapter 10 discusses the method in detail.

Chapter 10

New York City Taxi Data: Eigendecomposition and Graph Fourier Transform

In order to apply the graph Fourier transform to the statistics computed in the previous chapter, it remains to compute the eigendecomposition of the Manhattan network as well as the signal projections. This chapter expands on the issues described in Chapters 6.5 and 8 and presents details for computing the eigenvector matrix V for the non-diagonalizable adjacency matrix A of the Manhattan road network.

The Agile Inexact Method (AIM) for the graph Fourier transform developed in Chapter 6 is then applied to the statistics of Chapter 9. Our method provides a fine-grained analysis of city traffic that should be useful for urban planners, such as in improving emergency vehicle routing.

Sections 10.1– 10.4 describe our method to find the eigendecomposition of the Manhattan road network. Section 10.5 demonstrates the AIM for computing the graph Fourier transform.

10.1 Eigendecomposition of Manhattan Road Network

The Manhattan road network described in Section 9.1 provides the adjacency matrix $A \in \mathbb{R}^{6408 \times 6408}$ for which we compute the eigendecomposition. For this particular adjacency matrix, the eigenvector matrix $V_{\text{obs}} \in \mathbb{C}^{6408 \times 6408}$ generated by a standard eigenvalue solver such as MATLAB or LAPACK is not full rank, where the rank can be computed with either the singular value decomposition or the QR decomposition; i.e., A is not diagonalizable.

In order to apply the GFT (4.2) or the original projections onto each eigenvector as in [6], Jordan chain computations are required. These computations are costly as described in Section 6.6 and also potentially numerically unstable. In contrast, the AIM (6.2) does not require this step.

Since it is instructive to compare the GFT with the AIM results, it is necessary to compute the Jordan chains here. For this reason, the following sections detail our method for computing the Jordan chains for the Manhattan road network.

10.2 Eigenvalue Verification and Maximum Jordan Chain Length

Since the dimension of the null space of A (the geometric multiplicity of the zero eigenvalue) is 446, and there are 699 eigenvalues of magnitude close to zero (as seen in Figure 6-2a), $699 - 446 = 253$ generalized eigenvectors must be determined to complete the Fourier basis.

As discussed in Section 6.5.1, there is ambiguity in determining whether the 699 eigenvalues of small magnitude represent true zero eigenvalues. Consequently, accurately identifying eigenvalues of high algebraic multiplicity can be a hard problem – see, for example, [70, 71, 72, 73, 76, 34]. Furthermore, these eigenvalues of small magnitude form constellations with centers close to zero, as shown in Figure 6-2b. References [70] and [71] show that the presence of such constellations may indicate the presence of an eigenvalue of high algebraic multiplicity; [70] states that an approximation of this eigenvalue may be the center of the cluster of numerical eigenvalues, while [71] presents an iterative algorithm to approximate a "pseudoeigenvalue." It is unclear from pure inspection[1] whether the observed constellations are a result of the multiplicity of a zero eigenvalue or are the actual eigenvalues of A. For these reasons, it is necessary to verify the existence of a numerical zero eigenvalue before computing the Jordan chains. Our method is explained below; see also [35].

To verify a numerical zero eigenvalue, a result from [73] is applied, which states the following:

Definition 10.1 ([73]). *Consider a matrix $A \in \mathbb{C}^{N\times N}$ with singular values $\sigma_1(A) > \cdots > \sigma_N(A)$ that is scaled so that $\sigma_1(A) = 1$. Let $m_k = \left|\mathrm{Ker}(A^k)\right|$ denote the dimension of the null space of A^k . In addition, let α and δ be positive constants; δ is usually on the order of machine precision and α is significantly greater than δ. Then 0 is a numerically multiple eigenvalue with respect to α and δ if*

$$\sigma_{N-m_k}\left(A^k\right) > \alpha > \delta > \sigma_{N-m_k+1}\left(A^k\right), \tag{10.1}$$

for $k = 1, 2, \ldots, h$, where h is the maximum Jordan chain length for the zero eigenvalue.

[1]Considering double-precision floating point (64 bit) and that the number of operations to compute the eigenvalues of an $N \times N$ matrix is $O(N^3)$, the expected precision is on the order of 10^{-6} or 10^{-7}. Numerous eigenvalues in Figures 6-2a and 6-2b demonstrate this order of magnitude.

k	m_k	$N - m_k$	$\sigma_{N-m_k}\left(A^k\right)$	$\sigma_{N-m_k+1}\left(A^k\right)$
1	446	5962	1.9270×10^{-3}	1.2336×10^{-15}
2	596	5812	2.1765×10^{-6}	6.9633×10^{-16}
3	654	5754	1.4013×10^{-8}	3.4250×10^{-16}
4	678	5730	1.1853×10^{-10}	3.1801×10^{-16}
5	692	5716	2.0163×10^{-11}	8.4063×10^{-14}
6	700	5708	9.6533×10^{-11}	8.2681×10^{-11}

Table 10.1: Singular values to validate existence of a numerical zero.

Since the constants α and δ have different orders of magnitude, Equation (10.1) implies that singular value $\sigma_{N-m_k}(A^k)$ is significantly greater than $\sigma_{N-m_k+1}(A^k)$.

Definition 10.1 serves two purposes for our application. First, it verifies the existence of a numerical zero eigenvalue. It also implies that a value of k at which Equation (10.1) fails cannot be the maximum Jordan chain length of the zero eigenvalue. This suggests the following method to find the maximum Jordan chain length h: increment the value of k starting from $k = 1$, and let $k = k'$ be the first value of k such that Equation (10.1) fails. Then the maximum Jordan chain length for eigenvalue zero is $h = k' - 1$.

Table 10.1 is obtained by applying Definition 10.1 to the adjacency matrix A of the Manhattan road network. The columns of the table correspond to the power k of A, the dimension m_k of the null space of A^k, the index $N - m_k$ of the first singular value of A^k to examine, and the values of the singular values at indices $N - m_k$ and $N - m_k + 1$. The results in the table are reasonable since the computational machine precision was on the order of 10^{-16}, and the first four rows of the table display singular values for which δ is on the order of 10^{-15} or 10^{-16} and the constant α is larger by 6 to 12 orders of magnitude. Therefore, the inequality (10.1) holds for the first four rows of Table 10.1. The inequality begins to fail at $k = 5$; thus, the expected maximum numerical Jordan chain length is no more than 3 or 4.

10.3 Jordan Chain Computation

To find the eigenvectors for eigenvalue zero, the null space $\mathrm{Ker}(A)$ of A is first computed to find the corresponding eigenvectors. Each of these eigenvectors corresponds to a Jordan block in the Jordan decomposition of A, and the Jordan chains with maximum length h (as determined by Definition 10.1 from [73]) are computed by the recurrence equation [33]

$$Av_k = \lambda v_k + v_{k-1} = v_{k-1}, \tag{10.2}$$

where $k = 1, \ldots, h$ and $v_0 \in \mathrm{Ker}(A)$. If the number of linearly independent proper and generalized eigenvectors equals N, we are done; otherwise, the Fourier basis must be extended.

Numerical instability. According to [122], finding the Jordan chains starting from the proper eigenvectors is numerically unstable and may lead to solving inconsistent systems of equations. For this reason, the procedure shown in Section 6.6 is not implemented. Instead, the Jordan chains are generated from the vectors in the null space of A^h, where h is the maximum Jordan chain length; these computations are based on the SVD or the QR decomposition, and so are more stable. Then the Jordan chain can be constructed by direct application of the recurrence equation (10.2). The method we use here is related to that of [85], which demonstrates forward stability since it is based on stable null space computations.

Computation details. A single pass of the Jordan chain algorithm requires about a week to run on a 30-machine cluster of 16 GB RAM/16-core and 8 GB RAM/8-core machines; however, the algorithm does not return a complete set of 253 generalized eigenvectors. To maximize the number of recovered generalized eigenvectors, successive computations were run with different combinations of null space vectors as starting points for the Jordan chains. After three months of testing, the best run yielded 250 of the 253 missing Jordan chain vectors; the remaining eigenvectors were computed as in (6.14). The maximum Jordan chain length is two, which is consistent with Definition 10.1 and Table 10.1. As a result, the constructed eigenvector matrix captures a large part of the adjacency matrix structure. The next section describes our inexact alternative to full computation of the Jordan chains, which provides a significant speedup in the determination of a useful basis set.

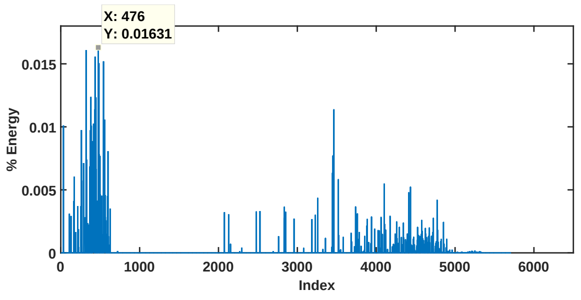
10.4 Generalized Eigenspace Computation

In order to compute the generalized eigenspace for the zero eigenvalue, the known eigenvectors are determined as in Section 10.3. Then the eigenvector matrix is computed as (6.14). This is a five minute computation on a 64-bit machine with 16 cores and 16 GB RAM.

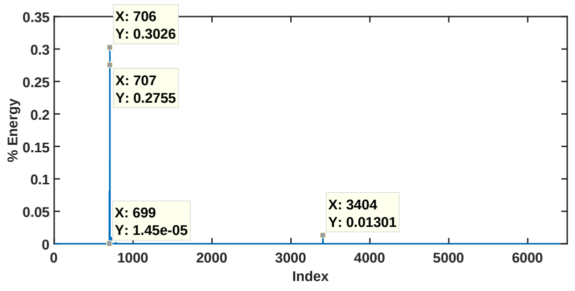
10.5 Demonstration of the Agile Inexact Method

In this section, the original graph Fourier transform, defined as eigenvector projections, is compared to the Agile Inexact Method (AIM) (6.14) for the signals computed in Chapter 9. The four-year average of trips that pass through Manhattan from June through August on Fridays from 9pm to 10pm is analyzed.

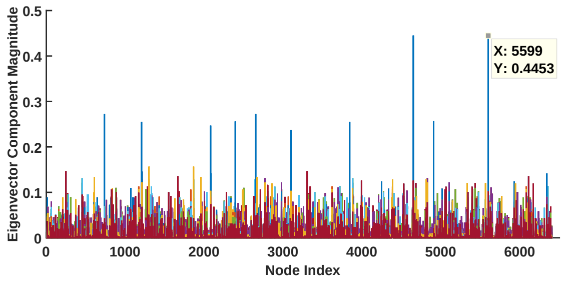
Figures 10-1a and 10-1b show the percentage of energy that is contained in the signal projections onto eigenvectors and onto generalized eigenspaces, respectively, using our previously defined energy metric



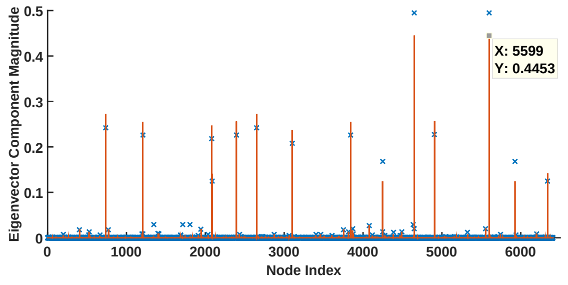
(a) Eigenvector spectral components.



(b) Generalized eigenspace spectral components.



(c) Eigenvector component magnitudes.

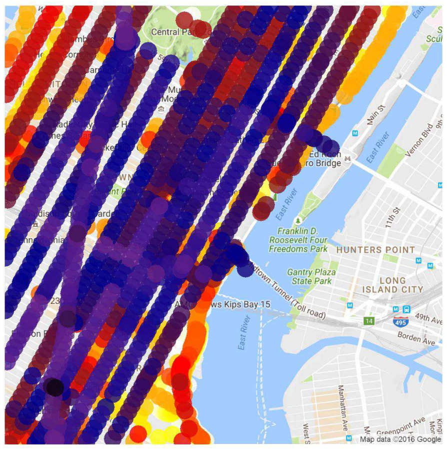


(d) Eigenvector component magnitudes.

Figure 10-1: (a) Percentage of energy captured by signal projections onto eigenvectors, including generalized eigenvectors. The horizontal axis corresponds to the eigenvector of the Manhattan road matrix based on the ascending order of the corresponding eigenvalue magnitudes. (b) Percentage of energy captured by signal projections onto generalized eigenspaces. The horizontal axis corresponds to the generalized eigenspace of the Manhattan road matrix based on the ascending order of corresponding eigenvalue magnitudes. The first projection energy onto the generalized eigenspace of $\lambda = 0$ is shown at index 699, corresponding to less than 1% of the total signal energy. The preceding indices are set to zero to match the indices of (a). The following indices correspond to the same eigenvectors as in (a). The most energy is captured by projections onto the eigenvector at index 706 (30% of the energy) and at index 707 (28% of the energy). (c) Eigenvectors of (a) that capture 60% of the signal energy. The horizontal axis corresponds to a node (geo-coordinate) in the Manhattan road network. The vertical axis corresponds to the magnitude of the corresponding eigenvector component. There are 84 eigenvectors corresponding to about 60% of the signal energy, each shown in a different color. (d) The two eigenvectors in (b) that correspond to 58% of the total signal energy, one depicted as red lines and the other as blue x's. The axes are identical to those in (c). The eigenvector components of highest magnitude correspond to the same nodes, or locations in New York City. Furthermore, the locations with highest eigenvector expression match those in (c).

(Section 6.2). The maximum energy contained in any of the eigenvector projections is 1.6% as shown in the data point in Figure 10-1a. The observed energy dispersal in the eigenvector projections corresponding to $\lambda = 0$ is a result of oblique projections. The computed Jordan chain vectors are nearly parallel to the proper eigenvectors (angles less than 1 degree), so the signal projection onto a proper or generalized eigenvector v of $\lambda = 0$ parallel to $\mathbb{C}^N \backslash \text{span}(v)$ has an augmented magnitude compared to an orthogonal projection. The reader is referred to [123] for more on oblique projectors.

In contrast, projecting onto the generalized eigenspaces with the AIM method (6.2) concentrates energy into two eigenspaces. The data points in Figure 10-1b at indices 706 and 707 contain 30% and 28% of the signal energy, respectively. On the other hand, the energy concentrated on the generalized eigenspace for


(a) Original signal.

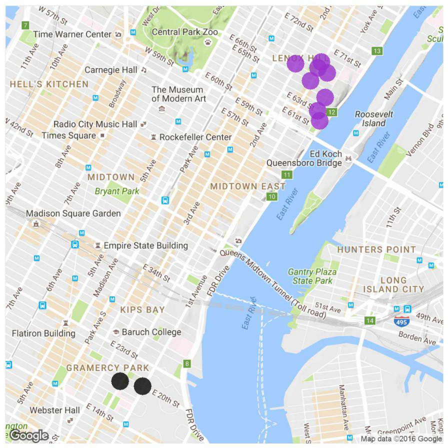

(b) Maximum eigenvector components.

Figure 10-2: (a) Original signal, representing the June-August four-year average number of trips through each node on the Manhattan road network. Colors denote 699 log bins (white: 0–20, beige to yellow: 20–230, orange: 230–400, red: 400–550, blue: 550–610, purple to black: 610–2,700). (b) Highly expressed eigenvector in Figures 10-1c and 10-1d, where the maximum components and second maximum components are represented by black and purple dots, respectively. These clusters correspond to locations through which high volumes of taxi trips flow.

eigenvalue zero is less than 1%, as shown by the data point at index 699 in Figure 10-1b. Thus, the AIM method decreases energy dispersal over the spectral components and reduces the effect of oblique projections shown in Figure 10-1a.

For both the eigenvector projection method and the AIM method, the spectral components are arranged by the energies they contain, in decreasing order, and those that contain 60% of the signal energy in the projections have their component magnitudes plotted in Figures 10-1c and 10-1d. There are 84 such eigenvectors for the projections onto eigenvectors, compared to only two eigenvectors (eigenspaces) that contain the most energy in the generalized eigenspace case; this illustrates that the eigenvector projections disperse more energy than the generalized eigenspace projections. Figures 10-1c and 10-1d show that the highly expressed eigenvector components are located at the same nodes for both methods. This demonstrates that the AIM can provide similar results to the original formulation, with a significant acceleration of execution time and less energy dispersal over the spectral components.

The highly expressed components of eigenvectors 706 and 707 in Figure 10-1d are plotted on the Manhattan grid in Figure 10-2b. There are two clusters on the east side of Manhattan, one in black in Gramercy Park and one in purple by Lenox Hill. These locations represent sites of significant behavior that suggest concentrations of taxi activity. The corresponding graph signal, shown in Figure 10-2a, confirms this. Since this signal represents the average number of taxi trips on Fridays from 9pm to 10pm, one possible explanation for this behavior is the proximity of the expressed locations to inexpensive restaurants that are popular among taxi passengers. Visual comparison of Figure 10-2a and 10-2b highlights the utility of our method in finding fine-grained behaviors that are otherwise non-obvious from the raw signal.

This chapter provides a detailed explanation of a Jordan chain computation for the Manhattan road network. The expensive nature of this computation is a significant drawback for many real-world applications. On the other hand, the inexact computation reveals the same eigenvector expression with computation time on the order of minutes. In addition, the AIM reduces energy dispersal among the spectral components corresponding to low-magnitude eigenvalues.

Chapter 11

Conclusion

This thesis develops new results in two main areas: in graph signal processing over defective adjacency matrices, and in formulating an approach for applying graph signal processing to signals derived from massive data sets over large, directed, and sparse real-world networks. We have defined and explored the properties of a spectral projector-based graph Fourier transform for which Jordan equivalence is shown over certain classes of graphs, and derived an inexact transform motivated by Jordan equivalence. The necessary steps for applying the method to real-world data are described and used to demonstrate our method using four years of NYC taxi data.

Section 11.1 summarizes the material covered in each chapter of this thesis. Section 11.2 highlights the contributions and explains their significance. Section 11.3 describes future work.

11.1 Summary

This section summarizes the key concepts that are covered in the four parts of the thesis.

Part I: Motivation and background. Chapter 2 presents a motivating example for the importance of directed graphs. The chapter describes an anomaly detection problem on a large mobile subscriber network that we solve by exploring directed subgraph structures to build a novel feature set that improves detector accuracy. Chapter 3 provides the necessary background for graph signal processing over graph adjacency matrices, which is the foundation for the main contributions of the thesis.

Part II: Spectral projector-based graph signal processing. Chapter 4 defines and proves properties of a graph Fourier transform based on projections onto the Jordan subspaces of the graph adjacency matrix. Properties such as uniqueness with respect to a choice of proper eigenvectors and a generalized Parseval's equality are presented. In addition, we find and formulate the concept of equivalence classes of graphs over which the GFT is equivalent. These equivalence classes include isomorphic graphs.

Chapter 5 discusses Jordan equivalence classes of graphs, which are characterized by the same Jordan normal forms and the same sets of Jordan subspaces. We show how the generalized Parseval's identity simplifies over certain Jordan equivalence classes. In addition, a ranking of spectral components in terms of total variation over a class of graphs is discussed.

In Chapter 6, the Agile Inexact Method (AIM) for computing the graph Fourier transform is developed for which the generalized eigenspaces are the spectral components. The generalized Parseval's identity for the inexact method is compared to that of the Jordan subspace projector method. In addition, we show that the total variation-based ordering of spectral components does not change under the inexact computation. The AIM simplifies the eigendecomposition step in real-world, sparse networks, and is associated with a fidelity-runtime trade-off.

Part III: Applying the GFT. Chapter 7 presents the general method of applying the GFT or AIM on real-world data sets. Specific considerations for the signal extraction, eigendecomposition, and GFT computation steps are discussed.

In addition, numerical instability in the eigendecompositions of particular defective and nearly defective graphs is demonstrated in Chapter 8. These examples show that the full Jordan decomposition, which includes determining the Jordan normal form, should not be computed in general; instead, the eigenvectors and eigenvalues should be computed first, followed by Jordan chain computations if necessary.

Part IV: New York City taxi application. Chapters 9 and 10 apply the GFT as described in Part III to New York City taxi trip data. Chapter 9 shows the details of extracting signals from four years of New York City taxi trip data using Dijkstra's algorithm to estimate taxi trajectories. Memory considerations and algorithm efficiency are discussed and analyzed. Chapter 10 discusses the eigendecomposition step, which involves verifying the presence of a numerical zero eigenvalue and computing Jordan chains. Finally, the AIM for the graph Fourier transform is applied and is shown to have less energy dispersal among the spectral components compared to eigenvector projections.

11.2 Contributions and Significance

The following contributions are made in this thesis:

- We define a spectral projector-based graph Fourier transform. Unlike methods based on projections onto generalized eigenvectors, which are non-unique in terms of the Jordan chain computation, our formulation is unique and unambiguous since it is not dependent on a choice of Jordan basis. Furthermore, our formulation decreases the signal energy dispersal among generalized eigenvectors that results from oblique projections.

- We define equivalence classes that enable computation of the graph Fourier transform over simpler graph topologies. The concept of isomorphic equivalence classes shows that the graph Fourier transform is invariant to re-orderings of node labels. In this way, methods that accelerate matrix-vector computations by exploiting sparse matrix structures can be applied. Moreover, the concept of Jordan equivalence classes demonstrates that the graph Fourier transform is equal over nonidentical defective matrices that have the same eigenvalues and eigenvalue multiplicities in addition to having Jordan chains that span the same Jordan subspaces. It is shown that knowledge of underlying graph structures can allow computations of graph Fourier transforms over graphs of simpler topologies. Arbitary graph structures, however, may not correspond to a Jordan equivalence class that contains a simple topology or algebraic structure to utilize. This leads us to define an inexact method.

- We present the Agile Inexact Method (AIM) for computing the graph Fourier transform. This method simplifies the choice of projection subspaces by projecting onto generalized eigenspaces instead of Jordan subspaces. Corresponding $\mathscr{G}$-equivalence classes for which the AIM yields the same transform are defined. For large, sparse, and directed real-world networks that are characterized by a single nontrivial generalized eigenspace, the $\mathscr{G}$-equivalence classes can dramatically reduce the computation time of the AIM's eigendecomposition. Since the resulting Fourier basis is an approximation of the original graph structure, higher fidelity to the original graph can be achieved by tuning the eigendecomposition with the trade-off between the number of computed Jordan vectors and the execution time.

- We formalize the applicability of our graph signal processing framework to real-world systems by illustrating an application to New York City taxi trip data over the Manhattan road network. The complete method consists of three steps: signal extraction, eigendecomposition, and computing the graph Fourier transform via signal projections. For the signal extraction step, we optimize Dijkstra path computations on a 30-machine cluster with 16-core/16 GB and 8-core/8 GB RAM machines and reduce computation time from 3,000 days to less than a day. For the eigendecomposition step, we illustrate the benefits of the AIM in attaining a fast approximation of the Fourier basis. Our results show that the inexact method disperses less energy over eigenvectors corresponding to zero eigenvalues while exciting the same highly expressed eigenvectors based on eigenvector projections.

11.3 Future Work

We briefly discuss future work in the following areas: leveraging and understanding graph structures when equivalence classes are applied, using our method in a supervised learning problem, and understanding correlation versus causation in the results.

Graph topologies of equivalence classes. Each of the equivalence classes discussed in this thesis (isomorphic, Jordan, and $\mathscr{G}$) can be studied further. Isomorphic classes, for example, can be applied to transform sparse adjacency matrices to nearly diagonal or low-bandwidth matrices with methods such as the Cuthill-McKee algorithm [124]. The signal projections and eigendecompositions for such matrices can leverage sparse matrix-vector multiplication strategies to reduce computation time. The types of graphs for which these re-orderings are useful can be studied in further detail.

For Jordan and $\mathscr{G}$-equivalence classes, the adjacency matrix $\widetilde{A} = \widetilde{V}J\widetilde{V}^{-1}$ of a graph has identical algebraic properties as the adjacency matrix $A = VJV^{-1}$ of a graph in the same class. Structural properties of the corresponding graph $\widetilde{\mathcal{G}} = \mathcal{G}(\widetilde{A})$ as compared to the original graph $\mathcal{G}(A)$ are not yet fully understood. For example, degree distributions, connectedness, and sparsity patterns of potential $\widetilde{\mathcal{G}}$ remain to be studied. One important aspect is determination of appropriate edge thresholds for an adjacency matrix that is numerically constructed to preserve the properties of an equivalence class.

Supervised learning. The graph signal processing framework presented in this thesis represents a solution to an unsupervised learning problem. For the example of NYC taxis on the Manhattan road network, a graph signal is defined as the number of trips that pass through each intersection or specified road location, and clusters of nodes that reflect taxi hotspots are determined by the eigenvectors or adjacency matrix subspaces that are highly expressed in the Fourier expansion of the signal. An interesting follow-up involves using the clusters discovered with our method as inputs to train classifiers such as deep neural networks [125, 126]. The outputs of such classifiers could relate hotspot locations to parameters such as time of day, business types (e.g., restaurants versus transportation hubs), and weather. Such a supervised learning scenario is useful for planning optimal paths to direct emergency first responders and firefighters to critical areas.

We emphasize that our method yields a fine-grained analysis of signal concentrations that leverages the directed nature of the underlying graph topology in comparison to those found with methods based on symmetric adjacency matrices, methods based on the graph Laplacian [127], and learned graph representations [128]. Building interpretable models based on our method in conjunction with other spectral methods and tensor analysis [129, 130] is future work with important ramifications in urban planning.

Correlation vs. causation. Our method yields highly expressed eigenvectors or subspaces whose components of highest magnitude correspond to hotspots such as locations of heavy taxi concentrations in Manhattan, NY. It is possible that several hotspots are identified by a single eigenvector or subspace, in which case they represent intrinsic co-behavior based on the road network eigenstructure, even when the hotspots represent non-adjacent geographical regions. While shutting down traffic at one of these hotspots will affect taxi flows at the others, it is unclear how the effect will manifest (e.g., whether the taxi concentrations will increase or decrease, or whether a new eigenvector will be expressed). In other words, our method characterizes correlation but not causation among hotspots. Studying the effects of localized perturbations of a graph signal on the location of taxi hotspots can lead to further understanding of the hotspot interrelationships.

To conclude, the novel graph signal processing methods we have developed in this thesis have the potential to significantly decrease computation times for spectral analysis of massive data sets over large-scale, directed networks. Our techniques provide effective methods to address previously intractable analyses of real-world networks with defective adjacency matrices. In addition, our methods can be applied to other types of data; other applications include studying the spread of viruses in populations or computer networks and characterizing information flows over social networks.

Bibliography

[1] H. Jeong, B. Tombor, R. Albert, Z.N. Oltvai, and A.L. Barabási, "The large-scale organization of metabolic networks," *Nature*, vol. 407, no. 6804, pp. 651–654, 2000.

[2] C. Chen, D. Zhang, P.S. Castro, N. Li, L. Sun, and S. Li, "Real-time detection of anomalous taxi trajectories from GPS traces," in *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pp. 63–74. Springer, 2012.

[3] P.S. Castro, D. Zhang, C. Chen, S. Li, and G. Pan, "From taxi gps traces to social and community dynamics: A survey," *ACM Computing Surveys*, vol. 46, no. 2, pp. 17:1–17:34, Nov. 2013.

[4] R. Ahlswede, N. Cai, S.-Y.R. Li, and R.W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[5] J.-P. Onnela, J. Saramäki, J. Hyvönen, G. Szabó, D. Lazer, K. Kaski, J. Kertész, and A.-L. Barabási, "Structure and tie strengths in mobile communication networks," *Proceedings of the National Academy of Sciences*, vol. 104, no. 18, pp. 7332–7336, 2007.

[6] A. Sandryhaila and J.M.F. Moura, "Discrete signal processing on graphs," *IEEE Transactions on Signal Processing*, vol. 61, no. 7, pp. 1644–1656, Apr. 2013.

[7] A. Sandryhaila and J.M.F. Moura, "Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure," *IEEE Signal Processing Magazine*, vol. 31, no. 5, pp. 80–90, Aug. 2014.

[8] A. Sandryhaila and J.M.F. Moura, "Discrete signal processing on graphs: Frequency analysis," *IEEE Transactions on Signal Processing*, vol. 62, no. 12, pp. 3042–3054, Jun. 2014.

[9] M. Püschel and J.M.F. Moura, "Algebraic signal processing theory: Foundation and 1-D time," *IEEE Transactions on Signal Processing*, vol. 56, no. 8, pp. 3572–3585, Aug. 2008.

[10] M. Püschel and J.M.F. Moura, "Algebraic signal processing theory: 1-D space," *IEEE Transactions on Signal Processing*, vol. 56, no. 8, pp. 3586–3599, Aug. 2008.

[11] M. Püschel and J. M. F. Moura, "Algebraic signal processing theory: Cooley-Tukey type algorithms for DCTs and DSTs," *IEEE Transactions on Signal Processing*, vol. 56, no. 4, pp. 1502–1521, April 2008.

[12] F. Franchetti, M. Püschel, Y. Voronenko, S. Chellappa, and J. M. F. Moura, "Discrete Fourier transform on multicore," *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 90–102, 2009.

[13] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine Series 6*, vol. 2, no. 11, pp. 559–572, 1901.

[14] H. Hotelling, "Analysis of a complex of statistical variables into principal components," *Journal of Educational Psychology*, vol. 24, no. 6, pp. 417–441, 498–âĂŞ520, 1933.

[15] H. Hotelling, "Relations between two sets of variates," *Biometrika*, vol. 28, no. 3/4, pp. 321–377, 1936.

[16] J. F. Tenenbaum, V. Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, pp. 2319–2323, 2000.

[17] S. Roweis and L. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, pp. 2323–2326, 2000.

[18] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[19] D. L. Donoho and C. Grimes, "Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data," *Proceedings of the National Academy of Sciences*, vol. 100, no. 10, pp. 5591–5596, 2003.

[20] D. Ganesan, B. Greenstein, D. Estrin, J. Heidemann, and R. Govindan, "Multiresolution storage and search in sensor networks," *ACM Transactions on Storage*, vol. 1, pp. 277–315, 2005.

[21] R. Wagner, H. Choi, R. Baraniuk, and V. Delouille, "Distributed wavelet transform for irregular sensor network grids," in *Proceedings of the 13th IEEE Workshop on Statistical Signal Processing*, 2005, pp. 1196–1201.

[22] R. Wagner, R. Baraniuk, S. Du, D. Johnson, and A. Cohen, "An architecture for distributed wavelet analysis and processing in sensor networks," in *Proceedings of the 5th ACM International Conference on Information Processing in Sensor Networks*, 2006, pp. 243–250.

[23] R. R. Coifman and M. Maggioni, "Diffusion wavelets," *Applied and Computational Harmonic Analysis*, vol. 21, no. 1, pp. 53–94, 2006.

[24] D. Shuman, S.K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, Apr. 2013.

[25] D.K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[26] S.K. Narang and A. Ortega, "Perfect reconstruction two-channel wavelet filter banks for graph structured data," *IEEE Transactions on Signal Processing*, vol. 60, no. 6, pp. 2786–2799, Jun. 2012.

[27] A. Agaskar and Y. Lu, "A spectral graph uncertainty principle," *IEEE Transactions on Information Theory*, vol. 59, no. 7, pp. 4338–4356, 2013.

[28] V. Ekambaraman, B Fanti, Ayazifar, and K. Ramchandran, "Multiresolution graph signal processing via circulant structures," in *Proceedings of the IEEE DSP/SPE Workshop*, 2013, pp. 112–117.

[29] X. Zhu and M. Rabbat, "Approximating signals supported on graphs," in *Proceedings of the 37th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Mar. 2012, pp. 3921–3924.

[30] S. Aksoy, F. Chung, and X. Peng, "Extreme values of the stationary distribution of random walks on directed graphs," *Advances in Applied Mathematics*, vol. 81, pp. 128–155, 2016.

[31] F. Chung, P. Horn, and L. Lu, "Diameter of random spanning trees in a given graph," *Journal of Graph Theory*, vol. 69, pp. 223–240, 2012.

[32] M. Püschel and J. M. F. Moura, "The algebraic approach to the discrete cosine and sine transforms and their fast algorithms," *SIAM Journal on Computing*, vol. 32, no. 5, pp. 1280–1316, 2003.

[33] P. Lancaster and M. Tismenetsky, *The Theory of Matrices*, New York, NY, USA: Academic, 2nd edition, 1985.

[34] G.H. Golub and C.F. Van Loan, *Matrix Computations*, Baltimore, MD, USA: JHU Press, 4 edition, 2013.

[35] J.A. Deri and J.M.F. Moura, "Taxis in New York City: A network perspective," in *Proceedings of the 49th Asilomar Conference on Signals, Systems, and Computers*, Nov. 2015, pp. 1829–1833.

[36] J.A. Deri, F. Franchetti, and J.M.F. Moura, "Big Data computation of taxi movement in New York City," in *To appear in Proceedings of the 1st International IEEE Big Data Conference Workshop on Big Spatial Data*, Dec. 2016.

[37] J.A. Deri and J.M.F. Moura, "Churn detection in large user networks," in *Proceedings of the 39th International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2014.

[38] J.L. de la Rosa, R. Mollet, M. Montaner, D. Ruiz, and V. Muñoz, "Kalman filters to generate customer behavior alarms," *Frontiers in Artificial Intelligence and Applications*, vol. 163, pp. 416–425, 2007.

[39] C. Archaux, A. Martin, and A. Khenchaf, "An SVM-based churn detector in prepaid mobile telephony," in *Proceedings of the 2004 IEEE International Conference on Information and Communication Technologies: From Theory to Applications*, Apr. 2004, pp. 459–460.

[40] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos, "Neighborhood formation and anomaly detection in bipartite graphs," in *Proceedings of the 5th IEEE International Conference on Data Mining*, 2005, pp. 418–425.

[41] L. Akoglu, M. McGlohon, and C. Faloutsos, "Oddball: Spotting anomalies in weighted graphs," *Advances in Knowledge Discovery and Data Mining*, pp. 410–421, 2010.

[42] P. Viola and M.J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[43] "Mobile subscriber data," provided courtesy of i-Lab of Carnegie Mellon University.

[44] L.A. Goodman, "Snowball sampling," *Annals of Mathematical Statistics*, vol. 32, no. 1, pp. 148–170, 1961.

[45] M. Gjoka, M. Kurant, C.T. Butts, and A. Markopoulou, "Walking in Facebook: A case study of unbiased sampling of OSNs," in *Proceedings of the 29th IEEE Conference on Computer Communications (INFOCOM)*, March 2010, pp. 1–9.

[46] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *12th ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD)*, 2006, pp. 631–636.

[47] M. Kurant, A. Markopoulou, and P. Thiran, "Towards unbiased BFS sampling," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1799 –1809, Oct. 2011.

[48] M.J. Salganik and D.D. Heckathorn, "Sampling and estimation in hidden populations using respondent-driven sampling," *Sociological Methodology*, vol. 34, pp. 193–239, 2004.

[49] J.A. Deri and J.M.F. Moura, "Graph sampling: Estimation of degree distributions," in *Proceedings of the 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2013.

[50] T.H. Cormen, C. Stein, R.L. Rivest, and C.E. Leiserson, *Introduction to Algorithms*, McGraw-Hill, 2nd edition, 2001.

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

[52] T.M. Mitchell, *Machine Learning*, New York, NY, USA: McGraw-Hill, 1997.

[53] H. Zhang, "The optimality of naive Bayes," in *Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference*, May 2004, pp. 562–567.

[54] J.R. Quinlan, *C4.5: Programs for machine learning*, vol. 1, Morgan Kaufmann, 1993.

[55] I. Gohberg, P. Lancaster, and L. Rodman, *Invariant Subspaces of Matrices with Applications*, vol. 51, Philadelphia, PA, USA: SIAM, 2006.

[56] R.A. Horn and C.R. Johnson, *Matrix Analysis*, Cambridge, U.K.: Cambridge Univ. Press, 2012.

[57] C. Jordan, *Traité des substitutions et des équations algébriques*, Paris, 1870.

[58] M. Vetterli, J. Kovačević, and V.K. Goyal, *Foundations of Signal Processing*, Cambridge, U.K.: Cambridge Univ. Press, 2014.

[59] A.V. Oppenheim, A.S. Willsky, and S.H. Nawab, *Signals and systems*, vol. 2, Englewood Cliffs, NJ, USA: Prentice-Hall, 1983.

[60] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, Englewood Cliffs, NJ, USA: Prentice-Hall, 2 edition, 1999.

[61] D.M. Cvetković, M. Doob, I. Gutman, and A. Torgašev, *Recent results in the theory of graph spectra*, vol. 36 of *Annals of Discrete Mathematics*, North-Holland, 1988.

[62] C. Candan, "On the eigenstructure of DFT matrices [DSP education]," *IEEE Signal Processing Magazine*, vol. 28, no. 2, pp. 105–108, 2011.

[63] L.K. Jørgensen, "On normally regular digraphs," Tech. Rep. R 94-2023, Univ. of Aalborg, Institute for Electronic Systems, Dept. of Mathematics and Computer Science, 1994.

[64] D.A. Cardon and B. Tuckfield, “The Jordan canonical form for a class of zero–one matrices,” *Linear Algebra and its Applications*, vol. 435, no. 11, pp. 2942–2954, 2011.

[65] H. Nina, R.L. Soto, and D.M. Cardoso, “The Jordan canonical form for a class of weighted directed graphs,” *Linear Algebra and its Applications*, vol. 438, no. 1, pp. 261–268, 2013.

[66] S. Mallat, *A Wavelet Tour of Signal Processingg*, New York, NY, USA: Academic, 3 edition, 2008.

[67] Baruch College: Baruch Geoportal, “NYC Geodatabase,” URL: `https://www.baruch.cuny.edu/confluence/display/geoportal/NYC+Geodatabase`, Accessed 31-Aug.-2014.

[68] L.A. Adamic and N. Glance, “The political blogosphere and the 2004 U.S. election: Divided they blog,” in *Proceedings of the 3rd ACM International Workshop on Link Discovery (LinkKDD)*, 2005, pp. 36–43.

[69] W. Kahan, “Conserving confluence curbs ill-condition,” Tech. Rep. AD0766916, Department of Computer Science, University of California, Berkeley, Berkeley, CA, USA, 1972.

[70] Z. Zeng, “Regularization and Matrix Computation in Numerical Polynomial Algebra,” in *Approximate Commutative Algebra*, L. Robbiano and J. Abbott, Eds., Texts and Monographs in Symbolic Computation, pp. 125–162. Springer-Verlag Vienna, 2010.

[71] Z. Zeng, “Sensitivity and Computation of a Defective Eigenvalue,” preprint at `http://orion.neiu.edu/~zzeng/Papers/eigit.pdf`, Apr. 2015.

[72] L.N. Trefethen and M. Embree, *Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators*, Princeton, NJ, USA: Princeton Univ. Press, 2005.

[73] A. Ruhe, “An algorithm for numerical determination of the structure of a general matrix,” *BIT Numerical Mathematics*, vol. 10, no. 2, pp. 196–216, 1970.

[74] Axel Ruhe, “Properties of a matrix with a very ill-conditioned eigenproblem,” *Numerische Mathematik*, vol. 15, no. 1, pp. 57–60, 1970.

[75] N. Guglielmi, M.L. Overton, and G.W. Stewart, “An efficient algorithm for computing the generalized null space decomposition,” *SIAM Journal on Matrix Analysis & Applications*, vol. 36, no. 1, pp. 38–54, Oct. 2014.

[76] G. H. Golub and J. H. Wilkinson, “Ill-Conditioned Eigensystems and the Computation of the Jordan Canonical Form,” *SIAM Review*, vol. 18, no. 4, pp. 578–619, 1976.

[77] Python, *version 2.7*, Python Software Foundation, 2015.

[78] MATLAB, *version 8.5.0 (R2015a)*, The MathWorks Inc., Natick, Massachusetts, 2015.

[79] J. Bezanson, S. Karpinski, V.B. Shah, and A. Edelman, "Julia: A Fast Dynamic Language for Technical Computing," *arXiv preprint (arXiv:1209.5145)*, Sep. 2012.

[80] Hadoop, *version 2.6.4*, Apache Software Foundation, 2015.

[81] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, 2010, p. 10.

[82] HTCondor, *version 8.2.10*, Center for High Throughput Computing, Univ. of Wisconsin-Madison, 2015.

[83] Z. Bujavonić, *Krylov Type Methods for Large Scale Eigenvalue Computations*, Ph.D. thesis, Univ. of Zagreb, Zagreb, Croatia, 2011.

[84] J.H. Wilkinson, *The Algebraic Eigenvalue Problem*, vol. 87, Oxford, UK: Clarendon Press, 1965.

[85] J. Wilkening, "An algorithm for computing Jordan chains and inverting analytic matrix functions," *Linear Algebra and its Applications*, vol. 427, no. 1, pp. 6–25, 2007.

[86] A. N. Yzelman and D. Roose, "High-level strategies for parallel shared-memory sparse matrixâĂŞvector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 116–125, 2014.

[87] J.W. Demmel, *A Numerical Analyst's Jordan Canonical Form*, Ph.D. thesis, Univ. of California, Berkeley, 1983.

[88] T. Kato, *Perturbation Theory for Linear Operators*, Springer-Verlag Berlin, 2nd edition, 1995.

[89] G.W. Stewart and J.G. Sun, *Matrix Perturbation Theory*, Boston, MA, USA: Academic, 1990.

[90] D.M. Cvetković, P. Rowlinson, and S. Simić, *An Introduction to the Theory of Graph Spectra*, Cambridge, U.K.: Cambridge Univ. Press, 2010.

[91] R.A. Brualdi and D.M. Cvetković, *A Combinatorial Approach to Matrix Theory and Its Applications*, Boca Raton, FL, USA: CRC press, 2008.

[92] F. Harary, "A graph theoretic method for the complete reduction of a matrix with a view toward finding its eigenvalues," *Journal of Mathematics and Physics*, vol. 38, no. 1, pp. 104–111, 1959.

[93] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos, "Epidemic spreading in real networks: An eigenvalue viewpoint," in *Proceedings of the 22nd IEEE International Symposium on Reliable Distributed Systems*, 2003, pp. 25–34.

[94] R. Lemonnier, K. Scaman, and N. Vayatis, "Tight bounds for influence in diffusion networks and application to bond percolation and epidemiology," in *Advances in Neural Information Processing Systems*, 2014, pp. 846–854.

[95] R. Brawer and M. Pirovino, "The linear algebra of the Pascal matrix," *Linear Algebra and Its Applications*, vol. 174, pp. 13–23, 1992.

[96] R.E. Greene and S.G. Krantz, *Function Theory of One Complex Variable*, vol. 40, Providence, RI, USA: American Math. Soc., 3rd edition, 2006.

[97] N. Ferreira, J. Poco, H.T. Vo, J. Freire, and C.T. Silva, "Visual exploration of big spatio-temporal urban data: A study of New York City taxi trips," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2149–2158, 2013.

[98] B. Donovan and D.B. Work, "Using coarse GPS data to quantify city-scale transportation system resilience to extreme events," *presented at Transportation Research Board 94th Annual Meeting (preprint: arXiv:1507.06011 [physics.soc-ph])*, Jan. 2015.

[99] "NYC Open Data," `https://nycopendata.socrata.com/`.

[100] New York City Dept. of Transportation, "Real time traffic information," URL: `http://dotsignals.org/`, Accessed 25-May-2016.

[101] B. Donovan and D.B. Work, "New York City Taxi Data (2010-2013)," Dataset, `http://dx.doi.org/10.13012/J8PN93H8`, 2014, Accessed 30-Jun.-2016.

[102] D. Chu, D. A. Sheets, Y. Zhao, Y. Wu, J. Yang, M. Zheng, and G. Chen, "Visualizing hidden themes of taxi movement with semantic transformation," in *2014 IEEE Pacific Visualization Symposium*, March 2014, pp. 137–144.

[103] X. He and H.X. Liu, "Modeling the day-to-day traffic evolution process after an unexpected network disruption," *Transportation Research Part B: Methodological*, vol. 46, no. 1, pp. 50–71, 2012.

[104] P.S. Castro, D. Zhang, and S. Li, "Urban traffic modelling and prediction using large scale taxi GPS traces," in *Pervasive Computing*, pp. 57–72. Springer, 2012.

[105] Y. Zheng, Y. Liu, J. Yuan, and X. Xie, "Urban computing with taxicabs," in *Proceedings of the 13th International Conference on Ubiquitous Computing*. ACM, 2011, pp. 89–98.

[106] R. Puzis, Y. Altshuler, Y. Elovici, S. Bekhor, Y. Shiftan, and A. Pentland, "Augmented betweenness centrality for environmentally aware traffic monitoring in transportation networks," *Journal of Intelligent Transportation Systems*, vol. 17, no. 1, pp. 91–105, 2013.

[107] Google, "Google Map of New York, New York," `https://goo.gl/maps/57U3mPQcdQ92`, Accessed 31-Aug.-2014.

[108] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the Condor experience," *Concurrency – Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[109] E.W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[110] S. Han, F. Franchetti, and M. Püschel, "Program generation for the all-pairs shortest path problem," in *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2006, pp. 222–232.

[111] D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Engineering route planning algorithms," in *Algorithmics of Large and Complex networks*, pp. 117–139. Springer, 2009.

[112] A.V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory," in *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005, pp. 156–165.

[113] C. Sommer, "Shortest-path queries in static networks," *ACM Computing Surveys*, vol. 46, no. 4, pp. 45:1–45:31, Apr. 2014.

[114] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *Experimental Algorithms*, vol. 5038, pp. 319–333. Springer Heidelberg, 2008.

[115] J. Zhang, "Efficient frequent sequence mining on taxi trip records using road network shortcuts," in *Big Data: Techniques and Technologies in Geoinformatics*, H.A. Karimi, Ed., pp. 193–206. CRC Press, 2014.

[116] A.V. Goldberg and R.E. Tarjan, "Expected performance of Dijkstra's shortest path algorithm," *NEC Research Institute Report*, Jun. 1996.

[117] N. Jasika, N. Alispahic, A. Elma, K. Ilvana, L. Elma, and N. Nosovic, "Dijkstra's shortest path algorithm serial and parallel execution performance analysis," in *Proceedings of the 35th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2012, pp. 1811–1815.

[118] B.V. Cherkassky, A.V. Goldberg, and T. Radzik, "Shortest paths algorithms: Theory and experimental evaluation," *Mathematical Programming*, vol. 73, no. 2, pp. 129–174, 1996.

[119] M.H. Sharker and H.A. Karimi, "Computing least air pollution exposure routes," *International Journal of Geographical Information Science*, vol. 28, no. 2, pp. 343–362, 2014.

[120] D. Kahle and H. Wickham, "ggmap: Spatial visualization with ggplot2," *The R Journal*, vol. 5, no. 1, pp. 144–161, 2013.

[121] OpenStreetMap contributers, "OpenStreetMap," `http://www.openstreetmap.org/`, Accessed 09-Jun.-2016.

[122] J.H. Kwak and S. Hong, *Linear Algebra*, Birkhäuser Boston, 2004.

[123] R.T. Behrens and L.L. Scharf, "Signal processing applications of oblique projection operators," *IEEE Transactions on Signal Processing*, vol. 42, no. 6, pp. 1413–1424, 1994.

[124] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th National Conference*, New York, NY, USA, 1969, ACM '69, pp. 157–172, ACM.

[125] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[126] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[127] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering.," in *NIPS*, 2001, vol. 14, pp. 585–591.

[128] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering," in *Proc. AAAI Conference on Artifical Intelligence*, 2014, pp. 1293–1299.

[129] J. Sun, D. Tao, and C. Faloutsos, "Beyond streams and graphs: Dynamic tensor analysis," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 374–383.

[130] A.R. Benson, D.F. Gleich, and J. Leskovec, "Tensor spectral clustering for partitioning higher-order network structures," in *SIAM International Conference on Data Mining (SDM)*, 2015.

[131] P. Hilton, D. Holton, and J. Pedersen, "Pascal, Euler, Triangles, Windmills,...," in *Mathematical Reflections*, Undergraduate Texts in Mathematics, pp. 185–248. Springer New York, 1997.

[132] ESRI, "ESRI Shapefile Technical Description," Tech. Rep., ESRI, Jul. 1998.

[133] F.C. Pereira, H. Costa, and N.M. Pereira, "An off-line map-matching algorithm for incomplete map databases," *European Transport Research Review*, vol. 1, no. 3, pp. 107–124, 2009.

[134] M.A. Quddus, W.Y. Ochieng, L. Zhao, and R.B. Noland, "A general map matching algorithm for transport telematics applications," *GPS Solutions*, vol. 7, no. 3, pp. 157–167, 2003.

[135] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk, "On map-matching vehicle tracking data," in *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, 2005, pp. 853–864.

Appendix A

Proofs for Section 8.1

A.1 Proof of Theorem 8.1

Proof of Theorem 8.1. *(i)* If $p = N$, then $H_N^N = J_{N,p} = 0_{N\times N}$; i.e., the Jordan form consists of p Jordan blocks for eigenvalue 0.

Let $p = N - k$ for $1 \leq k < N$. The matrix H_N^{N-k} consists of zeros except for k unit elements along the pth diagonal. Each non-zero element constrains a single variable in the system $H_N^p x = b$, $x, b \in \mathbb{R}^N$, so there are $p = N - k$ free variables; i.e, the dimension of the null space of H_N^{N-k} is $N - k = p$, so there are p Jordan blocks.

(ii) The maximum Jordan block size (chain length) l for matrix $A = H_N^p$ is found. Let $\mathrm{Ker}(A)$ denote the null space, or kernel, of A. The recurrence equation for the Jordan chain is

$$Av_i = \lambda v_i + v_{i-1} = v_{i-1}, \qquad i = 2, \ldots, l, \tag{A.1}$$

where $v_1 \in \mathrm{Ker}(A)$ and the ith vector v_i, $i \geq 2$, satisfies $v_i \in \mathrm{Ker}(A^i) \backslash \mathrm{Ker}(A^{i-1})$ [33]. Applying (A.1) more than l times will yield no other linearly independent vector in the generalized eigenspace corresponding to eigenvector v_1, or

$$\mathrm{Ker}(A^{l+1}) \backslash \mathrm{Ker}(A^l) = \emptyset. \tag{A.2}$$

Since H_N is nilpotent, $H_N^i = 0_{N\times N}$ for all $i \geq N$. The null space of such H_N^i has dimension N, so we satisfy (A.2) if $i \geq N$. Thus, the maximum Jordan chain length of H_N^p is the minimum integer $l \in [1, N)$ such that $(H_N^p)^l = 0_{N\times N} = H_N^N$. This equals the minimum l such that $pl \geq N$, or $l = \left\lceil \frac{N}{p} \right\rceil$.

(iii) The number of Jordan blocks k_r of size r has a lower bound given by the difference between the

dimensions of the null space of A^r and the null space of A^{r-1} [33, 56]:

$$k_r \geq |\mathrm{Ker}\,(A^r)| - \left|\mathrm{Ker}\left(A^{r-1}\right)\right|. \tag{A.3}$$

In addition, the number of Jordan blocks of size r equals [33]

$$k_r = 2\,|\mathrm{Ker}\,(A^r)| - \left|\mathrm{Ker}\left(A^{r-1}\right)\right| - \left|\mathrm{Ker}\left(A^{r+1}\right)\right|. \tag{A.4}$$

From the proof of (ii), $\left|\mathrm{Ker}(H_N^{pl})\right| = N$, and $\left|\mathrm{Ker}(H_N^{p(l-1)})\right| = p(l-1)$ from (i). Apply (A.3) with equality to compute the number of Jordan blocks k_l of maximum length $l = \left\lceil \frac{N}{p} \right\rceil$:

$$\begin{aligned} k_l &= \left|\mathrm{Ker}\left(H_N^{pl}\right)\right| - \left|\mathrm{Ker}\left(H_N^{p(l-1)}\right)\right| && \text{(A.5)} \\ &= N - p\,(l-1). && \text{(A.6)} \end{aligned}$$

Use (A.4) to compute the number of Jordan blocks of size $l-1$:

$$\begin{aligned} k_{l-1} = 2&\left|\mathrm{Ker}\left(H_N^{p(l-1)}\right)\right| \\ &\quad - \left|\mathrm{Ker}\left(H_N^{p(l-2)}\right)\right| - \left|\mathrm{Ker}\left(H_N^{pl}\right)\right| && \text{(A.7)} \\ &= 2p\,(l-1) - p\,(l-2) - N && \text{(A.8)} \\ &= pl - N. && \text{(A.9)} \end{aligned}$$

Since $k_l + k_{l-1} = p$ (the number of Jordan blocks by (i)), we get $k_{l-2} = \cdots = k_2 = k_1 = 0$ for $l \geq 3$. (We can also show by (A.4).) Thus, the Jordan blocks have size l or $l-1$.

Let p be a factor of N, i.e., there exists integer c such that $pc = N$. By (i), $c = l$ (the maximum block size), or $pl = N$. By (A.9), $k_{l-1} = 0$, so the minimum block size is l. Now suppose p is not a factor of N. Then, $k_{l-1} \neq 0$, so the minimum block size is $l - 1 = \left\lfloor \frac{N}{p} \right\rfloor$. □

A.2 Proof of Lemma 8.2

Proof of Lemma 8.2. The proof of Theorem 8.1(iii) shows that $J_{N,p}$ has blocks of size $\left\lceil \frac{N}{p} \right\rceil$ and $\left\lfloor \frac{N}{p} \right\rfloor$ with counts given by (A.6) and (A.9), respectively. These details result in (8.1). □

A.3 Proof of Theorem 8.3

Proof of Theorem 8.3. By Theorem 8.1(ii), the Jordan form has maximum chain length $l = \left\lceil \frac{N}{p} \right\rceil$ corresponding to the Jordan block $H_{\lceil \frac{N}{p} \rceil}$, the upper-left matrix in (8.2). The form of the bottom-right matrix is found next.

By Theorem 8.1(i) and(ii), $J_{N,p}$ can be decomposed as

$$J_{N,p} = \begin{bmatrix} H_{\lceil \frac{N}{p} \rceil} & \\ & \\ 0 & X \end{bmatrix}, \tag{A.10}$$

where the $(N - \left\lceil \frac{N}{p} \right\rceil) \times (N - \left\lceil \frac{N}{p} \right\rceil)$ matrix X is composed of $p-1$ Jordan blocks. Since the Segre characteristic of $J_{N,p}$ is known by Lemma 8.2, let $l = \left\lceil \frac{N}{p} \right\rceil$ and apply (A.6) and (A.9) to get the numbers of blocks of size l and size $l-1$:

$$k_l = N - pl + p \tag{A.11}$$

$$k_{l-1} = pl - N. \tag{A.12}$$

The number of blocks of size l and size $l-1$ in X must be

$$k_l^{(X)} = k_l - 1 = N - pl + p - 1 \tag{A.13}$$

$$k_{l-1}^{(X)} = k_{l-1} = pl - N. \tag{A.14}$$

This is the Segre characteristic of X. One possible X that satisfies (A.13) and (A.14) is $J_{N-\lceil \frac{N}{p} \rceil, p-1}$. In addition, since the Segre characteristic and eigenvalues $\lambda = 0$ are known, $X = J_{N-\lceil \frac{N}{p} \rceil, p-1}$ is unique up to the order of Jordan blocks. □

A.4 Proof of Theorem 8.4

Proof of Theorem 8.4. Solve the system of equations $H_N^p v = 0_N$. Matrix H_N^p has $N - p$ unit elements along the pth diagonal, so $v_i = 0$ for $i = p+1, \ldots, N$. It remains to determine $v_1, v_2, \ldots, v_p$ to get p linearly independent eigenvectors. (There are p proper eigenvectors since there are p Jordan blocks by Theorem 8.1.) Choosing the columns of $E_{N,p}$ satisfies these constraints. □

A.5 Proof of Theorem 8.8

Proof of Theorem 8.8. *(i)* The digraph $\mathcal{G}_{N,p}$ of unperturbed matrix H_N^p is not strongly connected and has nodes of in- and out-degree less than or equal to one; this means it is composed of directed paths and isolated nodes. The digraph $\mathcal{G}_{N,p}|_{\epsilon,j}$ adds weighted edge (v_j, v_1) to $\mathcal{G}_{N,p}$. A directed cycle exists in $\mathcal{G}_{N,p}|_{\epsilon,j}$ if a path exists from v_1 to v_j in $\mathcal{G}_{N,p}$, such as by choosing exponent $p = j$. The added edge (v_j, v_1) closes a single path, so $\mathcal{G}_{N,p}|_{\epsilon,j}$ has at most one directed cycle.

(ii) A directed cycle exists if a directed path exists from node v_1 to node v_j. This occurs if $j-1$ is a multiple of p. By inspection, the cycle length is $\frac{j-1}{p} + 1$, or $\lceil \frac{j}{p} \rceil$.

(iii) Suppose $\mathcal{G}_{N,p}|_{\epsilon,j}$ contains a directed cycle, i.e., an edge (v_j, v_1) closes a directed path from v_1 to v_j. If a path exists from v_j to v_k, $k > j$, the weakly connected component $\mathcal{G}_1$ containing the directed cycle is the union of the cycle and the path from v_j to v_k. The other components must be the directed paths and isolated nodes of unperturbed $\mathcal{G}_{N,p}$ but whose nodes are not contained in $\mathcal{G}_1$. If there is no directed cycle, the edge (v_j, v_1) augments but does not close a directed path in $\mathcal{G}_{N,p}$, so all components in $\mathcal{G}_{N,p}|_{\epsilon,j}$ are either directed paths or isolated nodes.

(iv) The result follows from (i) and (iii). □

A.6 Proof of Theorem 8.9

Proof of Theorem 8.9. The characteristic polynomial of $H_N^p|_{\epsilon,j}$ is the product of the characteristic polynomials of the strongly connected components in its digraph $\mathcal{G}_{N,p}|_{\epsilon,j}$ [92]. If $j-1$ is a multiple of p, the digraph has one directed cycle with $= \lceil \frac{j}{p} \rceil$ nodes by Theorem 8.8(i) and (ii). The characteristic polynomial of the cycle equals (8.9) with $N = l$. The other strongly connected components $\mathcal{G}_1, \ldots, \mathcal{G}_{N-l}$ are isolated nodes with characteristic polynomials $\varphi_{\mathcal{G}_i}(\lambda) = -\lambda$, $1 \leq i \leq N - l$. The eigenvalues of $H_N^p|_{\epsilon,j}$ are the roots of the characteristic polynomial of $H_N^p|_{\epsilon,j}$

$$\varphi_{H_N^p|_{\epsilon,j}}(\lambda) = \varphi_{H_l}(\lambda) \prod_{i=1}^{N-l} \varphi_{\mathcal{G}_i}(\lambda) \tag{A.15}$$

$$= (-1)^N \lambda^{N-l}(\lambda^l - \epsilon). \tag{A.16}$$

Suppose $j-1$ is not a multiple of p. Then the characteristic polynomial becomes $\varphi_{H_N^p|_{\epsilon,j}}(\lambda) = (-\lambda)^N$; i.e., all eigenvalues are zero. □

Appendix B

Proofs for Section 8.2

B.1 Proof of Theorem 8.15

Proof of Theorem 8.15. We proceed by induction. For $N = 1$, $A_N = 0$, so the single eigenvector $V_1 = 1$, which equals $P_1 = P_1^{-1}$.

Let v_N denote the first N elements of the $(N+1)$th column of P_{N+1} so that the ith element, $i = 1, \ldots, N$, is given by

$$[v_N]_i = \binom{N}{i-1}. \tag{B.1}$$

Assume $A_N = P_N^{-1} J_N P_N$, or $P_N A_N = J_N P_N$. We need to show $P_{N+1} A_{N+1} = J_{N+1} P_{N+1}$. The left side yields

$$P_{N+1} A_{N+1} = \begin{bmatrix} P_N & v_N \\ 0_N^T & 1 \end{bmatrix} \begin{bmatrix} A_N & 1_N \\ 0_N^T & 0 \end{bmatrix} \tag{B.2}$$

$$= \begin{bmatrix} P_N A_N & P_N 1_N \\ 0_N^T & 0 \end{bmatrix} \tag{B.3}$$

$$= \begin{bmatrix} J_N P_N & P_N 1_N \\ 0_N^T & 0. \end{bmatrix} \qquad \text{(induction).} \tag{B.4}$$

Since $\binom{n+1}{k+1} = \sum_{j=k}^{n} \binom{j}{k}$ for $1 \leq k \leq n$ [131], we can write

$$[P_N 1_N]_i = \sum_{j=i}^{N-1} \binom{j}{i} = \binom{N}{i} \tag{B.5}$$

Now consider $J_{N+1}P_{N+1}$:

$$J_{N+1}P_{N+1} = \begin{bmatrix} J_N & 0_N \\ 0_N^T & 0 \end{bmatrix} \begin{bmatrix} P_N & v_N \\ 0_N^T & 1 \end{bmatrix} \tag{B.6}$$

$$= \begin{bmatrix} J_N P_N & J_N v_N \\ 0_N^T & 0 \end{bmatrix}. \tag{B.7}$$

Multiplying v_N by J_N results in a circular shift of v_N's elements so that $[v_N]_i = \binom{N}{i}$, $i = 1, \ldots, N$, proving the equality of the upper-right block matrices in (B.4) and (B.7). □

B.2 Proof of Theorem 8.16

Proof of Theorem 8.16. It suffices to show that $A_{N+1} = V_{N+1}J_{N+1}V_{N+1}^{-1}$:

$$V_{N+1}J_{N+1}V_{N+1}^{-1}$$

$$= \begin{bmatrix} 1 & 0_N^T \\ 0_N & P_N^{-1} \end{bmatrix} \begin{bmatrix} 0 & \begin{bmatrix} 1 & 0 & \ldots & 0 \end{bmatrix} \\ 0_N & J_N \end{bmatrix} \begin{bmatrix} 1 & 0_N^T \\ 0_N & P_N \end{bmatrix} \tag{B.8}$$

$$= \begin{bmatrix} 1 & 0_N^T \\ 0_N & P_N^{-1} \end{bmatrix} \begin{bmatrix} 0 & \begin{bmatrix} 1 & 0 & \ldots & 0 \end{bmatrix} P_N \\ 0_N & J_N P_N \end{bmatrix} \tag{B.9}$$

$$= \begin{bmatrix} 0 & 1_N^T \\ 0_N & P_N^{-1} J_N P_N \end{bmatrix} \tag{B.10}$$

$$= \begin{bmatrix} 0 & 1_N^T \\ 0_N & A_N \end{bmatrix} \qquad \text{(Theorem 8.15)} \tag{B.11}$$

$$= A_{N+1}. \tag{B.12}$$

□

B.3 Proof of Lemma 8.17

Proof of Lemma 8.17. We prove by induction. Consider base cases $N \in \{2, 3\}$. For $N = 2$, $M_2 = 1$, so $\det(M_2) = 1 = (\lambda + 1)^0$. For $N = 3$, M_3 has determinant $\lambda + 1$.

Assume (8.24) holds for $N = k > 2$. Let $C_{p,q}^{(k)}$ be the $(k-2) \times (k-2)$ submatrix formed by removing the pth row and qth column of M_l. Let $c_{p,q}^{(k)}$ represent the pq-cofactor of M_k for $1 \leq p, q < k$. Compute $\det(M_{k+1})$ by expanding by minors along the first column:

$$\det(M_{k+1}) = \sum_{i=1}^{k+1} [M_{k+1}]_{i1}\, c_{i,1}^{(k+1)} \tag{B.13}$$

$$= c_{1,1}^{(k+1)} - \lambda c_{2,1}^{(k+1)}. \tag{B.14}$$

Note that $c_{1,1}^{(k+1)}$ corresponds to $(k-1) \times (k-1)$ matrix $C_{1,1}^{(k+1)}$, and that $C_{1,1}^{(k+1)} = C_{2,1}^{(k+1)} = M_k$ has cofactors $c_{1,1}^{(k+1)} = \det(M_k)$ and $c_{2,1}^{(k+1)} = -c_{1,1}^{(k+1)}$. Rewriting (B.14), we get

$$\det(M_{k+1}) = \det(M_k) - \lambda(-\det(M_k)) \tag{B.15}$$

$$= (1+\lambda)\det(M_k) \tag{B.16}$$

$$= (1+\lambda)(1+\lambda)^{k-2} \qquad \text{(induction)} \tag{B.17}$$

$$= (1+\lambda)^{k-1}. \tag{B.18}$$

□

B.4 Proof of Theorem 8.18

Proof of Theorem 8.18. Let $j = N$ and $B_{p,q}^{(N)}$ denote the $(N-1) \times (N-1)$ submatrix formed by removing the pth row and qth column of $A_N|_{\epsilon,N}$, $1 \leq p, q \leq N$. Let $b_{p,q}^{(N)}(\lambda) = (-1)^{p+q}\det(B_{p,q}^{(N)})$ denote the cofactors of $A_N|_{\epsilon,N}$. Expand by minors along the first column of $A_N|_{\epsilon,N} - \lambda I_N$:

$$\varphi_N|_{\epsilon,N}(\lambda) = \sum_{i=1}^{N} [A_N|_{\epsilon,N}]_{i1}\, b_{i,1}^{(N)} \tag{B.19}$$

$$= -\lambda b_{1,1}^{(N)} + \epsilon b_{N,1}^{(N)}. \tag{B.20}$$

The submatrix $B_{1,1}^{(N)}$ for $b_{1,1}^{(N)}$ is upper triangular of form

$$B_{1,1}^{(N)} = \begin{bmatrix} -\lambda & 1 & 1 & \dots & 1 \\ & -\lambda & 1 & \dots & 1 \\ & & \ddots & \ddots & \\ & & & \ddots & 1 \\ & & & & -\lambda \end{bmatrix}, \tag{B.21}$$

so $b_{1,1}^{(N)} = (-\lambda)^{N-1}$ and the first term in (B.19) is $(-\lambda)^N$. The submatrix $B_{N,1}^{(N)}$ corresponding to $b_{N,1}^{(N)}$ equals M_N as in Lemma 8.17. Then, by Lemma 8.17, the corresponding cofactor $b_{N,1}^{(N)} = (-1)^{N+1}(\lambda+1)^{N-2}$. We get

$$\varphi_N|_{\epsilon,N}(\lambda) = (-\lambda)^N + \epsilon(-1)^{N+1}(\lambda+1)^{N-2} \tag{B.22}$$

$$= (-1)^N\left(\lambda^N - \epsilon(\lambda+1)^{N-2}\right). \tag{B.23}$$

Let $2 \leq j < N$. Note that digraph $\mathcal{G}(A_N|_{\epsilon,j})$ has only one strongly connected component of size greater than one. This component has form $\mathcal{G}(A_N|_{\epsilon,j})$. Equation (8.26) results since the characteristic polynomial of $A_N|_{\epsilon,j}$ is the product of the characteristic polynomials of the strongly connected components in its digraph $\mathcal{G}(A_N|_{\epsilon,j})$ as shown in [92]. □

B.5 Proof of Theorem 8.20

Proof of Theorem 8.20. (i) By the Perron-Frobenius Theorem, $\lambda = \rho_{N,\epsilon}$ is real-valued for $A_N|_{\epsilon,N}$. Substituting in (8.29) and taking the logarithm yields

$$\log\frac{\rho_{N,\epsilon}}{\rho_{N,\epsilon}+1} = \frac{1}{N}\log\frac{\epsilon}{(\rho_{N,\epsilon}+1)^2}. \tag{B.24}$$

Since $\epsilon \in (0,1)$ and $\rho_{N,\epsilon}$ is a positive real number,

$$\rho_{N,\epsilon} > 0 > \sqrt{\epsilon} - 1,$$

so $(\rho_{N,\epsilon}+1)^2 \geq \epsilon$, or $\frac{\epsilon}{(\rho_{N,\epsilon}+1)^2} < 1$. We then get

$$\log \frac{\rho_{N,\epsilon}}{\rho_{N,\epsilon}+1} = \frac{1}{N} \log \frac{\epsilon}{\left(\rho_{N,\epsilon}+1\right)^2} < 0. \tag{B.25}$$

Let $N \to \infty$. Then, $\log \frac{\rho_{N,\epsilon}}{\rho_{N,\epsilon}+1} \to 0^-$, or $\frac{\rho_{N,\epsilon}}{\rho_{N,\epsilon}+1} \to 1^-$. This implies that $\rho_{N,\epsilon} \to \infty$.

(ii) Let $N \to 0$. Then, $\log \frac{\rho_{N,\epsilon}}{\rho_{N,\epsilon}+1} \to -\infty$ from (B.25), or $\frac{\rho_{N,\epsilon}}{\rho_{N,\epsilon}+1} \to 0^+$. This implies $\rho_{N,\epsilon} \to 0^+$.

(iii) Let $\epsilon = \alpha^N$, $0 < \alpha < 1$. From (B.24),

$$\log \frac{\rho_{N,\epsilon}}{\rho_{N,\epsilon}+1} = \log \alpha + \frac{1}{N} \log \frac{1}{\left(\rho_{N,\epsilon}+1\right)^2}. \tag{B.26}$$

If $N \to \infty$, then $\log \frac{\rho_{N,\epsilon}}{\rho_{N,\epsilon}+1} \to \log \alpha^-$, or $\frac{\rho_{N,\epsilon}}{\rho_{N,\epsilon}+1} \to \alpha^-$. The result follows for $\alpha \to 1$ and $\alpha \to 0$. □

B.6 Proof of Theorem 8.21

Proof of Theorem 8.21. (i) Taking the magnitude of both sides of (8.29) yields

$$\left| \left(\frac{\lambda}{\lambda+1} \right)^N \right| = \left| \frac{\epsilon}{(\lambda+1)^2} \right|, \text{ or,} \tag{B.27}$$

$$\left| \frac{\lambda}{\lambda+1} \right|^N = \frac{\epsilon}{|\lambda+1|^2}. \tag{B.28}$$

Take the logarithm of both sides and divide by N:

$$\log \left| \frac{\lambda}{\lambda+1} \right| = \frac{1}{N} \log \frac{\epsilon}{|\lambda+1|^2}. \tag{B.29}$$

Since $\operatorname{Re} \lambda > -\frac{1}{2}$, we get $\left| \frac{\lambda}{\lambda+1} \right| < 1$ by Lemma 8.19, so both sides of (B.28) are less than 1. Therefore,

$$\log \left| \frac{\lambda}{\lambda+1} \right| = \frac{1}{N} \log \frac{\epsilon}{|\lambda+1|^2} < 0. \tag{B.30}$$

Let $N \to \infty$. By (B.29), $\log \left| \frac{\lambda}{\lambda+1} \right| \to 0^-$, or $\left| \frac{\lambda}{\lambda+1} \right| \to 1^-$. If $\operatorname{Re} \lambda > 0$, then $|\lambda| < |\lambda+1|$, so $|\lambda| \to \infty$. Now let $\operatorname{Re} \lambda < 0$. We require $|\lambda| \to |\lambda+1|$. Substitute $\lambda = \operatorname{Re} \lambda + j \operatorname{Im} \lambda$ to get $\operatorname{Re} \lambda \to -\frac{1}{2}^+$.

(ii) Let $N \to 0$. Then, $\log \left| \frac{\lambda}{\lambda+1} \right| \to -\infty$ by (B.30). This implies $\left| \frac{\lambda}{\lambda+1} \right| \to 0^+$, or $|\lambda| \to 0^+$.

(iii) Assume $\epsilon = \alpha^N$, $0 < \alpha < 1$. Then, from (B.29),

$$\log \left| \frac{\lambda}{\lambda + 1} \right| = \log \alpha + \frac{1}{N} \log \frac{1}{|\lambda + 1|^2}. \tag{B.31}$$

If $N \to \infty$, then $\log \left| \frac{\lambda}{\lambda+1} \right| \to \log \alpha^-$, or $|\lambda| \to \alpha\, |\lambda + 1|^-$. Then, $|\lambda| \to 0$ for $\alpha \to 0$. If $\alpha \to 1$, the proof follows that of (i). □

B.7 Proof of Theorem 8.22

Proof of Theorem 8.22. It is first shown by contradiction that $\operatorname{Re}\lambda > -\frac{1}{2}$ must hold for all $\lambda \in \Lambda_N|_{\epsilon,N}$ to ensure that $|\lambda| \to 0$ as $N \to 0$. Let $\operatorname{Re}\lambda < -\frac{1}{2}$, so $\left| \frac{\lambda}{\lambda+1} \right| > 1$ by Lemma 8.19. Then, both sides of (B.28) are greater than one, so both sides of (B.29) are greater than zero. As $N \to 0$, $\log \left| \frac{\lambda}{\lambda+1} \right| \to \infty$, or $|\lambda| \to \infty$. This is a contradiction, so $\operatorname{Re}\lambda \geq -\frac{1}{2}$.

Next, ϵ' is found such that, for all $\epsilon < \epsilon'$, all eigenvalues converge to zero, or, equivalently, $\operatorname{Re}\lambda \geq -\frac{1}{2}$. By Theorem 8.21(ii), eigenvalue λ converges to zero as $N \to 0$ for all $\epsilon \in (0, 1)$ if $\operatorname{Re}\lambda > 0$, so it remains to find ϵ' such that $\operatorname{Re}\lambda$ has lower bound $-\frac{1}{2}$ when $\operatorname{Re}\lambda < 0$.

Assume $\lambda \in (-\frac{1}{2}, 0)$. From (8.28) and Lemma 8.19,

$$\epsilon = |\lambda + 1|^2 \left| \frac{\lambda}{\lambda + 1} \right|^N < |\lambda + 1|^2. \tag{B.32}$$

We require $\epsilon < |\lambda + 1|^2$ for all λ, i.e., $\epsilon < \inf |\lambda + 1|^2 = \epsilon'$. For $\lambda \in (-\frac{1}{2}, 0)$ it follows that $|\lambda + 1|^2 \in (\frac{1}{4}, 1+\lambda_y^2)$, so $\epsilon' = \frac{1}{4}$. Thus, $|\lambda|$ converges to zero for $\epsilon < \frac{1}{4}$. □

B.8 Proof of Theorem 8.23

Proof of Theorem 8.23. The characteristic polynomial (8.27) of $A_N|_{\epsilon,N}$ has $N - j$ zero roots and j roots from the characteristic polynomial (8.25) of $A_j|_{\epsilon,j}$. The eigenvalues of $A_j|_{\epsilon,j}$ are distinct with corresponding Jordan block $\operatorname{diag}(\Lambda_j|_{\epsilon,j})$. Since there are $N - j$ zero roots, eigenvalue zero has algebraic multiplicity $N - j$. The geometric multiplicity, or number of Jordan blocks, for zero equals the dimension of the null space of $A_N|_{\epsilon,j}$, which is one because there is one free variable in system $A_N|_{\epsilon,j}x = b$, where $x, b \in \mathbb{R}^N$. Therefore, the Jordan block for eigenvalue zero is H_{N-j}, yielding (8.31). □

B.9 Proof of Lemma 8.24

Proof of Lemma 8.24. We prove by induction. For $k = 1$, $v_{N-1} = \frac{1}{\lambda} v_N$ by (8.32), which equals $\frac{1}{\lambda}\left(1 + \frac{1}{\lambda}\right)^0 v_N$, so the base case holds.

Assume (8.34) holds and compute $\sum_{l=N-(k+1)}^{N} v_l$:

$$\sum_{l=N-(k+1)}^{N} v_l = v_{N-k-1} + \sum_{l=N-k}^{N} v_l \tag{B.33}$$

$$= \frac{1}{\lambda} \sum_{l=N-k}^{N} v_l + \sum_{l=N-k}^{N} v_l \text{ (by (8.32))} \tag{B.34}$$

$$= \left(1 + \frac{1}{\lambda}\right) \sum_{j=N-k}^{N} v_l \tag{B.35}$$

$$= \left(1 + \frac{1}{\lambda}\right) \left(1 + \frac{1}{\lambda}\right)^{k+1} v_N \text{ (induction)} \tag{B.36}$$

$$= \left(1 + \frac{1}{\lambda}\right)^{k+1} v_N. \tag{B.37}$$

□

Appendix C

Code for Memory Efficient HTCondor Solution

This chapter provides code snippets that illustrate the design decisions discussed in Sections 9.4, 9.5, and 9.6.

```
point_t nodearr[max_nodes];
edge_t edgearr[max_edges];
int degarr[max_nodes];
int adjlist[max_nodes][max_degree*2];
```

Figure C-1: Road network representation. Variables `max_nodes`, `max_edges`, and `max_degree` are set to $|V|$, $|E|$, and the maximum out-degree of the road network, respectively.

```
typedef struct node_t{
  int taxicount, tripcount;
  int taxicount_t[TIMERES];
  int taxicountS_t[TIMERES];
  int taxicountD_t[TIMERES];
  int tripcount_t[TIMERES];
  int tripcountS_t[TIMERES];
  int tripcountD_t[TIMERES];
  double tipfracsum_nt[TIMERES];
  double tipfracsumS_nt[TIMERES];
  double tipfracsumD_nt[TIMERES];
  int passcount_nt[TIMERES];
  int passcountS_nt[TIMERES];
  int passcountD_nt[TIMERES];
} node_t;
```

Figure C-2: Example C structure to extract taxi and trip counts, tips, and number of passengers for pickups ("S"), dropoffs ("D"), and along the Dijkstra paths. Variable `TIMERES` refers to the number of time points to track, e.g., 168 for each hour of the week.

```
for (i=0; i<degarr[v]; i++){
  w = adjlist[v][i*2];
  e = adjlist[v][i*2 + 1];
  currdist = dist[v] + edgearr[e].len;
  if (!visited[w] || currdist < dist[w]){
    dist[w] = currdist;
    endq = (endq + 1) % numnodes;
    insert_by_priority(distQ,Q,currdist,w);
    paths[w] = v;
}}
```

Figure C-3: Dijkstra inner loop at node `v` showing min-priority queue implementation with arrays `Q` and `distQ`.

Appendix D

Signal Extraction Implementation Details

This appendix provides additional details that complement the design decisions in Chapter 9. Section D.1 describes the parallelization of the Dijkstra and statistics computations with HTCondor. Sections D.2 and D.3 describe the step-by-step design decisions that were made for the shortest path computation and behavior extraction implementations.

D.1 Parallelization

The overall workflow is the following:

```
1: function MAIN
2:     SHORTEST PATH COMPUTATION
3:     BEHAVIOR EXTRACTION
4:     AVERAGING
5: end function
```

Lines 2 and 3 are separate high-throughput problems that can be run in HTCondor. The steps taken to run the shortest path computations and behavior extraction as jobs on HTCondor are described here. The requirements specified for job submission, the number of jobs to run per machine, and handling input and output are also described.

Shortest paths on HTCondor. The first step is to determine the requirements of the cluster machines in an HTCondor ClassAd. Since HTCondor writes the output as a text file to the scheduled

```
Requirements = (Arch == \"INTEL\"
                || Arch == \"X86_64\")
                && OpSys == \"LINUX\"
Request_Memory = 3G
```

Figure D-1: Example HTCondor ClassAd for shortest path computation.

machine, machines with enough physical memory to store uncompressed output are required. Assuming that a job computes 500,000 shortest paths, the output file requires at most 3GB of physical memory. In addition, machines that are either 32-bit or 64-bit with a Linux operating system are requested. A sample ClassAd is shown in Figure D-1.

Each HTCondor job must be relatively fast so that each job can compute as few shortest paths as possible. The cluster has 32 machines that are either 16-core, 16GB RAM or 8-core, 8GB RAM, so 300 to 500 cores are available at a time depending on user demand and the machines that are allocated by HTCondor. However, since the HTCondor output is written to the submit machine, the output files need to be compressed and transferred to a local machine. The submit and local machines both have limited physical memory, so it is not possible to wait for 700 million shortest paths before compressing the data. At the same time, many files cannot be compressed simultaneously, since the process consumes RAM and slows down the cluster. In other words, HTCondor cannot write output when a large number of jobs ends at the same time.

To account for these issues, each job runs about 500,000 shortest paths, which takes about an hour (see Table 9.2). The total computation for 700 million taxi trips requires 1,500 jobs. The submit machine can compress 30 CSV files of size 3GB simultaneously without too much weight on the system, so shell scripts were written to handle thread synchronization for job completion, output compression, and output transferral. After half of the original jobs are complete, another set of 30 jobs is added to the batch queue.

Behavior extraction on HTCondor. The `node_t` struct shown in Figure C-2 takes about 2GB of memory assuming 64-bit (see Section 9.4). The ClassAd of Figure D-1 includes this memory requirement.

The behavior extraction runs over the shortest path files, so the jobs are allocated to match the shortest path output files. Each job takes about 10-15 minutes to run, so a new batch of jobs can be submitted every 10 minutes.

The design considerations that were made for implementing Dijkstra's algorithm and behavior extraction in C are described next. Section D.2 presents the pre-processing solution. Section D.3 presents the post-processing solution.

D.2 Pre-Processing Implementation

The pre-processing workflow is as follows:

```
1: function SHORTEST PATH COMPUTATION
2:     Define NYC map.
3:     Load NYC road network.
4:     Open taxi data filestream.
5:     for each line in taxi data do
6:         Check for errors.
7:         Match coordinates to map.
8:         Compute shortest path and write to file.
9:     end for
10: end function
```

These steps are discussed in more detail below.

Defining the New York City map. The New York City geography is represented as a union of rectangular bounding boxes defined by the top-left, top-right, bottom-left, and bottom-right coordinates. Other geography representations include ESRI Shapefiles [132], which have the benefit of providing more detailed models for geographic boundaries such as coastlines and city boundaries. For each line of taxi data, it is necessary to verify whether the start and end coordinates are contained in New York City. Since this check is a frequent operation, the rough approximation provided by an array of bounding boxes is ideal for our case.

Loading the road network. The road network must be loaded at runtime to compute the shortest paths for our implementation; also see Section 9.4 for alternative methods.

Reading the taxi data. The 2010-2013 NYC taxi data consists of 16.7GB of compressed CSV files. While this size is small enough to store on a hard drive, it is large enough that the data cannot be stored in RAM at runtime since the cluster machines have 8 GB and 16 GB RAM.

Handling data errors. One important issue is the presence of errors in the NYC taxi data. These errors appear as invalid geo-coordinates and timestamps as well as invalid trip distances and durations. For example, certain geo-coordinates lie in the middle of Hudson River or on top of the Empire State building. A set of criteria is developed to determine whether a trip is spurious or not. This ensures that the extracted statistics are not spurious themselves.

The error checking works as follows. First, it is verified whether the start and end coordinates of a trip are contained in the New York City bounding boxes as defined above. Trips with geo-locations that lie outside these boxes are discarded. Other GPS errors that are detected include zero geo-coordinates and trip distances that are reported as less than the Euclidean distance between the start and end points. Trips of duration less than a minute are also discarded, as well as trips of distance less than 0.2 miles, since the resolution of the road network representation is such that each road segment is at least 0.2 miles long.

If both coordinates of a taxi trip satisfy the error-checking conditions, they are mapped to the road network. This method is described below.

Map matching. The geo-coordinates in the taxi data are mapped to the road network coordinates and stored in `nodearr` as shown in Figure C-1. The map-matching technique known as perpendicular mapping, or nearest-point estimation, is implemented; other map-matching methods are discussed in [133, 134, 135].

This map-matching method finds the "closest" point in the road network by computing the orthogonal distance to the road segments in the road network. The initial step involves finding a small subset $V_s \subset V$ of nodes with the same latitude as the given coordinate; using binary search on the sorted node array, this takes $O(\log |V|)$ time.

The next step is to compute the orthogonal distance from the geo-coordinate to each road segment that has at least one endpoint contained in the node subset; i.e., the distance is computed for all $(v_i, v_j) \in E$ such that $v_i \in V_s$ or $v_j \in V_s$. If (v_i^*, v_j^*) is the road segment that minimizes the distance, then the coordinate from the taxi data is mapped to the closest endpoint. Map matching is done for both the pick-up coordinate and the drop-off coordinate.

In some cases, the distance between the original coordinate and the matched coordinate is very large. The trip is discarded if this distance is greater than 0.2 miles for either the start or the end coordinate.

D.3 Post-Processing Implementation

In this section, the steps taken to implement behavior extraction of the NYC taxi data is described. The overall method is as follows:

```
1: function BEHAVIOR EXTRACTION
2:     Update node_t.
3:     Define NYC map.
4:     Load NYC road network.
5:     Open taxi data and shortest path filestreams.
6:     for each line in taxi data do
7:         Check for errors.
8:         Match coordinates to map.
```

```
9:         Sync line to shortest path data.
10:        Compute statistics and write to file.
11:     end for
12: end function
```

These steps are described below.

Updating `node_t`. The node struct is updated to include statistics of interest, such as total number of trips, total number of passenger, and total fare paid. These statistics are defined as arrays with length corresponding to the time resolution we desire. Weekly averages are computed and stored in 168-element arrays such that index 0 corresponds to Sunday 12am-1am, index 1 corresponds to Sunday 1am-2am, index 24 corresponds to Monday 12am-1am, etc.

Algorithm. For each valid trip, the data fields of interest are extracted. Then, for each node on the shortest path, the fields in the corresponding `node_t` struct are updated with the values from the taxi data. For a data-specific value such as total fare, the data value is extracted and used to update the corresponding node struct value. For updating a trip count, the corresponding counter in the node struct is incremented. These operations are constant-time but with a large constant because of the large number of paths to process.

It is straightforward to modify the operations to investigate other taxi behaviors as well. The simplicity of these update operations is important in order to extract statistics quickly. More complex statistics can then be computed in an analysis stage.

Writing output and computing averages. The output is written to a CSV file through HTCondor as described in Section 9.6. Converting the totals to average statistics requires one more pass over the files containing the statistics. The time to solution is described in Section 9.7.

Our post-processing step is easy to modify for research purposes. For example, the elements in the `node_t` struct can be updated to track different taxi behaviors. Furthermore, the set of operations in the statistics computation algorithm can be updated to handle more complex metrics. This step is designed to be streamlined for fast behavior extraction to enable analysis of taxi movement.