University of California
Santa Barbara

Mining and Modeling Processes on Graphs

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Arlei Lopes da Silva

Committee in charge:

Professor Ambuj Singh, Chair
Professor Xifeng Yan
Professor Subhash Suri
Ananthram Swami, PhD, ARL

September 2019

The Dissertation of Arlei Lopes da Silva is approved.

Professor Xifeng Yan

Professor Subhash Suri

Ananthram Swami, PhD, ARL

Professor Ambuj Singh, Committee Chair

August 2018

Mining and Modeling Processes on Graphs

To my family for their love and support.

Acknowledgements

I would like to thank my advisor Ambuj K. Singh for supporting my research and allowing me to learn so much as part of his group. Thanks also to Xifeng Yan, Subhash Suri and Ananthram Swami for being part of my committee, contributing to the improvement of this work and providing valuable mentorship. As a student at UCSB, I have learned and been inspired by Teofilo Gonzalez and John Gilbert. Earlier, Wagner Meira Jr. and Mohammed Zaki were the ones who motivated me to apply for a PhD.

During my PhD studies I was fortunate to have collaborated with Xuan-Hong Dang, Petko Bogdanov, Prithwish Basu, Gowtham Belalla, Mert Kosan, Zexi Huan, Daniel Hsu, Meredith Yang, Ramya Raghavendra, Mudhakar Srivatsa, Pranay Anchuri, Bo Zong, Jianwu Xu, and Haifeng Chen. I also had a great group of labmates: Hongyuan, Haraldur, Alex, Rachel, Omid, Victor, Yuanshun, Furkan, Minh, Anh, Nazli, Erdem, Yuning, Kyoungmin (Amy), and Wei. My special thanks goes to Dr. Sourav Medya. It is hard to imagine how my PhD experience would have been without him as a labmate, collaborator and friend. Thanks also to Greta, Tim, and Karen from the UCSB CS staff.

Being far from my Brazilian family, I was wholeheartedly adopted by the Bengali family of Santa Barbara: Soupitak, Swati, Ranajay, Pritam, Rima, Shoron, Alam, Soumy, Swagata, Kartick, Anish, Tahmid, Anindya, Amrita, Tanmoy, Anchal, Nadira, Chandi, Sranbanti, Souradeep, Tuktuky, Ranita, Shantonu, Ayan, Era and Risha. I'm also grateful to the Restore crew—Joel, Ruairi, Anna, Dan, and Tom—for so many fun Saturdays.

I have always received support and encouragement from my family and friends, specially my mother Ivone, my brother Andre, my friend David and my brother Alex, without whom I would never get anywhere close to this.

The research developed during my PhD research has been generously supported by the Army Lab Network Science CTA.

Curriculum Vitæ

Arlei Lopes da Silva

Education

2012-2019	PhD in Computer Science, University of California, Santa Barbara.
2008-2011	MSc in Computer Science, Universidade Federal de Minas Gerais, Belo Horizonte, Minas Gerais, Brazil.
2005-2008	BSc in Computer Science, Universidade Federal de Minas Gerais, Belo Horizonte, Minas Gerais, Brazil.

Publications

Refereed Journals and Conferences:

Arlei Silva, Ambuj Singh, Ananthram Swami. *Spectral Algorithms for Temporal Graph Cuts*. The Web Conference (WWW), 2018.

Sourav Medya, Arlei Silva, Ambuj Singh, Prithwish Basu, Ananthram Swami. *Group Centrality Maximization via Network Design*. SIAM International Conference on Data Mining (SDM), 2018.

Arlei Silva, Gowtham Bellala. *Privacy-Preserving Multi-party Clustering: An Empirical Study*. IEEE International Conference on Cloud Computing (CLOUD), 2017.

Arlei Silva, Xuan-Hong Dang, Prithwish Basu, Ambuj Singh, Ananthram Swami. *Graph Wavelets via Sparse Cuts*. ACM Conference on Knowledge Discovery and Data Mining (KDD), 2016.

Xuan-Hong Dang, Arlei Silva, Prithwish Basu, Ambuj Singh, Ananthram Swami. *Outlier Detection from Network Data with Subnetwork Interpretation*. IEEE International Conference on Data Mining (ICDM), 2016.

Arlei Silva, Petko Bogdanov, Ambuj Singh. *Hierarchical In-Network Attribute Compression via Importance Sampling*. IEEE International Conference on Data Engineering (ICDE), 2015.

Arlei Silva, Sara Guimaraes, Wagner Meira Jr, Mohammed Zaki. *ProfileRank: Finding Relevant Content and Influential Users based on Information Diffusion*. ACM International Workshop on Social Network Mining and Analysis (SNAKDD), 2013.

Arlei Silva, Wagner Meira Jr., Mohammed J. Zaki. *Mining Attribute-structure Correlated Patterns in Large Attributed Graphs*. Proceedings of the VLDB Endowment (PVLDB), 2012.

Mehdi Kaytoue, Arlei Silva, Loic Cerf, Wagner Meira Jr., Chedy Raissi. *Watch me playing, i am a professional: a first study on video game live streaming*. ACM International Workshop on Mining Social Network Dynamics (MSND), 2012.

Working Papers:

Arlei Silva, Pranay Anchuri, Jianwu Su, Bo Zong, Haifeng Chen, Ambuj Singh. *Learning interleaved markov processes*, 2019.

Sourav Medya, Tianyi Ma, Arlei Silva, Ambuj Singh. *K-core Resilience: A Game Theoretic Approach*, 2018.

Sourav Medya, Arlei Silva, Ambuj Singh. *Influence Minimization under Budget and Matroid Constraints*, 2018.

Abstract

Mining and Modeling Processes on Graphs

by

Arlei Lopes da Silva

Graphs are a powerful tool for the study of dynamic processes, where a set of interconnected entities change their states according to the time-varying behavior of an underlying complex system. For instance, in a social network, an individual's opinions are influenced by their contacts; while, in a traffic network, traffic conditions are spatially localized due to the fact that vehicles are often constrained to move along roads. Understanding the interplay between structure and dynamics in networked systems enables new models, algorithms, and data structures for managing and learning from large amounts of data arising from these processes.

This dissertation is focused on recent work on the analysis of dynamic graph processes. More specifically, we will show how sampling and spectral graph theory can be applied to effectively represent data from such processes. We also present efficient algorithms for learning Interleaved Markov Models (IHMMs). These are powerful latent variable models that enable the clustering of discrete sequences without training data and lead to interesting challenges in terms of inference. Furthermore, we also discuss how graphs can be modified in order to control a process of interest via network design and introduce three ongoing projects on the subject of this dissertation. The statement of the thesis is that mining and modeling processes on graphs leads often to problems that are not not only hard computationally but also in terms of inference. They can be solved using spectral, probabilistic, and combinatorial optimization algorithms, and must take into account the graph structure and also large amounts of data traces from these processes.

Permissions and Attributions

This dissertation contains material that has been published or is in the process of being published. The author of this dissertation claims major contributions in the development of the research works described below:

1. The content of Chapter 2 has been previously published as: Arlei Silva, Petko Bogdanov, Ambuj Singh. *Hierarchical In-Network Attribute Compression via Importance Sampling.* IEEE International Conference on Data Engineering (ICDE), 2015. DOI: 10.1109/ICDE.2015.7113347.

2. The content of Chapter 3 has been previously published as: Arlei Silva, Xuan-Hong Dang, Prithwish Basu, Ambuj Singh, Ananthram Swami. *Graph Wavelets via Sparse Cuts.* ACM Conference on Knowledge Discovery and Data Mining (KDD), 2016. DOI: 10.1145/2939672.2939777.

3. The content of Chapter 4 has been previously published as: Arlei Silva, Ambuj Singh, Ananthram Swami. *Spectral Algorithms for Temporal Graph Cuts.* The Web Conference (WWW), 2018. DOI: 10.1145/3178876.3186118.

4. The content of Section 6.1 has been previously published as: Sourav Medya, Arlei Silva, Ambuj Singh, Prithwish Basu, Ananthram Swami. *Group Centrality Maximization via Network Design.* SIAM International Conference on Data Mining (SDM), 2018. DOI: 10.1137/1.9781611975321.14.

5. The content of Section 6.2 has been previously published as: Xuan-Hong Dang, Arlei Silva, Prithwish Basu, Ambuj Singh, Ananthram Swami. *Outlier Detection from Network Data with Subnetwork Interpretation.* IEEE International Conference on Data Mining (ICDM), 2016. DOI: 10.1109/ICDM.2016.0101.

For ACM, authors can include partial or complete papers of their own in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are

included. If interested in reprinting or republishing ACM copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please contact the ACM.

For IEEE, requirements to be followed when using an entire IEEE copyrighted paper in a thesis: 1) The following IEEE copyright/ credit notice should be placed prominently in the references: [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]; 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line; 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/rights_link.html` to learn how to obtain a License from Rights Link.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A common characteristic of the problems investigated in this dissertation is that they are motivated by real applications and the availability of real data. Recently, an incipient field, called *Data Science*, has emerged as an umbrella for related research topics. Data Science is at the intersection between computer science, statistics, and application domain knowledge. According to David Donoho, a Statistician at Stanford: *"Data Science concerns the recognition, formalization and exploitation of data phenomenology emerging from the digital transformation of business, society, and science."*[1] Examples of this transformation include advances in hardware, IoT and cloud computing in addition to the popularization of the Web, social media and smartphones.

Graphs, or *networks*, are a powerful theoretical framework for modeling complex data. They are a common language in which different communities can communicate problems and their solutions. As a consequence, a significant part of this dissertation is on the design and application of graph algorithms. For instance, how can we identify attributes (eg., age, gender) that explain the rise of communities in society [1]? Or how to compress vehicle traffic data over time [2]? Or, given a large system log, how to break apart traces

[1]David Donoho. Data Science: The End of Theory? Talk at Univ. Wien.

of different components running concurrently in the system [3]? These problems can not only be described in the language of graphs but also be solved based on fundamental results from graph theory and algorithms.

Another emerging area also related to the topic of this dissertation is *Network Science*. The general paradigm of network science is that networks (or graphs) arising from different applications unexpectedly share many interesting properties. Thus, the motivation to create a science that studies networks. Newman et al.[4] emphasizes the following properties of network science: (1) Focuses on properties of real networks; (2) takes the view that networks evolve; and (3) applies networks as a framework for dynamical systems.

Most of the problems discussed in this dissertation assume a graph as a space in which a process is taking place. Broadly speaking, a process is a sequence of events that changes the state of a system. In the case of graph processes, event effects are correlated for some pairs of variables and these correlations can be represented as a graph. For instance, in a social network, an individual's opinions are influenced by their contacts; while, in a traffic network, traffic conditions are spatially localized due to the fact that vehicles are often constrained to move along roads. Understanding the interplay between structure and dynamics in networked systems enables new models, algorithms, and data structures for managing and learning from large amounts of data arising from these processes.

The main challenges related to mining and modeling graph processes are: (1) The scale and complexity of the system and (2) the limited knowledge of states and events. In particular, real graphs might have billions of vertices and edges representing heterogeneous time-evolving relationships. Moreover, the graph structure might be partially or completely missing, which requires efficient inference of the structure of such processes.

Figure 1.1 shows how the subject of this dissertation relate with a few research areas. In particular, we divide such areas into three groups. *Applications* (Databases, Web, Social Networks) are those areas that we use as a motivation and a source of data for

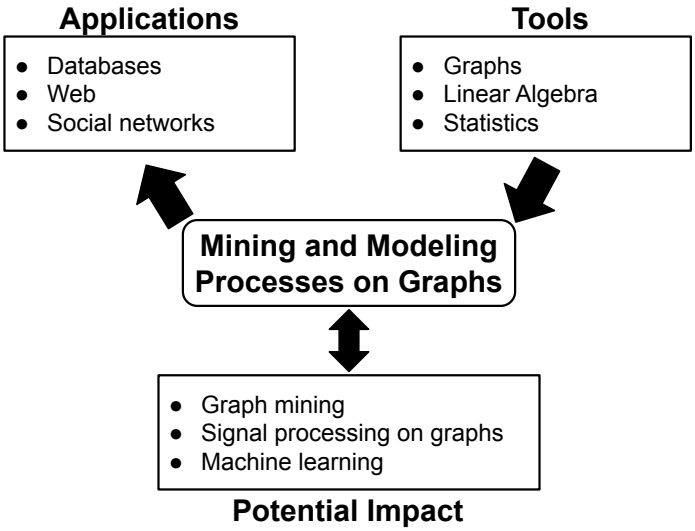


Figure 1.1: Relationship between the topic of this dissertation and other subject areas.

our research. *Tools* (Graphs, Linear Algebra, Statistics) are those areas that provide theoretical background for the solutions we propose. *Potential Impact* (Graph Mining, Signal Processing on Graphs, and Machine Learning) are areas that are the closest to our research and also where we aim to have a direct impact on.

1.1 Contributions, Organization and Thesis Statement

The contributions of this dissertation can be summarized as follows:

1. *Hierarchical In-Network Compression via Importance Sampling [5] (Chapter 2):* Many real-world complex systems can be modeled as dynamic networks with real-valued vertex/edge attributes. Examples include users' opinions in social networks and average speeds in a road system. When managing these large dynamic networks, compressing attribute values becomes a key requirement, since it enables the answering of attribute-based queries regarding a node/edge or network region

based on a compact representation of the data. To address this problem, we introduce a lossy network compression scheme called *Slice Tree (ST)*, which partitions a network into smooth regions with respect to node/edge values and compresses each value as the average of its region. ST applies a compact representation for network regions, called *slices*, that are defined as a center node and radius distance. We propose an importance sampling algorithm to efficiently prune the search space of candidate slices in the ST construction by biasing the sampling process towards the node values that most affect the compression error. The effectiveness of ST in terms of compression error, compression rate, and running time is demonstrated using synthetic and real datasets. ST scales to million-node instances and removes up to 87% of the error in attribute values with a 10^3 compression ratio. We also illustrate how ST captures relevant phenomena in real networks, such as research collaboration patterns and traffic congestions.

2. *Graph Wavelets via Sparse Cuts [2] (Chapter 3):* Modeling information that resides on vertices of large graphs is a key problem in several real-life applications, ranging from social networks to the Internet-of-things. Signal Processing on Graphs and, in particular, graph wavelets can exploit the intrinsic smoothness of these datasets in order to represent them in a compact and accurate manner. However, how to discover wavelet bases that capture the geometry of the data with respect to the signal as well as the graph structure remains an open problem. In this paper, we study the problem of computing graph wavelet bases via sparse cuts in order to produce low-dimensional encodings of data-driven bases. This problem is connected to known hard problems in graph theory (e.g. multiway cuts) and thus requires an efficient heuristic. We formulate the basis discovery task as a relaxation of a vector optimization problem, which leads to an elegant solution as a regularized eigenvalue

computation. Moreover, we propose several strategies in order to scale our algorithm to large graphs. Experimental results show that the proposed algorithm can effectively encode both the graph structure and signal, producing compressed and accurate representations for vertex values in a wide range of datasets (e.g. sensor and gene networks) and significantly outperforming the best baseline.

3. *Spectral Algorithms for Temporal Graph Cuts [6] (Chapter 4):* The sparsest cut problem consists of identifying a small set of edges that breaks the graph into balanced sets of vertices. The normalized cut problem balances the total degree, instead of the size, of the resulting sets. Applications of graph cuts include community detection and computer vision. However, cut problems were originally proposed for static graphs, an assumption that does not hold in many modern applications where graphs are highly dynamic. In this paper, we introduce sparsest and normalized cuts in temporal graphs, which generalize their standard definitions by enforcing the smoothness of cuts over time. We propose novel formulations and algorithms for computing temporal cuts using spectral graph theory, divide-and-conquer and low-rank matrix approximation. Furthermore, we extend temporal cuts to dynamic graph signals, where vertices have attributes. Experiments show that our solutions are accurate and scalable, enabling the discovery of dynamic communities and the analysis of dynamic graph processes.

4. *Learning Interleaved Hidden Markov Models [3] (Chapter 5):* Interleaved hidden Markov models (IHMMs) extend the classical hidden Markov models (HMMs) by simulating multiple HMMs that concurrently produce a sequence of observations. Our investigates the problem of learning the parameters of IHMMs from interleaved data. Exact inference for IHMMs is intractable, and thus we focus on approximate inference via Loopy Belief Propagation (LPB).

5. *Other Problems (Chapter 6):* Here, we grouped problems that, although related, are not part of the core idea of this dissertation.

   (a) *Network Design [7, 8, 9] (Section 6.1):* Network design is a recent area of study focused on modifying or redesigning a network in order to achieve a desired property. As networks become a popular framework for modeling complex systems (e.g. VLSI, transportation, communication), network design provides key controlling capabilities over these systems, especially when resources are constrained. As part of this dissertation, we have investigated three network design problems: (1) How to maximize the centrality of a target set of users in a social network [7]; (2) how to minimize the size of the k-core of a network [8]; and (3) how to minimize the influence of a target set of users in a social network [9]. In all these optimization problems, we assume that the number of network (edge) modifications is bounded by a budget. We have shown that these problems are NP-hard, and some of them are hard to approximate even by a constant. Moreover, we have proposed efficient algorithms for these problems, showing that they outperform their respective baselines in real datasets.

   (b) *Outlier Detection Outlier from Network Data with Subnetwork Interpretation [10] (Section 6.2):* Detecting a small number of outliers from a set of data observations is always challenging. This problem is more difficult in the setting of multiple network samples, where computing the anomalous degree of a network sample is generally not sufficient. In fact, explaining why a given network is exceptional, expressed in the form of subnetwork, is also equally important. We develop a novel algorithm to address these two key problems. We treat each network sample as a potential outlier and identify subnetworks that help discriminate it from nearby samples. The algorithm is developed in

the framework of network regression combined with the constraints on both network topology and L1-norm shrinkage to perform subnetwork discovery. Our method thus goes beyond subspace/subgraph discovery. We also show that the developed method converges to a global optimum. Empirical evaluation on various real-world network datasets demonstrates the advantages of our algorithm over various baseline methods.

Statement of the thesis: Mining and modeling processes on graphs leads often to problems that are not not only hard computationally but also in terms of inference. They can be solved using spectral, probabilistic, and combinatorial optimization algorithms, and must take into account the graph structure and also large amounts of data traces from these processes.

Chapter 2

Hierarchical In-Network Compression via Importance Sampling

2.1 Introduction

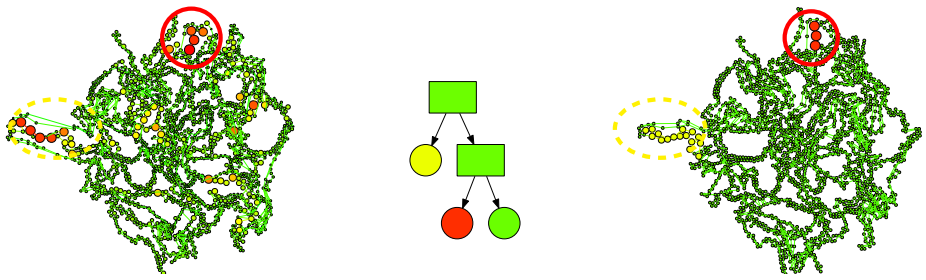
Managing large dynamic networks that represent data-rich complex systems is a challenge in terms of processing time, communication, and storage. In social networks, for instance, users' opinions regarding different topics along time generate a large amount of data that describes the network opinion dynamics. Similarly, sensors spread over a road network provide massive, real-time data about the traffic conditions across multiple regions (see Figure 2.1). In these scenarios, each node of the network has an attribute (e.g., opinion, speed) and these attributes describe properties of a subnetwork (e.g. a community's opinion on a topic) and the overall network, such as major traffic jams affecting several neighborhoods of a city. Therefore, compressing vertex/edge attributes is a fundamental problem in the management of these large dynamic networks.

An important step towards compressing real-world network attributes is understanding the processes that generate attribute values. In social networks, opinion dynamics is affected by word-of-mouth dissemination [11]. Similarly, bad traffic conditions in a given location are likely to affect nearby locations. These are examples of network processes, which change node/edge values via connections in the network [12]. Network processes arise in many domains, thus we can exploit them to better compress network attributes.

We can view network attributes as a signal over the network, where each node/edge has a value. Signals generated by network processes are expected to be smooth with respect to the network structure (i.e. values are similar at neighboring nodes) [13]. As a consequence, attributes can be expressed in terms of smooth regions, which are network regions with similar values. Examples of smooth regions in real networks include communities whose members share a similar opinion and neighborhoods affected by the same traffic conditions. In this work, we study the problem of compressing network attributes by decomposing the network into smooth regions.

While there is a rich body of research on compressing network topologies [14, 15], compressing network attributes is an emerging problem [13]. In a traffic network, for instance, the structure is mostly fixed but traffic conditions change along time due to seasonal patterns (e.g. rush hours) and events (e.g. accidents). In this context, the compressed data provides, in a compact form, key information for answering queries regarding the traffic conditions in a particular location or region. In a hypothetical case where traffic conditions are random with respect to the network, accurately answering such queries would require keeping a large fraction of node values. However, if conditions are smooth over the structure, we can exploit such locality and represent network attributes in terms of average values of few regions (e.g., streets, neighborhoods).

We propose a network compression strategy called *Slice Tree (ST)*, which is a hierarchical decomposition of a snapshot of the network in terms of smooth regions. ST



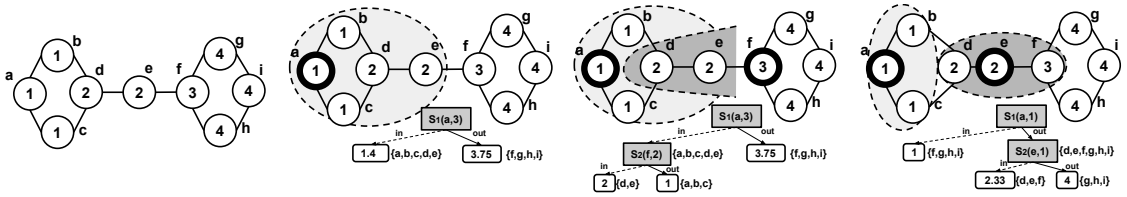
(a) Traffic network (b) Slice tree (c) Reconstructed state values

Figure 2.1: Slice Tree (ST) compression of a real traffic network (a). We set colors and sizes of nodes according to the average speed in the locations (large/red: very slow, medium/yellow: slow, small/green: fast). The attributes (speeds) are compressed as an ST with 2 slices (b) that decomposes the network into three smooth regions. The reconstruction of the original speed values (c) demonstrates that the compression captures the two major congestions in the network.

regions are represented in a compact form as a *center* and *radius*. A *slice* partitions a set of nodes into two regions: one composed of nodes within *radius* distance from a *center* node and the other containing the remaining nodes. An ST is constructed by recursive applications of such operations. Leaves of an ST define regions whose values are approximated as their average. To enable the reconstruction of regions, ST intermediate nodes retain information about each slice applied in the decomposition. Since ST is a lossy compression, we compute its error as the sum of squared errors (SSE) of the recovered values with respect to the original node values in the network.

Figure 2.1 shows the ST compression of a real traffic network (Figure 2.1a), where node values represent average speeds. Node colors and sizes emphasize low-speed locations. An ST (Figure 2.1b) decomposes the network into two congested regions and one non-congested region. Reconstructed speed values from the ST (Figure 2.1c) show that it captures the two major congestions (see more details in Section 2.6.4).

Figure 2.2 illustrates the ST compression of an example network (Figure 2.2a) with one (Figure 2.2b) and two slices (Figures 2.2c and 2.2d). We define the error of the network (14.22) w.r.t. its average (2.44). The slice S_1 (Figure 2.2b), which is centered



(a) Network, $SSE = 14.22$ (b) Slice 1, $SSE = 1.95$ (c) Slice 2, $SSE = 0.75$ (d) Opt. ST, $SSE = 0.65$

Figure 2.2: An illustrative example of the ST compression. A network with node values and error (SSE) w.r.t. the average (a). First (b) and second (c) slice STs with respective errors. A slice divides nodes into two regions, one with nodes within *radius* distance from a *center* node and other with the remaining nodes. STs are constructed by choosing slices recursively and leaves of the ST define regions. Node values are compressed as the average of their region. Optimal ST with 2 slices (d).

at node a and has radius 3, separates the nodes into regions $\{a, b, c, d, e\}$ and $\{f, g, h, i\}$ with averages 1.4 and 3.75, respectively. The error of the resulting ST (1.9) is the total SSE of the region values w.r.t. the original values. The ST given in Figure 2.2b can be extended by a new slice S_2 (center f and radius 2) that separates $\{a, b, c, d, e\}$ into two new regions $\{d, e\}$ (average 2) and $\{a, b, c\}$ (average 1). Figure 2.2d shows an alternative ST with 2 slices which is optimal, i.e. it minimizes the SSE of the region values.

Computing an optimal ST with k slices is an NP-hard problem. We propose a greedy scheme for ST construction which recursively selects the next slice that minimizes the compression error. While this solution achieves good results in practice, it is prohibitive for large networks due to the cost of searching over the set of possible slices. Therefore, we devise an efficient probabilistic approximation algorithm that prunes suboptimal slices with theoretical guarantees. In particular, we design an *importance sampling* strategy that biases the sampling process towards nodes for which the values are more likely to affect the error of the compression. We show that importance sampling requires less samples than uniform sampling for computing slices with the same error guarantees. Our main contributions are outlined as follows:

- We introduce Slice Tree as a novel network attribute compression strategy, show that computing an optimal ST, which minimizes the compression error, is NP-hard and present a greedy heuristic for ST construction.

- We propose an importance sampling algorithm, with associated approximation guarantees, which enables the application of ST to million-node networks by effectively pruning the search space of candidate slices.

- We show that ST is an effective compression scheme, removing up to 87% of the error in node values with a 10^3 compression ratio. Case studies illustrate how STs model relevant properties of real networks, such as collaboration patterns and traffic congestions.

2.2 Problem Definition

We define a network as a triple $G(V, E, W)$, where V and E are the sets of vertices and edges, while W is a function over the vertices ($W : V \to \mathbb{R}$) that gives the vertex attributes. The value of a vertex v is denoted as $w(v)$ and $d(u, v)$ is the shortest path distance between u and v. The network compression problem is posed as follows.

Definition 1 Network attribute compression. *Given a budget b and a network $G(V, E, W)$, compute a compression $\Gamma : V \to \mathbb{R}$, such that Γ can be encoded using b bits and $\Gamma(v)$ gives an approximate value for $w(v)$.*

The function Γ is a lossy compression of W. We quantify the compression error of Γ as a sum of squared errors: $\sum_{v \in V}(w(v) - \Gamma(v))^2$. A good compression recovers the values given by W with small error. Moreover, Γ must be computed efficiently for large networks. While we define the problem in terms of SSE minimization and node

attributes, our general framework can be adapted to other error metrics including those based on edge attributes. Also, we focus on the problem of compressing a single snapshot of the network at a time.

2.3 Slice Tree

Our approach for the network attribute compression problem is Slice Tree (ST), which decomposes a fixed network structure into hierarchical regular regions that are smooth with respect to the attribute values. ST encodes information for reconstructing the regions and their averages, thus providing a compact representation for attribute values based on the network structure.

A slice divides a set of nodes into two subsets, one containing nodes within radius distance from a center node and the other composed of the remaining nodes in the set.

Definition 2 Slice. *Given a network G, nodes $X \subseteq V$, a center $c \in V$, and a radius $r \geq 0$, a slice $s(c, r, X)$ partitions X into $P = \{u \in X | d(c, u) \leq r\}$ and $X \setminus P$.*

Slices are compact representations for regular partitions. A slice encodes a separation of a set of nodes X into two subsets (P and $X \setminus P$) using only two parameters (center and radius). Figure 2.2b, shows a slice $S_1(a, 3, V)$ applied to the set of nodes from the network shown in Figure 2.2a. New slices can be applied to the resulting partitions P and $X \setminus P$, producing a hierarchical data structure. The sequence of slices is a compressed representation for the final regions in the ST.

Definition 3 Slice Tree (ST). *A slice tree $ST(G, k)$ is a binary tree that encodes k recursive slices in G. The first slice is applied to V and subsequent slices are applied (recursively) to the resulting partitions (P and $X \setminus P$).*

Leaves of an ST define partitions (or regions) in the network and the representation cost of an ST is linear in the number of slices k. To compress the network attribute values W, we assign to each leaf node of the ST a value $\mu(P)$ that is the average value of nodes in P ($\mu(P) = 1/||P|| \sum_{v \in P} w(v)$). Values of nodes $v \in P_i$ are compressed as $\mu(P_i)$.

An ST with two slices (i.e. $k = 2$) for the network from Figure 2.2a is shown in Figure 2.2c. The partitions $\{a, b, c\}$, $\{d, e\}$, and $\{f, g, h, i\}$ are produced by the slices S_1 and S_2. We compress the value $w(f)$ as 3.75, since this is the average $\mu(\{f, g, h, i\})$. The error of an ST is computed as a sum of squared errors with respect to the original values. For instance, the error of the ST in Figure 2.2c is 0.75. An optimal Slice Tree $ST^*(G, k)$ minimizes the compression error $\sum_{P \in \{P_1, \ldots, P_{k+1}\}} \sum_{v \in P} (w(v) - \mu(P))^2$. Figure 2.2d shows an optimal ST (SSE= 0.65) with two slices for the network shown in Figure 2.2a. Computing $ST^*(G, k)$ is NP-hard.

Theorem 1 Optimal Slice Tree. *The slice tree compression problem is NP-hard.*

Proof: Let $VC(\mathcal{G}(\mathcal{V}, \mathcal{E}), q)$ be an instance of the (NP-complete) *Vertex Cover* problem, which asks whether there exists a set of vertices $\mathcal{V}' \in \mathcal{V}$ such that $|\mathcal{V}'| = q$ and, for each edge $(u, v) \in \mathcal{E}$, $u \in \mathcal{V}'$ or $v \in \mathcal{V}'$ (or both). There is a corresponding instance $ST(G(V, E, W), 2q)$ of the ST problem, where G is a 4-partite graph defined as follows. The set of nodes is $V = V^1 \cup V^2 \cup V^3 \cup V^4$, where for each node $v_i \in \mathcal{V}$ we create nodes $v_i^1 \in V^1$ and $v_i^2 \in V^2$, and for each edge $e_j \in \mathcal{E}$ we create nodes $e_{j,1}^3, e_{j,2}^3 \in V^3$ and $e_{j,1}^4, e_{j,2}^4 \in V^4$. The set E contains edges (v_i^1, v_i^2) for all nodes $v_i \in \mathcal{V}$, $(e_{j,1}^3, e_{j,1}^4)$ and $(e_{j,2}^3, e_{j,2}^4)$ for all edges $e_j \in \mathcal{E}$, and $(v_i^2, e_{j,1}^3)$ for all edges e_j adjacent to node v_i in $\mathcal{G}$. W is such that $w(v_i^h) = 1$ if $v_i^h \in V^3$ and $w(v_i^h) = 0$, otherwise.

We show that $VC(\mathcal{G}, q)$ is true iff $ST(G, 2q)$ has 0-error. Given a vertex cover $\mathcal{V}'$, we generate a 0-error ST by placing a slice $s(v_i^1, 2, X_i)$ followed by a slice $s(v_i^1, 1, X_i')$ for every $v_i \in \mathcal{V}'$. The order in which these slices are placed and the sets X_i and X_i'

involved do not matter. The resulting ST isolates nodes in V_3 from the remaining ones. Conversely, given a 0-error ST, we can generate a vertex cover by first applying some simple transformations to the ST and then selecting each vertex $v_i \in \mathcal{V}$ for which there is a slice $s(v_i^1, 2, X)$ as part of the cover $\mathcal{V}'$. These transformations enforce all slices in the ST to be centered at nodes in V^1 and the cases to be considered are slices centered at nodes in V^2 and V^3. Slices centered in V^2 with radius 1 can be replaced by slices centered in V^1 with radius 2 for an equivalent solution. Moreover, each pair of slices separating nodes $e_{j,1}^3$ and $e_{j,2}^3$ from the rest are replaced by two slices with any center v_i^1 such that e_j is adjacent to v_i, one with radius 2 followed by another with radius 1. ■

Theorem 1 shows that finding the best ST from G with k slices might require searching over an $O(d(G)||V||^k)$ space of candidate STs, where $d(G)$ is the diameter of G. Candidate STs combine possible choices of centers and radii and are order-sensitive. In the rest of this work, we devise efficient heuristics for computing accurate STs in large networks.

2.4 Fast Slice Tree Compression

We outline a greedy procedure that computes an ST by repeatedly selecting the slice that minimizes the compression error. To scale this solution to large networks, we introduce a probabilistic approximation algorithm that applies sampling and pruning in the ST construction.

2.4.1 Greedy Slice Tree Construction

While building an optimal ST is a computationally hard problem, a single optimal slice can be found in polynomial time by searching over all possible center-radius combinations. Therefore, a greedy algorithm, which builds an ST by selecting consecutive slices that minimize the compression error, is a natural approach for the ST problem.

We can express the benefit of adding a slice to an ST as follows.

Definition 4 *The* ***error reduction*** $\phi(s)$ *of a slice* $s(c, r, X)$ *is defined as:*

$$\phi(s) = SSE(X) - SSE(P) - SSE(X \setminus P),$$

where P *and* $X \setminus P$ *are regions produced by* s *and* $SSE(Y) = \sum_{v \in Y}(w(v) - \mu(Y))^2$.

An important property of $\phi(s)$ is that it can be computed in terms of the mean values and the sizes of partitions X and P (or $X \setminus P$), as shown in Theorem 2.

Theorem 2 *The error reduction* $\phi(s)$ *can be computed in terms of average values* $\mu(X)$, $\mu(P)$ *and* $\mu(X \setminus P)$*:*

$$\begin{aligned}\phi(s) &= (\mu(X) - \mu(P))^2 \frac{||P||||X||}{||X \setminus P||} \\ &= (\mu(X) - \mu(X \setminus P))^2 \frac{||X \setminus P||||X||}{||P||}\end{aligned}$$

Proof: The SSE of the average $\mu(Y)$ w.r.t. values $y \in Y$ can be expressed as:

$$\sum_{y \in Y}(y - \mu(Y))^2 = \sum_{y \in Y} y^2 - \frac{1}{||Y||}\left(\sum_{y \in Y} y\right)^2$$

Thus,

$$\begin{aligned}
\phi(s) &= \sum_{v\in X} w(v)^2 - \frac{1}{||X||}\left(\sum_{v\in X} w(v)\right)^2 \\
&\quad - \sum_{v\in P} w(v)^2 + \frac{1}{||P||}\left(\sum_{v\in P} w(v)\right)^2 \\
&\quad - \sum_{v\in X\setminus P} w(v)^2 + \frac{1}{||X\setminus P||}\left(\sum_{v\in X\setminus P} w(v)\right)^2 \\
&= -\frac{1}{||X||}\left(\sum_{v\in X} w(v)\right)^2 + \frac{1}{||P||}\left(\sum_{v\in P} w(v)\right)^2 \\
&\quad + \frac{1}{||X\setminus P||}\left(\sum_{v\in X\setminus P} w(v)\right)^2 \\
&= (\mu(X)-\mu(P))^2 \frac{||P||||X||}{||X\setminus P||}
\end{aligned}$$

We derive an expression w.r.t. $\mu(X \setminus P)$ by replacing $\mu(P)$ by $(\mu(X)||X|| - \mu(X \setminus P)||X \setminus P||)/||P||$ in this equation. ■

Intuitively, a good slice produces new regions, P and $X \setminus P$, for which averages, $\mu(P)$ and $\mu(X \setminus P)$, deviate from the average value of X. Theorem 2 supports an efficient approach for computing the error reduction of concentric slices of increasing radii. Moreover, we apply it to compute sampling-based upper bounds on the error reduction of candidate slices, as described in Section 2.4.2. Our greedy algorithm selects, at each step, the slice that maximizes the error reduction.

The *Slice* function (Algorithm 1), identifies the slice t with maximum error reduction $\phi(t)$ for a partition X. It iterates over all possible centers and increasing radii up to a maximum $radius(c, G, X)$ that splits X into non-empty regions. A center-radius pair (c, r) defines a candidate slice s in X (step 3). The error reduction of a candidate slice s is computed according to Theorem 2 (step 4) and the slice that maximizes the error

Require: Network G, set of nodes X
Ensure: Best slice t
1: **for** All centers $c \in X$ **do**
2: **for** All radii $r \in [0, radius(c, G, X)]$ **do**
3: $s \leftarrow (c, r, X)$
4: $\phi(s) \leftarrow (\mu(X) - \mu(P))^2 \frac{||P||||X||}{||X-P||}$
5: $t \leftarrow s$, if $\phi(s)$ is the largest observed reduction
6: **end for**
7: **end for**

Algorithm 1: Slice

Require: Network G, budget k
Ensure: ST
1: Slice $ST \leftarrow$ empty Slice Tree
2: Candidate slices $C \leftarrow \emptyset$
3: Add best slice $s(c, r) \leftarrow Slice(G, V)$ to C
4: **while** Number of slices less than k **do**
5: Retrieve best slice $t(c, r)$ from C
6: Add $t(c, r)$ to ST
7: Let P and $X \setminus P$ be the partitions produced by $t(c, r)$
8: Add best slice $s(c, r) \leftarrow Slice(G, P)$ to C
9: Add best slice $s(c, r) \leftarrow Slice(G, X \setminus P)$ to C
10: **end while**

Algorithm 2: Greedy Slice Tree

reduction is selected (step 5). Iterating over all possible radii for a given center c is equivalent to performing a breadth first search (BFS) starting from c.

Algorithm 2 describes our greedy strategy for computing an ST. It takes an input network G and a budget k and computes a slice tree $ST(G, k)$ by consecutively selecting the next slice with highest error reduction. The greedy ST algorithm proceeds in k iterations. In each iteration (steps 4-10), it selects the slice incurring highest error reduction, removes it from the priority queue C and inserts it in the corresponding branch of the ST (steps 4,5). Next, we compute the best slice for the new regions, P and $X \setminus P$, and add them to C (steps 8-9).

In our running example (Figure 2.2), the first slice S_1 (Figure 2.2b) selected by our

greedy algorithm has center a, radius 3 and an error reduction of $\phi(S_1) = 12.27$ (the error of the network with respect to its global mean $\mu(V)$ is 14.22). The next best slice S_2 (Figure 2.2c) further divides the partition $\{a, b, c, d, e\}$ into $\{a, b, c\}$ and $\{d, e\}$ using center f and radius 3. The error reduction $\phi(S_2)$ of S_2 is 1.25. The following Lemma gives the complexity of our greedy ST algorithm.

Lemma 2.4.1 *The greedy slice tree algorithm (Algorithm 2) runs in time* $O(k||V||||E||)$.

Proof: A slice s over X produces regions X_1 and X_2. Let $E(X) = \{(v_i, v_j) \in E | v_i, v_j \in X\}$. *Slice* applies $||X||$ BFSs, in time $O(||E(X)||)$ each. In the worst case, s generates X_1 and X_2 such that $||X_1|| = ||E(X_1)|| = 1$, $||X_2|| = ||X|| - 1$ and $||E(X_2)|| = ||E(X)|| - 1$, resulting in a complexity:

$$O\left(\sum_{i=0}^{k-1}(||V|| - i)(O(||E|| - 1))\right) = O(k||V||||E||)$$

■

This complexity imposes a challenge to the application of our greedy algorithm to large networks. The ST from Figure 2.2c was built using the greedy algorithm described in this section and we show that it is not optimal. An optimal ST with two slices for the same network is shown in Figure 2.2d. While the proposed algorithm achieves good results in practice (see Section 2.6), whether there is a constant-factor polynomial approximation for the ST problem remains an open question.

2.4.2 Error Reduction via Sampling

The challenge in selecting the next best slice in ST construction stems from the need to explore all the node values at a given radius from a center to compute its error reduction. The idea behind our sampling schemes is that we do not need to observe all nodes within

"good" slices to identify them, but instead can utilize a small sample of nodes. We propose two sampling schemes for computing upper bounds on the error reduction of slices: one based on uniform sampling and one based on importance sampling. The resulting bounds enable a scalable and accurate slice search algorithm.

As part of our *uniform sampling* approach, we generate a uniform node sample U with replacement from a set of nodes X. For a given slice $s(c, r, X)$, we estimate the average values of its regions $\mu(P)$ and $\mu(X \setminus P)$ based on values of nodes in $U \cap P$ and $U \cap (X \setminus P)$, respectively. We employ the Hoeffiding inequality [16] to compute probabilistic errors for these estimates. The average estimates and their errors are then used to obtain probabilistic upper bounds on the error reduction of a slice according to Theorem 2.

In the uniform sampling scheme, nodes are sampled with equal probability regardless of whether they are part of a region with outlier values. Theorem 2 shows that high error reduction slices create regions P and $X \setminus P$ with values deviating from the average $\mu(X)$. To increase the likelihood of nodes from high error reduction regions to be sampled, we apply *importance sampling* [17]. Importance sampling is a statistical technique that biases the sampling procedure towards values that are more relevant for computing a given estimate. In our case, we bias the samples based on how much their values deviate from the average $\mu(X)$.

Definition 5 *An **importance sample** from X is a multiset of nodes B where each node $v \in X$ is selected (with replacement) with probability:*

$$p(v) = \frac{|w(v) - \mu(X)|}{\lambda}$$

where $\lambda = \sum_{v \in X} |w(v) - \mu(X)|$

The importance sample B is biased and hence, in order to obtain unbiased estimates

for the average values in resulting regions $\mu(P)$ and $\mu(X \setminus P)$ we need to correct the sampled attribute values using a weighted average.

Definition 6 *The **weighted average** $\mu(B_P)$ is defined as:*

$$\mu(B_P) = \frac{1}{\Psi(B_P)} \sum_{v \in B_P} \frac{\lambda}{|w(v) - \mu(X)|} w(v)$$

where $B_P = B \cap P$ and $\Psi(B_P) = \sum_{v \in B_P} \frac{\lambda}{|w(v)-\mu(X)|}$.

We can compute a weighted average value $\mu(B_{X \setminus P})$ as an estimate of $\mu(X \setminus P)$ in a similar fashion. In the following lemma, we show that the weighting scheme produces an unbiased estimate for the average values of regions.

Lemma 2.4.2 *If $\Psi(B_P)$ and $\sum_{v \in B_P} \frac{\lambda}{|w(v)-\mu(X)||} w(v)$ are independent, then $\mu(B_P)$ is an unbiased estimate for $\mu(P)$.*

Proof: The expected value of $\mu(B_P)$ can be written as:

$$\mathbb{E}[\mu(B_P)] = \frac{1}{\mathbb{E}[\Psi(B_P)]} \mathbb{E}[\sum_{v \in B_P} \frac{\lambda}{|w(v) - \mu(X)|} w(v)]$$

Simplifying the two expectations:

$$\begin{aligned}
\mathbb{E}[\Psi(B_P)] &= ||B_P|| \mathbb{E}[\frac{\lambda}{|w(v) - \mu(X)|}] \\
&= ||B_P|| \sum_{v \in P} \frac{|w(v) - \mu(X)|}{\lambda} \times \frac{\lambda}{|w(v) - \mu(X)|} \\
&= ||B_P|| |P||
\end{aligned}$$

$$\mathbb{E}[\sum_{v\in B_P}\frac{\lambda}{|w(v)-\mu(X)||}w(v)] = ||B_P||\mathbb{E}[\frac{\lambda}{|w(v)-\mu(X)|}w(v)]$$
$$= ||B_P||\sum_{v\in P}\frac{|w(v)-\mu(X)|}{\lambda}\times\frac{\lambda}{|w(v)-\mu(X)|}w(v)$$
$$= ||B_P||\sum_{v\in P}w(v)$$

Combining the two:

$$\mathbb{E}[\mu(B_P)] = \frac{||B_P||}{||B_P||||P||}\sum_{v\in P}w(v) = \frac{1}{||P||}\sum_{v\in P}w(v) = \mu(P)$$

■

Lemma 2.4.2 provides an unbiased average estimate $\mu(B_P)$ that can be used as a uniform sampling-based estimate $\mu(U \cap P)$. In case the independence assumption does not hold, the covariance between the two variables must be considered. We omit this discussion here for simplicity. Next, we derive upper bounds on the error reduction of a slice based on unbiased estimates for region average values.

Theorem 3 *Given a (uniform or importance) sample S, a confidence parameter δ ($0 \leq \delta < 1$) and a slice s, the error reduction $\phi(s)$ can be bounded as:*

$$Pr[\phi(s) < \max\{\phi_1(s), \phi_2(s)\}] > 1-\delta$$

where:

$$\phi_1(s) = (\mu(X) - \mu(S_P) - \epsilon)^2 \frac{||P||||X||}{||X \setminus P||}$$
$$\phi_2(s) = (\mu(X) - \mu(S_P) + \epsilon)^2 \frac{||P||||X||}{||X \setminus P||}$$
$$\epsilon = \sqrt{\frac{-\theta^2}{2||S_P||} \log(\frac{\delta}{2})}$$
$$\theta = \max_{u,v \in X} |w(u) - w(v)|$$
$$S_P = S \cap P$$

Proof: According to the Hoeffiding inequality [16]:

$$Pr[|\mu(S_P) - \mu(P)| \geq \epsilon] \leq \delta$$

where $\epsilon = \sqrt{\frac{-\theta^2}{2||S_P||} \log(\frac{\delta}{2})}$

Thus,

$$Pr[\phi(s) < \max\{\phi_1(s), \phi_2(s)\}] = Pr[|\mu(X) - \mu(P)| < \max\{|\mu(X) - \mu(P) - \epsilon|, |\mu(X) - \mu(P) + \epsilon|\}]$$
$$= Pr[\mu(S_P) - \epsilon < \mu(P) < \mu(S_P) + \epsilon] > 1 - \delta$$

■

A similar upper bound with the same confidence exists based on the samples from $X \setminus P$, and hence we can apply either of them with the same probabilistic guarantees. The theorem applies to both uniform and importance sampling as long as an unbiased estimate of the average is used.

While the expected number of sampled values $||U \cap P||$ from a region P is proportional

to $||P||$ in a uniform sample U, regions that include many node values deviating from the average value $\mu(X)$ will be oversampled in the importance sample. For instance, consider the case of a small region P' with many values deviating from $\mu(X)$. Although a high error reduction slice might separate P' from X, only importance sampling is likely to produce enough samples from P' to induce the selection of such slice. In Theorem 4, we relate the size of $||P \cap B||$ to the error reduction $\phi(s)$ of the slice s, enabling a second upper bound for importance sampling.

Theorem 4 *For an importance sample B, confidence δ and slice s, the error reduction $\phi(s)$ can be bounded as:*

$$Pr[\phi(s) < \frac{(p' + \epsilon)^2 \lambda^2}{||P||} \frac{||X||}{||X \setminus P||}] > 1 - \delta$$

where $p' = \frac{||P \cap B||}{||B||}$ *and* $\epsilon = \sqrt{\frac{-1}{2||B||} \log(\delta)}$.

Proof: The probability p of selecting a node from P is:

$$p = \sum_{v \in P} \frac{|w(v) - \mu(X)|}{\lambda} \qquad \geq \frac{1}{\lambda} |\sum_{v \in P} w(v) - \mu(X)|$$
$$= \frac{||P|||\mu(P) - \mu(X)|}{\lambda}$$

The Hoeffiding inequality gives that:

$$Pr[p < p' + \epsilon] > 1 - \delta$$

Thus,

$$
\begin{aligned}
Pr[p' + \epsilon > \frac{||P|||\mu(P) - \mu(X)|}{\lambda}] & \\
&= Pr[|\mu(P) - \mu(X)| < \frac{(p' + \epsilon)\lambda}{||P||}] \\
&= Pr[\phi(s) < \frac{(p' + \epsilon)^2 \lambda^2}{||P||} \frac{||X||}{||X - P||}] \qquad > 1 - \delta
\end{aligned}
$$

■

According to this bound, regions that get few samples will likely have a low error reduction. Also, different from Theorem 3, this bound does not depend on the range (θ) of node values in the network. This is a desired property when dealing with outliers and skewed value distributions. In the next section, we introduce a sampling-based algorithm for identifying approximate optimal slices using uniform or importance sampling. This algorithm employs Theorems 3 and 4 to prune the search space of candidate slices.

2.4.3 Approximate Slice Tree Construction

In Section 2.4.1, we described a greedy algorithm for slice tree construction. It applies the *Slice* routine to choose the best slice in set of nodes X (see Algorithm 1). In order to speedup this exhaustive scheme, we next show how to find approximate optimal slices, with probabilistic guarantees using sampling.

Algorithm 3 (*Approx-Slice*) takes as input the network G, a set of nodes $X \subseteq V$, a confidence parameter δ ($0 < \delta \leq 1$), an approximation constant ρ ($0 \leq \rho \leq 1$), and a sampling rate π ($0 \leq \pi \leq 1$). As its output, it returns a slice $t = (c, r, X)$ such that $c \in X$ and the error reduction $\phi(t)$ is at least $\rho.OPT$ with probability $1 - \delta$, where OPT is the optimal error reduction for a slice in X. In particular, the algorithm finds the optimal slice with probability $1 - \delta$ if ρ is 1. We replace the previously defined *Slice* routine by

Require: Network G, Set of nodes X, Confidence parameter δ, Approximation constant ρ, Sampling rate π
Ensure: Approximate best slice t
1: $L \leftarrow \emptyset$
2: **for** All centers $c \in X$ **do**
3: **for** All radii $r \in [0, radius(c, G, X)]$ **do**
4: $s \leftarrow (X, c, r)$
5: $L \leftarrow L \cup \{s\}$
6: **end for**
7: **end for**
8: $\phi(t) \leftarrow 0$
9: $S \leftarrow \emptyset$
10: **while** L is not empty **do**
11: Add $\lceil \pi.||X|| \rceil$ new samples from X to S
12: **for** All candidate slices $s \in L$ **do**
13: Compute $\phi_{max}(s), \phi'(s)$
14: **end for**
15: $q \leftarrow \max_{s \in L}\{\phi'(s)\}$
16: **if** $\phi'(q) \geq \phi(t)$ **then**
17: Compute $\phi(q)$
18: **if** $\phi(q) \geq \phi(t)$ **then**
19: $t \leftarrow q$
20: **end if**
21: **end if**
22: $L \leftarrow \{s \in L | \rho \times \phi_{max}(s) \geq \phi(t)\}$
23: **end while**

Algorithm 3: Approx-Slice

Approx-Slice in Algorithm 2 after setting the additional parameters δ, ρ and π. This new version of our solution, which we call *Approximate Greedy Slice Tree*, returns an ST for which each slice is approximate optimal according to the given parameters.

Approx-Slice first generates the set of candidate slices L (steps 1-7) and then performs iterative sampling and pruning (steps 10-23) in the search for an approximate optimal slice. On each iteration, the set of samples S is increased by $\lceil \pi.||X|| \rceil$ samples from X (step 11). The sampling scheme applied can be either uniform sampling or importance sampling as long as proper upper bounds are considered. For each slice $s \in L$, we

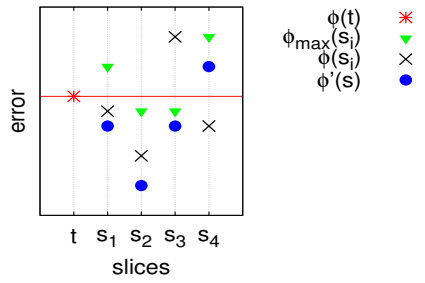


Figure 2.3: Example of slice pruning: t is the current top slice and s_1-s_4 are candidate slices with their respective error reduction upper bounds $\phi_{max}(s_i)$, actual error reductions $\phi(s_i)$, and estimated error reductions $\phi'(s_i)$. Slices s_2 and s_3 are pruned and the next slice to be computed is s_4.

compute two values based only on the samples in S: an upper bound on the error reduction $\phi_{max}(s)$ and an estimate on the error reduction $\phi'(s)$ (step 13). The value of $\phi_{max}(s)$ is set according to Theorem 3 for uniform sampling and both Theorems 3 and 4 for importance sampling. Also, it follows from Definition 4 that the error reduction for any slice in X cannot be larger than the error of X ($SSE(X)$).

For a given slice s, we can estimate the error reduction $\phi(s)$ based on the estimate for the average attribute value of the regions P and $X \setminus P$ and by considering the number of samples in P, in case importance sampling is applied. We compute $\phi'(s)$ as the harmonic mean of these different estimates to enforce that the slice with highest $\phi'(s)$ has high error reduction estimates across different measures.

Figure 2.3 illustrates the *Approx-Slice* pruning strategy for ρ set to 1, a current best slice t and 4 candidate slices (s_1-s_4). The actual error reduction is known only for t ($\phi(t)$). For all candidate slices, the algorithm computes the error reduction upper bound $\phi_{max}(s_i)$ and the error reduction estimate $\phi'(s_i)$ based on the sample S. The actual error reductions for candidate slices ($\phi(s_i)$) are shown only for explanation. Slices s_2 and s_3 are pruned, since their upper bounds ($\phi_{max}(s_2)$ and $\phi_{max}(s_3)$, respectively) are lower than $\phi(t)$. Nevertheless, the actual error reduction of s_3 ($\phi(s_3)$) is greater than the error

reduction of t, which means that s_3 should not be pruned. *Approx-Slice* is a probabilistic approximation algorithm in the sense that the probability of such an error is bounded by a small (user-defined) constant δ. The new set of candidate slices will contain only those slices with error reduction upper bound $\phi_{max}(s_i)$ greater than $\phi(t)$ (s_1 and s_4). Moreover, the slice s_4 is a candidate to replace t, since its error reduction estimate is greater then $\phi(t)$. After computing $\phi(s_4)$, the algorithm will not replace t because $\phi(s_4) < \phi(t)$.

In every iteration, new samples are added to S in order to improve the error reduction estimates and upper bounds of candidate slices until the candidate set L is empty. The execution time of *Approx-Slice* depends on how many iterations it takes to identify an approximate optimal slice. Some implementation decisions and associated complexity are discussed in Section 2.5. We show (see Section 2.6) that this algorithm can efficiently find a good slice s in X by pruning a large portion of the search space of candidate slices using a small sample S. It is important to notice that the constant factor approximation (ρ) given by our algorithm is for each single slice and not the resulting ST.

2.5 Implementation details

This section covers our implementation [1] of the ST algorithm (Algorithm 2).

Estimating the sizes of regions: Theorems 3 and 4 depend on the sizes of regions P ($||P||$) and $X \setminus P$ ($||X \setminus P||$), which cannot be computed based only on a sample of nodes. Computing exact sizes of regions for a candidate slice $s(c, r, X)$ takes a prohibitive $O(||E||)$ time in the worst case, since it requires a BFS starting from c up to distance r. To avoid such a cost, we propose simple upper bounds $\lceil ||P|| \rceil$ and $\lceil ||X \setminus P|| \rceil$ and lower bounds $\lfloor ||P|| \rfloor$ and $\lfloor ||X \setminus P|| \rfloor$ on $||P||$ and $||X \setminus P||$, respectively, that depend on the network structure and are calculated in constant time based on a pre-computed index.

[1] `https://code.google.com/p/graph-compression`

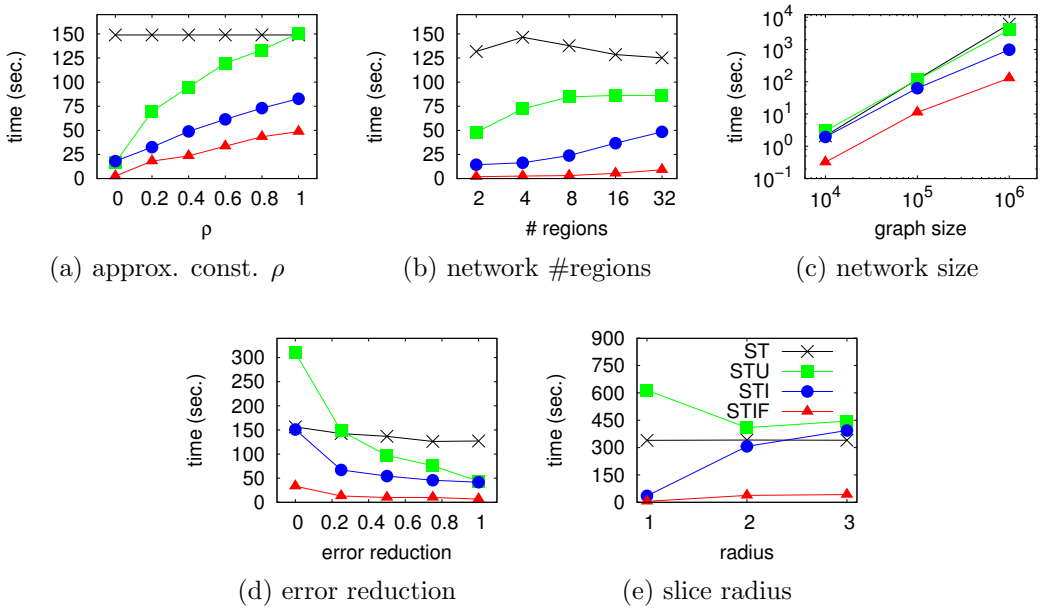
This index contains the number $z(v, r)$ of vertices at distance r from every vertex $v \in V$. For any slice s, $\lceil ||P|| \rceil = \max\{z(c, r), ||X||\}$ and $\lfloor ||X \setminus P|| \rfloor = \max\{||X|| - \lceil ||P|| \rceil, 0\}$. To compute the remaining bounds, we keep the maximum radius $r_f(v)$ for which a slice centered at v does not overlap with the border of any existing slice in the ST. We define $\lfloor ||P|| \rfloor = z(c, \min\{r, r_f(c)\})$ and $\lceil ||X \setminus P|| \rceil = \max\{||X|| - \lfloor ||P|| \rfloor, 1\}$. A region size is replaced by its upper bound if it appears as a numerator and by its lower bound if it appears as a denominator in Theorems 3 and 4.

Maximum radius: Our algorithm searches over all slices centered at every center $c \in X$ to identify the best one. However, we expect high error reduction slices to have small radius in real datasets, specially small-world networks [18]. Therefore, we set a maximum radius r_{max} as an extra parameter of our algorithm.

In-memory distance structure for sampling: One of the advantages of using sampling in ST construction is that we can keep a distance structure D which gives the set $D(c, r) \subseteq S$ of distinct samples at distance $r (r \leq r_{max})$ from c in G. Such a data structure requires $O(||V||||S'||)$ space and can be computed in worst-case time $O(||S'||||E||)$, where S' is the set of distinct samples in S. Computing and maintaining a similar data structure with full data would not be feasible for large networks. For each center c, we can traverse $D(c, r_{max})$ in worst-case time $O(||S'||)$ in order to compute the error reduction of all slices centered in c. For this implementation, each iteration of Algorithm 3 (steps 10-23) runs in $O(||S'||(||E|| + ||X||))$ time.

2.6 Experiments

We evaluate Slice Tree as a network attribute compression approach using real and synthetic networks. All algorithms are implemented in C++ and experiments were performed on a single core of a 2.67GHz Intel Core i7 with 12GB RAM.



(a) approx. const. ρ (b) network #regions (c) network size

(d) error reduction (e) slice radius

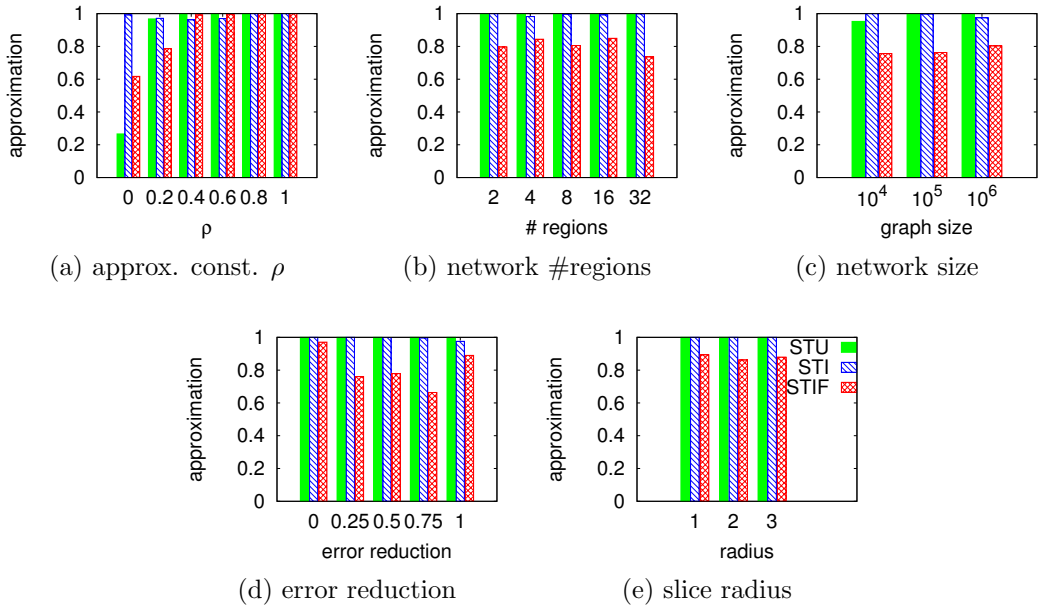
Figure 2.4: Execution time for finding the first slice ($k = 1$) in *synthetic* networks for greedy slice tree (ST), approximate slice tree with uniform sampling (STU), and two versions of slice tree with importance sampling using different parameters (STI and $STIF$) varying the approximation constant ρ and the number of regions, size ($||V||$), error reduction, and radius of slices of the network. By applying importance sampling, we can efficiently discover accurate STs for large networks in various settings.

2.6.1 Datasets

Table 2.1 lists four real-world datasets applied in our experiments. The *Traffic* dataset is the highway network of Los Angeles, CA (from the *PeMS* website[2]) with node values corresponding to average speeds at highway locations along time. The *Human* dataset is a gene network for *Homo sapiens* with gene and protein interactions as edges and tissue expression as node values (114 tissues) [19]. *DBLP* is an academic co-authorship network in which author nodes are annotated with publication counts for 15 research areas[3]. *Twitter* is a combination of the (undirected) social graph provided in [20], the tweets from [21] and the tweet sentiments from [22]. We averaged the sentiment of the

[2] `http://pems.dot.ca.gov/`

[3] Areas of venues were obtained from the classification in Wikipedia `http://en.wikipedia.org/wiki/List_of_computer_science_conferences`



(a) approx. const. ρ (b) network #regions (c) network size

(d) error reduction (e) slice radius

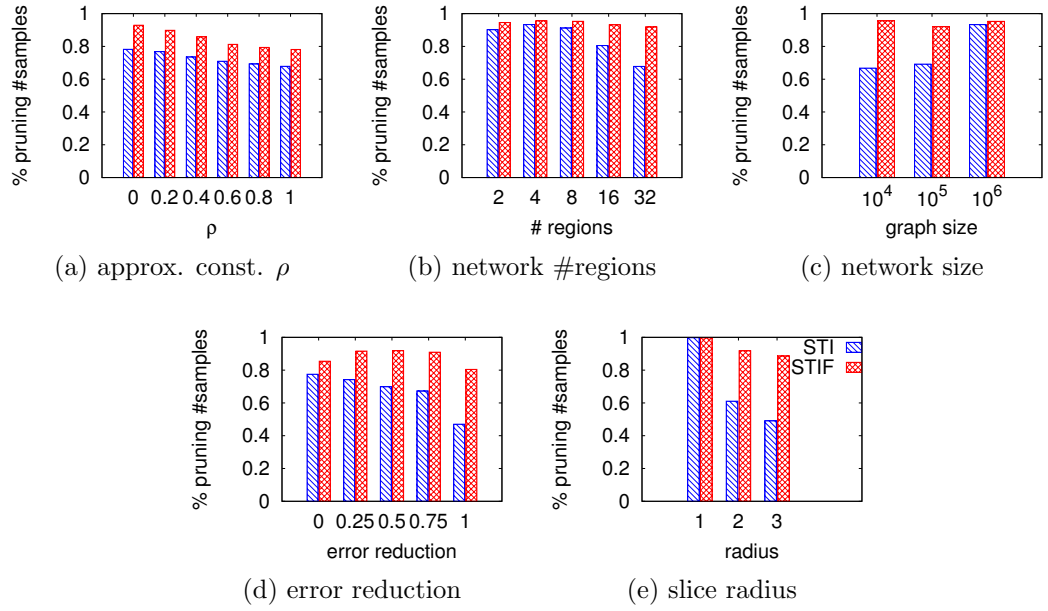
Figure 2.5: Approximation for finding the first slice ($k = 1$) in *synthetic* networks for greedy slice tree (ST), approximate slice tree with uniform sampling (STU), and two versions of slice tree with importance sampling using different parameters (STI and $STIF$) varying the approximation constant ρ and the number of regions, size ($||V||$), error reduction, and radius of slices of the network. By applying importance sampling, we can efficiently discover accurate STs for large networks in various settings.

tweets for each user and weighted these sentiments with the users' number of retweets as means to compute popularity-aware sentiments.

We also generate *synthetic* data by combining a network structure from the BA model [23] and node values defined by an ST. Values inside a region follow a Normal distribution with standard deviation set as to produce the desired SSE.

2.6.2 Scalability and Accuracy of Sampling

We start by evaluating our approximate algorithm applying different sampling strategies and under varied conditions using synthetic data. Figures 2.4, 2.5, and 2.6 compare the greedy slice tree construction algorithm (ST) and three versions of the approximate algorithm: STU, STI, and $STIF$. STU applies uniform sampling with $\delta = 0.1, \rho =$



(a) approx. const. ρ (b) network #regions (c) network size

(d) error reduction (e) slice radius

Figure 2.6: Effectiveness of the pruning based on the number of samples (Theorem 4) for finding the first slice ($k = 1$) in *synthetic* networks for greedy slice tree (ST), approximate slice tree with uniform sampling (STU), and two versions of slice tree with importance sampling using different parameters (STI and $STIF$) varying the approximation constant ρ and the number of regions, size ($||V||$), error reduction, and radius of slices of the network. By applying importance sampling, we can efficiently discover accurate STs for large networks in various settings.

$0.6, \pi = 0.1$. STI applies importance sampling with the same parameters as STU. $STIF$ also uses importance sampling, but with more relaxed parameters $\delta = 0.4, \rho = 0.1, \pi = 0.01$, thus being faster than STI. Maximum radius r_{max} is set to 2. We will make clear whenever these parameters are changed. Synthetic networks have 10^5 vertices, 5 edges for each new vertex, 32 regions (31 slices), slice radius 2, error of 2.10^5 and error reduction of 10^5, unless specified. The task for all algorithms is finding the first slice and we evaluate them in terms of average execution time and approximation (i.e. fraction of the optimal error reduction achieved). We also compute how often the pruning based on the number of samples (Theorem 4) is used by STI and STIF as means to measure the importance of this pruning strategy compared to the one based on mean estimates (Theorem 3).

name	**vertex**	**edge**	**value**	$\|\|V\|\|$	$\|\|E\|\|$
Traffic	sensor	highway	speed	2k	6k
Human	gene	interaction	expression	4k	9k
DBLP	author	collaboration	#papers	1.3m	5.2m
Twitter	user	following	sentiment	.7m	19.2m

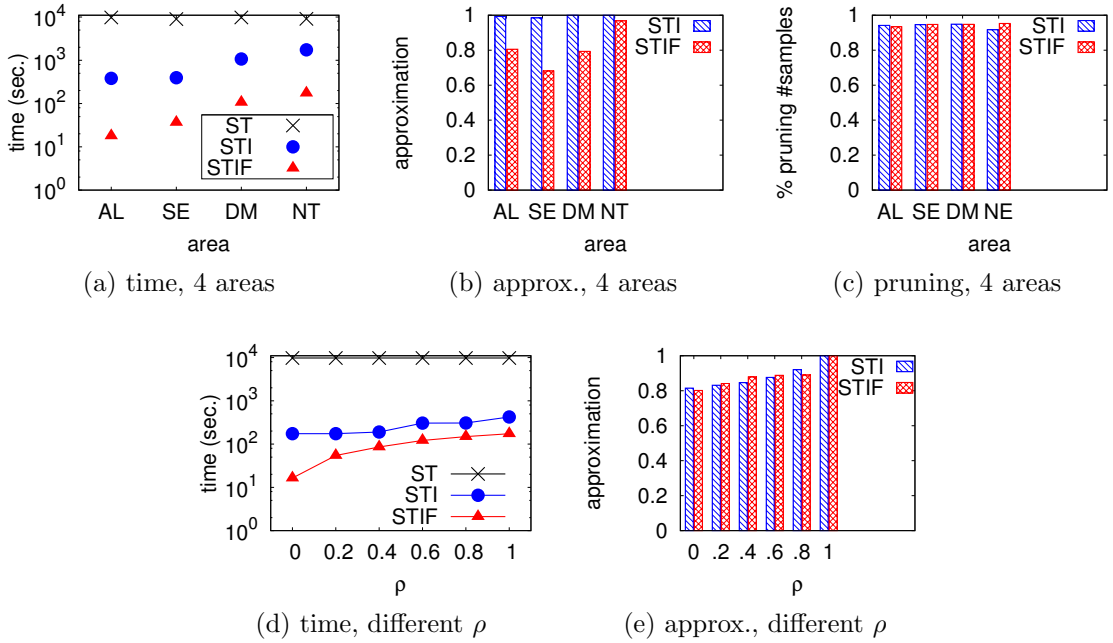
Table 2.1: Dataset description and statistics.

The constant ρ sets the compromise between the approximations given by *STU*, *STI*, and *STIF*, and their execution time (Figure 2.4a). However, the algorithms achieve much higher approximation than ρ in practice (Figure 2.5a). For instance, *STI* achieves at least a 0.95 approximation for any ρ and *STIF* obtains a 0.6 approximation in 2% of the time taken by *ST* ($\rho = 0$). Also, pruning based on the number of samples plays an important role for *STI* and *STIF* (Figure 2.6a).

Increasing the number of regions in the network has a small impact over the algorithms for all the evaluation metrics (Figures 2.4b, 2.5b and 2.6b). This shows that they can still quickly find a good slice in the presence of many candidates.

In terms of scalability with the network size (Figure 2.4c), *STU*, *STI*, and *STIF* achieve speedups up to 1.5×, 6× and 47× w.r.t. *ST* with at least 0.95, 0.97, and 0.75 approximation, respectively (Figure 2.4c). These results show how importance sampling enables the identification of approximate optimal slices using much less samples than the uniform sampling approach, which translates into effective pruning (Figure 2.6c).

The higher the input network error reduction, the better it matches an ST model, which leads to better performance for all the strategies except *ST* (Figure 2.4d) with no significant effect over approximation (Figure 2.5d). As the error reduction increases, uniform and importance sampling behave more similar and the pruning based on the number of samples becomes less effective for STI and STIF (Figure 2.6d). This analysis shows how a low-sampling version of the importance sampling algorithm enables users to assess whether ST is a suitable for the compression of a given network.



(a) time, 4 areas (b) approx., 4 areas (c) pruning, 4 areas

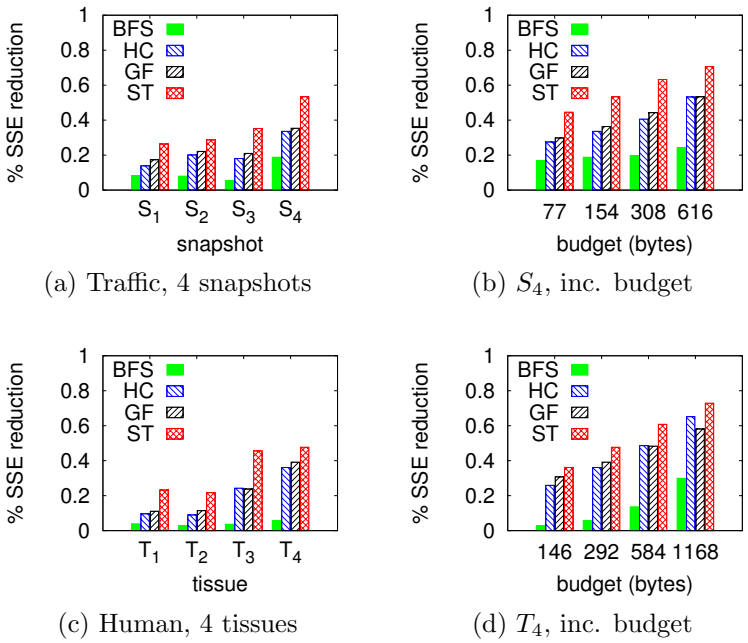
(d) time, different ρ (e) approx., different ρ

Figure 2.7: Execution time (a,d), approximation (b,e) and effectiveness of pruning based on the number of samples (c) in finding the first slice in *DBLP* for greedy slice tree (*ST*) and two versions of approximate slice tree using different parameters (*STI* and *STIF*). Four research areas (*algorithms* (AL), *security* (SE), *data management* (DM), and *networks* (NT)) are considered in (a,b,c). Results in (d,e) are for AL. Importance sampling enables the computation of accurate STs in large real networks.

Because uniform sampling is not value-sensitive, it needs more samples to detect slices with good approximation (Figures 2.5e and 2.4e)[4]. The use of the pruning reflects how importance sampling is more appropriate when slices are more condensed (Figure 2.6e).

After evaluating several aspects of different versions of our approximate algorithm using synthetic data, we next study its effectiveness on a real dataset. Figure 2.7 compares *ST*, *STI* and *STIF* in finding the first slice w.r.t. execution time, approximation, and use of the pruning based on number of samples for *DBLP*. *STU* was not considered in this evaluation because it does not scale to such a network. Default parameters for the importance sampling algorithm are set as follows: $\delta = 0.1, \rho = 0.9, , \pi = 0.02$ for *STI*;

[4]For this experiment, networks have $50k$ vertices and r_{max} is 3.



(a) Traffic, 4 snapshots (b) S_4, inc. budget

(c) Human, 4 tissues (d) T_4, inc. budget

Figure 2.8: Error reduction for the wavelets with BFS (*BFS*), Haar trees (*HC*), graph Fourier (*GF*), and slice tree (*ST*) in the *Traffic* (a,b) and *Human* (c,d) datasets. We selected four network attributes (tissues and snapshots) from each dataset (a,c) and also a fixed attribute for increasing budget (b,d). *ST* compresses attribute values with high accuracy in real datasets.

$\delta = 0.4, \rho = 0.1, \pi = 10^{-3}$ for *STIF* and $r_{max} = 2$ (*STI* and *STIF*).

Figures 2.7a, 2.7b, and 2.7c show the performance of the algorithms for four research areas: *algorithms*(AL), *security* (SE), *data management* (DM) and *networks* (NT). *STI* achieves speedups between 5× (NT) and 25× (AL) with average approximations between 0.79 (DM) and 1.0 (NT). *STIF* achieves speedups between 56× (NT) and 544× (DM) with approximations between 0.68 (SE) and 0.96 (NT). For all areas, the number of samples gives a tighter bound on the error reduction in more than 93% of the time.

In Figures 2.7d and 2.7d, we evaluate the performance of the algorithms for the research area *algorithms* (AL) varying the approximation constant ρ. *STI* and *STIF* obtain a similar approximation average of 0.80 with a speedup of 56× and 596×, respectively, when $\rho = 0$. When the best slice is to be returned with high probability ($\rho = 1$), *STI*

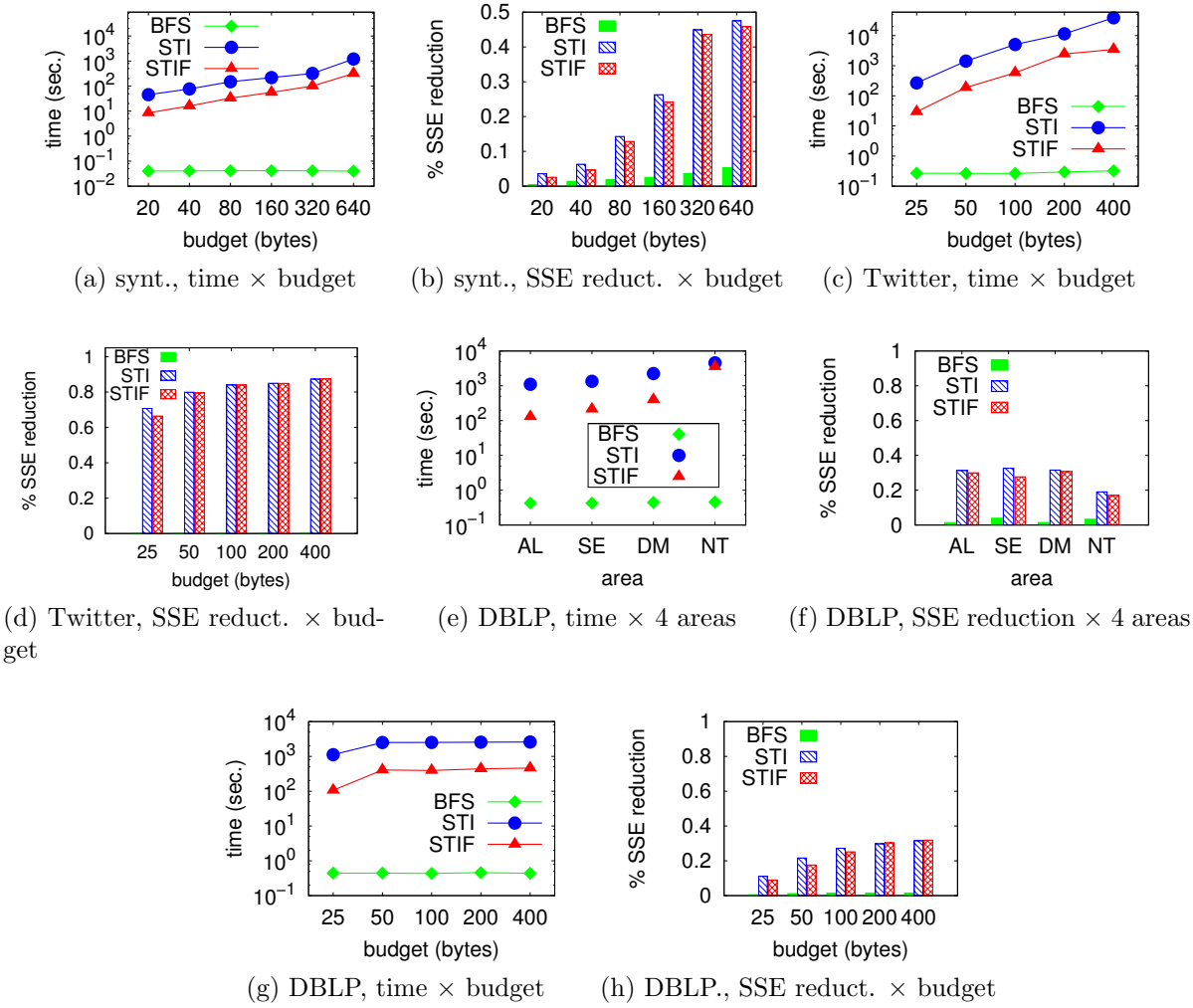
and *STIF* get speedups of 23× and 56×, respectively.

2.6.3 Slice Tree for Network Compression

We evaluate ST compression in terms of compression ratio, error, and running time using real and synthetic data. We also consider the following baselines: *Graph Fourier (GF)* [24], *Wavelets over a BFS vector (BFS)*, and *Haar trees with Average Linkage (HC)* [25]. *GF* applies the eigenvectors of the Laplacian matrix of the network as a basis to represent the attribute values. Compression is achieved by selecting a small set of high-energy coefficients in the spectral domain. *BFS* maps node values to a vector by performing a BFS starting from an arbitrary node. This search tends to assign nodes that are close in the network structure to close positions in the vector and the values are further compressed using *Haar wavelets* [26]. *HC* is based on a tree representation of the network structure using hierarchical clustering [27]. Node values projected over this hierarchical structure are further compressed using a *Haar-like* wavelet basis. *Average Linkage* [25] and shortest path distances were applied in the hierarchical clustering.

Here, instead of computing the budget of an ST in number of slices (k), we convert this measure to *bytes* (see Definition 1). This enables a fair comparison between ST and the baselines. Figure 2.8 shows relative error reduction results for *ST*, *BFS* and *HC*. The relative SSE reduction is the ratio between the SSE of the dataset with respect to its overall average and the SSE of the compression. Since these datasets are small (up to $4k$ edges), we do not show the running time results. All the methods run in time in the order of seconds for both datasets. We first evaluate the compression approaches across different network attributes (Figures 2.8a and 2.8c). For *Traffic*, we selected four snapshots of the network that cover well the span of compression results. Similarly, we show results for four tissues[5] from the *Human* dataset. Budgets were set to 1% of the size

[5] T_1: frontal cortex, T_2 : stomach pylorus, T_3 : placenta and T_4 : tonsil.



(a) synt., time × budget (b) synt., SSE reduct. × budget (c) Twitter, time × budget

(d) Twitter, SSE reduct. × budget (e) DBLP, time × 4 areas (f) DBLP, SSE reduction × 4 areas

(g) DBLP, time × budget (h) DBLP., SSE reduct. × budget

Figure 2.9: Execution time (a,c,e,g) and error reduction (b,d,f,h) for *BFS* and two versions of approximate slice tree with importance sampling using different parameters (*STI* and *STIF*) in *synthetic* datasets (a,b), *Twitter* (c,d) and *DBLP* (e-g). For *synthetic*, *Twitter*, and *DBLP* (DM), we evaluate how the execution time and error reduction increase with the budget (a-d, g,h). We also show time and error reduction results for a fixed budget and four areas in *DBLP* (e,f). *BFS* is more efficient than *STI* and *STIF* but achieves much poorer performance in terms of compression error.

of the dataset (154 and 292 *bytes* for *Traffic* and *Human*, respectively). We also show results for a single attribute (S_4 and T_4) increasing the budget (Figures 2.8b and 2.8d). *ST* consistently outperforms the baselines for both datasets in all settings considered, achieving up to a 2-fold improvement over the best baseline (*GF*).

Next, we evaluate ST compression using large networks (*synthetic*, *DBLP* and *Twitter*). However, because *HC* requires the computation of a distance matrix and *GF* requires an spectral decomposition of the Laplacian matrix of the network, these methods cannot be applied to such large datasets. We restrict our results to our approximate algorithm using importance sampling and *BFS*. Figures 2.9a and 2.9b show the running time and *SSE* reduction as the budget is increased for two versions of the approximate algorithm using $\delta = 0.1, \rho = 0.6, \pi = 0.1$ (*STI*) and $\delta = 0.4, \rho = 0.1, \pi = 0.01$ (*STIF*) in synthetic networks. Maximum radius r_{max} is set to 2. The parameters used in the generation of the networks are the same ones described in Section 2.6.2 and thus the optimal relative *SSE* reduction achievable using an ST with 32 regions (350 *bytes*) is 0.5. *BFS* is the most efficient algorithm, but it obtains poor error reduction results. For a budget of 320 *bytes*, *BFS*, *STI* and *STIF* achieve average error reductions of 0.04, 0.45 and 0.44, respectively. As expected, once this budget is reached and there are no more new regions to be captured in the network, the benefit of adding extra budget is reduced.

Figures 2.9c and 2.9d show the compression results for the *Twitter* dataset. Default parameters for the versions of the importance sampling algorithm are set as: $\delta = 0.1, \rho = 0.9, \pi = 0.02$ for *STI*; $\delta = 0.4, \rho = 0.1, \pi = 0.001$ for *STIF* and $r_{max} = 2$ (*STI* and *STIF*). Maximum radius r_{max} is set to 1. As for the synthetic networks, *BFS* is fast but cannot produce accurate compression. While both *STI* and *STIF* achieve 87% relative *SSE* reduction with only 400 bytes (10^3 compression rate), *BFS* is not able to achieve any significant error reduction in any of the experiments. These results contrast with the performance of *BFS* for small datasets, where it was able to achieve up to 50% of the relative *SSE* reduction of *ST*.

Compression results for *DBLP* are given in Figures 2.9e-2.9h. Default parameters for the algorithms are set as for the experiments using *Twitter*. Maximum radius r_{max} is set to 2. Figures 2.9e and 2.9f show the execution time and error reduction of the algorithms

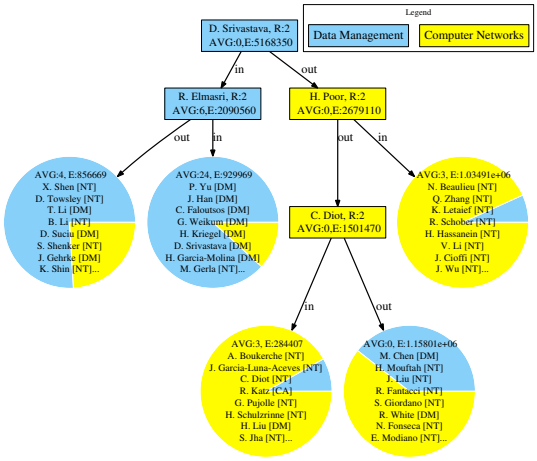


Figure 2.10: First slices for *DBLP* combining paper counts for Data Management and Networks. ST captures publications patterns across different research areas.

for four research areas and a fixed budget of 400 *bytes* (31 slices). The compression results for *STI* and *STIF* vary according to the research area considered. For instance, the compression of *Algorithms* (AL) resulted in twice as much relative error reduction as achieved for *Networks* (NT) (Figure 2.9f). This is due to the fact that NT is a much more prolific area, with three times as many publications and active researchers compared to AL. Similarly, compressing NT takes 4 and 25 times longer than AL for *STI* and *STIF*, respectively. The gains in *SSE* reduction vanish quickly as we increase the budget for a fixed research area (AL). Figure 2.9h shows that both *STI* and *STIF* achieve a maximum relative *SSE* reduction of 30% with a 10^3 compression rate.

2.6.4 Slice Trees in Real-World Networks

In what follows, we illustrate ST compressions of real-world networks. We show that STs capture in a compact form important properties of networks, such as collaboration patterns in *DBLP* and major congestions in the *Traffic* network.

Figure 2.10 shows how authors from two areas (*Data Management (DM) and Networks (NT)*) are distributed among the first slices of an ST based only on combined publication

counts. Moreover, we assign authors to a single main area in which they are most prolific as means to characterize the regions discovered. We color-code both slice nodes (rectangles) and resulting regions (circles) based on the fraction of authors in DM and NT. The leaf region nodes list the 10 most prolific authors with their respective areas. The ST naturally separates the communities from the two areas as they are both distinct within the structure (researchers within areas are more likely to collaborate among each other) and also in terms of number of publications. The first slice separates the majority of the DM community within 2 hops from *Divesh Srivastava* from the NT community. Further slices separate authors based on how prolific they are (average paper count for each leaf ST node is reported in the first line of the label).

In Figure 2.1, we show how one snapshot of *Traffic* (Figure 2.1a) is compressed using 2 slices (Figure 2.1b). Nodes in the network have their sizes and colors set based on average speed values. The largest (red) node has an average speed of $10mph$ while the smallest (green) node has an average speed of $85mph$. Medium (yellow) nodes have average speed around $45mph$. Intermediate and leaf regions of the ST are also colored according to their average speeds (from left to right: 41, 15, and 65). The values reconstructed from the ST are projected back in the network (Figure 2.1c). ST captures two major congestions with smooth regions while the remaining locations are covered by a third region. Further slices can isolate more congested regions.

2.7 Related Work

The majority of existing work on network summarization and compression focuses on providing compact representations of a network structure [14, 15]. Prior work on network compression has proposed information-theoretic approaches for encoding structural data [14, 28]. More recent studies applied OLAP-like aggregations to networks, providing

a multi-resolution summarization of nodes and their relationships [29, 15]. Our work is complementary to these efforts, since we focus on compressing attribute values.

The compression of network attributes is related to a recent effort to generalize signal processing techniques to network data [27, 30, 13]. More specifically, although harmonic analysis has been successfully applied to problems that range from image compression [31] to query processing [26], applying these techniques to networks remains a challenge. A natural solution would be first to embed the network structure into an Euclidean space and then process the embedded values using existing signal processing approaches. However, Bourgain [32] has shown that such an embedding requires, in the worst case, a logarithmic number of dimensions w.r.t. to the size of the graph. An existing alternative, which is known as graph Fourier, constructs an orthonormal basis for a network using spectral analysis [24, 13]. The first eigenvectors of the network Laplacian matrix define a basis that is expected to capture the smoothness of node values over the network.

Similarly to the traditional Fourier analysis in Euclidean spaces, graph Fourier is unable to localize signals in both time (space) and frequency domains [27, 33, 13]. Recent efforts attempt to address this limitation by generalizing the wavelet analysis [31] to networks. Examples of such efforts include graph wavelets [34, 33], diffusion wavelets [35], and Haar trees [27]. Similar to ST, graph wavelets [34] also average values within a radius distance from a center node. However, these wavelets are based on deviations in values on a disc and a surrounding ring, which is suitable for summarizing communication data but not for compressing network attributes under budget constraints. In [36], the authors propose a lossy compression of attributes over a grid structure. However, their approaches do not assume budget constraints and depend on either a rigid topology or the ordering of node identifiers. We show that ST achieves better compression accuracy than graph Fourier and Haar trees under a fixed budget. The main advantage of ST compared to these existing techniques is that ST performs an efficient value-sensitive search for slices

that maximize the compression accuracy using sampling.

ST applies the network as a space where attributes are embedded and can be represented in a compact form using slices. It differs from existing work on traditional clustering [25], which does not consider the network structure, and community detection based on attributes, which has no constraints on the representation cost [37]. Finally, while compression is a key problem in sensor networks, existing approaches [38] are focused on power consumption and routing, which is out of the scope of this work.

2.8 Conclusions

We introduced Slice Tree as a network compression strategy for attribute values. ST is a hierarchical decomposition scheme using slices, which partition the network into regions that are smooth w.r.t. attribute values. Computing an ST that minimizes the compression error is NP-hard, thus we proposed an efficient greedy heuristic for ST construction. In order to scale ST to large networks, we devised an importance sampling strategy that efficiently computes approximate optimal slices with high probability.

We evaluated our approach in terms of compression error and scalability using synthetic and real-world datasets. Results showed that ST produces accurate compression, achieving up to 87% error reduction in node values, with 10^3 compression ratio, and up to 2-fold improvements over the best baseline method considered. Moreover, importance sampling enables the efficient compression of million-node networks with speedups up to 47× over our scheme using full data. We also demonstrated the effectiveness STs in capturing relevant phenomena in real networks, such as collaboration patterns in co-authorship networks and congestions in traffic networks.

This work opens promising directions for future research. A key question is how to update STs over time. We also plan to generalize our framework to other error metrics

and more complex data (e.g. attribute vectors). Finally, distributed algorithms might enable the compression of even larger networks than those considered in this work.

Chapter 3

Graph Wavelets via Sparse Cuts

3.1 Introduction

Graphs are the model of choice in several applications, ranging from social networks to the *Internet-of-things (IoT)*. In many of these scenarios, the graph represents an underlying space in which information is generated, processed and transferred. For instance, in social networks, opinions propagate via social interactions and might produce large cascades across several communities. In IoT, different objects (e.g. cars) collect data from diverse sources and communicate with each other via the network infrastructure. As a consequence, exploiting the underlying graph structure in order to manage and process data arising from these applications has become a key challenge.

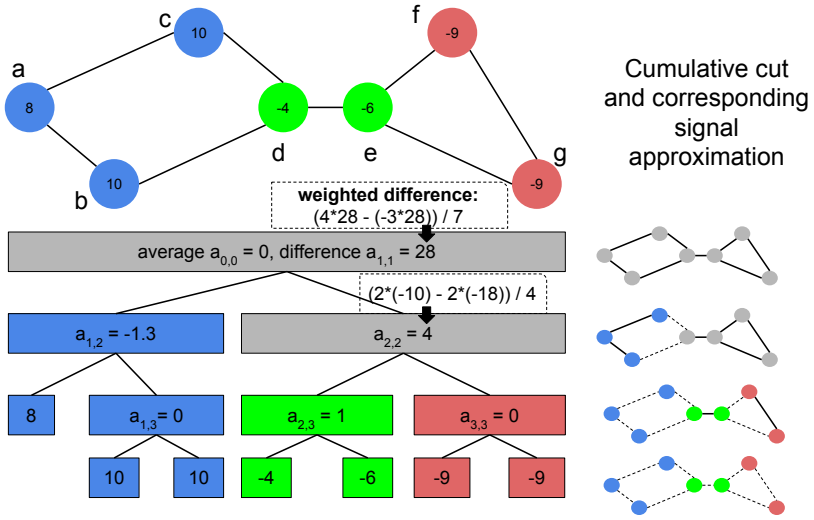
Signal processing on graphs (SPG) is a framework for the analysis of data residing on vertices of a graph [13, 39]. The idea generalizes traditional signal processing (e.g. compression, sampling) as means to support the analysis of high-dimensional datasets. In particular, SPG has been applied in the discovery of traffic events using speed data collected by a sensor network [40]. Moreover, graph signals are a powerful representation for data in machine learning [27, 41]. As in traditional signal processing, the fundamental

operation in SPG is the *transform*, which projects the graph signal in the frequency (or other convenient) domain. Real signals are expected to be smooth with respect to the graph structure—values at nearby vertices are similar—and an effective transform should lead to rapidly decaying coefficients for smooth signals. The most popular transform in SPG, known as Graph Fourier Transform [30, 13], represents a signal as a linear combination of the eigenvectors of the graph Laplacian. However, as its counterpart in traditional signal processing, Graph Fourier fails to localize signals in space (i.e. within graph regions). This limitation has motivated recent studies on graph wavelets [34, 33, 42].

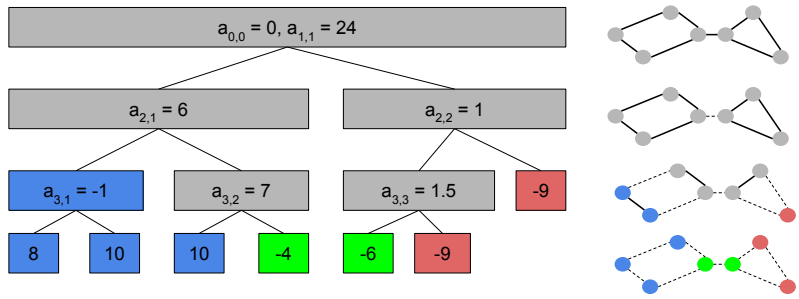
An open issue in SPG is how to link properties of the signal and underlying graph to properties of the transform [13]. Gavish et al. [27] makes one of the first efforts in this direction, by relating the smoothness of the signal with respect to a tree structure of increasingly refined graph partitions and the fast decay of the wavelet coefficients in a Haar-like expansion. However, as explicitly stated in their paper, their approach *"raises many theoretical questions for further research, in particular regarding construction of trees that best capture the geometry of these challenging datasets"*.

In this work, we study the problem of computing wavelet trees that encode both the graph structure and the signal information. A wavelet tree defines a hierarchical partitioning used as basis for a graph wavelet transform. Good wavelet trees should produce fast decaying coefficients, which support a low-dimensional representation of the graph signal. The particular application scenario we consider is the lossy graph signal compression. This task arises in many relevant data management and analytics applications, including IoT and social networks, where values associated to interconnected entities have to be represented in a compact form.

Figure 3.1 shows two wavelet trees, A and B, and their respective transforms for a piecewise smooth graph signal defined over seven vertices. The wavelet transform



(a) Wavelet basis A



(b) Wavelet basis B

Figure 3.1: Graph wavelet transforms for two different wavelet trees and the same piecewise smooth graph signal (values set to vertices). A wavelet tree contains one *average coefficient* and several *weighted difference coefficients* associated with vertex partitions. Basis A is better than B because it produces fast decaying difference coefficients. Moreover, basis A can be approximately encoded as a sequence of sparse graph cuts (first $\{(b,d),(c,d)\}$ then $\{(e,f),(e,g)\}$), which leads to a compact and accurate representation of the graph signal.

contains a single *average coefficient* and a set of *weighted difference coefficients* associated to each node of the tree. Weighted difference coefficients are computed as a function of the values in each partition and the partition sizes (see Equation 3.2 for a formal definition). Notice that these two bases produce very different wavelet transforms for the same signal. While tree A is characterized by fast decaying difference coefficients, tree B has relatively large coefficients at every level. This indicates that tree A supports a

better representation for the signal than does tree B. However, good wavelet trees must also capture properties of the graph structure.

We measure the relationship between a wavelet tree and the graph structure using the notion of sparse cuts. A graph cut is a set of edges that connect two disjoint sets of vertices and sparse cuts (i.e. a small number of edges) are a natural way to model graph partitions [43]. As each node of the wavelet tree separates a set of vertices into two subsets, a sparse wavelet tree can be approximately encoded by a sequence of sparse cuts. This work is the first effort to connect graph cuts and graph signal processing. In particular, we show how problems that arise in the construction of optimal wavelet trees are related to hard cut problems, such as *graph bisection* [44] and *multiway-cuts* [45].

In Figure 3.1, we also show the cuts associated to each level of the wavelet tree together with the signal approximation for the respective level. Basis A can be approximately encoded by the cutting four edges: $\{(b,d),(c,d)\}$ (level 1) and $\{(e,f),(e,g)\}$ (level 2). The resulting compact wavelet tree can effectively represent the graph signal using only the two top wavelet coefficients, leading to a relative L_2 error of 1%. On the other hand, basis B does not have such a compact approximation with small error via sparse cuts.

In this work, we formalize the problem of computing sparse wavelet bases (or trees) for graph wavelet transforms. This problem, which we call *sparse graph wavelet transform* (SWT) consists of identifying a sequence of sparse graph cuts that leads to the minimum error in the reconstruction of a given graph signal. We show that this problem is NP-hard, even to approximate by a constant. In fact, we are able to show that computing each individual cut in the tree construction is an NP-hard problem.

As the main contribution of this work, we propose a novel algorithm for computing an SWT via spectral graph theory. The algorithm design starts by formulating a relaxation of our problem as an eigenvector problem, which follows the lines of existing approaches for *ratio-cuts* [46], *normalized-cuts* [47] and *max-cuts* [48]. We further show how the

proposed relaxation leads to a regularization of pairwise values by the graph Laplacian, which relates to existing work on graph kernels [49, 50]. In order to improve the computational efficiency of our algorithm, we design a *fast graph wavelet transform* (FSWT) using several techniques including *Chebyshev Polynomials* and the *Power method*.

3.2 Related Work

Generalizing the existing signal processing framework to signals that reside on graphs is the main focus of *Signal Processing on Graphs (SPG)* [13, 39]. Operations such as filtering, denoising, and downsampling, which are well-defined for signals in regular Euclidean spaces, have several applications also when signals are embedded in sparse irregular spaces that can be naturally modeled as graphs. For instance, sensor networks [40], brain imaging [51], computer network traffic [34], and statistical learning [49, 50, 41], are examples of scenarios where graph signals have been studied. The main idea in SPG is the so called *Graph Fourier Transform (GFT)* [30], which consists of applying eigenvectors of the Laplacian matrix of a graph as a basis for graph signals. Laplacian eigenvectors oscillate at different frequencies over the graph structure, capturing a notion of frequency similar to complex exponentials in the standard Fourier Transform.

As is the case for its counterpart for Euclidean spaces, GFT fails to localize graph signals, i.e. capture differences within graph regions. This aspect has motivated the study of graph wavelets [34, 27, 33, 42]. Crovella and Kolaczyk [34] introduced wavelets on graphs for the analysis of network traffic. Their design extracts differences in values within a disc (i.e. a center node and a fixed radius) and a surrounding ring as means to identify traffic anomalies. Coiffman and Maggioni [42] proposed a more sophisticated design, known as *diffusion wavelets*, based on compressed representations of dyadic powers of a diffusion operator. In [33], Hammond et al. present a wavelet design using kernel

functions that modulate eigenvectors around vertices at multiple scales.

An assumption shared by existing work on graph wavelets is that good bases can be computed based solely on the graph structure. However, as shown in Figure 3.1, a proper choice of graph wavelet bases can lead to significantly more effective transforms. In this work, we study the problem of computing optimal graph wavelet bases for a given signal via sparse graph cuts. A graph cut partitions the vertices of a graph into two disjoint subsets and optimization problems associated with graph cuts are some of the most traditional problems in graph theory [52, 53]. In particular, graph cuts (e.g. min-cut, max-cut) are a natural way to formulate graph partitioning problems [43]. Here, we constraint the size of the cut, in number of edges, associated to a graph wavelet basis in order to discover bases that are well-embedded in the graph. A similar constraint also appears in the min-cut [52], graph bisection [44], and multiway-cut [45] problems.

Learning bases tailored for classes of signals is an important problem in signal processing, known as *dictionary learning* [54]. This problem differs from ours since our wavelet bases are adapted to each signal, which leads to more compact representations. In [5], the authors show how importance sampling can support the discovery of center-radius partitions for attribute compression. However, their approach does not generalize to arbitrarily shaped partitions.

Many relevant problems on graphs have been solved using the framework of Spectral Graph Theory (SPG) [55], which studies combinatoric graph properties via the spectrum of matrices associated with them. For instance, the relationship between eigenvectors of the Laplacian and graph partitions can be traced back to Cheeger's inequality [56]. More recently, SPG has led to efficient graph partitioning algorithms (e.g. ratio-cuts [46], normalized-cuts [47]). Here, we propose a spectral algorithm for computing sparse graph wavelet bases. Interestingly, our analysis show that these bases are related to existing work on graph kernels [49, 50], including the wavelet design by Hammond et al. [33].

3.3 Wavelets on Graphs

A graph is a tuple $G(V, E)$, where V is a set of n vertices and E is a set of m (unweighted) edges, respectively. A signal $W{:}V \to \mathbb{R}$ is a real-valued function defined on the set of vertices V. In other words, $W(v)$ is the value of the signal for a vertex $v \in V$. In Figure 3.1 we show an example of a graph G for which we define a signal W.

A graph wavelet tree is a binary tree structure $\mathcal{X}(G)$ that partitions the graph recursively as follows. A root node X_1^1 contains all the vertices in the graph (i.e. $X_1^1 = V$). In general, $X_k^\ell \subseteq V$ is the k-th node at level ℓ with children $X_i^{\ell+1}$ and $X_j^{\ell+1}$ at level $\ell+1$ such that $X_i^{\ell+1} \cap X_j^{\ell+1} = \emptyset$ and $X_i^{\ell+1} \cup X_j^{\ell+1} = X_k^\ell$. We focus on binary trees since they have the same encoding power as n-ary trees in this model.

The tree $\mathcal{X}(G)$ defines spaces of functions, $\mathcal{V}_\ell$ and $\mathcal{W}_\ell$, analogous to Haar wavelet spaces in harmonic analysis [31]. The space $\mathcal{V}_1$ contains functions that are constant on V. And, in general, $\mathcal{V}_\ell$ contains functions that are piecewise constant on nodes in X_k^ℓ at the ℓ-level of $\mathcal{X}(G)$. Let $\mathcal{V}$ be the space of functions that are constant on individual nodes in V. Bases to span such spaces can be constructed using functions $\mathbf{1}_{X_k^\ell}$ equal to 1 for $v \in X_k^\ell$ and 0, otherwise (box functions). This formulation leads to a multiresolution $\mathcal{V}_1 \subset \mathcal{V}_2 \subset \ldots \mathcal{V}$ for function spaces. Another set of function spaces in the form $\mathcal{W}_\ell$ contains *wavelet functions* $\psi_{k,\ell}$ with the following properties: (1) are piecewise constant on $X_i^{\ell+1}$ and $X_j^{\ell+1}$, (2) are orthogonal to $\mathbf{1}_{X_k^\ell}$ defined on X_k^ℓ and (3) are 0 everywhere else. It follows that any function in $\mathcal{W}_\ell$ can be represented using $\mathcal{V}_{\ell+1}$. Also, for any level ℓ, $\mathcal{V}_\ell \perp \mathcal{W}_\ell$ and $\mathcal{V}_\ell \oplus \mathcal{W}_\ell = \mathcal{V}_{\ell+1}$, where $\oplus$ is the orthogonal sum.

We combine wavelet functions with $\mathbf{1}_V$ to produce an orthonormal basis for G. Intuitively, this basis supports the representation of any graph signal W as a linear combination of the average $\mu(W)$ plus piecewise functions defined on recursive partitions of the vertices V (see Figure 3.1). A graph wavelet transform φW is a set of difference

coefficients $a_{k,\ell}$:

$$a_{k,\ell} = \begin{cases} \mu(W), & \text{if } \ell = k = 0 \\ \langle W, \psi_{k,\ell} \rangle, & \text{otherwise} \end{cases} \tag{3.1}$$

In particular, except for $a_{0,0}$, we can write $a_{k,\ell}$ as:

$$a_{k,\ell} = \frac{|X_j^{\ell+1}|}{|X_k^\ell|} \sum_{v \in X_i^{\ell+1}} W(v) - \frac{|X_i^{\ell+1}|}{|X_k^\ell|} \sum_{v \in X_j^{\ell+1}} W(v) \tag{3.2}$$

The sizes $|X_k^\ell|$, $|X_i^{\ell+1}|$ and $|X_j^{\ell+1}|$ are taken into account because partitions might be unbalanced. Analogously, the wavelet inverse $\varphi^{-1}W$ is defined as:

$$\varphi^{-1}W(v) = a_{0,0} + \sum_k \sum_\ell \nu_{k,\ell}(v) a_{k,\ell} \tag{3.3}$$

where:

$$\nu_{k,\ell}(v) = \begin{cases} 1/|X_i^{\ell+1}|, & \text{if } v \in X_i^{\ell+1} \\ -1/|X_j^{\ell+1}|, & \text{if } v \in X_j^{\ell+1} \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

Figure 3.1a shows the graph wavelet transform for a toy example. For instance, the value of $a_{2,2} = (2.(-4 + (-6)) - 2.(-9 + (-9)))/4 = 4$ and the inverse $\varphi^{-1}W(e) = 0 + (-28)/4 + 4/2 + (-1)/1 = -6 = W(e)$. An important property of the graph wavelet transform, known as Parseval's relation, is that the signal and its transform are equivalent representations (i.e. $\varphi^{-1}\varphi W = W$) for any signal W and wavelet tree $\mathcal{X}(G)$. More formally, we can define the L_2 energy of a graph wavelet coefficient as:

$$||a_{k,\ell}||_2 = \frac{|X_i^{\ell+1}|.a_{k,\ell}^2}{|X_i^{\ell+1}|^2} + \frac{|X_j^{\ell+1}|.a_{k,\ell}^2}{|X_j^{\ell+1}|^2} = \frac{a_{k,\ell}^2}{|X_i^{\ell+1}|} + \frac{a_{k,\ell}^2}{|X_j^{\ell+1}|} \tag{3.5}$$

Using Equation 3.2, we can show the Parseval's relation:

$$\sum_k \sum_\ell ||a_{k,\ell}||_2 = \sum_v |W(v)|^2 \tag{3.6}$$

In particular, a lossy compressed representation of W can be constructed by the following procedure: (1) Compute transform φW, (2) set the lowest energy coefficients $a_{k,\ell}$ to 0, (3) return the non-zero wavelet coefficients $\varphi' W$ from φW. In this setting, the error of the compression is the sum of the energies of the dropped coefficients. If W has a sparse representation in the transform (frequency domain), where most of the energy is concentrated in a few high-level coefficients, it can be compressed with small error.

Figure 3.1a illustrates a sparse representation of a graph signal W (basis A). The fast decay of the difference coefficients $a_{k,\ell}$ in the wavelet transform as the level ℓ increases leads to a high compression using the aforementioned algorithm. The signal can be approximated within L_2 error of 1% using the top coefficients $a_{1,1}$ and $a_{2,2}$. However, by keeping the top coefficients for basis B (Figure 3.1b), the error is 22%.

In [27] (see theorems 1-3), the authors show that, if the energy of a wavelet coefficient $a_{k,\ell}$ is bounded as a function of the size of its corresponding vertex set X_k^ℓ and the tree $\mathcal{X}(G)$ is almost balanced, then there is a sparse representation of W as a wavelet transform. Here, we tackle the problem from a more practical and data-driven perspective, where a tree $\mathcal{X}(G)$ that leads to a sparse representation of W is unknown. Moreover, we add sparsity constraints to the description size of $\mathcal{X}(G)$ in order to enforce wavelet bases that are embedded in the graph structure. In the next section, we formalize the problem of computing wavelet basis using sparse cuts and characterize its hardness.

3.4 Wavelet Bases via Sparse Cuts

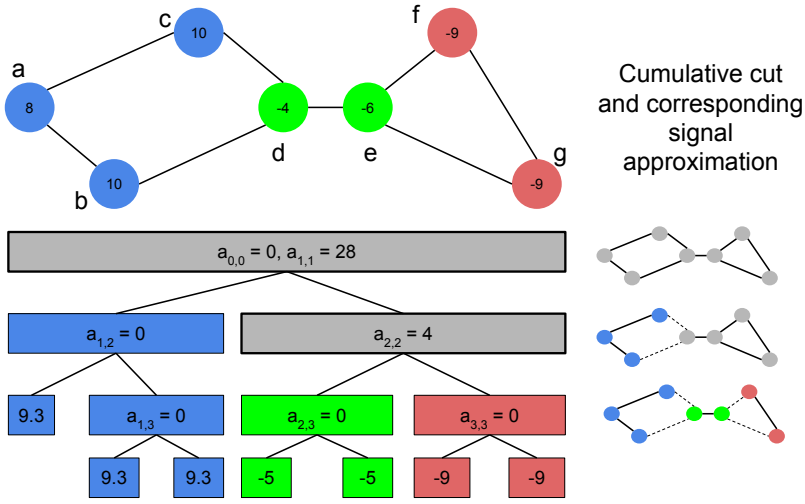
The existence of a good basis (or tree) for a signal W in a graph G provides relevant information about both W and G. We measure the description length of a wavelet tree $\mathcal{X}(G)$ as the size $|\mathcal{X}(G)|_E$ of its edge cut. The edge cut of a wavelet tree is the number of edges in the set $E' \subseteq E$ that, if removed, separates the leaf nodes of $\mathcal{X}(G)$. In other words, there is no path between any pair of vertices $u \in X_i^a$, $v \in X_j^b$ in $G(V, E - E')$ whenever X_i^a and X_j^b are leaves of $\mathcal{X}(G)$. A tree $\mathcal{X}(G)$ associated with a *sparse cut* requires a few edges to be removed in order to disconnect its leaf nodes.

If $|\mathcal{X}(G)|_E < |E|$, the energy of at least one coefficient a_k^ℓ of any transform φW will be always set to 0 and, as a consequence, the inverse $\varphi^{-1}\varphi W(v)$ will be the same for any vertex $v \in X_k^\ell$. As graphs have a combinatorial number of possible cuts, we formalize the problem of finding an optimal sparse wavelet basis in terms of (L_2) error minimization.

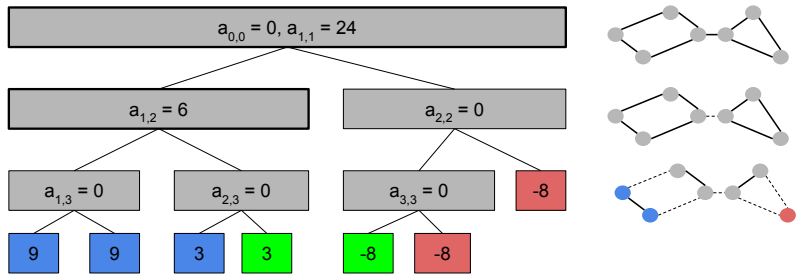
Definition 7 ***Optimal graph wavelet basis via sparse cuts.*** *For a given graph $G(V, E)$, a signal W, and a cut size q compute a wavelet tree $\mathcal{X}(G)$ with a cut $|\mathcal{X}(G)|_E$ of size q that minimizes $||W - \varphi^{-1}\varphi W||_2$.*

Figure 3.2 shows two candidate wavelet trees with cut size $q = 4$ for the same graph signal shown previously in Figure 3.1a. While the tree from Figure 3.2b achieves an error of 22%, the one from Figure 3.2a is the optimal basis of cut size 4 for our example, with an error of 1%. As discussed in Section 3.3, a good basis generates sparse transforms, which maximize the amount of energy from the signal that is conserved in a few coefficients. In the remainder of this section, we analyze the hardness of computing sparse wavelet bases by connecting it to well-known problems in graph theory.

Theorem 5 *Computing an optimal graph wavelet basis is NP-hard.*



(a) Optimal wavelet basis



(b) Alternative wavelet basis

Figure 3.2: Two graph wavelet bases with cut of size 4 for the same signal. Reconstructed values are set to leaf nodes. The basis from Figure 3.2a achieves 1% error and is optimal. An alternative basis with 22% error is shown in Figure 3.2b.

Please refer to the extended version of this work [57] for proofs of Theorems 5, 6 and 10. Theorem 5 shows that finding an optimal basis is NP-hard using a reduction from the 3-multiway cut problem [45], which leads to the question of whether such problem can be approximated within a constant factor in polynomial time. Theorem 6 shows that our problem is also NP-hard to approximate by any constant.

Theorem 6 *The optimal graph wavelet basis is NP-hard to approximate by a constant.*

Connecting the construction of sparse wavelet basis to a hard problem such as the

3-multiway cut is a key step for proving Theorems 5 and 6. However, these constructions assume wavelet trees $\mathcal{X}(G)$ with a number of levels ℓ strictly larger than 2 (i.e. with more than two partitions). A final question we ask regarding the hardness of our problem is whether there is an efficient algorithm for partitioning a set of nodes X_k^ℓ into children $X_i^{\ell+1}$ and $X_j^{\ell+1}$. If so, one could apply such an algorithm recursively in a top-down manner in order to construct a reasonably good wavelet basis. We can pose such a problem using the notion of L_2 energy of graph wavelet coefficients from Equation 3.5.

Definition 8 ***Optimal graph wavelet cut.*** *Given a graph $G(V, E)$, a signal W, a constant k, and a set of nodes $X_k^\ell \subseteq V$, compute a partition of X_k^ℓ into $X_i^{\ell+1}$ and $X_j^{\ell+1}$ that maximizes $||a_{k,\ell}||_2$.*

Theorem 7 rules out the existence of an efficient algorithm that solves the aforementioned problem optimally.

Theorem 7 *Computing an optimal sparse graph wavelet cut is NP-hard.*

Our proof (in the appendix) is based on a reduction from the graph bisection [53] and raises an interesting aspect of good graph wavelet bases, which is *balancing*. The problem of finding balanced partitions in graphs has been extensively studied in the literature, specially in the context of VLSI design [46], image segmentation [47] and other applications of spectral graph theory [55]. In the next section, we propose a spectral algorithm for computing graph wavelet bases.

3.5 Spectral Algorithm

Our approach combines structural and signal information as a vector optimization problem. By leveraging the power of spectral graph theory, we show how a relaxed

version of this formulation is a regularized eigenvalue problem, which can be solved using 1-D search and existing eigenvalue computation procedures. Our discussion focuses on computing a single cut (Definition 8) and extends to the computation of a complete basis. Section 3.5.3 is focused on performance.

3.5.1 Formulation

First, we introduce some notation. The degree d_v of a vertex v is the number of vertices $u \in V$ such that $(u, v) \in E$. The degree matrix $\mathcal{D}$ of G is an $n \times n$ diagonal matrix with $D_{v,v} = d_v$ for every $v \in V$ and $D_{u,v} = 0$, for $u \neq v$. The adjacency matrix A of G is an $n \times n$ matrix such that $A_{u,v} = 1$ if $(u, v) \in E$ and $A_{u,v} = 0$, otherwise[1]. The Laplacian of G is defined as $L = D - A$. We also define a second matrix $C = n\mathbf{I} - \mathbf{1}_{n\times n}$, where $\mathbf{I}$ is the identity matrix and $\mathbf{1}_{n\times n}$ is an $n \times n$ matrix of 1's. The matrix C can be interpreted as the Laplacian of a complete graph with n vertices. The third matrix, which we call S, is a matrix of pairwise squared differences with $S_{u,v} = (W(u) - W(v))^2$ for any pair of nodes $u, v \in V$. Notice that these matrices can also be computed for an induced subgraph $G'(X_k^\ell, E')$, where $E' = \{(u, v)|u \in X_k^\ell \wedge v \in X_k^\ell\}$.

In order to formulate the problem of finding an optimal sparse wavelet cut in vectorial form, we define a $|X_k^\ell|$ dimensional indicator vector x for the partition of X_k^ℓ into $X_i^{\ell+1}$ and $X_j^{\ell+1}$. For any $v \in X_k^\ell$, $x_v = -1$ if $v \in X_i^{\ell+1}$ and $x_v = 1$ if $v \in X_j^{\ell+1}$. By combining the matrices (C, S, L) and the indicator vector x, the following Theorem shows how the problem from Definition 8 can be rewritten as an optimization problem over vectors (see appendix for the proof).

Theorem 8 *The problem of finding an optimal sparse graph wavelet partition (Definition*

[1]Our method can be generalized to weighted graphs.

8) can be written as:

$$x* = \min_{x \in \{-1,1\}^n} a(x) \qquad st. \quad x^\intercal Lx \leq 4q \tag{3.7}$$

where $a(x) = \frac{x^\intercal CSCx}{x^\intercal Cx}$ *and* q *is the maximum cut size.*

Theorem 8 does not make the problem of computing an optimal wavelet basis any easier. However, we can now define a relaxed version of our problem by removing the constraint that $x_i \in \{-1, 1\}$. Once real solutions ($x_i \in \mathbb{R}$) are allowed, we can compute an approximate basis using eigenvectors of a well-designed matrix. The next corollary follows directly from a variable substitution and properties of Lagrange multipliers in eigenvalue problems [58, chapter-12].

Corollary 9 *A relaxed version of the problem from Definition 8 can be solved as a regularized eigenvalue problem:*

$$\begin{aligned} x* &= \min_x a(x) \\ &= \min_x \frac{x^\intercal CSCx}{x^\intercal Cx + \beta x^\intercal Lx} \\ &= ((C + \beta L)^+)^{\frac{1}{2}} y* \end{aligned} \tag{3.8}$$

where $y* = \min_y \frac{y^\intercal My}{y^\intercal y}$, $M = ((C+\beta L)^+)^{\frac{1}{2}} CSC((C+\beta L)^+)^{\frac{1}{2}}, y = (C+\beta L)^{\frac{1}{2}} x$, $(C+\beta L)^+$ *is the pseudoinverse of* $(C + \beta L)$ *and* β *is a regularization factor.*

This eigenvalue problem is well-defined due to properties of the matrix M, which is real and symmetric. In fact, M is negative semidefinite, since the energy $||a_{k,\ell}||_2$ of a wavelet coefficient is non-negative. We apply the pseudoinverse $(C+\beta L)^+$ because C and L are positive semidefinite and thus their standard inverses are not well-defined—they both have at least one zero eigenvalue.

At this point, it is not clear how the matrix M captures both signal and structural information as means to produce high-energy sparse wavelet cuts. In particular, we want to provide a deeper insight into the role played by the regularization factor β in preventing partitions that are connected by many edges in G. To simplify the notation and without loss of generality, let us assume that $X_k^\ell = V$ and that V has 0-mean. The next theorem gives an explicit form for the entries of M based on the node values and graph structure:

Theorem 10 *The matrix M is in the form:*

$$M_{ij} = 2n^2 \sum_{v=1}^{n} \left(\sum_{u=1}^{n} \left(\sum_{r=2}^{n} \left(\frac{1}{\sqrt{\lambda_r}} e_{r,i} e_{r,u} \right) W(u).W(v) \right) \sum_{r=2}^{n} \frac{1}{\sqrt{\lambda_r}} e_{r,v} e_{r,j} \right) \tag{3.9}$$

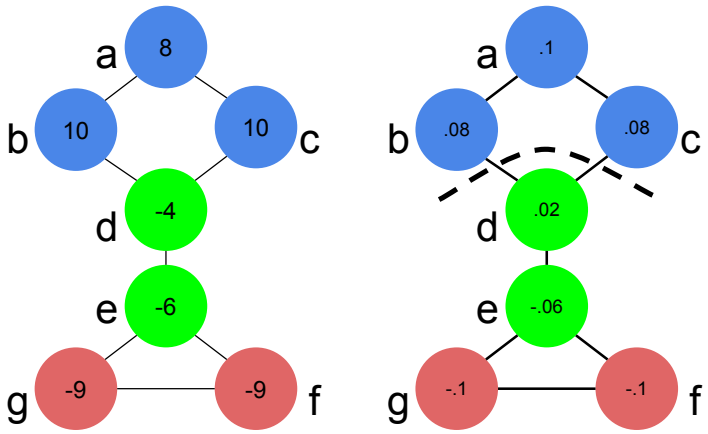
where (λ_r, e_r) is an eigenvalue-eigenvector pair of the matrix $(C + \beta L)$ such that $\lambda_r > 0$.

Based on Theorem 10, we can interpret M as a Laplacian regularized matrix and Expression 3.8 as a relaxation of a maximum-cut problem in a graph with Laplacian matrix $-M$. In this setting, the largest eigenvalue of $-M$ is known to be a relaxation of the maximum cut in the corresponding graph. The matrix $(C + \beta L)$ is the Laplacian of a graph G'' associated to G with the same set of vertices but edge weights $w_{u,v} = 1 + \beta$ if $(u, v) \in G$, and $w_{u,v} = 1$, otherwise. Intuitively, as β increases, G'' becomes a better representation of a weighted version of G with Laplacian matrix βL. For instance, if $\beta = 0$, G'' is a complete graph with all non-zero eigenvalues equal to n and G has no effect over the weights of the cuts in M. In other words, the wavelet cut selected will simply maximize the sum of (negative) products $-W(u).W(v)$ and separate nodes with different values. On the other hand, for large β, the eigenvalues λ_r will capture the structure of G and have a large magnitude. The relative importance of a product

$-W(u).W(v)$ will be reduced whenever u and v are well-connected to nodes i and j, respectively, in G. As a consequence, the cuts selected will rather cover edge pairs (i, j) for which far away nodes u and v in G have different values for the signal W.

Require: Graph G, values W, set X_k^ℓ, regularization constant β, cut size q
Ensure: Partitions $X_i^{\ell+1}$ and $X_j^{\ell+1}$
1: $C \leftarrow n \times n$ Laplacian of complete graph
2: $L \leftarrow n \times n$ Laplacian of G
3: $S \leftarrow n \times n$ squared difference matrix of G
4: $x* \leftarrow \min_x a(x)$
5: $(X_1, X_2)_z \leftarrow$ cut $(\{1, 2 \dots z\}, \{z+1 \dots n\})$
6: $(X_i^{\ell+1}, X_j^{\ell+1}) \leftarrow \max_{(X_1,X_2)_j} ||a_{k,\ell}||_2$ st. cut size $|(X_1, X_2)| \leq q$

Algorithm 4: Spectral Algorithm



(a) Graph signal (b) Eigenvector/cut

Figure 3.3: Example of a cut of size $q = 2$ found by the spectral algorithm. The eigenvector x is rounded using a sweep procedure and the best wavelet cut is selected.

Expressions in the form $\sum_r g(\lambda_r) e_i e_i^\intercal$ define regularizations via the Laplacian, which have been studied in the context of kernels on graphs [49, 50] and also wavelets [33, 51].

Notice that the regularization factor β is not known a priori, which prevents the direct solution of the relaxation given by Expression 3.8. However, we can apply a simple 1-D search algorithm (e.g. golden section search [59]) in order to compute an approximate optimal β within a range $[0, \beta_{max}]$.

$$(x*, \beta*) = \min_{\beta} \min_{x} a(x) \qquad st. \quad x^{\intercal} L x \leq 4q \tag{3.10}$$

3.5.2 Algorithm

Algorithm 4 describes our spectral algorithm for computing sparse graph wavelet cuts. Its inputs are the graph G, the signal W, a set of nodes X_k^ℓ from G, the regularization constant β, and the cut size q. As a result, it returns a cut $(X_i^{\ell+1}, X_j^{\ell+1})$ that partitions X_k^ℓ by maximizing the energy $||a_{k,\ell}||_2$ and has at most q edges. The algorithm starts by constructing matrices C, L and S based on G and W (lines 1-3). The best relaxed cut $x*$ is computed using Equation 3.8 (line 4) and a wavelet cut is obtained using a standard *sweeping approach* [47] (lines 5-6). Vertices in X_k^ℓ are sorted in non-decreasing order of their value in $x*$. For each value x_u, the algorithm generates a candidate cut $(X_1, X_2)_j$ by setting $x_v = -1$ if $v < u$, and $x_v = 1$, otherwise (line 5). The cut with size $|(X_i^{\ell+1}, X_j^{\ell+1})|$ at most q that maximizes the energy $||a_{k,\ell}||$ is selected among the candidate ones (line 6) and is returned by the algorithm.

Figure 3.3 illustrates a wavelet cut of size $q = 2$ discovered by our spectral algorithm. The input graph and its signal are given in Figure 3.3a. Moreover, we show the value of the eigenvector x that maximizes Expression 3.8 for each vertex and the resulting cut after rounding in Figure 3.3b. Notice that x captures both signal and structural information, assigning similar values to vertices that have small difference regarding the signal and are near in the graph. The energy $||a_{k,\ell}||_2$ associated with the cut is 457 (96% of the energy of the signal), which is optimal in this particular setting.

We evaluate Algorithm 4 using several datasets in our experiments. However, an open question is whether such an algorithm provides any quality guarantee regarding its solution (for a single cut). One approach would be computing a lower bound on the L_2

energy of the wavelet cuts generated by the rounding algorithm, similar to the *Cheeger's inequality* for the sparsest cut [55]. Unfortunately, proving such a bound has shown to be quite challenging and will be left as future work. For a similar proof regarding an approximation for the max-cut problem, please refer to [48].

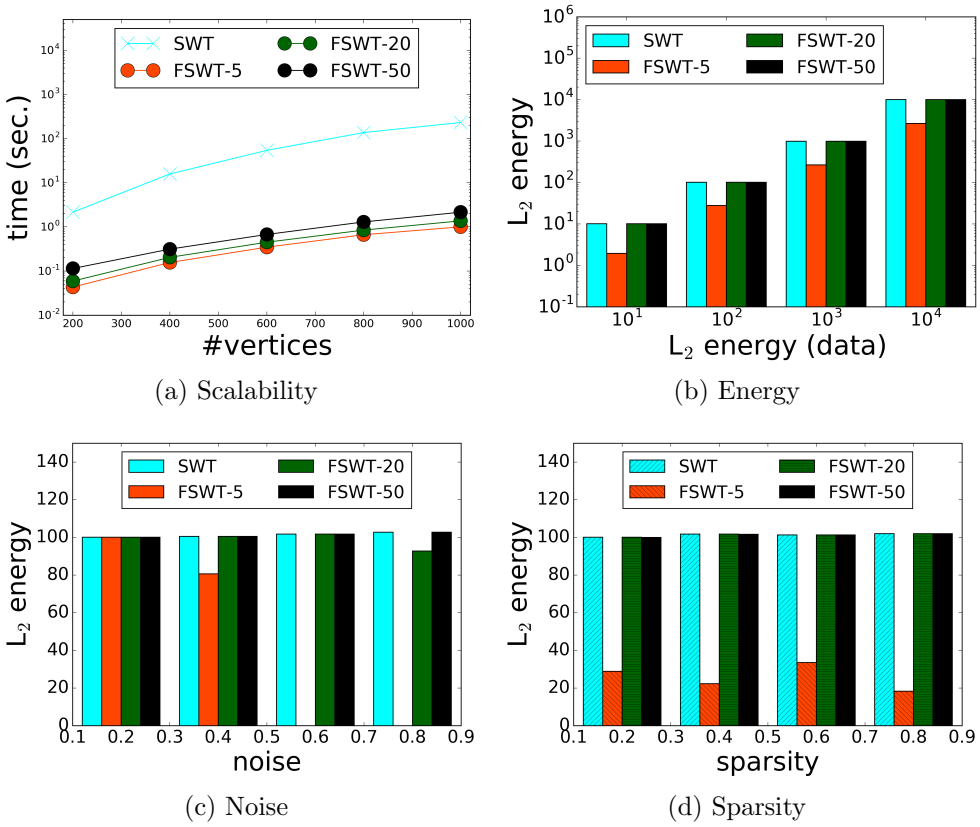
We apply Algorithm 4 recursively in order to construct a complete graph wavelet basis. Starting with the set of nodes V, we repeatedly compute new candidate wavelet cuts and select the one with maximum L_2 energy (i.e. it is a greedy algorithm). Once there is no feasible cut given the remaining budget of edges, we compute the remaining of the basis using ratio-cuts, which do not depend on the signal.

3.5.3 Efficient Approximation

Here, we study the performance of the algorithm described in the previous section and describe how it can be approximated efficiently. Although performance is not the main focus of this work, we still need to be able to compute wavelets on large graphs. The most complex step of Algorithm 4 is computing the matrix M (see Corollary 9), which involves (pseudo-)inverting and multiplying dense matrices. Moreover, the algorithm also requires the computation of the smallest eigenvalue/eigenvector of M.

A naive implementation of our spectral algorithm would take $O(n^3)$ time to compute the pseudo-inverse $(C + \beta L)^+$, $O(n^3)$ time for computing matrix products, and other $O(n^3)$ time for the eigen-decomposition of M. Assuming that the optimal value of β (Equation 3.10) is found in s iterations, the total complexity of this algorithm is $O(sn^3)$, which would hardly enable the processing of graphs with more than a few thousand vertices. Therefore, we propose a fast approximation of our algorithm by removing its dependence of β and using Chebyshev polynomials and the Power Method.

Our original algorithm searches for the optimal value of the regularization constant



(a) Scalability (b) Energy

(c) Noise (d) Sparsity

Figure 3.4: Scalability and L_2 energy associated to the cuts discovered by the sparse wavelet transform (**SWT**) and its fast approximation (**FSWT-p**) for different number of polynomial coefficients (p) and varying the graph size (a), the energy of the cut in the data (b), the noise level (c), and the sparsity of the cut (d) using synthetic datasets. Our fast approximation is up to 100 times faster than the original algorithm and achieves accurate results even when p is relatively small (20).

β using golden-search, which requires several iterations of Algorithm 4. However, our observations have shown that typical values of β found by the search procedure are large, even for reasonable values of q, compared to the number of edges in G. Thus, we propose simplifying Equation 3.8 to the following:

$$\frac{x^\intercal CSCx}{x^\intercal Lx} \tag{3.11}$$

As a consequence, we can compute a wavelet cut with a single execution of our

spectral algorithm. Using Theorem 10, we can show that dropping the matrix C from the denominator has only a small effect over the resulting matrix M. First, consider the eigenvalue-eigenvector pairs (λ_r, e_r) of $(C + \beta L)$ and let (λ_l, e_l) and (λ_c, e_c) be the eigenvalue-eigenvector pairs for non-zero eigenvalues of L and C, respectively. Given that C is the Laplacian of a complete graph, we know that $\lambda_c = n$, for any c, and every vector orthogonal to the constant vector $\mathbf{1}_n$ is an eigenvector of C. In particular, any eigenvector e_l of L is an eigenvector of C. Thus, we get that $(C + \beta L)e_l = (n + \beta\lambda_l)e_l$ and thus $(n + \beta\lambda_l, e_l)$ is an eigenvalue-eigenvector pair of $(C + \beta L)$.

Nevertheless, computing all the eigenvalues of the graph Laplacian L might still be prohibitive in practice. Thus, we avoid the eigen-decomposition by computing an approximated version of M using Chebyshev polynomials [33]. These polynomials can efficiently approximate an expression in the form $\langle \upsilon, f \rangle$, where $\upsilon_i = \sum_r g(\lambda_r) e_{r,i} e_{r,j}$ and f is a real vector. We can apply the same approach to approximate the product $((L^+)^{\frac{1}{2}} \times CSC)_{i,j}$ by setting g and f as:

$$g(\lambda_r) = \frac{1}{\sqrt{\lambda_r}}, \quad f = CSC_{:,j} \tag{3.12}$$

where $\lambda_r \in [1, n]$ and $:, j$ is an index for a matrix column.

Chebyshev polynomials can be computed iteratively with cost dominated by a matrix-vector multiplication by L. By truncating these polynomials to p terms (i.e. iterations), each one with cost $O(mn)$, where m is the number of edges, and n is the number of nodes, we can approximate this matrix product in $O(pmn)$ time. For sparse matrices $(m = O(n))$ and small p, $pmn \ll n^3$, which leads to significant performance gains over the naive approach. In order to compute M, we can repeat the same process with $f = ((L^+)^{\frac{1}{2}} \times CSC)_{j,:}$, where $j, :$ is an index for a matrix row.

Once the matrix M is constructed, it remains to compute its eigenvector associated

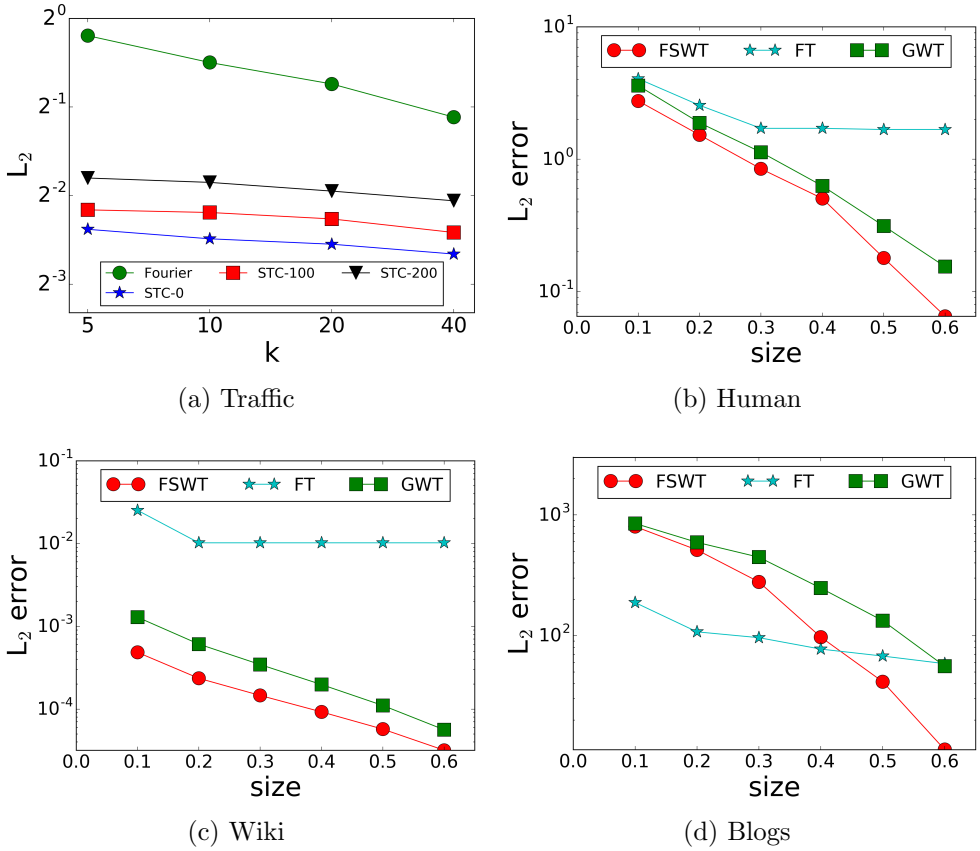
to the smallest eigenvalue. A trivial solution would be computing all the eigenvectors of M, which can be performed in time $O(n^3)$. However, due to the fact that our matrix is negative semidefinite, its smallest eigenvector can be approximated more efficiently using the Power Method [60], which requires a few products of a vector and M. Assuming that such method converges to a good solution in t iterations, we can approximate the smallest eigenvalue of M in time $O(tn^2)$. Moreover, the computation of x from y using $(L^+)^{\frac{1}{2}}$ can also be performed via Chebyshev polynomials in time $O(pm)$.

The time taken by our improved algorithm to compute a single cut is $O(pmn + tn^2)$, where p is the number of terms in the Chebyshev polynomial, $m = |E|$, $n = |V|$, and t is the number of iterations of the Power method. This complexity is a significant improvement over the $O(sn^3)$ time taken by its naive version whenever p, m, and t are small compared to n. For computing all the cuts, the total worst-case time complexity of the algorithm is $O(qpmn + qtn^2)$, where q is the size of the cut of the wavelet tree $\mathcal{X}(G)$. However, notice that good bases tend to be balanced (see Theorem 7) and in such case our complexity decreases to $O(pmn + tn^2)$.

3.6 Experiments

We evaluate our algorithms for computing sparse wavelet bases using synthetic and real datasets. We start by analyzing the scalability and quality of our efficient approximation compared to the original algorithm. Next, we compare our approach against different baselines and using four real datasets in the signal compression task. This section ends with some visualizations of the sparse wavelet formulation, which provides further insights into our algorithm. All the implementations are available as open-source and we also provide the datasets applied in this evaluation[2].

[2] `https://github.com/arleilps/sparse-wavelets`



(a) Traffic (b) Human

(c) Wiki (d) Blogs

Figure 3.5: Compression results for the *Traffic*, *Human*, *Wiki*, and *Blogs*. Our approach (FSWT) outperforms the baselines in most of the settings considered. In particular, FSWT achieves up to 80 times lower error than the best baseline (GWT).

3.6.1 Scalability and Approximation

The results discussed in this section are based on a synthetic data generator for both the graph and an associated signal. Our goal is to produce inputs for which the best wavelet cut is known. The data generator can be summarized in the following steps: (1) Generate sets of nodes V_1 and V_2 such that $|V_1| = |V_2|$; (2) Generate m edges such that the probability of an edge connecting vertices in V_1 and V_2 is given by a sparsity parameter h; (3) Assign average values μ_1 and μ_2 to V_1 and V_2, respectively, so that the energy of the cut (V_1, V_2) is equal to an energy parameter α; (4) Draw values from a Gaussian distribution $N(\mu_i, \sigma)$ for each vertex set V_i, where σ is a noise parameter.

Proper values for the averages are computed using Equation 3.5. We set default values for each parameter as follows: number of vertices $n = 500$ and edges $m = 3n$, sparsity $h = .5$, and noise $\sigma = |\mu_i|$. These parameters are varied in each experiment presented in Figure 3.4. For SWT, we fix the value of β_{max} in the golden search to 1000 and, for the fast approximation (FSWT), we vary the number of Chebyshev polynomials applied (5, 20, and 50). The number of iterations of the Power method to approximate the eigenvectors of M is fixed at 10, which achieved good results in our experiments. Figure 3.4a compares FSWT and the original algorithm (SWT) varying the graph size (n), showing that FSWT is up to 100 times faster than SWT. In Figures 3.4b-3.4d, we compare the approaches in terms of the energy $||a_{1,1}||_2$ of the first wavelet cut discovered varying the synthetic signal parameters. The results show that FSWT achieves similar or better results than SWT for relatively few coefficients ($p = 20$) in all the settings.

3.6.2 Compression

We evaluate our spectral algorithm for sparse wavelet bases in the signal compression task. Given a graph G and a signal W, the goal is to compute a compact representation W' that minimizes the L_2 error ($||W - W'||_2$). For the baselines, the *size* of the representation is the number of coefficients of the transform kept in the compression, relative to the size of the dataset. We also take into the account the representation cost of the cuts ($\log(m)$ bits/edge) for our approach.

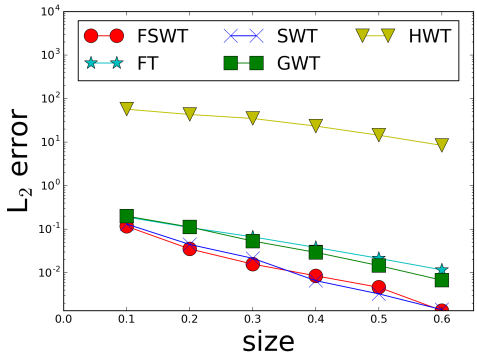
Datasets: Four datasets are applied in our evaluation. *Small Traffic* and *Traffic* are road networks from California for which vehicle speeds –measured by sensors– are modeled as a signal, with $n = 100$ and $m = 200$, and $n = 2K$ and $m = 6K$, respectively [61]. *Human* is a gene network for *Homo Sapiens* with expression values as a signal where $n = 1K$ and $m = 1K$ [19]. *Wiki* is a sample of Wikipedia pages where the (undirected)

link structure defines the graph and the signal is the number of page views for each page with $n = 5K$ and $m = 25K$. *Blogs* is a network of blogs with political leaning (-1 for left and 1 for right) as vertex attributes [62] ($n = 1K$ and $m = 17K$). Notice that these graphs have sizes in the same scale as the ones applied by existing work on signal processing on graphs [13, 27, 41]. We normalize the values to the interval $[0, 1]$ to make the comparisons easier.

Baselines: We consider the Graph Fourier Transform (FT) [30, 13] and the wavelet designs by Hammond et al. (HWT) [33] and Gavish et al. (GWT) [27] as baselines. Instead of the original bottom-up partitioning algorithm proposed for GWT, we apply ratio-cuts [46], which is more scalable and achieves comparable results in practice.

Figure 3.6a shows compression results for *Small Traffic*. The best baselines (GWT and FT) incur up to 5 times larger error than our approaches (SWT and FSWT). Figures 3.5a-3.5d show the results for FSWT, GWT, and FT using the remaining datasets. Experiments for HWT and SWT took too long to finish and were terminated. FSWT outperforms the baselines in most of the settings, achieving up to 5, 6, 2, and 80 times lower error than the best baseline (GWT) for *Traffic*, *Human*, *Wikipedia*, and *Blogs*, respectively. FT performs surprisingly well for *Blogs* because vertex values are almost perfectly separated into two communities, and thus some low frequency eigenvectors are expected to approximately match the separation (see [62, Fig. 3]). As the size of the representation increases, FSWT is the only method able to separate values at the border of the communities.

These results offer strong evidence that our sparse wavelet bases can effectively encode both the graph structure and the signal. The main advantage of our approach is building bases that are adapted to the signal by cutting few edges in the graph. The compression times of our algorithm are comparable with the baselines, as shown in Table 3.6b.



(a) Compression for *Small Traffic*

	Small Traffic	*Traffic*	*Human*	*Wiki*	*Blogs*
HWT	8	-	-	-	-
FT	1	35	2	381	7
GWT	1	5	11	386	47
SWT	1	-	-	-	-
FSWT	1	18	14	425	38

(b) Average compression times (in secs).

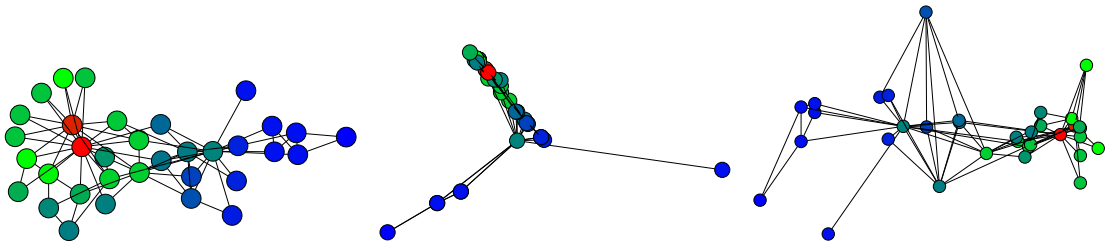
Figure 3.6: Compression results for *Small Traffic* and compression times for all methods and the datasets. Our approaches (SWT and FSWT) outperform the baselines while taking comparable compression time.

3.6.3 Visualization

Finally, we illustrate some interesting features of our sparse wavelet formulation using graph drawing. Eigenvectors of the Laplacian matrix are known to capture the community structure of graphs, and thus can be used to project vertices in space. In particular, if e_2 and e_3 are the second (*Fiedler*) and the third eigenvectors of the Laplacian matrix, we can draw a graph in 2-D by setting each vertex $v_i \in V$ to the position $(e_2(i), e_3(i))$. Following the same approach, we apply the smallest eigenvectors of the matrix M (see Corollary 9) to draw graphs based on both the structure and a signal.

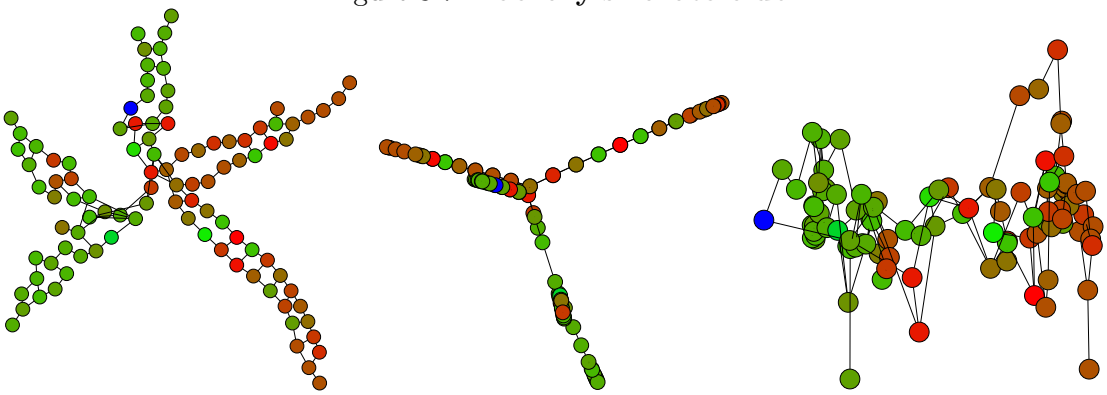
Figure 3.9 presents drawings for two graphs, one is the traditional Zachary's Karate network with a synthetic heat signal starting inside one community and the other is *Small Traffic*. Three different drawing approaches are applied: (1) The Scalable Force Directed Placement (SFDP) [63][3], the Laplacian eigenvectors, and the wavelet eigenvectors. Both

[3]Implemented by *GraphViz*: `http://www.graphviz.org/`



(a) SFDP (b) Laplacian (c) Wavelet

Figure 3.7: **Zachary's karate club.**



(a) SFDP (b) Laplacian (c) Wavelet

Figure 3.8: **Traffic.**

Figure 3.9: Drawing graphs using SFDP (a,d) and Laplacian (b,e) and wavelet eigenvectors (c,f). Vertices are colored based on values (red is high, green is average and blue is low). Different from the other schemes, wavelet eigenvectors are based on both signal and structure (better seen in color).

SFDP and the Laplacian are based on the graph structure only. The drawings demonstrate how our wavelet formulation separates vertices based on values and structure.

3.7 Conclusion

Signal Processing in Graphs (SPG) is a powerful framework for modeling complex data arising from several applications. A major challenge in SPG is relating properties of the graph signal, the graph structure and the transform. Graph wavelets are able to effectively model a smooth graph signal conditioned to the existence of a hierarchical

partitioning of the graph that captures the geometry of the graph structure as well as the signal. Our work is the first effort to build such hierarchies in a compact fashion. We first introduced the problem of computing graph wavelet bases via sparse cuts and show that it is NP-hard—even to approximate by a constant—by connecting it to existing problems in graph theory. Then, we proposed a novel algorithm for computing sparse wavelet bases by solving regularized eigenvalue problems using spectral graph theory. While naively considering both structure and values can lead to computationally intensive operations, we introduced an efficient solution using several techniques. These approaches are evaluated using several datasets and the results provide strong evidence that our solution produces compact and accurate representations for graph signals in practice.

This work opens several lines for future investigation: (i) It remains an open question whether approximating a single optimal wavelet cut is NP-hard. (ii) The wavelet design applied in this work maps only to a particular type of wavelets (*Haar*); extending our approach to other wavelet functions (e.g. *Mexican hat, Meyer* [31]) might lead to better representations for particular classes of signals. (iii) Generalizing the ideas presented here to time-varying graph signals might lead to novel algorithms for anomaly detection, event discovery, and data compression.

Chapter 4

Spectral Algorithms for Temporal Graph Cuts

4.1 Introduction

Temporal graphs represent how a graph changes over time, being ubiquitous in data mining and Web applications. Users in social networks present dynamic behavior, leading to the evolution of communities [64]. In hyperlinked environments, such as blogs, new topics drive modifications in content and link structure [65]. Communication, epidemics and mobility are other scenarios where temporal graphs can enable the understanding of complex processes. However, several key concepts and algorithms for static graphs have not been generalized to temporal graphs [66, 67].

This work focuses on cut problems in temporal graphs, which consist of finding a small sets of edges that break the graph into balanced sets of vertices. Two traditional graph cut problems are the *sparsest cut* [68, 46] and the *normalized cut* [47, 55]. In sparsest cuts, the resulting partitions are balanced in terms of size, while in normalized cuts, the balance is in terms of total degree (or volume) of the resulting sets. Graph cuts have

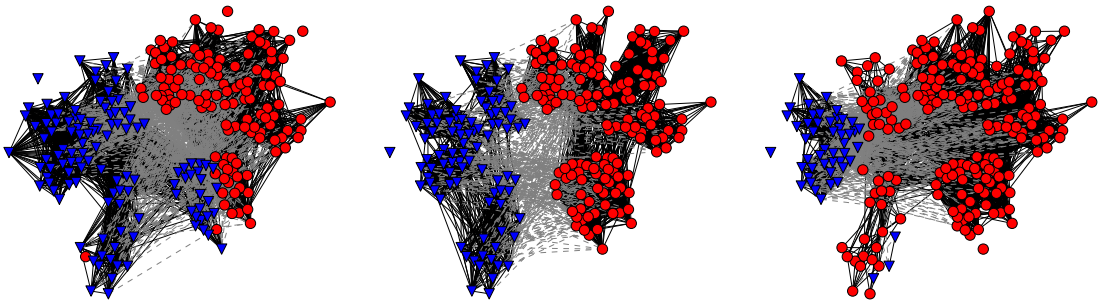
applications in community detection, image segmentation, clustering, and VLSI design. Moreover, the computation of graph cuts based on eigenvectors of graph-related matrices is one of the earliest results in spectral graph theory [55], a subject with great impact in information retrieval [69], graph sparsification [70], and machine learning [50].

One of our motivations to study graph cuts in this new setting is the emerging field of *Signal Processing on Graphs (SPG)* [13]. SPG is a framework for the analysis of data residing on vertices of a graph, as a generalization of traditional signal processing. Temporal cuts can be applied as bases for signal processing on dynamic graphs.

Our Contributions. We propose formulations of sparsest and normalized cuts in a sequence of graph snapshots. The idea is to extend classical definitions of these problems while enforcing the smoothness (or stability) of cuts over time. Our formulations can be understood using a multiplex view of the temporal graph, where additional edges connect the same vertex in different snapshots.

Figure 4.1 shows a sparse temporal graph cut for a school network [71], where children are connected based on proximity. Vertices are naturally organized into communities resulting from classes. However, there is a significant amount of interaction across classes (e.g. during lunch). Major changes in the contact network can be noticed during the experiment, causing several vertices to move across partitions—identified with vertex shapes/colors. The temporal cut is able to capture such trends while keeping the remaining vertex assignments mostly unchanged.

Traditional spectral solutions, which compute approximated cuts as rounded eigenvectors of the Laplacian matrix, do not generalize to our setting. Thus, we propose new algorithms, still within the framework of spectral graph theory, for the computation of temporal cuts. We further exploit important properties of our formulation to design efficient approximation algorithms for temporal cuts combining *divide-and-conquer* and low-rank matrix approximation.



(a) Time I (b) Time II (c) Time III

Figure 4.1: Temporal graph cut capturing major changes in the network interactions. Figure is better seen in color.

In order to also model dynamic data embedded on the vertices of a graph, we apply temporal cuts as data-driven wavelet bases. Our approach exploits smoothness in both space and time, illustrating how the techniques presented in this work provide a powerful and general framework for the analysis of dynamic graphs.

Summary of contributions.

- We generalize sparsest and normalized cuts to temporal graphs; we further extend temporal cuts to graph signals.
- We propose efficient approximate algorithms for temporal cuts via spectral graph theory and divide-and-conquer.
- We evaluate our methods extensively, applying them for community detection and signal processing on graphs.

Related Work. Computing graph cuts is a traditional problem [68, 72] with a diverse set of applications, ranging from image segmentation [47] to community detection [73]. We focus on the sparsest and normalized cut problems, which are of particular interest due to their connections with the spectrum of the Laplacian matrix, mixing time of random walks, geometric embeddings, effective resistance, and graph expanders [46, 70, 55].

Community detection in temporal graphs has attracted great interest in recent years. An evolutionary spectral clustering technique was proposed in [74]. The idea is to minimize a cost function $\alpha.CS + \beta.CT$, where CS is a snapshot cost and CT is a temporal cost. FacetNet [75] and estrangement [76] apply a similar approach under different clustering models. An important limitation of these solutions is that they perform community assignments in a step-wise manner, being highly subject to local optima. In incremental clustering [77, 78], the main goal is to avoid recomputation, and not to capture long-term structural changes. Multi-view clustering [79, 80] combines different subsets of features (or views) from a given dataset but does not model how objects navigate across clusters over time. Different from spatio-temporal data clustering [81, 82], we do not assume that our data is embedded in an Euclidean space.

A formulation for temporal modularity that simultaneously partitions snapshots using a multiplex graph [83] was proposed in [84]. A similar idea was applied in [85] to generalize eigenvector centrality. We propose generalizations for temporal cut problems by studying spectral properties of multiplex graphs [86, 87]. As one of our contributions, we exploit the link between multiplex graphs and block tridiagonal matrices to efficiently approximate temporal cuts [88, 89]. While extending spectral graph theory to tensors seems to be a more natural approach to our problems, eigenvectors are well-studied only for symmetric tensors [90], which is not our case due to the time dimension.

Our definition of temporal cuts is a special case of non-uniform cuts [91]—the second graph is a sequence of disconnected cliques to enforce cuts over time. Different from the general case, which requires more sophisticated (and computationally intensive) schemes [92], a relaxation for temporal cuts can be computed as an eigenvector of a linear combination of two matrices (see Theorem 11).

Signal processing on graphs [13, 2] is an interesting application of temporal cuts. Traditional signal processing operations are also relevant when the signal is embedded

into sparse irregular spaces. For instance, in machine learning, object similarity can be represented as a graph and labels as signals to solve semi-supervised learning tasks [27, 93]. In this work, we show how temporal cuts can be applied as bases for representing dynamic graph signals, even in the case where the graph structure also changes over time.

4.2 Temporal Graph Cuts

This section introduces temporal cuts (Section 4.2.1) and spectral algorithms for these problems (Section 4.2.2). Faster algorithms, using divide-and-conquer and low-rank matrix approximation, are presented in Section 4.2.3. We also discuss theoretical guarantees (Section 4.2.4) and generalizations for temporal cuts (Section 4.2.5).

4.2.1 Definitions

A temporal graph $\mathcal{G}$ is a sequence of snapshots $\langle G_1, G_2, \dots G_m \rangle$ where G_t is the snapshot at timestamp t. G_t is a tuple (V, E_t, W_t) where V is a fixed set with n vertices, E_t is a dynamic set of undirected edges and $W_t : E_t \to \mathbb{R}$ is an edge weighting function.

We model temporal graphs as *multiplex graphs*, which connect vertices from different graph layers. We denote as $\chi(\mathcal{G}) = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ the multiplex view of $\mathcal{G}$, where $\mathcal{V} = \{v_t | v \in V \wedge t \in [1, m]\}$ ($|\mathcal{V}| = nm$) and $\mathcal{E} = E_1 \cup \dots E_t \cup \{(v_t, v_{t+1}) | v \in V \wedge t \in [1, m-1]\}$. Thus, $\mathcal{E}$ also includes 'vertical' edges between nodes v_t and v_{t+1}. The edge weighting function $\mathcal{W} : \mathcal{E} \to \mathbb{R}$ is defined as:

$$\mathcal{W}(u_r, v_s) = \begin{cases} W_t(u, v), & \text{if } (u, v) \in E_t \wedge r = t \wedge s = t \\ \beta, & \text{if } u = v \wedge |r - s| = 1 \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

As a result, each vertex $v \in V$ has m representatives $\{v_1, \dots v_m\}$ in $\chi(\mathcal{G})$. Besides the intra-layer edges corresponding to the connectivity of each snapshot E_t, temporal edges (v_t, v_{t+1}) connect consecutive versions of a vertex v at different layers, which is known as *diagonal coupling* [94]. Intra-layer edge weights are the same as in $\mathcal{G}$ while inter-layer weights are set to β.

Sparsest Cut

A graph cut $(X, \overline{X})$ divides V into two disjoint sets: $X \subseteq V$ and $\overline{X} = V - X$. We denote the weight of a cut $|(X, \overline{X})| = \sum_{u \in X, v \in \overline{X}} W(u, v)$. The *cut sparsity* σ is the ratio of the cut weight and the product of the sizes of the sets [68]:

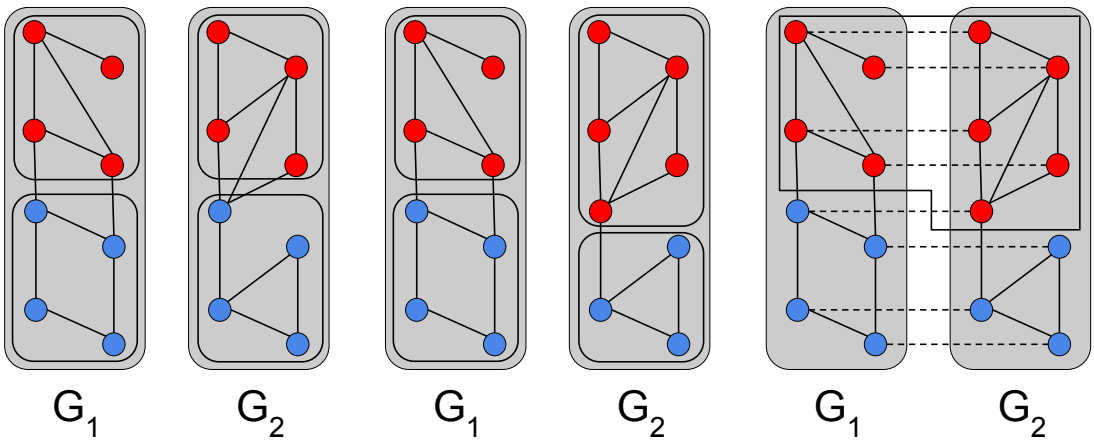
$$\sigma(X) = \frac{|(X, \overline{X})|}{|X||\overline{X}|} \tag{4.2}$$

Here, we extend the notion of cut sparsity to temporal graphs. A temporal cut $\langle (X_1, \overline{X}_1), \dots (X_m, \overline{X}_m) \rangle$ is a sequence of graph cuts where $(X_t, \overline{X}_t)$ is a cut of the graph snapshot G_t. The idea is that in temporal graphs, besides the cut weights and partition sizes, we also care about the smoothness (i.e. stability) of the cuts over time. We formalize the temporal cut sparsity σ as follows:

$$\sigma(X_1, \dots X_m; \beta) = \frac{\sum_{t=1}^{m} |(X_t, \overline{X}_t)| + \sum_{t=1}^{m-1} \Delta(X_t, \overline{X}_{t+1})}{\sum_{t=1}^{m} |X_t||\overline{X}_t|} \tag{4.3}$$

where $\Delta(X_t, \overline{X}_{t+1}) = \beta|(X_t, \overline{X}_{t+1})|$ is the number of vertices that move from X_t to $\overline{X}_{t+1}$ (or $|X_t \cap \overline{X}_{t+1}|$) times the constant β, which allows different weights to be given to the cut smoothness.

Figure 4.2 shows two alternative cuts for a temporal graph ($\beta = 1$). Cut I (Figure 4.2a) is smooth, since no vertex changes partitions, and it has weight 5. Cut II (Figure 4.2b) is a sparser temporal cut, with weight 3 and only one vertex changing partitions.



(a) Temporal cut I (b) Temporal cut II (c) Multiplex cut II

Figure 4.2: Two temporal cuts for the same graph and multiplex view of cut II. For $\beta = 1$, cut I has a sparsity of $(2+3+0)/(4\times4+4\times4) = 0.16$ while cut II has sparsity of $(2+1+1)/(4\times4+5\times3) = 0.13$. Thus, II is a sparser cut.

Notice that cut I becomes sparser than cut II if β is set to 2 instead of 1. We formalize the sparsest cut problem in temporal graphs as follows.

Definition 9 ***Sparsest temporal cut.*** *The sparsest cut of a temporal graph $\mathcal{G}$, for a constant β, is defined as:*

$$\arg\min_{X_1 \ldots X_m} \sigma(X_1, \ldots X_m; \beta)$$

The sparsest temporal cut is a generalization of the sparsest cut problem and thus also NP-hard [46].

An interesting property of the multiplex model is that temporal cuts in $\mathcal{G}$ become standard—single graph—cuts in the multiplex view $\chi(\mathcal{G})$. We can evaluate the sparsity of a cut in $\mathcal{G}$ by applying the original formulation (Expression 4.2) to $\chi(\mathcal{G})$, since both edges cut and partition changes in $\mathcal{G}$ become edges cut in $\chi(\mathcal{G})$. As an example, we

show the multiplex view of cut II (Figure 4.2b) in Figure 4.2c. However, notice that not every standard cut in $\chi(\mathcal{G})$ is a valid temporal cut. For instance, cutting all the temporal edges (i.e. separating the two snapshots in our example) would be allowed in the standard formulation, but would lead to an undefined value of sparsity as the denominator in Expression 4.3 will be 0. Therefore, we cannot directly apply existing sparsest cut algorithms to $\chi(\mathcal{G})$ and expect to achieve a sparse temporal cut for $\mathcal{G}$.

The connection between temporal cuts and multiplex networks is one of the main motivations for our formulation. Moreover, Equation 4.3 is general enough to capture different dynamic behaviors depending on the constant β. More specifically, if $\beta \to \infty$, the sparsity σ will be minimized for a constant cut over the snapshots. On the other hand, if $\beta \to 0$, σ approximates the sparsity of single snapshot cuts. These two extreme regimes have been studied in the context of random-walks on dynamic graphs [95].

Normalized Cut

A limitation of Equation 4.2, is that it favors sparsity over partition size balance. In community detection, this often leads to *"whisker communities"* [73, 96]. Normalized cuts [47] take into account the *volume* (i.e. sum of the degrees of vertices) of the partitions, being less prone to this effect. The normalized version of the cut sparsity is defined as:

$$\phi(X) = \frac{|(X, \overline{X})|}{vol(X).vol(\overline{X})} \tag{4.4}$$

where $vol(Y) = \sum_{v \in Y} deg(v)$ and $deg(v)$ is the degree of v.

We also generalize the normalized sparsity ϕ to temporal graphs:

$$\phi(X_1, \dots X_m; \beta) = \frac{\sum_{t=1}^{m} |(X_t, \overline{X}_t)| + \sum_{t=1}^{m-1} \Delta(X_t, \overline{X}_{t+1})}{\sum_{t=1}^{m} vol(\overline{X}_t) vol(X_t)} \tag{4.5}$$

Next, we define the normalized cut problem for temporal graphs.

Definition 10 ***Normalized temporal cut***. *The normalized temporal cut of $\mathcal{G}$, for a constant β, is defined as:*

$$\arg\min_{X_1 \dots X_m} \phi(X_1, \dots X_m; \beta)$$

Computing optimal normalized temporal cuts is also NP-hard. The next section introduces spectral approaches for temporal cuts.

4.2.2 Spectral Approaches

Similar to the single graph case, we also exploit spectral graph theory in order to compute good temporal graph cuts. Let A be an $n \times n$ weighted adjacency matrix of a graph $G(V, E)$, where $A_{u,v} = W(u, v)$ if $(u, v) \in E$ or 0, otherwise. The degree matrix D is a an $n \times n$ diagonal matrix, with $D_{v,v} = deg(v)$ and $D_{u,v} = 0$ for $u \neq v$. The Laplacian matrix of G is defined as $L = D - A$. Let L_t be the Laplacian matrix of the graph snapshot G_t and I_z be an $z \times z$ identity matrix. We define the Laplacian of the temporal graph $\mathcal{G}$ as the Laplacian of its multiplex view $\chi(\mathcal{G})$:

$$\mathcal{L} = \begin{pmatrix} L_1 + \beta I_n & -\beta I_n & 0 & \dots & 0 \\ -\beta I_n & L_2 + 2\beta I_n & -\beta I_n & \dots & 0 \\ \vdots & & \ddots & \dots & -\beta I_n \\ 0 & 0 & \dots & -\beta I_n & L_m + \beta I_n \end{pmatrix}$$

The matrix $\mathcal{L}$ can be also written in a more compact form using the Kronecker product as $\mathbf{diag}(L_1, \dots L_m) + \beta(L^\ell \otimes I_n)$, where L^ℓ is the Laplacian of a line graph. Similarly, we define the degree matrix $\mathcal{D}$ of $\mathcal{G}$ as $\mathbf{diag}(D_1, D_2 \dots D_m)$, where D_t is the degree matrix of G_t. Let $C = nI_n - \mathbf{1}_{n \times n}$ be the Laplacian of a clique with n vertices. We define another

$nm \times nm$ Laplacian matrix $\mathcal{C} = I_m \otimes C$, which is the Laplacian of a graph composed of m isolated cliques. This matrix will be applied to enforce valid temporal cuts over the snapshots of $\mathcal{G}$. Further, we define a size-nm indicator vector $\mathbf{x}$ where each vertex $v \in V$ is represented m times, one for each snapshot. The value $\mathbf{x}[v_t] = 1$ if $v_t \in X_t$ and $\mathbf{x}[v_t] = -1$ if $v_t \in \overline{X}_t$.

Sparsest Cut

The next lemma shows how the matrices $\mathcal{L}$ and $\mathcal{C}$ can be applied to compute the sparsity of temporal cuts.

Lemma 4.2.1 *The sparsity σ of a temporal cut is equal to $\frac{\boldsymbol{x}^\intercal \mathcal{L} \boldsymbol{x}}{\boldsymbol{x}^\intercal \mathcal{C} \boldsymbol{x}}$.*

Proof: Since $\mathcal{L}$ is the Laplacian of $\chi(\mathcal{G})$:

$$\begin{aligned}\mathbf{x}^\intercal \mathcal{L} \mathbf{x} &= \sum_{(u,v)\in\mathcal{E}} (\mathbf{x}[u] - \mathbf{x}[v])^2 W(u,v) \\ &= 4\sum_{t=1}^{m} |(X_t, \overline{X}_t)| + 4\sum_{t=1}^{m-1} \Delta(X_t, \overline{X}_{t+1})\end{aligned}$$

Regarding the denominator:

$$\begin{aligned}\mathbf{x}^\intercal \mathcal{C} \mathbf{x} &= \sum_{t=1}^{m} \sum_{u \neq v} (\mathbf{x}[v_t] - \mathbf{x}[u_t])^2 \\ &= 4\sum_{t=1}^{m} |X_t||\overline{X}_t|\end{aligned}$$

■

Based on Lemma 4.2.1, we can obtain a relaxed solution, $\mathbf{x} \in [-1, 1]^{nm}$, for the sparsest temporal cut problem (Definition 9) in $O(n^3m^3)$ time via generalized eigenvalue computation [60], or an approximate solution in $O(n^2 m \log^2(n^2 m))$ time using a fast

Laplacian linear system solver [97, 98]. Solutions are later rounded to adhere to the original (discrete) constraint $\mathbf{x} \in \{-1, 1\}^{nm}$. The next lemma supports a more efficient solution for our relaxation.

Lemma 4.2.2 *The matrices $\mathcal{C}$ and $\mathcal{L}$ commute.*

Proof: First, we show that C commutes with any Laplacian L_t:

$$CL_t = (nI_n - \mathbf{1}_{n\times n})L_t = nL_t = L_tC$$

Using the Kronecker product notation (see Section 4.2.2):

$$\begin{aligned}\mathcal{C}\mathcal{L} &= (I_m \otimes C)[\mathbf{diag}(L_1, \dots L_m) - \beta(L^\ell \otimes I_n)] \\ &= \mathbf{diag}(CL_1, \dots CL_m) - \beta(I_mL^\ell) \otimes (CI_n) \\ &= \mathbf{diag}(L_1C, \dots L_mC) - \beta(L^\ell I_m) \otimes (I_nC) \\ &= \mathcal{L}\mathcal{C}\end{aligned}$$

■

A relaxation of the sparsest temporal cut can be computed based on a linear combination of matrices $\mathcal{C}$ and $\mathcal{L}$.

Theorem 11 *A relaxed solution for the sparsest temporal cut problem can, alternatively, be computed as:*

$$\boldsymbol{y}* = \underset{\boldsymbol{y}\in[-1,1]^{nm}}{\arg\max} \frac{\boldsymbol{y}^\intercal[(3(n+2\beta)\mathcal{C} - \mathcal{L}]\boldsymbol{y}}{\boldsymbol{y}^\intercal\boldsymbol{y}} \tag{4.6}$$

which is the largest eigenvector of $3(n+2\beta)\mathcal{C} - \mathcal{L}$.

Proof:

The spectrum of C is in the form $(\mathbf{e}_1, \lambda_1) = (\mathbf{1}_n, 0)$ and $\lambda_2 = \ldots = \lambda_n = n$ for any vector $\mathbf{e}_i \perp \mathbf{1}_n$. As a consequence, the spectrum of $\mathcal{C}$ is in the form $\lambda_1 = \ldots = \lambda_m = 0$

and $\lambda_{m+1} = \dots \lambda_{nm} = n$ for any $\mathbf{e}_i \perp \mathbf{span}\{e_1, \dots e_m\}$. Lemma 4.2.2 implies that any linear combination of $\mathcal{C}$ and $\mathcal{L}$ has the same eigenvectors as $\mathcal{L}$ [99]. Upper bounding the eigenvalues of a Laplacian matrix [100]:

$$0 \leq \frac{\mathbf{y}^\intercal \mathcal{L} \mathbf{y}}{\mathbf{y}^\intercal \mathbf{y}} \leq 2(n + 2\beta) \tag{4.7}$$

This implies that:

$$\frac{(\mathbf{y}*)^\intercal \mathcal{C} (\mathbf{y}*)}{(\mathbf{y}*)^\intercal (\mathbf{y}*)} = n \tag{4.8}$$

as it guarantees a strictly positive ratio (i.e. eigenvalue) in Equation 4.6. Based on Equation 4.8, we can re-write $\mathbf{y}*$ as:

$$\mathbf{y}* = \underset{\mathbf{y} \in [-1,1]^{nm}, \mathbf{y}^\intercal \mathcal{C} \mathbf{y} > 0}{\arg\min} \frac{\mathbf{y}^\intercal \mathcal{L} \mathbf{x}}{\mathbf{y}^\intercal \mathbf{y}} \tag{4.9}$$

Moreover, from Lemma 4.2.1:

$$\mathbf{x}* = \underset{\mathbf{x} \in [-1,1]^{nm}, \mathbf{x}^\intercal \mathcal{C} \mathbf{x} > 0}{\arg\min} \frac{\mathbf{x}^\intercal \mathcal{L} \mathbf{x}}{\mathbf{x}^\intercal \mathcal{C} \mathbf{x}} \tag{4.10}$$

Equations 4.9 and 4.10 are related to an *eigenvalue* and a *generalized eigenvalue problem*, respectively, and can be written as follows:

$$\mathcal{L}\mathbf{y} = \lambda \mathbf{y}, \qquad \mathcal{L}\mathbf{x} = \lambda' \mathcal{C} \mathbf{x} \tag{4.11}$$

where λ and λ' are minimized and $\mathbf{y}^\intercal \mathcal{C} \mathbf{y}, \mathbf{x}^\intercal \mathcal{C} \mathbf{x} > 0$. From Equation 4.8, we know that $\mathcal{C}\mathbf{y} = n\mathbf{y}$. Thus, $\mathcal{L}\mathbf{y} = (\lambda/n)\mathcal{C}\mathbf{y}$ is a corresponding solution (same eigenvector) to the generalized problem. ■

The matrix $3(n + 2\beta)\mathcal{C} - \mathcal{L}$ is a Laplacian of a multiplex graph in which temporal edges have weight $w'(v_t, v_{t+1}) = -\beta$ and intra-layer edges have weight $w'(u, v) = 3(n +$

$2\beta) - w(u, v)$. This leads to a reordering of the spectrum of $\mathcal{L}$ where cuts containing only temporal edges have negative associated eigenvalues and sparse cuts for each Laplacian L_t become dense cuts for a new Laplacian $[3(n + 2\beta))C - L_t]$. In terms of complexity, computing the difference $3(n + 2\beta)\mathcal{C} - \mathcal{L}$ takes $O(n^2m)$ time and the largest eigenvector of $[3(n+2\beta)\mathcal{C}-\mathcal{L}]$ can be calculated in $O(n^2m)$. The resulting complexity is a significant improvement over the $O(n^3m^3)$ alternative, specially if the number of snapshots is large.

Normalized Cut

We follow the steps of the previous section to compute normalized temporal cuts. *See the extended version of this work [101] for proofs of Lemma 4.2.3 and Theorem 12.*

Lemma 4.2.3 *The normalized sparsity ϕ of a temporal cut is equal to* $\frac{\boldsymbol{x}^\intercal \mathcal{L} \boldsymbol{x}}{\boldsymbol{x}^\intercal \mathcal{D}^{\frac{1}{2}} \mathcal{C} \mathcal{D}^{\frac{1}{2}} \boldsymbol{x}}$.

We also define an equivalent of Theorem 11 for normalized cuts.

Theorem 12 *A relaxed solution for the sparsest temporal cut problem can, alternatively, be computed as:*

$$\boldsymbol{y}* = \underset{\boldsymbol{y} \in [-1,1]^{nm}}{\arg\max} \frac{\boldsymbol{y}^\intercal [(3(n + 2\beta)\mathcal{C} - (\mathcal{D}^+)^{\frac{1}{2}} \mathcal{L} (\mathcal{D}^+)^{\frac{1}{2}}] \boldsymbol{y}}{\boldsymbol{y}^\intercal \boldsymbol{y}} \tag{4.12}$$

the largest eigenvector of $3(n + 2\beta)\mathcal{C} - (\mathcal{D}^+)^{\frac{1}{2}} \mathcal{L} (\mathcal{D}^+)^{\frac{1}{2}}$.

The interpretation of matrix $3(n + 2\beta)\mathcal{C} - (\mathcal{D}^+)^{\frac{1}{2}} \mathcal{L} (\mathcal{D}^+)^{\frac{1}{2}}$ is similar to the one for sparsest cuts, with temporal edges having negative weights. Moreover, the complexity of computing the largest eigenvector of such matrix is also $O(n^2m)$. This quadratic cost on the size of the graph, for both sparsest and normalized cut problems, becomes prohibitive even for reasonably small graphs. The next section is focused on faster algorithms for temporal graph cuts.

4.2.3 Fast Approximations

By definition, sparse temporal cuts are sparse in each snapshot and smooth across snapshots. Similarly, normalized temporal cuts are composed of a sequence of good normalized snapshot cuts that are stable over time. This motivates *divide-and-conquer* approaches for computing temporal cuts that first find a good cut on each snapshot (*divide*) and then combine them (*conquer*). These solutions have the potential to be much more efficient than the ones based on Theorems 11 and 12 if the conquer step is fast. However, they could lead to sub-optimal results, as optimal temporal cuts might not be composed of each snapshot's best cuts. Instead, better *divide-and-conquer* schemes can explore multiple snapshot cuts in the conquer step to avoid local optima. Since we are working in the spectral domain, it is natural to take eigenvectors of blocks of $\mathcal{M}_S = 3(n+2\beta)\mathcal{C} - \mathcal{L}$ and $\mathcal{M}_N = 3(n+2\beta)\mathcal{C} - (\mathcal{D}^+)^{\frac{1}{2}}\mathcal{L}(\mathcal{D}^+)^{\frac{1}{2}}$, as continuous notions of snapshot cuts. This strategy is supported by the well-known connections between higher-order eigenvectors of the Laplacian matrix and the sparsity of multiway cuts [102, 103].

This section will describe our general *divide-and-conquer* approach. We will focus our discussion on the sparsest cut problem and then briefly show how it can be generalized to normalized cuts. The following theorem is the basis of our algorithm.

Theorem 13 *The eigenvalues of the matrix $\mathcal{M}_S = 3(n+2\beta)\mathcal{C} - \mathcal{L}$ are the same as the ones for the matrix $\mathcal{Q}$:*

$$\mathcal{Q} = \mathbf{\Lambda} - \beta \begin{pmatrix} I_n & -U_1^\intercal U_2 & 0 & \dots & 0 \\ -U_2^\intercal U_1 & 2I_n & -U_2^\intercal U_3 & \dots & 0 \\ \vdots & & \ddots & \dots & U_{m-1}^\intercal U_m \\ 0 & 0 & \dots & U_m^\intercal U_{m-1} & I_n \end{pmatrix}$$

where $U_t \Lambda_t U_t^\intercal$ is the eigendecomposition of the matrix $M_t = (3(n+2\beta)C - L_t)$ and $\mathbf{\Lambda} =$

$\boldsymbol{diag}(\Lambda_1 \dots \Lambda_m)$. *An eigenvector* $\boldsymbol{e}_j$ *of* $\mathcal{M}_S$ *is computed as* $\mathcal{U}.\boldsymbol{e}_j^{\mathcal{Q}}$, *where* $\mathcal{U} = \boldsymbol{diag}(U_1 \dots U_m)$ *and* $\boldsymbol{e}_j^{\mathcal{Q}}$ *is an eigenvector of* $\mathcal{Q}$.

Proof: We show that $\mathcal{M}_S$ and $\mathcal{Q}$ are similar matrices under the change of basis $\mathcal{U}^\intercal$ and thus $\mathcal{M} = (\mathcal{U}^\intercal)^{-1}\mathcal{Q}\mathcal{U}^\intercal$. Let's define matrices $\mathcal{B} = \beta(L^\ell \otimes I_n)$ and $M_t = 3(n - 2\beta)C - L_t$. Because L_t is symmetric, $\mathcal{U}^{-1} = \mathcal{U}^\intercal$. For an eigenvector matrix $\mathcal{U}$, $\mathcal{U}\mathcal{U}^\intercal$ is an $nm \times nm$ identity matrix $\mathcal{I}$. We rewrite $\mathcal{M}_S$ as:

$$\begin{aligned}
\mathcal{M}_S &= \mathbf{diag}(M_1, M_2 \dots M_m) - \mathcal{B} \\
&= \mathcal{U}\mathbf{\Lambda}\mathcal{U}^\intercal - \mathcal{I}\mathcal{B}\mathcal{I} \\
&= \mathcal{U}\mathbf{\Lambda}\mathcal{U}^\intercal - \mathcal{U}\mathcal{U}^\intercal\mathcal{B}\mathcal{U}\mathcal{U}^\intercal \\
&= \mathcal{U}(\mathbf{\Lambda} - \mathcal{U}^\intercal\mathcal{B}\mathcal{U})\mathcal{U}^\intercal \\
&= (\mathcal{U}^\intercal)^{-1}\mathcal{Q}\mathcal{U}^\intercal
\end{aligned}$$

■

$\mathcal{Q}$ has $O(n^2m)$ non-zeros, being asymptotically as sparse as $\mathcal{M}_S$. However, $\mathcal{Q}$ can be block-wise sparsified using low-rank approximations of the matrices M_t. Given a constant $r \leq n$, we approximate each M_t as $U_t\Lambda'_tU_t^\intercal$, where Λ'_i contains only the top-r eigenvalues of M_t. The benefits of such a strategy are the following: (1) The cost of computing the eigendecomposition of M_t changes from $O(n^3)$ to $O(rn^2)$; (2) the cost of multiplying eigenvector matrices decreases from $O(n^3)$ to $O(r^3)$; and (3) the number of non-zeros in $\mathcal{Q}$ is reduced from $O(n^2m)$ to $O(r^2m)$. Similar to the case of general block tridiagonal matrices [89], we can show that the error associated with such approximation is bounded by $2\lambda_{r+1}^{max}$, where λ_{r+1}^{max} is the largest $(r+1)$-nth eigenvalue of the approximated matrices M_t.

We improve our approach by speeding-up the eigendecomposition of the matrices M_t. The idea is to operate over the original Laplacians L_t, which are expected to be sparse.

The eigendecomposition of a matrix with $|E|$ non-zeros can be performed in time $O(n|E|)$ and for real-world graphs $|E| << n^2$. The following Lemma shows how the spectrum of M_t can be computed based on L_t.

Require: Temporal graph $\mathcal{G}$, rank r, constant β
Ensure: Temporal cut $\langle (X_1, \overline{X}_1), \dots (X_m, \overline{X}_m) \rangle$
1: **for** $G_t \in \mathcal{G}$ **do**
2: Compute bottom-r eigendecomposition $U'_t \Lambda'^L_t U'^\intercal_t \approx L_t$
3: Fix $\Lambda_t \leftarrow \mathbf{diag}(0, 3(n+2\beta)n - \lambda_n, \dots 3(n+2\beta)n - \lambda_2)$
4: **end for**
5: $\mathcal{Q} \leftarrow \mathbf{diag}(\Lambda_1, \dots \Lambda_m) - \mathcal{B}$
6: **for** $t \in [1, m-1]$ **do**
7: $\mathcal{Q}_{t,t+1} \leftarrow U'^\intercal_t U'_t, \quad \mathcal{Q}_{t+1,t} \leftarrow U'^\intercal_t U'_t$
8: **end for**
9: Compute largest eigenvector $\mathbf{x*} \leftarrow \arg\max_{\mathbf{x}} \mathbf{x}^\intercal \mathcal{Q}\mathbf{x} / \mathbf{x}^\intercal \mathbf{x}$
10: **return** rounded cut $\mathbf{sweep}(\mathcal{U}.\mathbf{x}*, \mathcal{G}, \beta)$

Algorithm 5: Spectral Algorithm

Lemma 4.2.4 *Let $\lambda^L_1, \lambda^L_2 \dots \lambda^L_n$ be the eigenvalues of a Laplacian matrix L in increasing order with associated eigenvectors $e^L_1, e^L_2 \dots e^L_n$. The eigenvectors e^L_i are also eigenvectors of $3(n+2\beta)C - L$ with eigenvalues $\lambda_1 = 0$ and $\lambda_i = 3(n+2\beta)n - \lambda_{n-i+1}$ for $i > 0$.*

Proof: The spectrum of C is $(\mathbf{e}_1, \lambda_1) = (\mathbf{1}_n, 0)$ and $\lambda_2 = \dots = \lambda_n = n$ for any vector $\mathbf{e}_i \perp \mathbf{1}_n$. As M_i is also a Laplacian, it follows that $\lambda_1 = 0$ and $e_1 = e^L_1$. Also, by definition $L.e^L_i = \lambda^L_i.e^L_i$, and thus $(3(n+2\beta)C - L)e^L_i = (3(n+2\beta)n - \lambda_i)e^L_i$ for $i > 0$. ■

Algorithm 5 describes our *divide-and-conquer* approach for approximating the sparsest temporal cut. Its inputs are the temporal graph $\mathcal{G}$, the rank r that controls the accuracy of the algorithm, and a constant β. It returns a cut $\langle (X_1, \overline{X}_1) \dots (X_m, \overline{X}_m) \rangle$ that (approximately) minimizes the sparsity ratio defined in Equation 4.3. In the *divide* phase, the top-r eigenvalues/eigenvectors of each matrix M_t—related to the bottom-r

eigenvalues/eigenvectors of L_t—are computed using Lemma 4.2.4 (steps 1-4). The *conquer* phase (steps 5-9) consists of building the matrix $\mathcal{Q}$—based on the blocks $\mathcal{Q}_{t,t+1}$ adjacent to the diagonal—and then computing its largest eigenvector as a relaxed version of a temporal cut. The resulting eigenvector is discretized using a standard sweep algorithm (**sweep**) over the vertices sorted by their corresponding value of $\mathbf{x}^*$. The selection criteria for the sweep algorithm is the sparsity ratio (Equation 4.3).

The time complexity of our algorithm is $O(mr\sum_{t=1}^{m}|E_t| + mr^3)$. The *divide* step has cost $O(mr\sum_{t=1}^{m}|E_t|)$, which corresponds to the computation of r eigenvectors (and eigenvalues) of matrices L_t with $O(|E_t|)$ non-zeros each. As snapshots are processed independently, this part of the algorithm can be easily parallelized. In the *conquer* step, the most time consuming operation is computing $m-1$ $r \times r$ matrix products in the construction of $\mathcal{Q}$, which takes $O(r^3m)$ time. Our algorithm has space complexity of $O(r^2m)$ due to the number of non-zeros in the sparse representation of $\mathcal{Q}$.

We follow the same general approach discussed in this section to efficiently compute normalized temporal cuts. As in Theorem 13, we can compute the eigenvectors of $\mathcal{M}_N$ using divide-and-conquer. However, each block M_t will be in the form $3(n-2\beta)C - (D_t^+)^{\frac{1}{2}}L_t(D_t^+)^{\frac{1}{2}}$. Moreover, similar to Lemma 4.2.4, we can also compute the eigendecomposition of M_t based on $(D_t^+)^{\frac{1}{2}}L_t(D_t^+)^{\frac{1}{2}}$.

4.2.4 Approximation Guarantees

The algorithms presented in Sections 4.2.2 and 4.2.3 are based on eigenvector computations that are relaxations of temporal cut problems. A natural question to ask is: *Do they provide any approximation guarantees with respect to the optimal solution for the problems?* Notice that, for the fast solutions discussed in the previous section, the number of top eigenvalues (r) considered by Algorithm 5 gives some control over the

quality of the approximations. Therefore, we focus on bounding the error induced by the relaxations described in Lemmas 4.2.1 and 4.2.3 and by Theorems 11 and 12.

Our analysis is heavily based on a recent generalization of the Cheeger inequality [104, 92], which relates the ratio between the weights of the same cut on two graphs, G and H, with the same set of vertices, and the generalized eigenvalue involving their Laplacian matrices L_G and L_H. This generalization is the main ingredient to proof the following Lemma regarding our spectral algorithm for the sparsest temporal cut problem (Definition 9):

Lemma 4.2.5 *The temporal sparsity ratio $\lambda(\mathcal{G})$ achieved by our relaxation is such that:*

$$\lambda(\mathcal{G}) \geq \frac{\varphi(\chi(\mathcal{G}))\min(\sigma_{X_1,\dots X_m}(X_1,\dots X_m;\beta))}{8} \tag{4.13}$$

where $\lambda(\mathcal{G}) = \min_{\boldsymbol{x}^\intercal \mathcal{C}\boldsymbol{x}>0} \frac{\boldsymbol{x}^\intercal \mathcal{L}\boldsymbol{x}}{\boldsymbol{x}^\intercal \mathcal{C}\boldsymbol{x}}$ and $\varphi(\chi(\mathcal{G}))$ is the (standard) conductance of the multiplex view of $\mathcal{G}$.

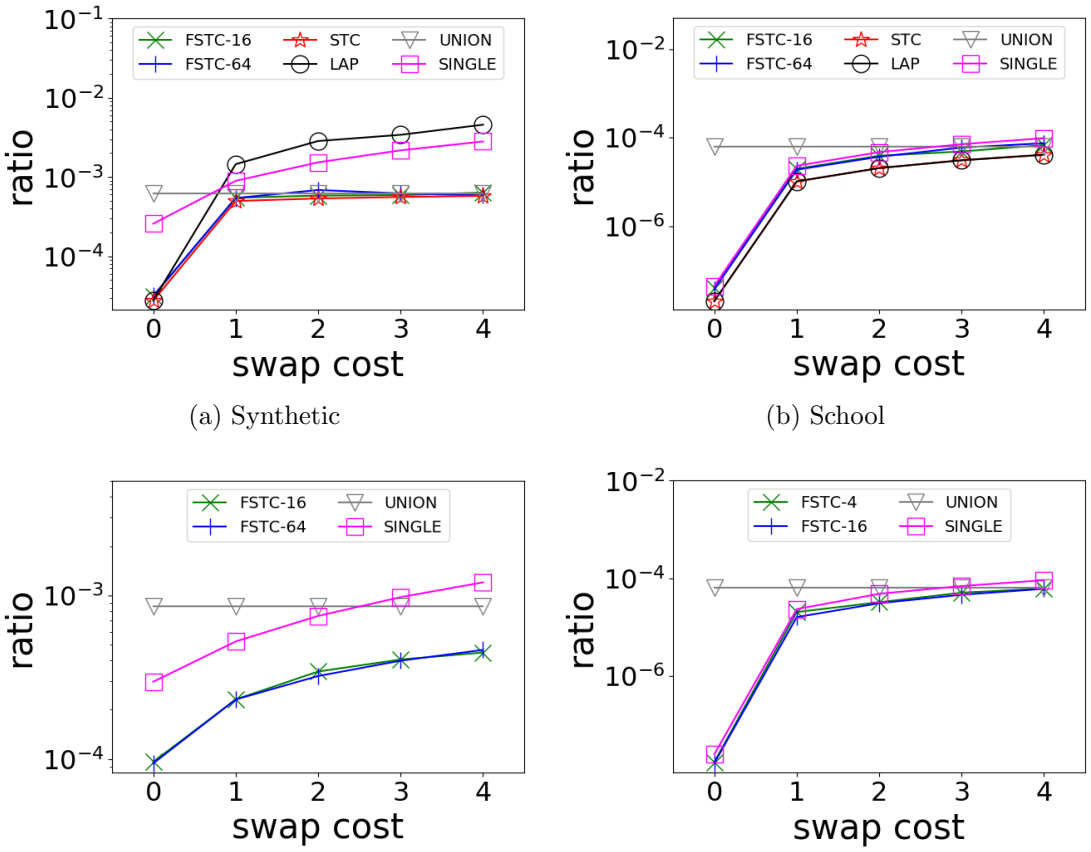
The lemma follows directly from [104, Theorem 1], by setting G to our temporal graph $\mathcal{G}$ and H to the sequence of cliques with Laplacian $\mathcal{C}$, and thus the proof is omitted. Similar to the classical Cheeger inequality [55], the proof of its generalized version is also constructive, and the rounding algorithm applied by our solutions achieves this bound. We can interpret Lemmma 4.2.5 as follows. If the temporal graph $\mathcal{G}$ has a low-sparsity cut compared to the conductance of its multiplex view $\chi(\mathcal{G})$, then our relaxations will find a good approximate (possibly different) temporal cut. A similar bound also holds for normalized temporal cuts (Definition 10).

4.2.5 Generalizations

Here, we briefly address several generalizations of temporal cuts that aim to increase the applicability of this work.

Arbitrary swap costs: While we have assumed uniform swap costs β, generalizing our formulation to arbitrary (non-negative) costs for pairs (v_t, v_{t+1}) is straightforward.

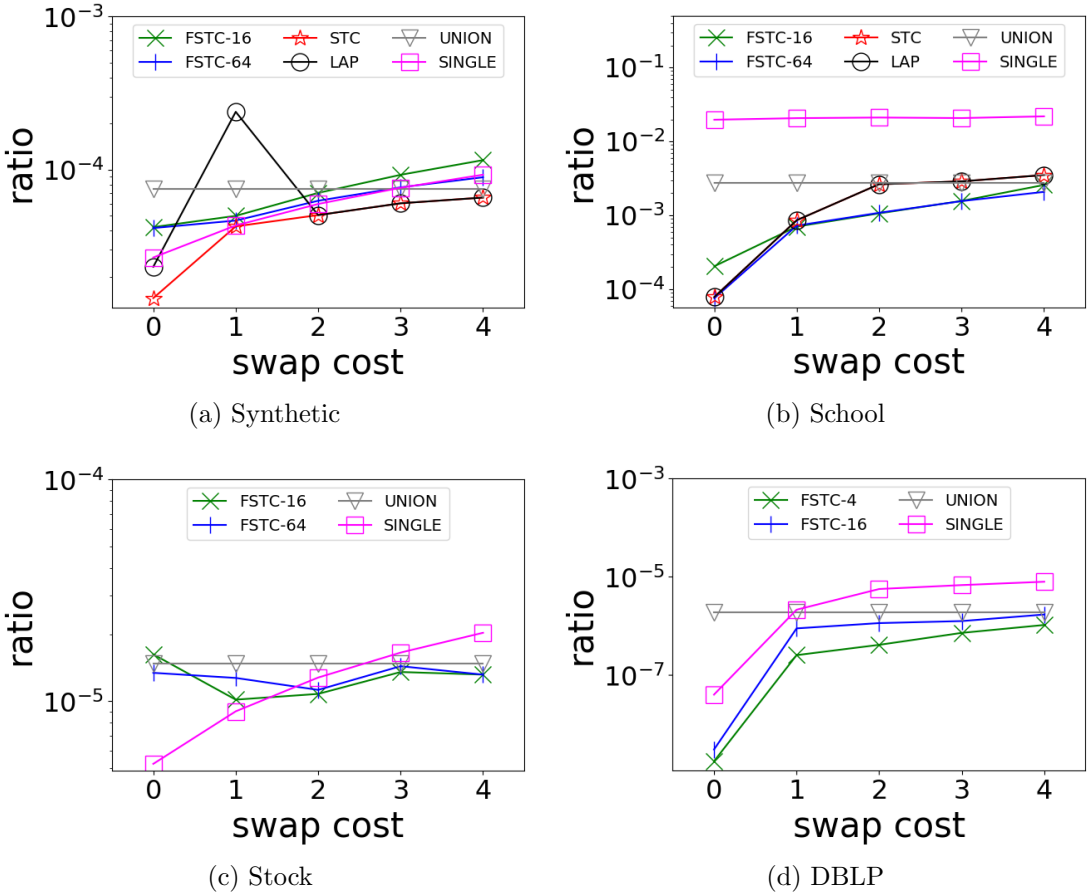
Multiple cuts: Multi-cuts can be computed based on the top eigenvectors of our temporal cut matrices, as proposed in [47]. We use k-means to obtain a k-way partition.



(a) Synthetic (b) School

(c) Stock (d) DBLP

Figure 4.3: Sparsity ratios for sparsest cuts. *STC* achieves the smallest ratio in most of the settings. *FSTC* also achieves good results, specially for $r = 64$, being able to adapt to different swap costs.



(a) Synthetic (b) School

(c) Stock (d) DBLP

Figure 4.4: Sparsity ratios for normalized cuts. *STC* achieves the smallest ratio in most of the settings. *FSTC* also achieves good results, specially for $r = 64$, being able to adapt to different swap costs.

4.3 Signal Processing on Graphs

We apply graph cuts as data-driven wavelet bases for dynamic signals. Given a sequence of signals $\langle \mathbf{f}_{(1)}, \ldots \mathbf{f}_{(m)} \rangle$, $\mathbf{f}_{(i)} \in \mathbb{R}^n$, on a temporal graph $\mathcal{G}$, our goal is to discover a temporal cut that is sparse, smooth, and separates vertices with dissimilar signal values. A previous work [2] has shown that a relaxation of the L_2 energy (or importance) $||a||_2$ of a wavelet coefficient a for a single graph snapshot with signal $\mathbf{f}$ can be computed as:

$$\frac{(|X| \sum_{v \in \overline{X}} \mathbf{f}[v] - |\overline{X}| \sum_{u \in X} \mathbf{f}[u])^2}{|X||\overline{X}|} \propto -\frac{\mathbf{x}^\intercal C S C \mathbf{x}}{\mathbf{x}^\intercal C \mathbf{x}} \tag{4.14}$$

where $\mathbf{x}$ is an indicator vector and $S_{u,v} = (\mathbf{f}[v] - \mathbf{f}[u])^2$. Sparsity is enforced by adding a Laplacian regularization factor $\alpha \mathbf{x}^\intercal L \mathbf{x}$, where α is a user-defined constant, to the denominator of Equation 4.14. This formulation supports an algorithm for computing graph wavelets, which we extend to dynamic signals. Following the same approach as in Section 4.2.1, we apply the multiplex graph representation to compute the energy of a dynamic wavelet coefficient:

$$\frac{\sum_{t=1}^{m} \Theta_t^2 + \sum_{t=1}^{m-1} \Theta_t \Theta_{t+1}}{\sum_{t=1}^{m} |X_t||\overline{X}_t|} \tag{4.15}$$

where $\Theta_t = |X_t| \sum_{v \in \overline{X}_t} \mathbf{f}[v] - |\overline{X}_t| \sum_{u \in X_t} \mathbf{f}[u]$. The first term in the numerator of Equation 4.15 is the sum of the numerator of Equation 4.14 over all snapshots. The second term acts on sequential snapshots and enforces the partitions to be consistent over time —i.e. X_t's to be jointly associated with either large or small values. Intuitively, the energy is maximized for partitions that separate different values and are also balanced in size. The next theorem provides a spectral formulation for the energy of dynamic wavelets.

Theorem 14 *The energy of a dynamic wavelet is proportional to* $-\frac{\boldsymbol{x}^\intercal \mathcal{C}\mathcal{S}\mathcal{C}\boldsymbol{x}}{\boldsymbol{x}^\intercal \mathcal{C}\boldsymbol{x}}$*, where* $\mathcal{S}_{u,v} = (\boldsymbol{f}[u] - \boldsymbol{f}[v])^2$ *for values within one snapshot from each other.*

We apply Theorem 14 (see proof in [101]) to compute a relaxation of the optimal dynamic wavelet as a regularized eigenvalue problem:

$$\mathbf{x}* = \underset{\mathbf{x} \in [-1,1]^{nm}}{\arg\min} \frac{\mathbf{x}^\intercal \mathcal{C}\mathcal{S}\mathcal{C}\mathbf{x}}{\mathbf{x}^\intercal \mathcal{C}\mathbf{x} + \alpha \mathbf{x}^\intercal \mathcal{L}\mathbf{x}} \tag{4.16}$$

where $\mathcal{C}$ and $\mathcal{L}$ are matrices defined in Section 4.2.1. Optimizations discussed in [2] can be applied to efficiently approximate Equation 4.16. The resulting algorithm has complexity $O(pn \sum_t |E_t| + qn^2m^2)$, where p and q are small constants. Similar to Algorithm 5, we apply a sweep procedure to obtain a cut from $\mathbf{x}*$.

4.4 Experiments

4.4.1 Eigenvector Computation

We applied the Lanczos method [60, 105] to implement the algorithms discussed in this work[1]. However, our algorithms can also be implemented using other eigensolvers from the literature.

4.4.2 Datasets

School is a contact network where vertices represent children from a primary school and edges are created based on proximity detected by sensors [71], with 242 vertices, $17K$ edges and 3 snapshots. *Stock* is a correlation network of US stocks' end of the day prices[2] with 500 vertices, $27K$ edges, and 26 snapshots (one for each year in the interval 1989-2015). *DBLP* is a sample from the DBLP collaboration network. Vertices corresponding to two authors are connected in a given snapshot if they co-authored a paper in the corresponding year. We selected authors who published at least 5 papers one of the following conferences: KDD, CVPR, and FOCS. The resulting temporal network has $3.4K$ vertices, $16.4K$ edges, and 4 snapshots.

We also use a synthetic data generator. Its parameters are a graph size n, partition size $k < n$, number of hops h, and noise level $0 \leq \epsilon \leq 1$. Edges are created based on a $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ grid, where each vertex is connected to its h-hop neighbors. A partition is a sub-grid initialized with $\lceil\sqrt{k}\rceil \times \lceil\sqrt{k}\rceil$ dimensions ($k = n/2$). A value $\pi(v) = 1. + N(0, \epsilon)$ is assigned to vertices inside the partition and the remaining vertices receive iid realizations of a Gaussian $N(0, \epsilon)$. Given the node values, the weight w of an edge (u, v) is set as $\exp(|\pi(v) - \pi(u)|)$. To produce the dynamics, we move the partition along the main

[1]Data and code: `https://github.com/arleilps/time-cuts`
[2]Qualdl data: `https://www.quandl.com/data/`

diagonal of the grid.

To evaluate our wavelets for dynamic signals, we apply our approach to *Traffic* [2], a road network from California with 100 vertices, 200 edges, and 12 snapshots. Average vehicle speeds measured at the vertices were taken as a dynamic signal for the timespan of a Friday in April, 2011. Moreover, we apply the *heat equation* to generate synthetic signals over the *School* network. Different from *Traffic*, which has a static structure, the resulting dataset (*School-heat*) is dynamic in structure and signal (see details in [101]).

4.4.3 Approximation and Performance

Two general approaches for computing temporal cuts, for both sparsest and normalized cuts, are evaluated in this section. The first approach, *STC*, combines Theorems 11 and 12, for sparsest and normalized cuts, respectively, and the same rounding scheme applied by Algorithm 5. The second approach, *FSTC-r*, for a rank r, applies the fast approximation described in Section 4.2.3.

We consider three baselines in this evaluation. *SINGLE* discovers the best cut on each snapshot and then binds them into one temporal cut. *UNION* computes the best average cut over all the snapshots. *LAP* is similar to our approach, but operates directly on the Laplacian matrix $\mathcal{L}$. Notice that each of these baselines can be applied to either sparsest and normalized cuts as long as the appropriate (standard or normalized) Laplacian matrix is used. Each experiment was repeated 10 times and we report the average results.

Figures 4.3 and 4.4 show quality results (sparsity ratios) of the methods. We vary the swap cost (β) within a range that enforces local and global (or stable) patterns. The values of β shown are normalized to integers for ease of comparison. *STC* and *LAP* took too long to finish for the *Stock* and *DBLP* datasets, and thus their results are omitted. Our approach, *STC*, achieves the best results (smallest ratios) in most

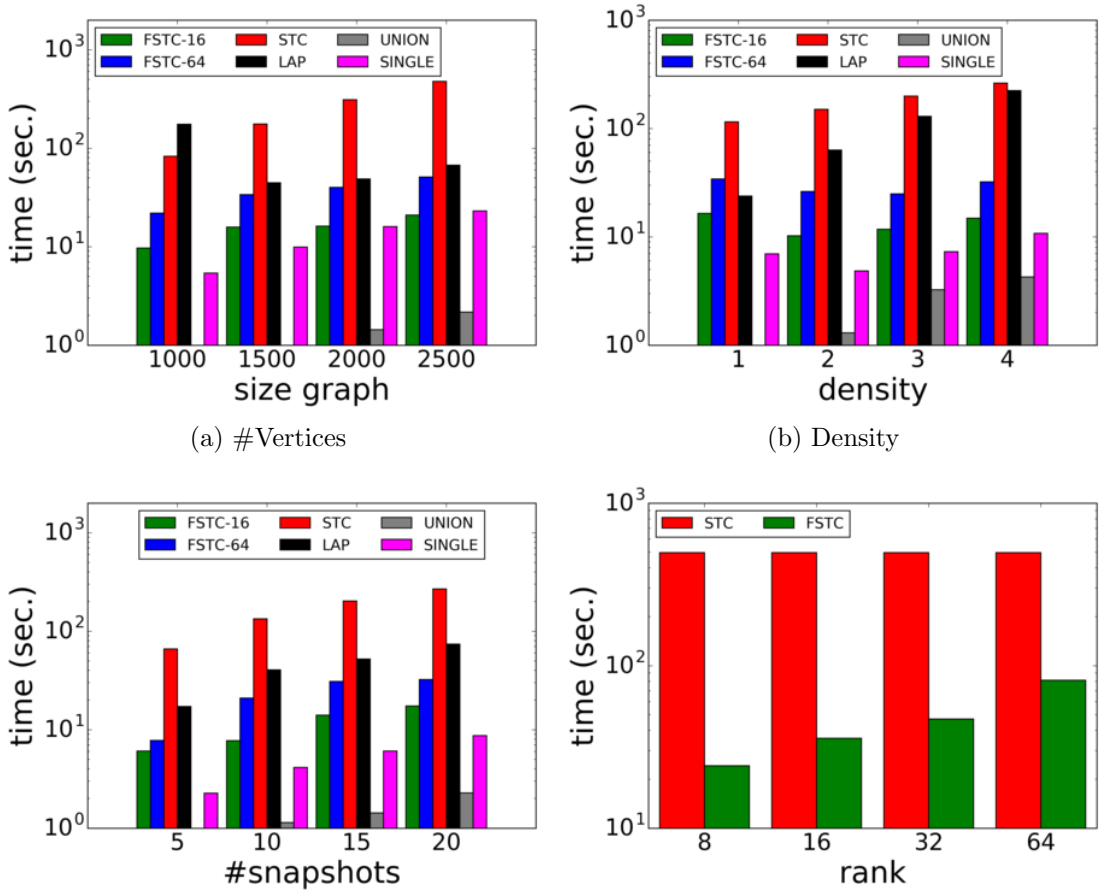
of the settings. For the *School* dataset, *LAP* also achieves good results, which is due to the small number of snapshots in the graph. As expected, *UNION* performs well for large swap costs, while *SINGLE* achieves good results when swap costs are close to 0. Our fast approximation (*FSTC*) is able to identify low-sparsity cuts in most of the settings, outperforming *SINGLE* and *UNION*. Notice that even though a larger value of r generates a better approximation for the temporal matrix, as discussed in Section 4.2.3, the quality of the temporal cut is not guaranteed to increase monotonically with the value of r (see Figure 4.4c). However, a larger r often leads to a better approximation (i.e. lower sparsity ratio).

Figures 4.5 and 4.6 show the performance results (running time) using synthetic data for sparsest (Figure 4.5a-4.5d) and normalized (Figures 4.6a-4.6d) cuts. We vary the number of vertices, density, number of snapshots, and also the rank of FSTC. Similar conclusions can be drawn for both problems. *UNION* is the fastest method, as it operates over an $n \times n$ matrix. *STC* and *LAP*, which process $nm \times nm$ matrices, are the most time consuming methods. *STC* is even slower than *LAP*, due to its denser matrix. *SINGLE* and *FSTC* achieve similar performance, with running times close to *UNION*'s. Figures 4.5d and 4.6d illustrate how the rank r of the matrix approximation performed by *FSTC* enables significant performance gains compared to *STC*.

4.4.4 Community Detection

Dynamic community detection is an interesting application for temporal cuts. Two approaches from the literature, *FacetNet* [75] and *GenLovain* [94], are used as the baselines. We focus our evaluation on *School* and *DBLP*, which have most meaningful communities. The following metrics are considered for comparison:

Cut: Total weight of edges across partitions, $\sum_{t=1}^{m} \sum_{v,u \in G_t} w(u,v)(1 - \delta(c_v, c_u))$,



(a) #Vertices (b) Density

(c) #Snapshots (d) Rank

Figure 4.5: Running time results for sparsest cuts on synthetic data.

where c_v and c_u are the partitions to which v and u are assigned, respectively.

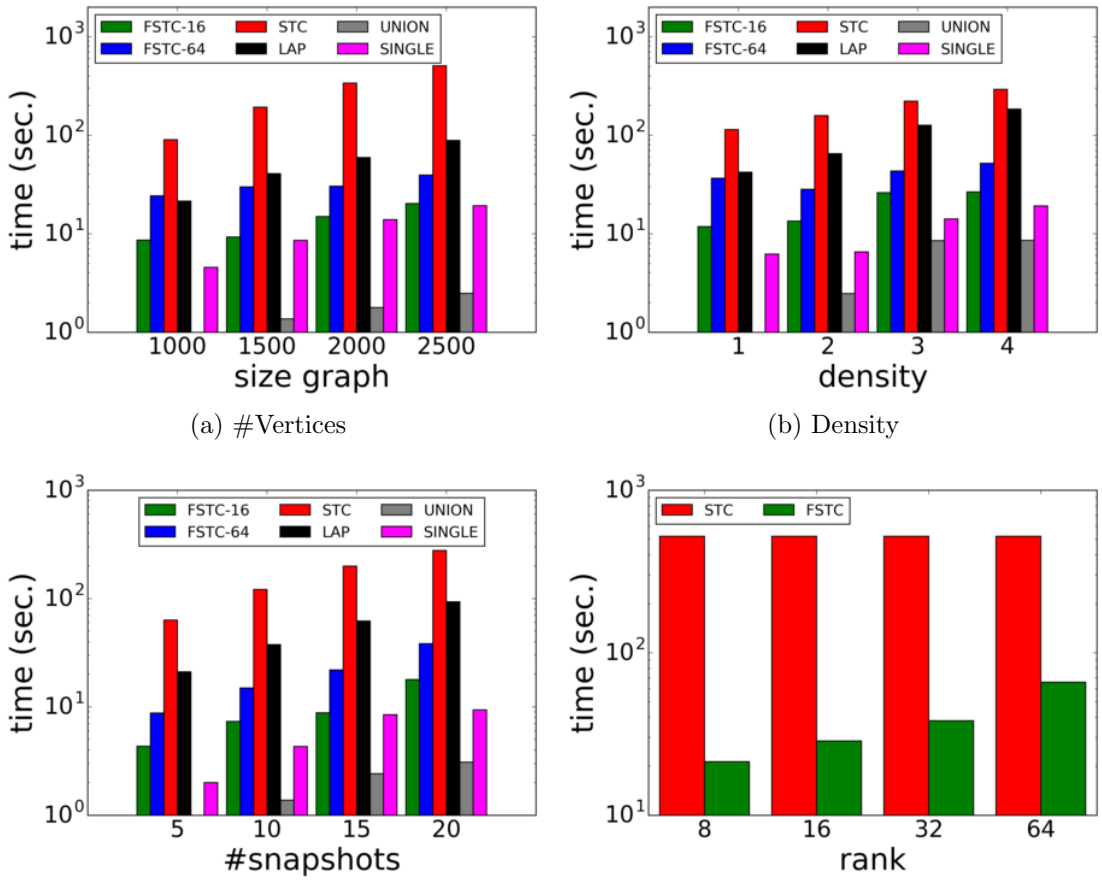
Sparsity: Sparsity ratio (Equation 4.3) for k-cuts [47]:

$$\sum_{i=1}^{k} \frac{\sum_{t=1}^{m} |(X_{i,t}, \overline{X}_{i,t}| + \beta \sum_{t=1}^{m-1} |(X_{i,t}, \overline{X}_{i,t+1})|}{\sum_{t=1}^{m} |X_{i,t}||\overline{X}_{i,t}|}$$

N-sparsity: Normalized k-cut ratio (similar to sparsity).

Modularity: Temporal modularity, as defined in [94].

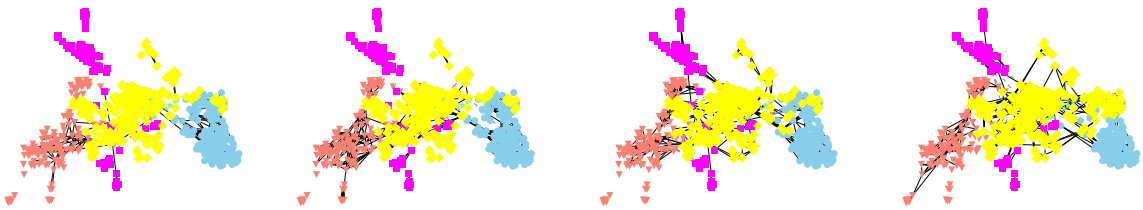
Baseline parameters were varied within a range of values and the best results were chosen. For *GenLovain*, we fixed the number of partitions by agglomerating pairs that



(a) #Vertices (b) Density

(c) #Snapshots (d) Rank

Figure 4.6: Running time results for normalized cuts on synthetic data.



(a) I (b) II (c) III (d) IV

Figure 4.7: Dynamic communities discovered using *sparsest cuts* for the *DBLP* dataset (4 snapshots). *Better seen in color.*

maximize modularity [94] and for *FacetNet*, we assign each vertex to its highest weight partition.

Community detection results, for 2 and 5 communities, are shown in Table 4.1. For *School*, both *GenLovain* and our methods found the same communities ($\beta = 0.25$) when $k = 2$, outperforming *FacetNet* in all the metrics. However, for $k = 5$, different communities were discovered by the methods, with *Sparsest* and *Normalized Cuts* achieving the best results in terms of sparsity and n-sparsity, respectively. Our methods also achieve competitive results in terms of modularity. Similar results were found using *DBLP* ($\beta = 0.5$), although *Sparsest* and *Normalized Cuts* switch as the best method for each other's metric in some settings. This is possible because our algorithms are approximations (i.e. not optimal). We illustrate the communities found in the *School* ($k = 2$) and *DBLP* ($k = 5$) datasets in Figures 4.1 and 4.7, respectively.

S

k	Method	*Cut*	*Sparsity*	*N-sparsity*	*Modularity*
2	*GenLovain*	**2.6**	**1.0e-4**	**5.0e-3**	**102.0**
	Facetnet	6.0	3.8e-4	.012	95.7
	Sparsest	**2.6**	**1.0e-4**	**5.0e-3**	**102.0**
	Norm.	**2.6**	**1.0e-4**	**5.0e-3**	**102.0**
5	*GenLovain*	8.0	6.8e-4	2.7e-2	**110.0**
	Facetnet	10.0	8.4e-4	3.0e-2	106.0
	Sparsest	8.3	**6.4e-4**	2.6e-2	109.0
	Norm.	**6.1**	9.9e-4	**1.8e-2**	**110.0**

(a) School

k	Method	*Cut*	*Sparsity*	*N-sparsity*	*Modularity*
2	*GenLovain*	80.	3.9e-4	1.3e-5	**38,612**
	Facetnet	267.0	2.6e-3	8.9e-5	33,091
	Sparsest	**9.0**	**7.6e-5**	**3.6e-6**	38,450
	Norm.	19.0	1.2e-4	3.8e-6	38,516
5	*GenLovain*	174.	1.3e-3	4.1e-5	**39,342**
	Facetnet	501.0	7.2e-3	2.8e-4	30,116
	Sparsest	40.0	5.2e-4	6.2e-5	38,498
	Norm.	**31.0**	**4.0e-4**	**1.0e-5**	39,015

(b) DBLP

Table 4.1: Community detection results for *Sparsest and Normalized Cuts* (and two baselines) using *School* and *DBLP* datasets. Our methods achieve the best results for most of the metrics and are competitive in terms of modularity.

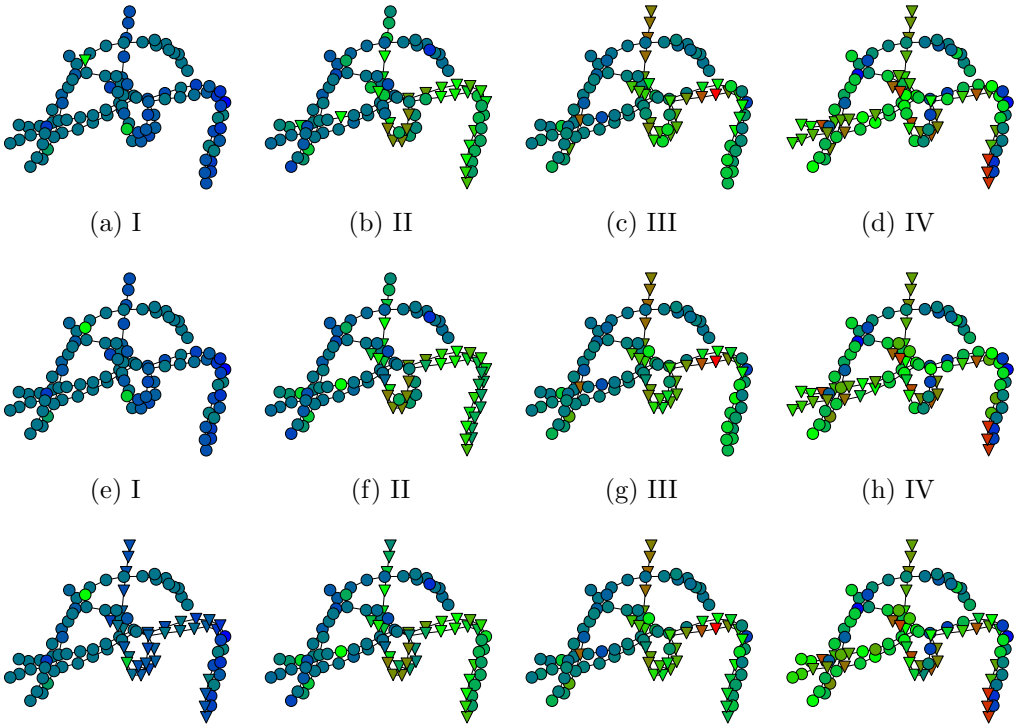
4.4.5 Signal Processing on Graphs

We finish our evaluation with the analysis of dynamic signals on graphs. In Figure 4.8, we illustrate three dynamic wavelets for *Traffic* discovered using our approach under different settings. First, in Figures 4.8a-4.8d, we consider cuts that take only the graph signal into account by setting both the regularization parameter α and the smoothness parameter β to 0, which leads to a cut that follows the traffic speeds but has many edges and is not smooth. Next (Figures 4.8e-4.8h), we increase α to 200, producing a much sparser cut that is still not smooth. Finally, in Figures 4.8i-4.8l, we increase the smoothness β to 10, which forces most of the vertices to remain in the same partition despite of speed variations.

We also evaluate our approach in signal compression, which consists of computing a compact representation for a dynamic signal. As a baseline, we consider the *Graph Fourier* scheme [13] applied to the temporal graph (i.e. the multiplex view of the graph). The size of the representation (k) is the number of partitions and the number of top eigenvectors for our method and *Graph Fourier*, respectively. Figures 4.9a and 4.9b show the compression results in terms of L_2 error using a fixed representation size k for the *Traffic* and *School-heat* datasets, respectively. We vary the value of the regularization parameter α, which controls the impact of the network structure over the wavelets computed, for our method. As expected, a larger value of α leads to a higher L_2 error. However, even for a high regularization, our approach is still able to compute wavelets that accurately compress the signal, outperforming the baseline.

4.5 Conclusion

This work studied cut problems in temporal graphs. Extensions of two existing graph cut problems, sparsest and normalized cuts, by enforcing the smoothness of cuts over

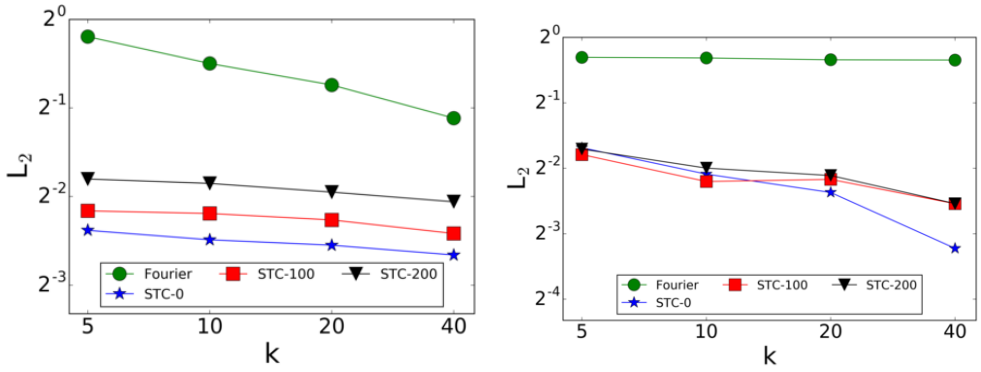


(a) I (b) II (c) III (d) IV

(e) I (f) II (g) III (h) IV

(i) I (j) II (k) III (l) IV

Figure 4.8: Wavelet cut of a 4-snapshot dynamic traffic network with vehicle speeds as a signal. Vertex colors correspond to speed values (red for high and blue for low) and shapes indicate the partitions for 3 different settings: $\alpha = 0.$ and $\beta = 1$ (a-d, no network effect), $\alpha = 200.$ and $\beta = 1.$ (e-h, large network effect with low smoothness), and $\alpha = 200.$ and $\beta = 10.$ (i-l, large network effect and high smoothness) .



(a) Traffic (b) School-heat

Figure 4.9: L_2 error with different representation sizes k for Graph Fourier and our approach while setting the regularization parameter α to 200, 100, and 0.

time, were introduced. To solve these problems, we have proposed spectral approaches based on multiplex graphs by computing relaxed temporal cuts as eigenvectors. Scalable versions of our solutions using divide-and-conquer and low-rank matrix approximation were also presented. In order to compute cuts that take into account also graph signals, we have extended graph wavelets to the dynamic setting. Experiments have shown that our temporal cut algorithms outperform the baseline methods in terms of quality and are competitive in running time. Moreover, temporal cuts enable the discovery of dynamic communities and the analysis of dynamic graph processes.

This work opens several lines for investigation: (i) temporal cuts can be applied to many scenarios other than the ones considered in this work (e.g., computer vision); (ii) Perturbation Theory can support fast updates for temporal cuts in graph streams [85]; finally, (iii) we want to investigate the relationship between cuts and random-walks on temporal graphs [95].

Chapter 5

Learning Interleaved Hidden Markov Models

5.1 Introduction

Hidden Markov models (HMMs) are popular probabilistic models for sequential data [106]. More recently, extensions of HMMs have been proposed to support modern applications in speech and activity recognition, bioinformatics, intrusion detection, among others [107, 108]. Interleaved hidden Markov models (IHMMs), as one of such extensions, assumes that a set of HMMs concurrently produce a single sequence of observations [109]. This work is focused on the problem of learning IHMMs from data.

The main motivation for this work is *syslog analytics* [110, 111, 112]. Syslogs are sequential textual files with traces of system events, providing valuable information for failure detection and diagnosis. However, these systems often integrate multiple workflows (e.g., login, credit card processing) executing concurrently, and thus isolating events from different workflows is required. The most accurate solution for this problem is instrumentation (records are tagged with global IDs), but many enterprise systems have

black-box components (e.g. proprietary software) that cannot be annotated [113]. This work investigates whether these workflows, with their associated events, can be recognized from a single unlabeled syslog (see Figure 5.1).

There are many other scenarios where interleaved processes occur. In the *internet-of-things*, where sensors collect data from multiple users in an environment, interleaved activity recognition has become an important challenge [109, 114]. In particular, tracking many activities of a user might enable its identification even if activities are anonymized. Interleaved processes also have applications in bioinformatics, where genome sequences combine both coding and noncoding regions [115, 116].

While IHMMs have been introduced in the last decade [109], many questions about their inference remain unanswered. For instance, there is no efficient algorithm for learning IHMM parameters in general settings. In fact, even the conditions that enable IHMM inference (e.g. identifiability and sampling complexity) are still not well understood. These are challenges addressed in this work.

As in other extensions of HMMs, exact inference for IHMMs is intractable. Variational Inference (VI) [117, 118], an alternative to Markov Chain Monte Carlo (MCMC) [119], is a promising approach to approximate such complex probabilistic models. One of the contributions of this work is an efficient algorithm for learning IHMMs based on loopy belief propagation (LBP). IHMMs are a generalization of HMMs, and thus we can apply several existing from HMM inference to our problem. For instance, IHMMs are also hard to learn in general [120] and might not even be identifiable [121].

5.2 Related work

From an empirical standpoint, it is a common understanding that many relevant processes are naturally interleaved. In both the workflow mining and the systems research

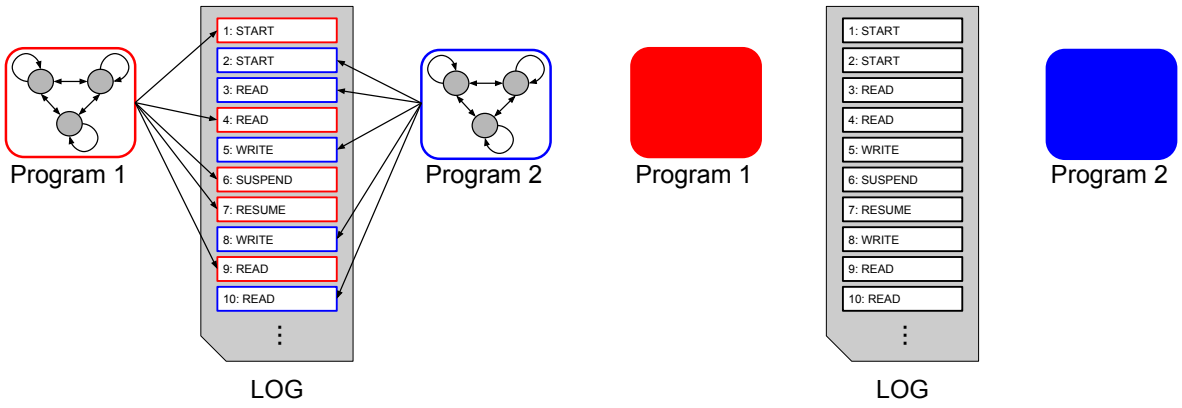


Figure 5.1: Interleaved syslog analysis of *black-box* components. Programs, for which workflows are modeled as Markov chains, write concurrently to a system event log (left). Due to the lack of instrumentation, events are not tagged with program IDs (right). Using interleaved hidden Markov processes, we can learn program workflows directly from log data.

literature, interleaving is mostly treated as noise, and thus some combination of counting and thresholding is expected to detect transitions within the same workflow/system component [111, 113]. A costlier but more accurate alternative is to isolate processes in controlled experiments [122]. The development of statistical models for interleaving processes can have a great impact in these application scenarios.

In the machine learning community, latent models for interleaved processes have been motivated by other extensions of HMMs, such as factorial hidden Markov models [107], mixed memory Markov models [123], and mixed hidden Markov models [124]. The work closest to ours is the chainwise Viterbi algorithm for the inference of the most likely sequence of hidden states in an interleaved process [109]. However, the author assumes a supervised setting and leaves parameter estimation as future work. More recent studies focused on parameter estimation in simplified settings, such as disjoint alphabets [125], identical sources [126], and observed interleaving [116] or state sequences [127].

This work investigates how the parameters of an IHMM can be learned from data, similar to the Baum-Welch algorithm for HMMs [106]. Exact IHMM inference is intractable,

due to its combinatorial structure, and thus we resort to the classical Markov Chain Monte Carlo (MCMC) sampling [119, 128] and variational approaches [129, 118, 130, 117].

5.3 The Model

We first briefly review classical Hidden Markov Models (HMMs). An HMM is a probabilistic model for a sequence of observations $\mathbf{Y} = Y_1, \dots Y_T$, where $Y_t \in \{1 \dots W\}$ is one of W discrete symbols. Each observation Y_t is generated by a hidden state $X_t \in \{1, \dots K\}$ with probability $P(Y_t|X_t)$ and a sequence of hidden states $\mathbf{X} = X_1 \dots X_T$ is associated to the observations $\mathbf{Y}$. Hidden states change according to first-order Markov chain with transition probabilities $P(X_t|X_{t-1})$ and the first state is chosen with probability $P(X_1)$. We show a graphical representation of an HMM in Figure 5.2 (a). The joint distribution of observations $\mathbf{Y}$ and hidden states $\mathbf{X}$ in the model is given by:

$$P(\mathbf{X}, \mathbf{Y}) = P(X_1)P(Y_1|X_1) \prod_{t=2}^{T} P(X_t|X_{t-1})P(Y_t|X_t)$$

We apply the following notation for HMM parameters: $\pi_k = P(X_1 = k)$, $A_{i,j} = P(X_t = j|X_{t-1} = i)$, and $b_{k,w} = P(Y_t = w|X_t = k)$. For a detailed review on HMMs and their applications, please refer to [106].

An IHMM combines multiple concurrent Markov processes that produce a single sequence of observations $\mathbf{Y}$. At each time t, the active process $Z_t \in \{1 \dots M\}$ transitions to a new state X_t and emits Y_t while the remaining processes are kept frozen. The switching behavior can then be described by the sequence of active models $\mathbf{Z} = Z_1 \dots Z_T$ with independent process probabilities $c_m = P(Z_t = m)$.

To simplify the notation, we add a fixed starting state $X_1^{(m)} = 0$, which does not emit any symbol, to the set of states of each Markov chain. As a consequence, processes will

often be described in terms of their transition probabilities $A_{i,j}^{(m)}$ and emission probabilities $b_{k,w}^{(m)}$ only. Symbols emitted by the processes may overlap and we assume, without loss of generality, that the models have the same number of states K. We also assume that the number of processes and states is known a priori.

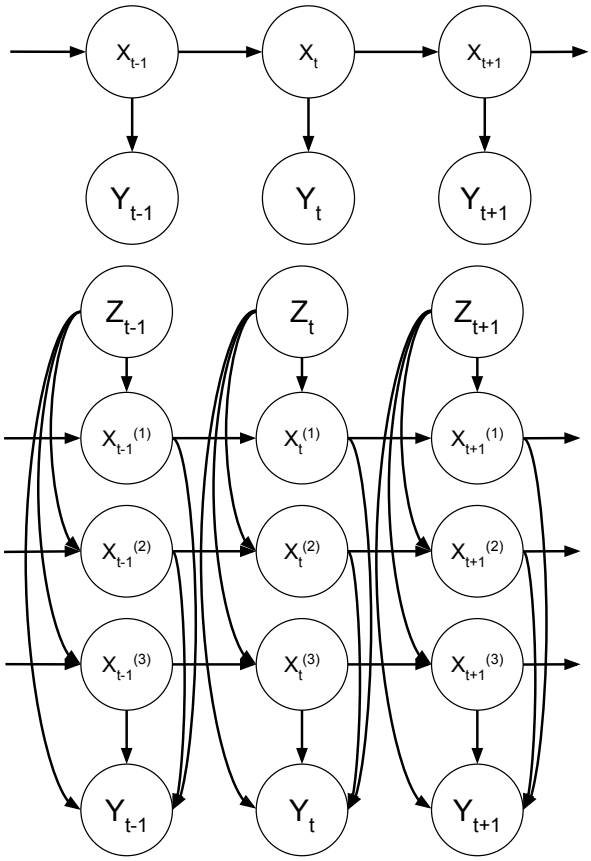


Figure 5.2: Hidden Markov model (top) and interleaved independent hidden Markov model (bottom).

The joint probability for the sequence of activations, states and observations for IHMMs can be factored as:

$$P(\mathbf{Z}, \mathbf{X}, \mathbf{Y}) = \prod_{t=1}^{T} P(Z_t) P(Y_t | X_t^{(1)} \dots X_t^{(M)}, Z_t) \times \prod_{m=1}^{M} P(X_t^{(m)} | X_{t-1}^{(m)}, Z_t) \tag{5.1}$$

At each timestamp t, only the active process Z_t emits a symbol, as described by the following emission probability distribution for the interleaved process:

$$P(Y_t = w | X_t^{(1)} = k_1, \dots X_T^{(M)} = k_M, Z_t = m) = b_{k_m,w}^{(m)}$$

Moreover, only the active process transitions to the next state. Inactive processes remain at their current state:

$$P(X_t^{(m)} = j | X_{t-1}^{(m)}, Z_t = \ell) = \begin{cases} A_{i,j}^{(m)} & m = \ell \\ \delta(i,j) & m \neq \ell \end{cases} \tag{5.2}$$

where $\delta(i,j)$ is the Kronecker delta function.

Figure 5.2 shows the graphical representation of an IHMM (Figure 5.2(a)) compared with the same representation for a standard HMM (Figure 5.2(b)). Notice that our model differs from the one proposed by [109] as we assume that processes are activated independently.

5.4 Inference

As a generalization of HMMs, IHMMs share many inference problems with its predecessor. In this work, we focus on how to learn the parameters of IHMMs, in the same spirit as the Baum-Welch algorithm [106]. In particular, we also apply Expectation Maximization (EM) to solve our estimation problem (Section 5.4.1). Furthermore, we propose both exact (Section 5.4.2) and approximate (Section 5.4.3) inference algorithms for IHMM inference.

5.4.1 EM Algorithm

Categorical variables in the IHMM are represented using an 1-to-K scheme. The following properties hold for activations, states, and symbols, respectively: $Z_t^{(m)} \in \{0,1\}$ with $\sum_{m=1}^{M} Z_t^{(m)} = 1$, $X_{t,k}^{(m)} \in \{0,1\}$ with $\sum_{k=1}^{K} X_{t,k}^{(m)} = 1$, and $Y_{t,w} \in \{0,1\}$ with $\sum_{w=1}^{W} Y_{t,w} = 1$. We apply such representation to formulate IHMM probabilities:

$$P(Z_t) = \prod_{m=1}^{M} c_m^{Z_t^{(m)}}$$

$$P(Y_t|\mathcal{X}_t) = \prod_{m=1}^{M}\prod_{w=1}^{W}\prod_{k=1}^{K}(b_{k,w}^{(m)})^{Y_{t,w}X_{t,k}^{(m)}Z_t^{(m)}}$$

$$P(X_t^{(m)}|X_{t-1}^{(m)}, Z_t) = \prod_{i,j=1}^{K}[(A_{i,j}^{(m)})^{Z_t^{(m)}}\delta^{1-Z_t^{(m)}}]^{X_{t-1,i}^{(m)}X_{t,j}^{(m)}}$$

where $\mathcal{X}_t = \langle Z_t, X_t^{(1)}, \ldots X_t^{(M)}\rangle$ and we applied δ as a short for $\delta(i,j)$. In the E step of the algorithm, parameters $\Theta = \{c_m, A_{i,j}^{(m)}b_{k,w}^{(m)},\}$ are updated to maximize the expected log likelihood of the complete data:

$$\begin{aligned}
\mathcal{Q}(\Theta^{new}|\Theta) =& E\{\log P(\mathbf{Z},\mathbf{X},\mathbf{Y}|\Theta^{new})|\Theta,\mathbf{Y}\} \\
=& -\log\mathcal{Z} + \sum_{t=1}^{T}\sum_{m=1}^{M}\langle Z_t^{(m)}\rangle \log c_m \\
&+ \sum_{t=1}^{T}\sum_{m=1}^{M}\sum_{w=1}^{W}\sum_{k=1}^{K} Y_{t,w}\langle Z_t^{(m)}X_{t,k}^{(m)}\rangle \log b_{k,w}^{(m)} \\
&+ \sum_{t=1}^{T}\sum_{m=1}^{M}\sum_{i=1}^{K}\sum_{j=1}^{K}\langle Z_t^{(m)}X_{t-1,i}^{(m)}X_{t,j}^{(m)}\rangle \log A_{i,j}^{(m)} \\
&+ \sum_{t=1}^{T}\sum_{m=1}^{M}\sum_{i=1}^{K}\sum_{j=1}^{K}\langle (1-Z_t^{(m)})X_{t-1,i}^{(m)}X_{t,j}^{(m)}\rangle \log \delta(i,j)
\end{aligned}$$

where $\langle Z_t^{(m)} \rangle$, $\langle Z_t^{(m)} X_{t,k}^{(m)} \rangle$, $\langle Z_t^{(m)} X_{t-1,i}^{(m)} X_{t,j}^{(m)} \rangle$, and $\langle (1 - Z_t^{(m)}) X_{t-1,i}^{(m)} X_{t,j}^{(m)} \rangle$ are expected values involving hidden variables and $\mathcal{Z}$ is a constant. Parameters are updated in the M step using the following equations:

$$c_m = \frac{\sum_{t=1}^{T} \sum_{k=1}^{K} \langle Z_t^{(m)}, X_{t,k}^{(m)} \rangle}{\sum_{m=1}^{M} \sum_{t=1}^{T} \sum_{k=1}^{K} \langle Z_t^{(m)}, X_{t,k}^{(m)} \rangle}$$

$$A_{i,j}^{(m)} = \frac{\sum_{t=2}^{T} \langle Z_t^{(m)}, X_{t-1,i}^{(m)}, X_{t,j}^{(m)} \rangle}{\sum_{k=1}^{K} \sum_{t=2}^{T} \langle Z_t^{(m)}, X_{t-1,i}^{(m)} X_{t,k}^{(m)} \rangle}$$

$$b_{k,w}^{(m)} = \frac{\sum_{t=1}^{T} Y_{t,w} \langle Z_t^{(m)}, X_{t,k}^{(m)} \rangle}{\sum_{t=1}^{T} \langle Z_t^{(m)}, X_{t,k}^{(m)} \rangle}$$

In the next sections, we discuss how to compute the statistics $\langle . \rangle$ using both exact and approximate inference. Notice that the parameters do not depend on $\langle (1 - Z_t^{(m)}) X_{t-1,i}^{(m)} X_{t,j}^{(m)} \rangle$, due to the fact that this value is always zero (from Equation 5.2). This property will be a major challenge in the design of approximate inference algorithms for IHMMs, which will be discussed in Section 5.4.3.

5.4.2 Exact Inference

Inference in IHMMs is trivial if process activations Z_t are known. Emitted symbols can be separated according to the process that generated them and the Baum-Welch algorithm can be applied for the inference within each process. While we do not assume that Z_t's are given, this simple approach can still be applied to learn the expected values $\langle . \rangle$ by marginalizing over the $O(M^T)$ possible activation settings. Time complexity of such algorithm is $O(K^2 T M^T)$, making it prohibitive in practice.

An alternative exact algorithm for IHMM inference can be designed by observing the graphical model from Figure 5.2 (b). The interleaved process is modeled as the cartesian

product of $M+1$ variables representing hidden states and activations. More specifically, a larger Markov chain with states in the form $\mathcal{X}_t = \langle Z_t, X_t^{(1)}, \ldots X_t^{(M)} \rangle$ simulates the IHMM. The standard Baum-Welch algorithm can be applied to learn the parameters of the joint model that are used to compute expectations $\langle . \rangle$. However, the new chain has $O(MK^M)$ states, leading to a, still prohibitive, time complexity of $O(M^2K^{2M}T)$.

Faster exact inference for HMMs can be performed via the Junction tree algorithm for directed graphical models [131]. More specifically, we can obtain an undirected representation for the graphical model shown in Figure 5.2 (b) after *moralization* and *triangulation*. Inference is performed via local message passing over a spanning tree of the resulting undirected graph. The complexity of the resulting algorithm is $O(MK^{M+1}T)$.

Due to the performance limitations of the exact inference approaches presented in this section, the next two sections are focused on approximate inference for IHMMs.

5.4.3 Approximate Inference: Loopy Belief Propagation

The graphical model for IHMMs (see Figure 5.2) has loops, and thus we apply LBP to design an approximate inference algorithm for our problem. We define T factors $g_t = P(Z_t)$ and $M \times T$ factors $f_t^{(m)}$ as follows:

$$f_t^{(m)} = P(Y_t|X_t^{(m)}, Z_t^{(m)})P(X_t^{(m)}|X_{t-1}^{(m)}, Z_t^{(m)})$$

The joint probability for IHMMs (Equation 5.1) can then be expressed as a product of factors g_t and $f_t^{(m)}$:

$$P(\mathbf{Z}, \mathbf{X}, \mathbf{Y}) = \prod_{t=1}^{T} g_t \prod_{m=1}^{M} f_t^{(m)}$$

Inference is performed via messages between hidden variables, ($\mathbf{X}$,$\mathbf{Z}$), and factors $f_t^{(m)}$. Let $m_{x \to y}$ be a message from x to y, we use the following notation to describe messages

in the IHMM factor graph:

$$\alpha_{k,m,t} = m_{f_t^{(m)} \to X_t^{(m)}}(X_t^{(m)} = k)$$
$$\beta_{k,m,t} = m_{f_{t+1}^{(m)} \to X_t^{(m)}}(X_t^{(m)} = k)$$
$$\gamma_{m,m',t} = m_{Z_t \to f_t^{(m)}}(Z_t = m')$$
$$\lambda_{m,m',t} = m_{f_t^{(m)} \to Z_t}(Z_t = m')$$

According to BP, these probabilities are related as follows:

$$\begin{aligned}
\alpha_{k,m,t} =& b_{t,k}^{(m)} \gamma_{m,m,t} \sum_{i=1}^{K} A_{i,k}^{(m)} \alpha_{i,m,t-1} \\
&+ (1 - \gamma_{m,m,t}) \alpha_{k,m,t-1} \\
\beta_{k,m,t} =& \gamma_{m,m,t+1} \sum_{j=1}^{K} A_{k,j}^{(m)} b_{t+1,j}^{(m)} \beta_{j,m,t+1} \\
&+ (1 - \gamma_{m,m,t+1}) \beta_{k,m,t+1} \\
\gamma_{m,m',t} =& c_{m'} \prod_{\ell=1, \ell \neq m} \lambda_{\ell,m',t} \\
\lambda_{m,m',t} =& \begin{cases} \sum_{i=1}^{K} \alpha_{i,m,t-1} \sum_{j=1}^{K} A_{i,j}^{(m)} b_{t,j}^{(m)} \beta_{j,m,t} & m = m' \\ \sum_{k=1}^{K} \alpha_{k,m,t-1} \beta_{k,m,t} & m \neq m' \end{cases}
\end{aligned}$$

where $b_{t,k}^{(m)} = \sum_{w=1}^{W} Y_{t,w} b_{k,w}^{(m)}$. The M-step of our EM algorithm requires expected values $\langle . \rangle$, which can be approximated based on the messages α, β, γ, and λ:

$$\begin{aligned}
\langle Z_t^{(m)}, X_{t,k}^{(m)} \rangle \propto& c_m \beta_{k,m,t} b_{k,t}^{(m)} \\
&\times \prod_{\ell=1, \ell \neq m}^{M} \lambda_{\ell,m,t} \sum_{i=1}^{K} A_{i,k}^{(m)} \alpha_{i,m,t-1} \\
\langle Z_t^{(m)}, X_{t-1,i}^{(m)}, X_{t,j}^{(m)} \rangle \propto& A_{i,j}^{(m)} b_{k,t}^{(m)} \alpha_{i,m,t-1} \beta_{j,m,t} \gamma_{m,m,t}
\end{aligned}$$

One iteration of our algorithm—assuming that every message is sent once—takes time $O(MT(M^2 + MK + K^2))$ and estimating expected values $\langle . \rangle$ takes time $O(MK^2T)$.

5.5 Experiments

We evaluate several algorithms for exact and approximate inference for IHMMs in terms of accuracy and running time using synthetic datasets.

5.5.1 Accuracy Metrics

These metrics will be applied to evaluate IHMM accuracy:

- *Likelihood (Like)*: Log-likelihood of a sequence of observations to be generated by the IHMM. For synthetic data, we distinguish between *Like-train* and *Like-test*, for training and test sequences, respectively, and also show the likelihood relative to the True model.
- *Adjusted Rand-score (Rand)*: Modification of the rand-score—ratio of agreement between two assignments of observations to processes—adjusted for chance.
- *KL-Divergence (KLD)*: KL-divergence between IHMMs estimated via Monte Carlo [132].

5.5.2 Approaches and Experimental Settings

We consider the following approaches in our experiments:

- *True*: Available only for synthetic data, this the true IHMM that generated the input sequence.

- *BOW*: Simple bag-of-words classifier that predicts the process of an observation without learning an IHMM.
- Offline: Each process is learned independently using the standard Baum-Welch algorithm.
- Single: Standard Baum-Welch algorithm assuming a single process with K^M states, thus being able to compute likelihoods but not to de-interleave an input sequence.
- *IHMM-E*: Exact inference for IHMMs using the junction tree algorithm for $M=2$ and the cartesian product scheme for $M>2$ (see Section 5.4.2).
- *IHMM-G*: Approximate inference for IHMMs using Gibbs sampling, as described in the Appendix.
- *IHMM-(V/H/R)*: Approximate IHMM inference based on BP (see Section 5.4.3). The V, H, and R stand for vertical, horizontal, and residual inference.

The number of iterations of EM was set to 50 for all the approaches. For an input sequence, we run each solution with 10 different random initializations and select the resulting model with highest likelihood. For Gibbs sampling, we warm-up the sampling procedure by throwing away the first 100 samples and then apply the following 100 ones for inference. For BP (IHMM-V/H/R), we perform a maximum of 100 message-passing iterations or stop at convergence—if message values do not change by more than 10^{-5}.

5.5.3 Results

For a fixed number of processes M and states K, we generate 20 $K \times K$ transition matrices $A^{(m)}$ and $K \times MK$ emission matrices $b^{(m)}$ ($m \in \{1, \dots M\}$). Entries of matrices $A^{(m)}$ are sampled uniformly in the interval $[0, 1]$. For emission matrices $b^{(m)}$, we assign

M	K	Method	Like-train	Like-test	Rand-train	Rand-test	KLD
2	2	True	0. ± 0.	0. ± 0.	0.96 ± 0.02	0.96 ± 0.03	0. ± 0.
		Offline	-129.56 ± 8.78	16.44 ± 31.1	0.96 ± 0.02	0.95 ± 0.03	0.41 ± 1.13
		Single	-9.71 ± 4.08	23.77 ± 22.32	-	-	0.36 ± 0.57
		IHMM-E	-4.35 ± 2.91	6.34 ± 4.47	0.46 ± 0.29	0.46 ± 0.31	0.12 ± 0.15
		IHMM-G	3.07 ± 5.02	10.62 ± 10.71	0.07 ± 0.2	0.07 ± 0.21	0.15 ± 0.27
		IHMM-V	-2.27 ± 2.85	11.45 ± 11.68	0.71 ± 0.37	0.72 ± 0.37	0.25 ± 0.43
		IHMM-H	-1.97 ± 3.19	9.46 ± 7.58	0.72 ± 0.34	0.72 ± 0.33	0.27 ± 0.64
		IHMM-R	-2.09 ± 3.3	10.14 ± 9.28	0.69 ± 0.37	0.69 ± 0.38	0.17 ± 0.28
	4	True	0. ± 0.	0. ± 0.	0.92 ± 0.03	0.92 ± 0.03	0. ± 0.
		Offline	-148.49 ± 4.89	99.6 ± 64.18	0.94 ± 0.04	0.8 ± 0.08	1.79 ± 1.84
		Single	-90.86 ± 12.08	884.0 ± 296.77	-	-	7.46 ± 4.11
		IHMM-E	-25.82 ± 4.92	68.4 ± 26.65	0.21 ± 0.18	0.17 ± 0.14	0.7 ± 0.53
		IHMM-G	5.22 ± 5.99	32.64 ± 13.15	0.01 ± 0.04	0.01 ± 0.03	0.72 ± 0.51
		IHMM-V	-14.44 ± 5.53	44.83 ± 28.54	0.29 ± 0.27	0.22 ± 0.23	0.64 ± 0.68
		IHMM-H	-16.56 ± 5.39	54.99 ± 20.8	0.23 ± 0.22	0.17 ± 0.18	0.74 ± 0.67
		IHMM-R	-12.78 ± 6.21	49.63 ± 38.82	0.18 ± 0.17	0.14 ± 0.12	0.68 ± 0.64
3	2	True	0. ± 0.	0. ± 0.	0.95 ± 0.03	0.94 ± 0.03	0. ± 0.
		Offline	-209.38 ± 6.59	16.31 ± 21.47	0.95 ± 0.03	0.94 ± 0.03	0.49 ± 0.75
		Single	-41.13 ± 6.16	98.64 ± 35.85	-	-	1.45 ± 1.22
		IHMM-E	-10.25 ± 3.16	13.03 ± 8.5	0.31 ± 0.16	0.32 ± 0.18	0.19 ± 0.3
		IHMM-G	1.55 ± 8.32	17.51 ± 10.39	0.1 ± 0.16	0.11 ± 0.19	0.37 ± 0.47
		IHMM-V	-2.95 ± 5.74	15.44 ± 7.75	0.35 ± 0.21	0.36 ± 0.24	0.25 ± 0.34
		IHMM-H	-2.42 ± 5.77	14.45 ± 6.07	0.33 ± 0.23	0.35 ± 0.23	0.25 ± 0.35
		IHMM-R	-3.83 ± 4.5	11.74 ± 8.32	0.4 ± 0.26	0.4 ± 0.25	0.18 ± 0.22

Table 5.1: Accuracy results (averages and standard deviations) for synthetic data varying the number of processes (M) and states per process (K) for some of the approaches described in Section 5.5.2 using the metrics given in Section 5.5.1.

one distinct symbol to each state of each process (MK symbols) and add a small (1%) noise uniformly at random. Training and test sequences with 200 symbols are generated from each IHMM. Three configurations for M and K were considered: (i) $M=2$, $K=2$; (ii) $M=2$, $K=4$; and (iii) $M=3$, $K=2$. The KL-divergence was estimated using 100 sequences sampled from the model with 20 observations each.

Table 5.1 shows the results obtained by True, Offline, Single, IHMM-E, IHMM-G, IHMM-V, IHMM-H, and IHMM-R using synthetic data. Offline achieves high accuracy, but it overfits the data, as shown by its likelihood and divergence results. Single is even more prone to overfitting due to its exponential model complexity. IHMM-E is less af-

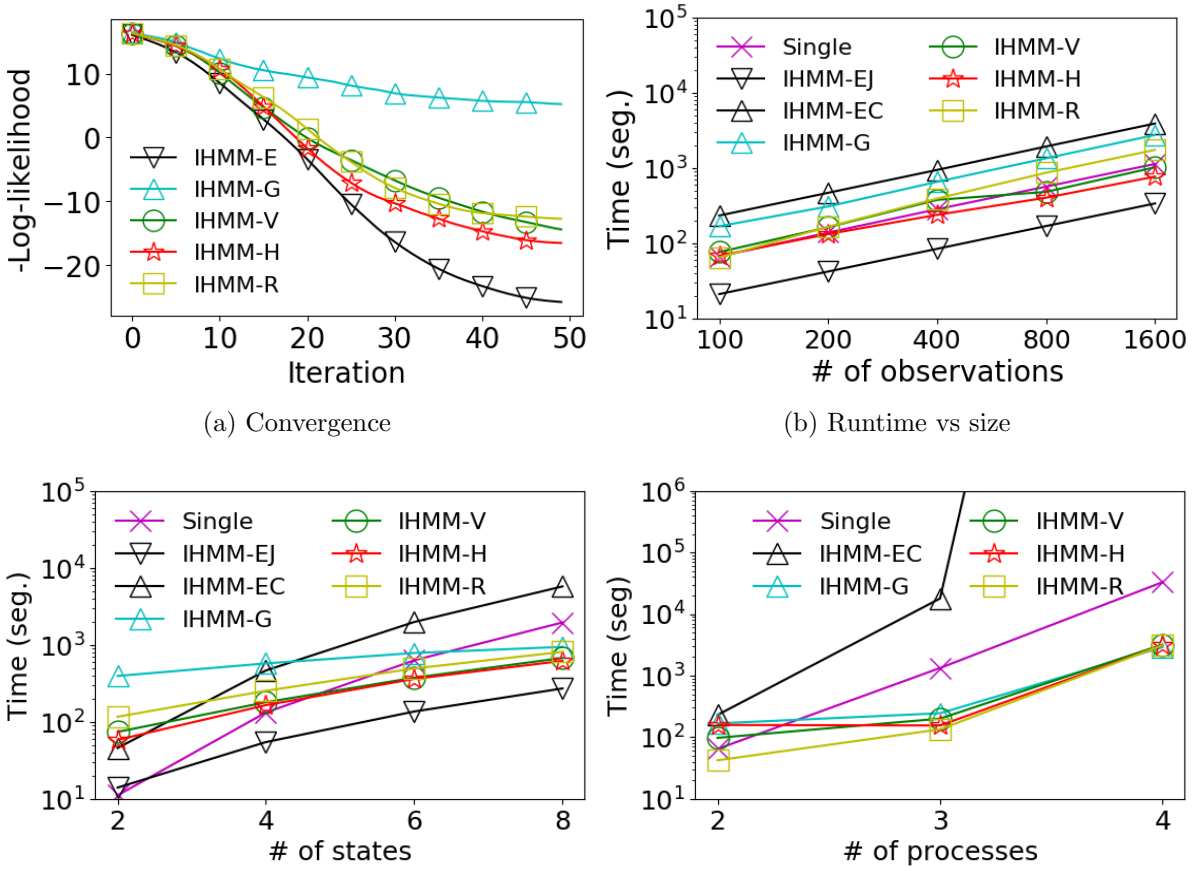
fected by overfitting than Offline, but also achieves significantly lower accuracy results. IHMM-G did not perform well for most of the metrics, even when the number of samples applied turned its running time prohibitive. BP approaches (IHMM-V/H/R) often outperform exact inference in terms of rand-score and were less affected by overfitting when $M=2$ and $K=4$—the same did not hold for $K=2$. A similar trend can be noticed for the KL-divergence. A more complex message schedule (IHMM-R) did not consistently lead to better inference results, being competitive with IHMM-H and IHMM-V.

Figure 5.3(a) shows the convergence of IHMM inference methods for increasing number of EM iterations ($M = 2$,$K = 4$). Results agree with our analysis of Table 5.1, with IHMM-G underfitting, IHMM-E overfitting, and BP providing a tradeoff between convergence and generalization.

We also evaluate the impact of different parameters over the running time of the algorithms in Figures 5.3(b-d). Default values for number of observations (T), states (K), and processes (M) were set to 100, 5, and 2, respectively. When varying the number of observations and processes, we consider both the cartersian product (IHMM-EC) and the junction tree (IHMM-EJ) approaches for exact inference. IHMM-EJ is the fastest among the algorithms due to the small number of processes—but would still be prohibitive for larger M. BP solutions, especially IHMM-H and IHMM-V, achieve good scalability across parameters.

5.6 Conclusions and Ongoing Work

In this work, we have studied the problem of learning interleaved hidden Markov models (IHMMs). After formalizing IHMMs, we have introduced exact inference algorithms for the model, which are shown to be intractable. Thus, we focused on the problem of efficiently learning IHMM parameters via approximate inference. In particular, we



(a) Convergence

(b) Runtime vs size

(c) Runtime vs # states

(d) Runtime vs # processes

Figure 5.3: Convergence and running times of the IHMM inference approaches using synthetic data.

have proposed a Loopy Belief Propagation (LBP) algorithm for IHMMs. Using synthetic data, we show that our algorithm is a good alternative to exact inference in terms of running time, outperforming a different solution based on Gibbs sampling. However, we have noticed in our experiments that IHMMs with many processes and states can easily overfit without a large number of observations. However, such settings are challenging even for our LBP algorithm. This is the main motivation for our ongoing work.

We are currently working on a different approach for IHMM inference using the so called *method-of-moments* (MoM) [133, 134]. The idea is to compute low-order moments

of an IHMM process—frequencies of sequences with one, two, and three symbols—and learn a model by matching these moments. A major advantage of this approach is allowing polynomial-time minima-free learning (consistency). However, such an estimator is known to be biased, which is often remedied by the application of an iterative method (e.g. exact inference of belief propagation) after the MoM.

Chapter 6

Other Problems

In this chapter, we summarize our work on a few problems that, though related to the topic of this thesis, are not its core contributions. First, we discuss network design, which is a set of problems where, given a graph, the goal is induce some property of interest using a budgeted number of (node or edge) modifications. Three design problems are introduced—centrality maximization, k-core minimization, and influence limitation—and our main results are briefly described. Please refer to [7, 8, 9] for more details. Next, we change our focus to outlier detection on graphs. The problem consists not only on finding abnormal graph instances, based on both the graph structure and real-valued node attributes, but also identifying subnetworks that explain a given outlier. We address this problem using a classification framework (SVM), which is shown to achieve effective results. More details on this work are given in [10].

6.1 Network Design

6.1.1 Overview

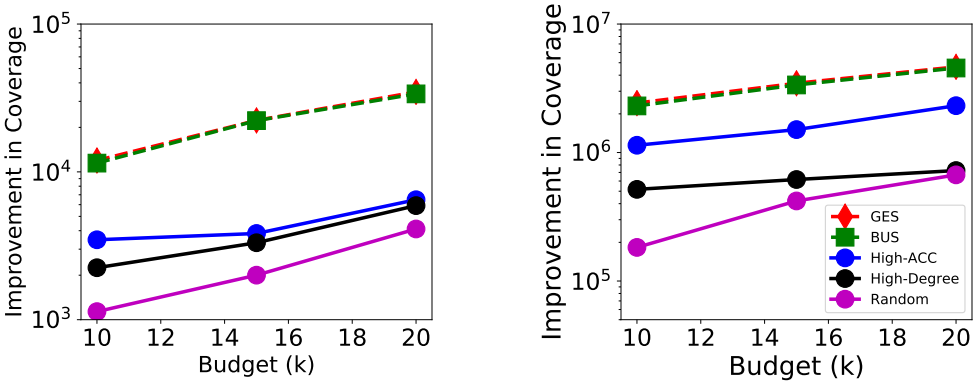
Network design is a recent area of study focused on modifying or redesigning a network in order to achieve a desired property [135]. As networks become a popular framework for modeling complex systems (e.g. VLSI, transportation, communication, society), network design provides key controlling capabilities over these systems, especially when resources are constrained. Existing work has investigated the optimization of global network properties, such as minimum spanning tree [136], shortest-path distances [137, 138, 139], diameter [140], and information diffusion-related metrics [141, 142] via a few local (e.g. vertex, edge-level) upgrades. Due to the large scale of real networks, computing a global network property becomes time-intensive. For instance, computing all-pairs shortest paths in large networks is prohibitive. As a consequence, design problems are inherently challenging. Moreover, because of the combinatorial nature of these local modifications, network design problems are often NP-hard, and thus, require the development of efficient approximation algorithms.

6.1.2 Group Centrality Maximization via Network Design

We focus on a novel network design problem, that improves the *group centrality*. Given a node v, its coverage centrality is the number of distinct node pairs for which a shortest path passes through v, whereas its betweenness centrality is the sum of the fraction of shortest paths between all distinct pair of nodes passing through v. The centrality of a group X is a function of the shortest paths that go through members of X [143]. *Our goal is to maximize group centrality, for a target group of nodes, via a small number of edge additions.*

Dataset Name	$\|V\|$	$\|E\|$
email-Eu-core (EU)	1k	25k
ca-GrQc (CG)	5K	14K
email-Enron (EE)	36K	183K
loc-Brightkite (LB)	58K	214K
loc-Gowalla (LG)	196K	950K
web-Stanford (WS)	280K	2.3M
DBLP (DB)	1.1M	5M

Table 6.1: Dataset description and statistics.



(a) Quality on EU (b) Quality on CG

Figure 6.1: BUS vs. Greedy: Improvement in coverage centrality produced by different algorithms.

The main contributions of this work can be summarized as follows: (1) We study a novel general network design problem, the group centrality optimization, and prove that it is NP-hard as well as APX-hard; (2) we propose a greedy algorithm and faster sampling algorithms for group centrality maximization; and (3) we show the effectiveness of our algorithms on several datasets and also prove their theoretical guarantees for a constrained version of the problem.

We evaluate our algorithms on real-world networks. All experiments were conducted on a 3.30GHz Intel Core i7 machine with 30 GB RAM and Ubuntu. Algorithms were implemented in Java. We consider three baselines in our experiments: 1) *High-ACC [143, 144]:* Finds the top k central nodes based on *maximum adaptive centrality coverage*

	Coverage of BUS (relative to baselines)				Time [sec.]			# Samples
Budget	GES	High-ACC	High-Degree	Random	GES	High-ACC	BUS	BUS
$k = 10$	0.96	3.3	5.1	10.1	271	2.3	1.8	2093
$k = 15$	0.97	5.8	6.7	11.1	423	2.4	3.4	3139
$k = 20$	0.97	5.2	5.7	8.2	531	2.5	4.5	4186

Table 6.2: EU data: Comparison of our sampling algorithm (BUS) and the baselines using the EU dataset.

and adds edges between target nodes X and the set of top-k central nodes; 2) *High-Degree:* Selects edges between the target nodes X and the top k high degree nodes; 3) *Random:* Randomly chooses k edges from Γ. We compare our sampling algorithm (BUS) against our Greedy solution (GES) and show that BUS is more efficient while producing similar results in terms of quality. The quality of a solution set (edges produced by the algorithm) is the number of newly covered pairs by the target set of nodes after addition of these edges to the initial graph. We call it *improvement in coverage*. The datasets applied in the experiments are summarized in Table 6.1.

Fig. 6.1 shows the number of new pairs covered by the algorithms. Table 6.2 shows the running times and the quality of BUS relative to the baselines—i.e. how many times more pairs are covered by BUS compared to a given baseline on EU data. BUS and GES produce results at least 2 times better than the baselines. Moreover, BUS achieves results comparable to GES while being 2-3 orders of magnitude faster.

We compare our sampling algorithm against the baseline methods on large graphs (EE, LB, LG, WS and DB). Due to the high cost of computing all-pairs shortest paths, we estimate the centrality based on $10K$ randomly selected pairs. For High-ACC, we also use sampling for adaptive coverage centrality computation [143, 144] and the same number of samples is used. The budget and target set sizes are set as 20 and 5, respectively.

Table 6.3 shows the results, where the quality is relative to BUS results. BUS takes a few minutes ($8, 15, 17, 45, 85$ minutes for EE, LB, WS, LG and DB respectively) to run

and significantly outperforms the baselines. This happens as the existing approaches do not take into account the dependencies between the edges selected. BUS selects the edges sequentially, considering the effect of edges selected in previous steps.

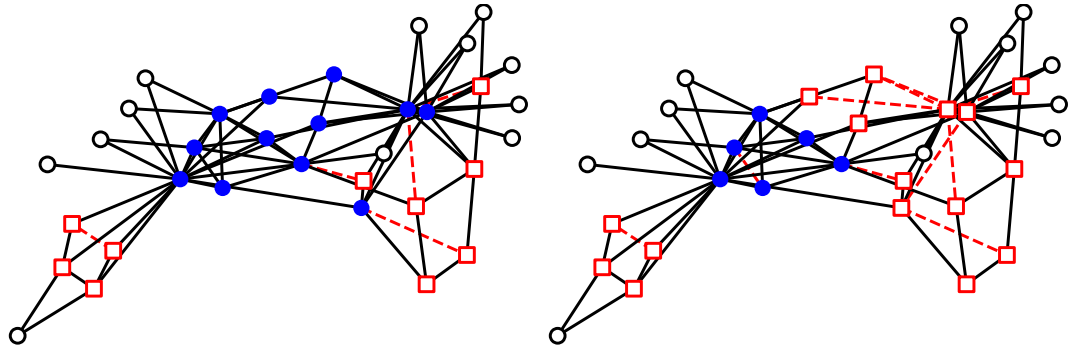
	BUS (relative to baselines)			**# Samples**
Data	High-ACC	High-Degree	Random	BUS
EE	4.88	2.74	51	6462
LB	3.3	2.3	33.8	6796
LG	3.3	4.2	62	4255
WS	1.89	1.95	4.8	2000
DB	2.5	1.6	5	875

Table 6.3: Coverage centrality of BUS relative to baselines.

6.1.3 A Game Theoretic Approach For Core Resilience

K-cores play an important role in revealing the higher-order organization of networks. A k-core [145] is a maximal induced subgraph where all vertices have internal degree of at least k. These cohesive subgraphs have been applied to model users' engagement and viral marketing in social networks [146, 147]. Other applications include anomaly detection [148], community discovery [149], protein function prediction [150], and visualization [151, 152]. However, the k-core structure can be quite unstable under network modification. For instance, removing only a few edges from the graph might lead to the collapse of its core structure. This motivates the k-core minimization problem: *Given a graph G and constant k, find a small set of b edges for which the removal minimizes the size of the k-core structure [153].*

The algorithm for k-core minimization proposed in this work applies the concept of *Shapley values* (SVs), which, in the context of cooperative game theory, measure the contribution of players in coalitions [154]. Our algorithm selects edges with largest Shapley value to account for the joint effect (or cooperation) of multiple edges. Since



(a) $b = 5$ (b) $b = 10$

Figure 6.2: K-core ($k = 3$) minimization on the Newman's Karate network: (a) $b = 5$ and (b) $b = 10$. Unfilled circle nodes are not in the 3-core of the original network. After removal of b dashed (red) edges, filled (blue) circle nodes remain in the 3-core and unfilled (red) square nodes are removed from the 3-core.

computing SVs is NP-hard, we approximate them in polynomial time via a randomized algorithm with quality guarantees.

Our main contributions are summarized as follows: (1) We study k-core minimization (KCM), which consists of finding a small set of edges, removal of which minimizes the size of the k-core structure of a network; (2) we show that KCM is NP-hard, even to approximate by a constant for $k \geq 3$, and also discuss the parameterized complexity of KCM and show the problem is $W[2]$-hard for the same values of k; (3) given the above inapproximability result, we propose a randomized Shapley Value based algorithm that efficiently accounts for the interdependence among the candidate edges for removal; and (4) we show that our algorithm is both accurate and efficient using several datasets and illustrate how KCM can be applied to profile the structural resilience of real networks.

In Figure 6.2, we demonstrate the application of our algorithm for KCM using the popular Newman's Karate network with two different budget settings, $b = 5$ and $b = 10$, and k fixed to 3. Unfilled circles are nodes initially out of the 3-core. The dashed (red) edges are removed by our algorithm—often connecting fringe nodes. Filled (blue) circles and unfilled (red) squares represent nodes that remain and are removed from the

3-core, respectively, after edge removals. All our experiments were conducted on a 2.59 GHz Intel Core i7-4720HQ machine with 16 GB RAM running Windows 10. Algorithms were implemented in Java. The real datasets used in our experiments are available online and are mostly from SNAP[1]. The Human and Yeast datasets are available in [19]. In these datasets the nodes and the edges correspond to genes and interactions (protein-protein and genetic interactions) respectively. The Facebook dataset is from [155]. Table 6.4 shows dataset statistics, including the largest k-core (a.k.a. degeneracy). These are undirected and unweighted graphs from various applications: EE is from email communication; FB is an online social network, WS is a Web graph, DB is a collaboration network and CA is a product co-purchasing network. We also apply a random graph (ER) generated using the Erdos-Renyi model.

Our algorithm is the *Shapley Value Based Cut (SV)*. Besides Greedy Cut (GC) [153], we also consider three more baselines in our experiments. *Low Jaccard Coefficient (JD)* removes the k edges with lowest Jaccard coefficient. Similarly, *Low-Degree (LD)* deletes k edges for which adjacent vertices have the lowest degree. We also apply *Random (RD)*, which simply deletes k edges from the candidate set Γ uniformly at random. Notice that while LD and JD are quite simple approaches for KCM, they often outperform GC.

We apply the percentage $DN(\%)$ of vertices from the initial graph G that leave the k-core after the deletion of a set of edges B (produced by a KCM algorithm):

$$DN(\%) = \frac{N_k(G) - N_k(G^B)}{N_k(G)} \times 100 \tag{6.1}$$

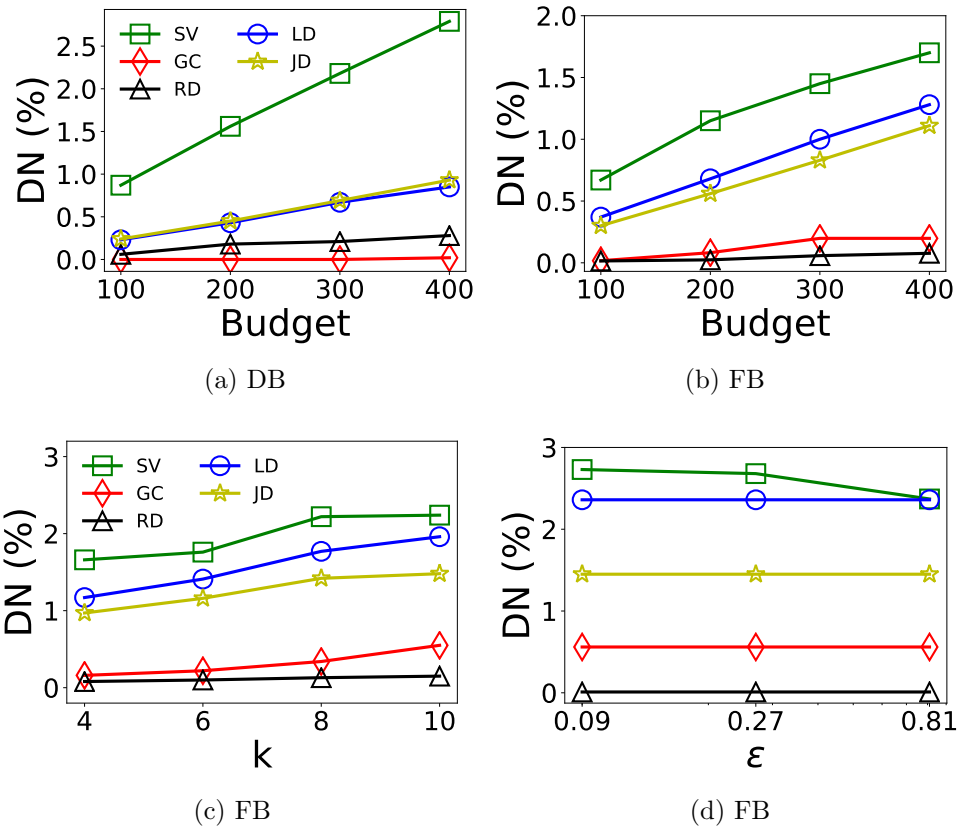
KCM algorithms are compared in terms of quality (DN(%)) for varying budget (b), core value k, and the error of the sampling scheme applied by the SV algorithm (ϵ).

Figure 6.3 presents k-core minimization results for $k = 5$—similar results were found

[1] `https://snap.stanford.edu`

Dataset Name	$\|V\|$	$\|E\|$	k_{max}	Type
Yeast	1K	2.6K	6	Biological
Human	3.6K	8.9K	8	Biological
Facebook (FB)	60K	1.5M	52	OSN
DBLP (DB)	317K	1M	113	Co-authorship

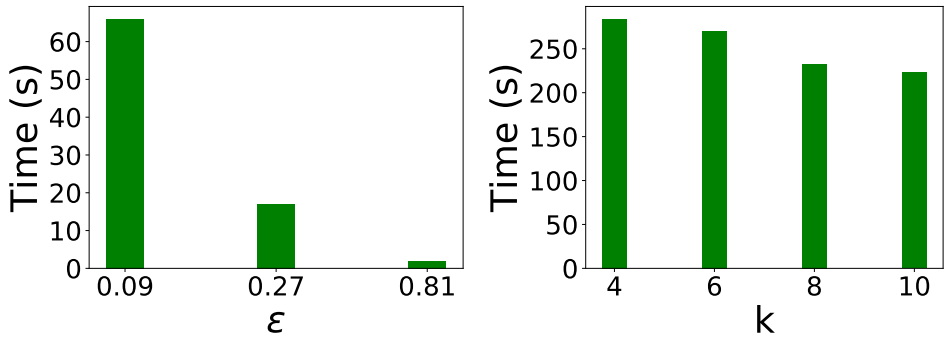
Table 6.4: Dataset descriptions and statistics. The value of k_{max} (or degeneracy) is the largest k among all the values of k for which there is a k-core in the graph.



(a) DB (b) FB

(c) FB (d) FB

Figure 6.3: **K-core minimization (DN(%)) for different algorithms varying (a-d) the number of edges in the budget; (e-f) the core parameter k; (g-h) and the sampling error ϵ. Some combinations of experiments and datasets are omitted due to space limitations, but those results are consistent with the ones presented here. The Shapley Value based Cut (SV) algorithm outperforms the best baseline (LD) by up to 6 times. On the other hand, the Greedy approach (GC) achieves worse results than the baselines, with the exception of RD, in most of the settings. SV error increases smoothly with ϵ and LD becomes a good alternative for large values of k.**



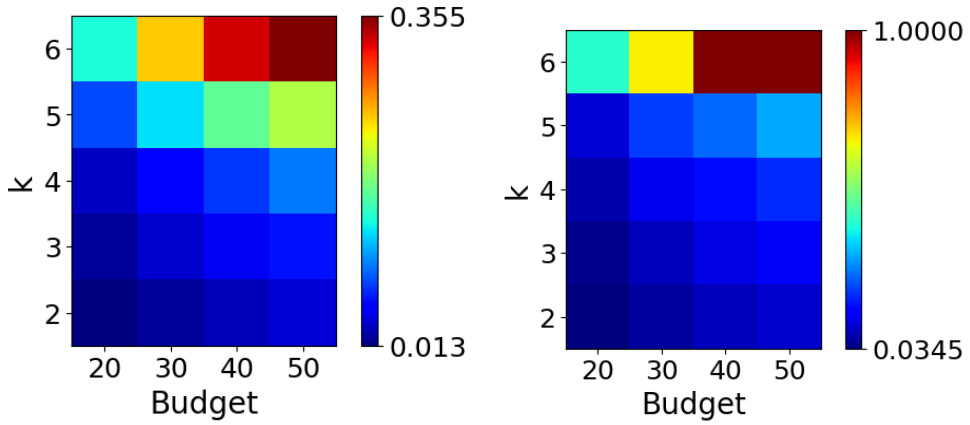
(a) Varying ϵ (b) Varying k

Figure 6.4: Running times by SV using FB while varying (a) the sampling error ϵ and (b) the core parameter k; SV is efficient even for small values of sampling error and its running time decreases with k.

for $k = 10$—using four different datasets. SV outperforms the best baseline by up to six times. We evaluate the impact of k over quality for the algorithms using FB in Figure 6.3c. The budget (b) is set to 400. As in the previous experiments, SV outperforms the competing approaches. The parameter ϵ controls the sampling error of the SV algorithm. We show the effect of ϵ over the quality results for FB in Figure 6.3d. The values of b and k are set to 400 and 12 respectively. The trade-off between ϵ and the running time of our algorithm enables both accurate and efficient k-core minimization.

Running times for SV varying the sampling error (ϵ) and the core parameter (k) using the FB dataset are given in Figures 6.4a and 6.4b, respectively. Even for small error, the algorithm is able to process graphs with tens of thousands of vertices and millions of edges in, roughly, one minute.

K-cores have been previously applied in the analysis of functional modules in protein-protein networks [151, 156]. Here, we compare the k-core stability of Human and Yeast (Figs. 6.5a, 6.5b). Human is shown to be more stable, as can be inferred from the range of values in the profile—1% to 35% for Human and 3.4% to 100% for Yeast. Moreover, the profile for Human is smoother than Yeast. These results confirm our



(a) Human (Core resilience) (b) Yeast (Core resilience)

Figure 6.5: Core resilience for the Human and Yeast protein-protein interaction networks.

intuition that proteins have a more complex functional structure in Humans compared to other organisms [157].

6.1.4 Data-driven Influence Limitation

Online social networks, such as Facebook and Twitter, were popularized mostly as platforms for sharing entertaining content and maintaining friendship. However, they have been quickly transformed into major battlegrounds for political campaigns, viral marketing, and the dissemination of news. With this shift, the increase in the number of "bad actors", such as tyrannical governments, spammers, and bullies exploiting these platforms has become a key challenge for their administrators, businesses and society. A questionable approach to control the diffusion of misinformation in social platforms is via stricter laws and regulations by governments. This control often happens in detriment of the democratic and organic structure that are central to these platforms. Instead, a more sensible approach is to limit the impact of bad actors in the network while minimizing the disruption of its structure.

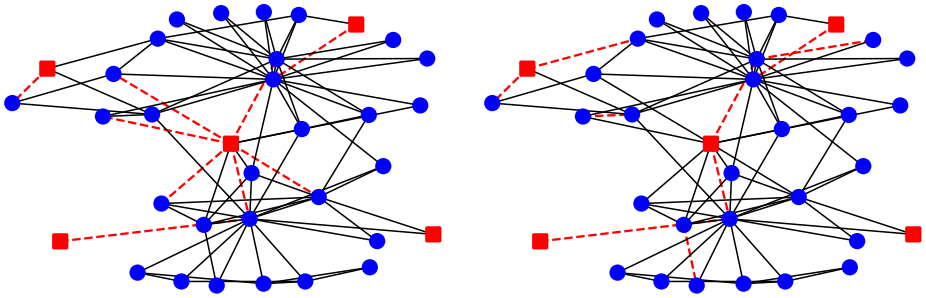
This work formalizes the *influence limitation problem*. We focus on a setting where

the network is modified via the removal (or blocking) of a few edges or nodes. These modifications can be implemented by social network administrators or induced by other organizations or governments via advertising campaigns. Although we focus on influence limitation, our problem is also relevant from the perspective of an agent that aims to maintain the influence of a set of users. Nodes/edges discovered by our algorithm are those that should be protected by such an agent. Similarly, while we focus on the edge version of our problem, the techniques discussed here also apply to the node version.

We propose a data-driven approach for influence minimization based on historical data. Moreover, we study the influence limitation problem not only under a budget constraint but also under a more general set of matroid constraints [158, 159]. Our contributions are summarized as follows: (1) We investigate the novel data-driven influence limitation problem via node/edge removals, showing that the edge version is more general and covers the node version of the problem; (2) we study our problem under both budget and matroid constraints, discussing how these affect algorithmic design; (3) we show that the influence limitation problem is APX-hard and propose constant-factor approximations for both versions of the problem—deterministic and probabilistic approximation for the budget and matroid version, respectively; and (4) we show that our methods outperform baseline solutions by up to 35% while scaling to large graphs.

BIL (Budgeted Influence Limitation) assumes that any k edges in the candidate set can be removed. As a consequence, an optimal solution for BIL might make the network disconnected or disproportionately affect particular portions of the network. Fig. 6.6 exemplifies this issue using the Newman's karate[2] network. BIL modifications are strongly biased towards a small set of nodes. ILM (Influence Limitation under Matroid) enforces network modifications that are more uniformly distributed across the network. Notice that a valid solution for the budget constrained version (BIL) might not necessarily be a

[2]http://www-personal.umich.edu/ mejn/netdata/



(a) BIL (b) ILM

Figure 6.6: We perform our methods for BIL (a) and ILM (b) on the Newman's Karate network with $|X| = 5, k = 9$. Square (red) nodes are in the target set, $|X|$, and dotted (red) edges are in the solution set. The edges are incident to few nodes in the solution for BIL, being strongly biased towards a small set of nodes. For ILM, we have considered $b = 2$, which leads to a solution with more uniform set of edges.

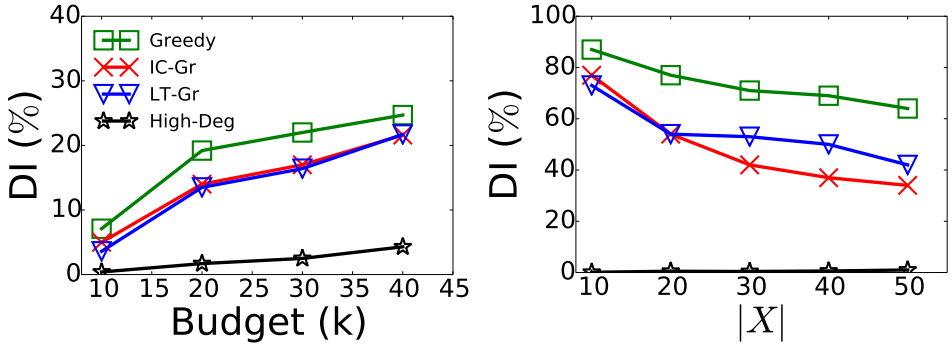
valid solution for ILM and vice-versa.

Our solutions were implemented in Java and experiments were conducted on 3.30GHz Intel core with 30 GB RAM. We show results using the **ca-AstroPh (CA)** dataset, with $18K$ vertices, $197K$ edges $1K$ actions and $56K$ tuples. Influence probabilities are learned using the method proposed in [160]. The quality of a solution set B (a set of edges) is the percentage of *Decrease in Influence (DI)* of X:

$$DI(B) = \frac{(\sigma_{cd}(G, X) - \sigma_{cd}(G^m, X))}{\sigma_{cd}(G, X)} \times 100 \tag{6.2}$$

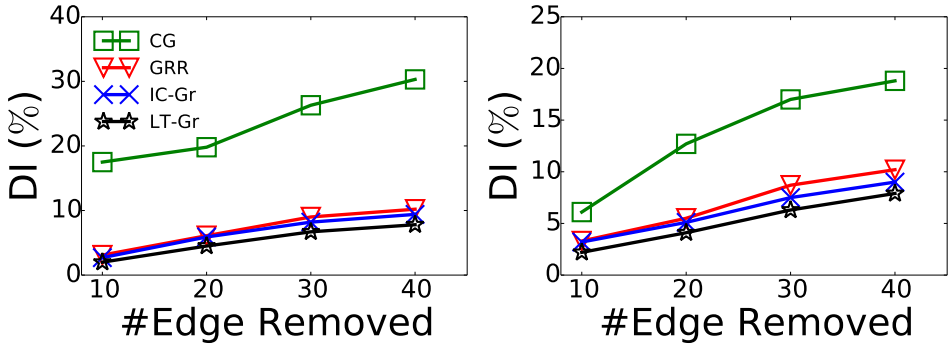
The set X is randomly selected from the set of top 150 nodes with highest number of actions. The candidate set C contains edges that appear at least once in any action graph. The number of MC simulations for IC and LT-based baselines is at least 1000.

Baselines for BIL: 1) **IC-Gr [161]:** Finds the top k edges based on the greedy algorithms that minimize influence via edge deletion under the IC model. 2) **LT-Gr [141]:**Finds the top k edges based on the greedy algorithm proposed in [141]. Here, the authors minimize the influence of a set of nodes according to the LT model via edge



(a) CA (varying k) (b) CA (varying $|X|$)

Figure 6.7: [BIL] (a) Decrease in Influence (DI) produced by different algorithms. (b) DI produced by different algorithms varying the size of the target set, X with $k = 30$.



(a) CA (b=1) (b) CA (b=2)

Figure 6.8: **[ILM]** Decrease in Influence produced by different algorithms on CA. Our algorithm, CG outperforms the baselines by up to 20%.

deletion. Note that we also apply optimization techniques proposed in [141] for both of these baselines. 3) **High-Degree:** Selects edges between the target nodes X and the top-k high degree nodes. Other heuristics (*Friends of a Friend* and random selection) did not produce better results than High-Deg.

We compare our Greedy algorithm against the baselines on three datasets (CA, FXS and FCS) in Figure 6.7a (target size is set as 30). Greedy takes a few seconds to run and significantly outperforms the baselines (by up to 35%) in terms of DI(%). The running time of Greedy is low as it avoids expensive Monte-Carlo simulations. For CA, the action graphs are generated by IC model and hence, IC-Gr produces good results. We also analyze the impact of varying the number of target nodes ($|X|$) in Figure 6.7b. Greedy provides better DI (by up to 35%) across all $|X|$ and datasets.

Baselines for ILM: 1) **Greedy with Restriction (GRR):** Finds the feasible edges (respecting the matroid constraint) using BIL (greedy). (2-3) We also apply **IC-Gr** and **LT-Gr** with the edge removal constraint. The number of samples and iterations used in CG are $s = 20$ and $\tau = 100$, respectively. After obtaining the solution vector from CG, we run randomized rounding for 50 times and choose the best solution.

We compare the Continuous Greedy (CG) algorithm against the baseline methods on CA ($|X| = 30$) varying the number of edges removed in Figure 6.8 using $b = 1$ and $b = 2$. CG outperforms the baselines by up to 20%. GRR does not produce good results as it has to select the feasible edge that does not violate the maximum edge removal constraint b. While maintaining feasibility, GRR cannot select the current true best edge.

6.2 Outlier detection on graphs with subnetwork interpretation

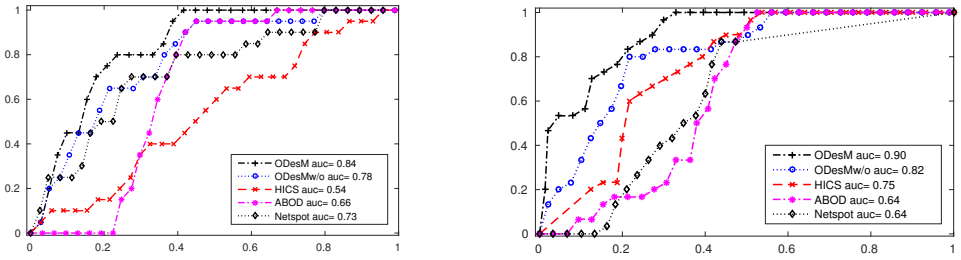
6.2.1 Overview

Detecting and characterizing exceptional patterns is an important task in many domains ranging from fraud detection, environmental surveillance, to various health care applications [162]. This problem is often referred to as *outlier* or *anomaly* detection in the literature. Although identifying anomalous subjects has been widely studied in high dimensional data and recently extended to the network context [162], the problem remains very challenging. In the network setting, most existing works focus on searching individual nodes [163], or groups of linked nodes [164] whose structures or behaviors are irregular. Though these studies have provided intuitive concepts about outlying patterns defined in the respect of network connectivity, most results are limited to the setting of a single static network. Other recent studies have extended the scope of analysis to evolving networks [165], but the focus is on event/change detection where the temporal dimension is a key factor for defining outliers.

In this work, we address the problem of identifying anomalous networks from a database of multiple network samples while at the same time investigating *why* a network is exceptional. An outlier is defined at the global level of an entire network sample but we use local subnetworks to explain its exceptionality. Although the outlierness of a network sample can be quantified via the outlier degree, such a single measure only bears limited explanatory information [166] since it lacks the capability of showing in what data view, i.e. local subnetworks, an anomalous network is most exceptional. Moreover, although two networks may have similar outlier degrees, the local subnetworks that make them abnormal might be quite different since the anomalous networks themselves are usually

not homogeneous. For example, exploring a database of gene networks for outliers can lead to the isolation of subjects suffering from cancer. However, the gene pathway (local subnetwork) that causes the disease can vary from subject to subject due to the complexity of the disease [167], or even depending on different stages of the disease. Spotting an unhealthy subject is generally not sufficient. Figuring out what abnormal gene subnetwork leads to the disease is usually more important since it helps to develop possible and effective treatments.

We develop a novel algorithm that exploits network regression models combined with network topology regularization to concurrently address the two important problems mentioned above. Specifically, we treat each network sample as a potential outlier and determine local subnetworks that help discriminate it from nearby regular network samples. Our objective function is formulated under the framework of network regression where we first upsample the outlier candidate network in order to make the binary regression problem balanced. The objective function is then regularized by the network topology and further penalized by L1-norm shrinkage to perform subnetwork discovery. It can be shown that the combined objective function has a form closely related to the dual SVM [168, 169], which can be further optimized in the primal form using Newton's method. The objective function is proven to be convex, which is key to guaranteeing the convergence of the algorithm. Our algorithm, therefore, goes beyond the simple strategy of subspaces/subgraphs examination by directly learning the most discriminative subnetworks with respect to each network sample. Consequently, the outlier degree can be appropriately computed within the space spanned by these selected subnetworks and, collectively, they form a ranking of all network samples based on the outlier scores.



(a) CMUFace (b) LATraffic

Figure 6.9: ROC curve performance of all algorithms on identifying outlier networks from the CMUFace and LATraffic datasets.

6.2.2 Results

We name our algorithm ODeSM (Outlier Detection with Subgraph Mining), and compare its performance against techniques in both network studies and high dimensional studies: (1) Netspot [165] without temporal constraint so allowing it to uncover network regions from each individual network; (2) HiCS [170] that seeks outliers through contrast subspaces for high dimensional data; (3) ABOD [171] which discovers outliers via variance of angles between vector triples; (4) ODesMw/o, a variant of our method without exploiting network regularization. The parameter setting for ODesM and ODesMw/o follows the best-effort-approach [172].

CMUFace graph data

Since most network datasets (presented next) lack ground-truth subnetworks, we conduct an experiment on the CMUFace image data (http://archive.ics.uci.edu) as it allows us to evaluate the relevance of uncovered subnetworks *via visualization*. The network dataset is generated with the procedure described in [173], with the following modification: we select all networks with open-eye images as inliers, and randomly select one with sunglasses from any of 4 poses (straight/up/left/right) as an outlier. This allows us to evaluate whether an algorithm can deal with the heterogeneity in the data. The

dataset consists of 303 regular network samples and 20 anomalous ones.

Outlier identification: In Fig.6.9a, we plot the ROC curve performance of all five algorithms. As seen from this figure, both techniques HiCS and ABOD designed for high dimensional data perform moderately well on this dataset. ABOD explores the variance over angles between an outlier candidate and every pair of other two samples, so its approach explores global outliers deviating from a single distribution of inliers. For this dataset, however, we have multiple distributions, and thus true outliers are harder determined by solely relying on the angles. HiCS, on the other hand, attempts to find most contrasting subspaces using a bottom-up approach and it starts with those of 2-dimension (from a pool of $\binom{1920}{2} = 1844160$ possible subspaces). If such low dimensional subspaces are not well sampled, the quality of contrasting subspaces found in higher dimensional subspaces can be compromised. Netspot performs better than these two techniques by relying on the p-value defined at each node. However, by converting to a p-value, Netspot also removes the contrast among node values and thus is less successful in seeking the most potential seed-nodes. ODesM performance is the best with its AUC at 0.84, as compared to 0.78 obtained by the second best ODesMw/o. This large gap in AUC also confirms the key role of network topology exploited by ODesM.

Explanatory subnetworks: We further explore the set of subnetworks discovered by ODesM as the explanation for top ranking outliers. Out of top 20 anomalous networks, 8 are true outliers. We plot in Fig.6.10(a-c) the three top ranked networks that are also truly labeled as outliers, and their corresponding images. As observed, despite coming from different poses, the outlier networks are still well-identified and the subnetworks located around the sunglasses are appropriately selected by ODesM, though they can vary across different outliers. Fig.6.10(d) shows a network sample ranked high by ODesM but not a true outlier according to the labeling based on sunglasses.

ABOD, ODesMw/o and HiCS are not network-based techniques. Hence, we se-

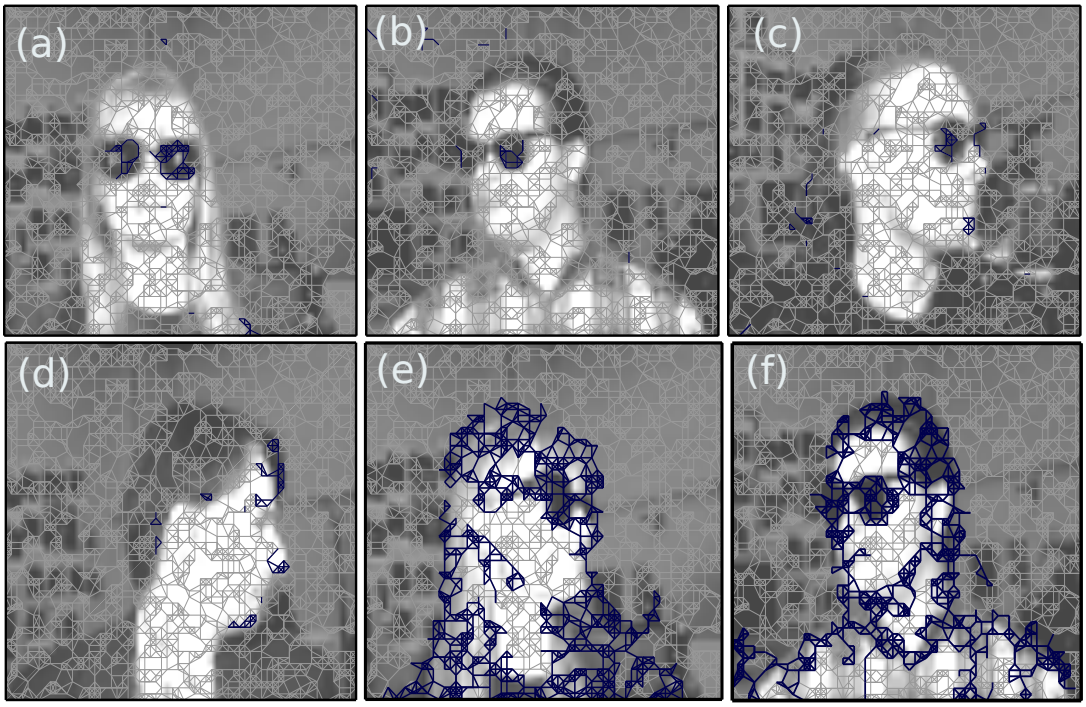


Figure 6.10: Subnetworks selected by ODesM ((a)-(d)) and Netspot ((e)-(f)) in CMUFace. In each figure, the network topology is shown in grey and the selected subnetworks are shown in blue, while the corresponding full image is shown in background with dimmed colors to improve the visualization.

lect Netspot for comparison based on its discovered anomalous subnetwork regions. In Fig.6.10(e-f), we plot two typical true outliers found from 20 top networks ranked by Netspot. Unlike the subnetworks discovered by our method, it is harder to justify why the corresponding images are exceptional though the uncovered subnetworks are strongly connected. In both figures, the substructures from entire faces have been selected. This performance probably comes from the fact that, other than p-value, Netspot also relies on the adjacency of network samples to derive the time interval at which significant anomalous regions can appear. However, once the interval is set to 1 (i.e. for each individual network and no temporal development), it has limited information to justify the relevance of a network region. Thus, the p-value computed at each node is likely playing the key role. And as long as its values do not change abruptly, Netspot tends to select all of them, forming a large subnetwork structure. The patterns discovered by Netspot and our ODesM are thus fundamentally different. For this reason, we do not attempt to compare their uncovered subnetworks in subsequent experiments.

Road traffic networks

The last dataset we use for evaluation is LATraffic—the highway traffic network data of Los Angeles, California (http://pems.dot.ca.gov) during April 2011. Based on the distribution of the average speed computed for each network snapshot, we randomly select 300 snapshots around the mean of this distribution to label as regular networks, and other 30 snapshots from two extreme tails (15 each) to label as anomalous networks.

Outlier identification: The ROC curve performance of all algorithms on the LATraffic is shown in Fig.6.9b. HiCS handles the subspace candidates well and its Monte-Carlo sampling based approach tends to select highly contrasting subspaces. Netspot is less successful in uncovering both types of low and high speed outliers. Among all examined techniques, ODesM is still the best performer with an AUC of 0.9.

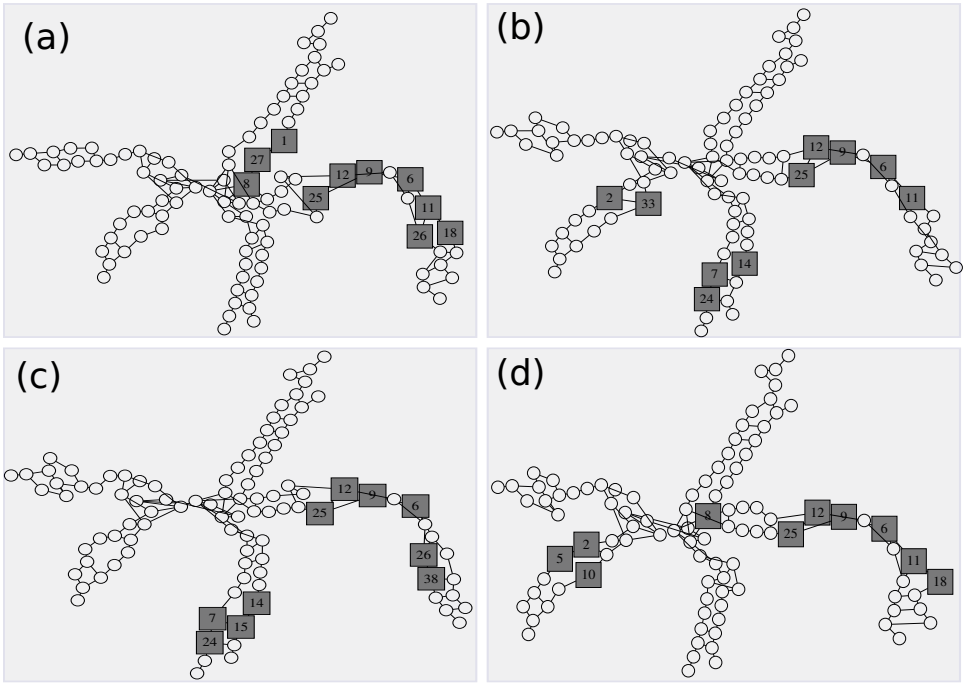


Figure 6.11: Top four outlier networks discovered by ODesM from the LATraffic dataset. Road segments involved in the explanatory subnetworks are shaded.

Explanatory subnetworks: We further explore the set of subnetworks ranked top by ODesM. Fig.6.11 depicts the uncovered subnetworks for the top four outlier networks. The networks in (a) and (d) are the true outliers with low speed while the ones in (b) and (c) are the true outliers with high speed. The sets of discovered subnetworks in both cases are quite consistent. An interesting point emerges upon closer inspection of these explanatory substructures. We would expect the explanatory subnetworks for two types of outliers to be different since one considers low speeds while the other one considers high speeds. However, it turns out that they share one large subnetwork spanned by nodes 11, 6, 9, 12 and 25. The common selection of this subnetwork may suggest that such a set of adjacent road segments is highly sensitive to traffic congestion. For monitoring purposes, these road segments should be the top candidate since they are likely to reflect the overall condition of the entire traffic network.

Chapter 7

Conclusions, Ongoing and Future Work

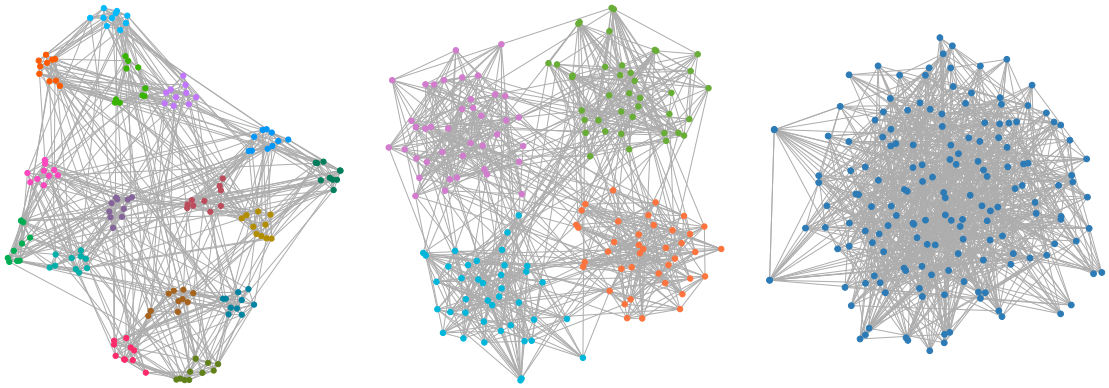
In this closing section, we first summarize three ongoing projects (Section 7.1), re-visit the main conclusions of this work including the statement of the thesis (Section 7.2), and list a few promising directions for future research (Section 7.3).

7.1 Ongoing Work

This section summarizes three ongoing projects that were motivated by the research developed as part of this thesis.

7.1.1 Multiscale Network Embedding

Many relevant network processes are multiscale. For instance, flights in a airport network might be local, national, or international. Another example is hyperlinked content, such as the Web and Wikipedia, where links between pages might be due to semantic relationships at multiple levels (e.g. shared topic or sub-topic). Understanding the impact



(a) t=10^{-5} (b) t=10^{2} (c) t=10^{5}

Figure 7.1: Embedding single network at multiple scales.

of processes at multiple scales is particularly relevant for learning representations (or embeddings) for networked data, such as nodes, edges and subgraphs. Representation learning is a powerful approach for learning problems on graphs—e.g. link prediction and node classification. In these settings, embeddings might be optimized for different scales depending on the task at hand. In the case of link prediction, an embedding able to predict links inside dense communities might perform poorly when predicting links across communities. Similarly, an embedding suitable for classifying pages into sub-topics might not be ideal for higher level topics. Here, we propose addressing these challenges via multiscale embedding.

Our approach is based on the concept of Markov Stability [174], which is a function of the auto-covariance of a Random-Walk process within bounded time. Our goal is to efficiently embed nodes in a network by approximating the auto-covariance matrix using a few dimensions. Figure 7.1 shows the embedding of a synthetic network using the proposed scheme at multiple time scales—small scale corresponds to small communities and large scale to large communities. We will apply the resulting embedding to solve multiscale downstream task on several datasets (power grid, airport, and Wikipedia) and analyze its performance against state-of-the-art approaches [175].

7.1.2 Event Detection on Dynamic Graphs

Learning complex graph features (e.g. cuts, subgraphs) is often computationally hard. However, in many applications the end goal is not to compute the features themselves but to apply them in order to perform a machine learning task (e.g. classification, regression, outlier detection) [176, 175]. In these settings, one natural question to be asked is whether relevant features can be learned directly from data? Automatic feature engineering is at the core of recent advances in machine learning, specially on deep learning applied to natural language processing, speech recognition and computer vision tasks. Our goal is to apply the same approach to graph settings.

The task we will focus on in our work is is event detection [177, 178, 179]. Given a dynamic graph with a few labels marking the occurrence of events of interest, the goal is to learn how to detect such events based on the graph evolution. Figure 7.2 illustrates our framework. We will train a machine learning model (a Neural Network) to predict events based on multiple global and local graph features over time. We will apply our framework to several real datasets, including the Enron emails, NY cab traces, and Twitter data.

7.1.3 Semantic Segmentation with Social Context

The Reddit Place experiment was a large-scale online social experiment launched by Reddit in 2017 [180]. Reddit users were allowed change the colors of few pixels/minute in a 1000x1000 canvas. In total, 16 colors were available and pixels changed by one user could be overwritten by others. After 72 hours, 1 million users placed 16 million updates (time,user,x,y,color) to the canvas. The dataset was later released to the Reddit community and several related projects were developed using the experiment data[1]. One of the projects was an atlas were users tagged more than 1500 different artworks in the

[1] `https://www.reddit.com/r/place/`

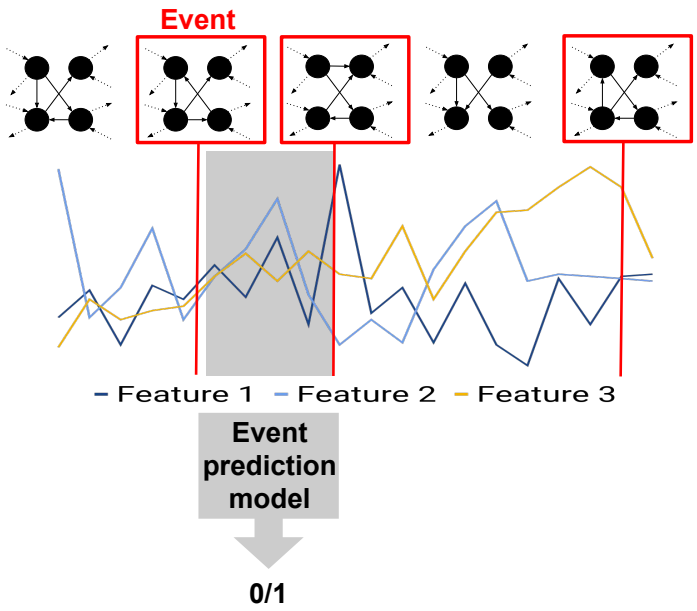
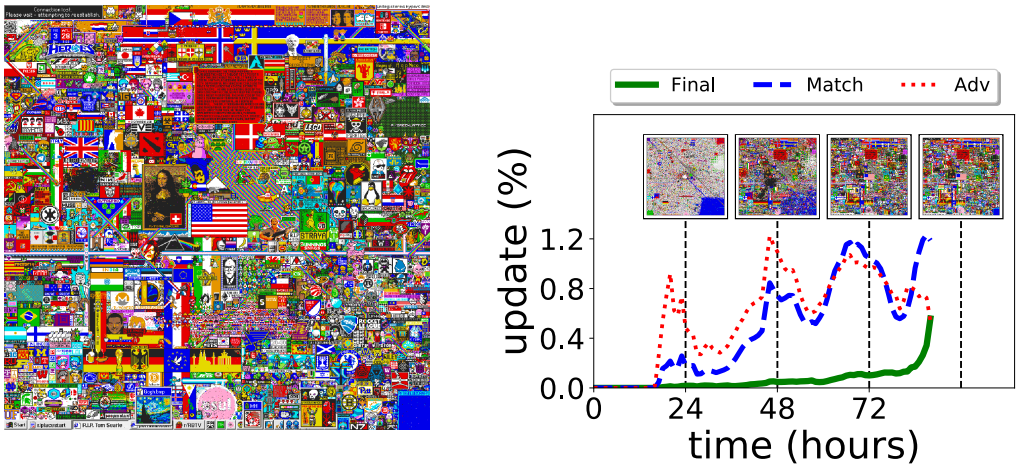


Figure 7.2: Event Detection Framework.

canvas. The atlas dataset was also made available to the community[2].

Our goal is to apply machine learning and graph algorithms to understand the user dynamics in the experiment. In particular, we are interested in analyzing engagement, collaboration, and competition patterns. In order to achieve such a goal, we first will segment the updates in the dataset into artworks. This will allow us to capture the intention of each update made to the canvas. Notice that this segmentation task is quite challenging due to the limited amount of information associated to each update. We will extend existing semantic segmentation algorithms[181] to exploit specific properties of the experiment (e.g. user information). Moreover, we will also apply more recent segmentation schemes based on Convolutional Neural Networks [182].

[2]https://www.reddit.com/r/place/comments/64zn88/the_rplace_atlas_an_interactive_map_with_details/



(a) Final Canvas (b) Progress

Figure 7.3: Reddit Place Experiment

7.2 Conclusions

In this thesis, we have investigated how to mine and model graph processes, where a set of interconnected entities change their states according to the time-varying behavior of an underlying complex system. Several problems and their respective solutions were discussed and validated using real datasets arising from many applications. In particular, we have emphasized the computational hardness of these problems and the theoretical justification and or empirical effectiveness of the algorithms proposed. The breadth and depth of the problems and techniques applied in this work together with their rigorous analysis has led as to the following thesis:

Mining and modeling processes on graphs leads often to problems that are not not only hard computationally (Chapters 2-4 and Section 6.1) but also in terms of inference (Chapters 2 and 5 and Section 6.1). They can be solved using spectral (Chapters 3, 4 and 5 and Sections 6.2 and 7.1.1), probabilistic (Chapter 2 and 5 and Section 6.1), and combinatorial optimization (Chapters 2-4 and Section 6.1) algorithms, and must take into account the graph structure and also large amounts of data traces from these

processes (Chapters 2-5 and Sections 7.1.1-7.1.3)

From a more personal perspective, these are some of the lessons learned along the development of the research described in this dissertation:

- Surprising effectiveness of spectral theory: Spectral graph theory, which connects combinatorial optimization problem on graphs to linear algebra, is problem one of the most elegant results in Algorithms. The more recent discovery of moment-based inference algorithms for graphical models is a result of similar stature that we expect to attract the attention of the research community in the upcoming years.

- Importance of real, high-quality data: Data science research is mostly driven by the availability of real data and by the challenges that arise when models and algorithms are applied in the real-world. However, and unfortunately, real high-quality data is often unavailable for academic research. Besides partnering with an organization that owns the data or performing expensive large-scale experiments, there seems to be no clear solution for this problem.

- Challenges in multidisciplinary research: While there is an increasing emphasis on multidisciplinary research recently, there are a few challenges that prevent such type of research efforts. Communication seems to be the major one. Moreover, it is hard to share the credit among the parts involved in an interdisciplinary project. Researchers tend to see their field as central and other fields as complementary.

- The hardest part of optimization is often selecting the objective: Many problems in data science involve proposing a novel objective function and then searching for efficient ways to optimize it. However, the novelty of the objective and the hardness of its optimization are often not of interest to the potential users of the solution being proposed. Selecting the right objective is key to the impact of the research.

- The route from theory to practice is long: It is easy to assume that an elegant theoretical result will work on practice, but it might not be the case. For instance, the connection between sparse graph cuts and Laplacian eigenvectors is a quite powerful idea. However, many real-world graphs do not seem to have sparse cuts as the ones assumed by the formulation. Moreover, real graphs might be disconnected, or have many trivial cuts that have low values of conductance/sparsity. Validating the main assumptions of a theoretical result on real data is a good first step before investing further time on applying the result on a data science problem.

- Only well-established problems deserve a sophisticated solution: A new problem might look quite hard at first, which induces the development of a sophisticated solution for any problem. However, the problem might later be found not to be relevant a significant number of people. Thus, in case the problem is not well-established, it is more effective to propose a simpler solution and then see if there is a real need for a better solution (or for a solution at all). Often times a solution to a different problem can be easily adapted to the new problem at hand.

Again, these lessons were not validated by experiments but are personal opinions based on the experience of writing this dissertation.

7.3 Future Work

Our work opens several venues for future research, which we summarize as follows:

Controlling network processes: In this thesis, we have focused on the modelling and mining of network processes. However, a more challenging task is to change the process towards a desired property. This line of work is related to network design (Section 6.1), but the models considered in our work were quite simple. In order to control

many relevant large-scale processes that arise in real applications, we need to better understand *causality* [183]. More specifically, we are interested in studying how models of counterfactual inference, can be combined with network design. Another related field of interest is *control theory* [184, 185], which, different from network design, does not focus on the combinatorial effect of decisions. Instead, control theory often imposes stronger assumptions on the behavior of a system (e.g. it is linear) and then studies conditions for the observability, identifiability, and controlability of the system. We are interested in connecting control theory, network design, and the analysis of network processes based on data traces. One particular scenario we will focus in the study of physical networks (e.g. power grids, water distribution systems, and road networks).

Probabilistic inference via spectral methods: Graphical models can capture complex relationships between different types of objects in a dataset. However, as discussed in Chapter 5, learning dense graphical models is often intractable in general. There are three major alternatives to address this challenge in the literature. The first is sampling (e.g. Gibbs) [119, 128], which is known to be an accurate but often slow solution. An alternative is approximate inference, which, as the name says, attempts to approximate the original model by a more treatable one [129, 118, 130, 117]. The second line of work is to define a class of models that enables efficient inference using the method of moments [133, 134]. While the second approach is more recent and promising, its application is much more difficult than for the first approach, as discussed for the case of IHMMs (Section 5). Moreover, the consistency guarantees of the MoM come at the cost of bias. Our knowledge of the trade-off between these three approaches, both theoretically and empirically is very limited. Addressing such questions might enable efficient inference for graphical models way beyond the ones we are able to process today.

Applications: While developing this thesis, we have found several interesting applications of network processes that we were unable to study in depth due to the lack

of time. The Reddit place project (Section 7.1.3) is a good example of such project, where the focus is more on the findings than on the novelty of the methods employed. As discussed in the introduction, data science is a field that emerged on the frontier between computer science, statistics and specific applications. However, these applications often require some domain knowledge and additional steps to enable the use of existing algorithms such as those discussed here. As businesses, governments, and the general public increase their expectations and awareness regarding the use data for decision-making, other requirements, such as the privacy [186], fairness [187], and accountability [188] of algorithms will have increasing importance. Today, data scientists do not possess the proper tools to explore these intricate trade-offs.

Bibliography

[1] A. Silva, W. Meira, Jr., and M. J. Zaki, *Mining attribute-structure correlated patterns in large attributed graphs*, in *VLDB*, 2012.

[2] A. Silva, X. H. Dang, P. Basu, A. Singh, and A. Swami, *Graph wavelets via sparse cuts*, in *KDD*, (New York, NY, USA), pp. 1175–1184, ACM, 2016.

[3] A. Silva, P. Anchuri, B. Zong, H. Chen, and A. Singh, *Learning interleaved markov processes*, in *Working Paper*, 2018.

[4] M. E. Newman, A.-L. E. Barabási, and D. J. Watts, *The structure and dynamics of networks.* Princeton university press, 2006.

[5] A. Silva, P. Bogdanov, and A. K. Singh, *Hierarchical in-network attribute compression via importance sampling*, in *ICDE*, 2015.

[6] A. Silva, A. Singh, and A. Swami, *Spectral algorithms for temporal graph cuts*, in *Proceedings of the 2018 World Wide Web Conference*, pp. 519–528, 2018.

[7] S. Medya, A. Silva, A. Singh, P. Basu, and A. Swami, *Group centrality maximization via network design*, in *SIAM International Conference on Data Mining (SDM)*, 2018.

[8] S. Medya, T. Ma, , A. Silva, and A. Singh, *A game theoretic approach for core resilience*, in *Under Submission*, 2019.

[9] S. Medya, A. Silva, and A. Singh, *Influence minimization under budget and matroid constraints*, in *Working Paper*, 2019.

[10] X.-H. Dang, A. Silva, A. Singh, A. Swami, and P. Basu, *Outlier detection from network data with subnetwork interpretation*, in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 847–852, IEEE, 2016.

[11] J. J. Brown and P. H. Reingen, *Social ties and word-of-mouth referral behavior.*, *Journal of Consumer research* (1987).

[12] M. Newman, *Networks: An Introduction*. Oxford, 2010.

[13] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, *The emerging field of signal processing on graphs*, *IEEE Signal Processing Magazine* (2013).

[14] M. Naor, *Succinct representation of general unlabeled graphs*, *Discrete Applied Mathematics* (1990).

[15] Y. Tian, R. A. Hankins, and J. M. Patel, *Efficient aggregation for graph summarization*, in *SIGMOD*, 2008.

[16] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, *Journal of the American Statistical Association* (1963).

[17] S. Asmussen and P. Glynn, *Stochastic Simulation: Algorithms and Analysis*. Springer, 2007.

[18] D. Watts and S. Strogatz, *Collective dynamics of 'small-world' networks*, *Nature* (1998).

[19] F. Moser, R. Colak, A. Rafiey, and M. Ester, *Mining cohesive patterns from graphs with feature vectors.*, in *SDM*, 2009.

[20] H. Kwak, C. Lee, H. Park, and S. Moon, *What is twitter, a social network or a news media?*, in *WWW*, 2010.

[21] J. Yang and J. Leskovec, *Patterns of temporal variation in online media*, in *WSDM*, 2011.

[22] A. Go, R. Bhayani, and L. Huang, *Twitter sentiment classification using distant supervision*, *Project Report* (2009).

[23] A.-L. Barabási and R. Albert, *Emergence of scaling in random networks*, *Science* (1999).

[24] Z. Karni and C. Gotsman, *Spectral compression of mesh geometry*, in *SIGGRAPH*, 2000.

[25] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani, *The elements of statistical learning*. Springer, 2009.

[26] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim, *Approximate query processing using wavelets*, *The VLDB Journal* (2001).

[27] M. Gavish, B. Nadler, and R. R. Coifman, *Multiscale wavelets on trees, graphs and high dimensional data*, in *ICML*, 2010.

[28] Y. Choi, *Fast algorithm for optimal compression of graphs.*, in *ANALCO*, 2010.

[29] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu, *Graph olap: Towards online analytical processing on graphs*, in *ICDM*, 2008.

[30] A. Sandryhaila and J. Moura, *Discrete signal processing on graphs*, in *ICASSP*, 2013.

[31] S. Mallat, *A wavelet tour of signal processing*. Academic press, 1999.

[32] J. Bourgain, *On lipschitz embedding of finite metric spaces in hilbert space*, *Israel Journal of Mathematics* (1985).

[33] D. K. Hammond, P. Vandergheynst, and R. Gribonval, *Wavelets on graphs via spectral graph theory*, *Applied and Computational Harmonic Analysis* (2011).

[34] M. Crovella and E. Kolaczyk, *Graph wavelets for spatial traffic analysis.*, in *INFOCOM*, 2003.

[35] M. Maggioni, J. BremerJr, R. Coifman, and A. Szlam, *Biorthogonal diffusion wavelets for multiscale representations on manifolds and graphs*, in *SPIE*, 2005.

[36] J. Iverson, C. Kamath, and G. Karypis, *Fast and effective lossy compression algorithms for scientific datasets*, in *Euro-Par*, 2012.

[37] Y. Zhou, H. Cheng, and J. X. Yu, *Graph clustering based on structural/attribute similarities*, in *VLDB*, 2009.

[38] T. Srisooksai, K. Keamarungsi, P. Lamsrichan, and K. Araki, *Practical data compression in wireless sensor networks: A survey*, *Journal of Network and Computer Applications* (2012).

[39] A. Sandryhaila and J. Moura, *Big data analysis with signal processing on graphs*, *IEEE Signal Processing Magazine* **31** (2014) 80–90.

[40] D. Mohan, M. T. Asif, N. Mitrovic, J. Dauwels, and P. Jaillet, *Wavelets on graphs with application to transportation networks*, in *ITSC*, 2014.

[41] A. Gadde, A. Anis, and A. Ortega, *Active semi-supervised learning using sampling theory for graph signals*, in *SIGKDD*, 2014.

[42] R. Coifman and M. Maggioni, *Diffusion wavelets*, *Applied and Computational Harmonic Analysis* **21** (2006) 53–94.

[43] S. Fortunato, *Community detection in graphs*, *Physics Reports* **486** (2010) 75–174.

[44] T. Bui, S. Chaudhuri, F. Leighton, and M. Sipser, *Graph bisection algorithms with good average case behavior*, *Combinatorica* **7** (1987) 171–191.

[45] E. Dahlhaus, D. Johnson, C. Papadimitriou, P. Seymour, and M. Yannakakis, *The complexity of multiway cuts*, in *STOC*, 1992.

[46] L. Hagen and A. B. Kahng, *New spectral methods for ratio cut partitioning and clustering*, *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems* **11** (1992), no. 9 1074–1085.

[47] J. Shi and J. Malik, *Normalized cuts and image segmentation*, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22** (2000) 888–905.

[48] L. Trevisan, *Max cut and the smallest eigenvalue*, *SIAM Journal on Computing* **41** (2012) 1769–1786.

[49] A. Smola and R. Kondor, *Kernels and regularization on graphs*, in *Learning theory and kernel machines*, vol. 2777, pp. 144–158. 2003.

[50] J. Lafferty and G. Lebanon, *Diffusion kernels on statistical manifolds*, *JMLR* **6** (2005) 129–163.

[51] N. Leonardi and D. Van De Ville, *Tight wavelet frames on multislice graphs*, *IEEE Transactions on Signal Processing* **61** (2013) 3357–3367.

[52] J. Edmonds and R. M. Karp, *Theoretical improvements in algorithmic efficiency for network flow problems*, *Journal of the ACM* **19** (1972) 248–264.

[53] M. R. Garey and D. S. Johnson, *Computers and intractability*. WH Freeman, 2002.

[54] I. Tošić and P. Frossard, *Dictionary learning*, *IEEE Signal Processing Magazine* **28** (2011) 27–38.

[55] F. R. Chung, *Spectral graph theory*. American Mathematical Society, 1997.

[56] J. Cheeger, *A lower bound for the smallest eigenvalue of the laplacian*, *Problems in analysis* **625** (1970) 195–199.

[57] A. Silva, X.-H. Dang, P. Basu, A. Singh, and A. Swami, "Graph wavelets via sparse cuts." http://arxiv.org/abs/1602.03320, 2016.

[58] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer, 2001.

[59] J. Kiefer, *Sequential minimax search for a maximum*, in *Proceedings of the AMS*, 1953.

[60] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012.

[61] M. Mongiovi, P. Bogdanov, and A. Singh, *Mining evolving network processes*, in *ICDM*, 2013.

[62] L. A. Adamic and N. Glance, *The political blogosphere and the 2004 us election: divided they blog*, in *Workshop on Link Discovery*, 2005.

[63] Y. Hu, *Efficient, high-quality force-directed graph drawing*, *Mathematica* **10** (2005) 37–71.

[64] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, *Group formation in large social networks: membership, growth, and evolution*, in *KDD*, (New York, NY, USA), pp. 44–54, ACM, 2006.

[65] R. Kumar, J. Novak, P. Raghavan, and A. Tomkins, *On the bursty evolution of blogspace*, *World Wide Web* **8** (2005), no. 2 159–178.

[66] O. Michail, *An introduction to temporal graphs: An algorithmic perspective*, *Internet Mathematics* **12** (2016), no. 4 239–280.

[67] T. Erlebach, M. Hoffmann, and F. Kammer, *On temporal graph exploration*, in *ICALP*, (Berlin, Heidelberg), pp. 444–455, Springer, 2015.

[68] T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms*, in *FOCS*, (White Plains, NY), pp. 422–431, IEEE, 1988.

[69] A. N. Langville and C. D. Meyer, *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press, Princeton, New Jersey, 2011.

[70] D. A. Spielman and S.-H. Teng, *Spectral sparsification of graphs*, *SIAM Journal on Computing* **40** (2011), no. 4 981–1025.

[71] J. Stehlé, N. Voirin, A. Barrat, C. Cattuto, L. Isella, J.-F. Pinton, M. Quaggiotto, W. Van den Broeck, C. Régis, B. Lina, *et. al.*, *High-resolution measurements of face-to-face contact patterns in a primary school*, *PloS one* **6** (2011), no. 8 e23176.

[72] S. Arora, S. Rao, and U. Vazirani, *Expander flows, geometric embeddings and graph partitioning*, *Journal of the ACM* **56** (2009), no. 2 5.

[73] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, *Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters*, *Internet Mathematics* **6** (2009), no. 1 29–123.

[74] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng, *Evolutionary spectral clustering by incorporating temporal smoothness*, in *KDD*, (New York, NY, USA), pp. 153–162, ACM, 2007.

[75] Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng, *Facetnet: A framework for analyzing communities and their evolutions in dynamic networks*, in *WWW*, (New York, NY, USA), pp. 685–694, ACM, 2008.

[76] V. Kawadia and S. Sreenivasan, *Sequential detection of temporal communities by estrangement confinement*, *Scientific reports* **2** (2012) 794–794.

[77] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, *Incremental clustering and dynamic information retrieval*, *SIAM Journal on Computing* **33** (2004), no. 6 1417–1440.

[78] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang, *Incremental spectral clustering by efficiently updating the eigen-system*, *Pattern Recognition* **43** (2010), no. 1 113–127.

[79] S. Bickel and T. Scheffer, *Multi-view clustering.*, in *ICDM*, (Washington, DC, USA), pp. 19–26, IEEE, 2004.

[80] C. Xu, D. Tao, and C. Xu, "A survey on multi-view learning." https://arxiv.org/pdf/1304.5634.pdf, 2013.

[81] Y. Li, J. Han, and J. Yang, *Clustering moving objects*, in *KDD*, (New York, NY, USA), pp. 617–622, ACM, 2004.

[82] J. Rosswog and K. Ghose, *Detecting and tracking spatio-temporal clusters with adaptive history filtering*, in *ICDMW*, (Washington, DC, USA), pp. 448–457, IEEE, 2008.

[83] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter, *Multilayer networks*, *Journal of complex networks* **2** (2014), no. 3 203–271.

[84] P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, and J.-P. Onnela, *Community structure in time-dependent, multiscale, and multiplex networks*, *Science* **328** (2010), no. 5980 876–878.

[85] D. Taylor, S. A. Myers, A. Clauset, M. A. Porter, and P. J. Mucha, *Eigenvector-based centrality measures for temporal networks*, *Multiscale Modeling & Simulation* **15** (2017), no. 1 537–574.

[86] A. Sole-Ribalta, M. De Domenico, N. E. Kouvaris, A. Diaz-Guilera, S. Gomez, and A. Arenas, *Spectral properties of the laplacian of multiplex networks*, *Physical Review E* **88** (2013), no. 3 032807.

[87] S. Gomez, A. Diaz-Guilera, J. Gomez-Gardenes, C. J. Perez-Vicente, Y. Moreno, and A. Arenas, *Diffusion dynamics on multiplex networks*, *Physical review letters* **110** (2013), no. 2 028701.

[88] J. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, *Numerische Mathematik* **36** (1980), no. 2 177–195.

[89] W. N. Gansterer, R. C. Ward, R. P. Muller, and W. A. Goddard III, *Computing approximate eigenpairs of symmetric block tridiagonal matrices*, *SIAM Journal on Scientific Computing* **25** (2003), no. 1 65–85.

[90] C. J. Hillar and L.-H. Lim, *Most tensor problems are np-hard*, *Journal of the ACM (JACM)* **60** (2013), no. 6 45.

[91] L. Trevisan, "Is cheeger-type approximation possible for nonuniform sparsest cut?." https://arxiv.org/pdf/1303.2730.pdf, 2013.

[92] M. Cucuringu, I. Koutis, S. Chawla, G. Miller, and R. Peng, *Simple and scalable constrained clustering: a generalized spectral method*, in *AISTATS*, (Cadiz, Spain), pp. 445–454, PMLR, 2016.

[93] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, *Geometric deep learning: going beyond euclidean data*, *IEEE Signal Processing Magazine* **34** (2017), no. 4 18–42.

[94] M. Bazzi, M. A. Porter, S. Williams, M. McDonald, D. J. Fenn, and S. D. Howison, *Community detection in temporal multilayer networks, with an application to correlation networks*, *Multiscale Modeling & Simulation* **14** (2016), no. 1 1–41.

[95] D. Figueiredo, P. Nain, B. Ribeiro, E. de Souza e Silva, and D. Towsley, *Characterizing continuous time random walks on time varying graphs*, in *SIGMETRICS*, (New York, NY, USA), pp. 307–318, ACM, 2012.

[96] K. Lang, *Fixing two weaknesses of the spectral method*, in *NIPS*, (Cambridge, MA, USA), pp. 715–722, MIT Press, 2006.

[97] D. A. Spielman and S.-H. Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in *STOC*, (New York, NY, USA), pp. 81–90, ACM, 2004.

[98] I. Koutis, G. L. Miller, and R. Peng, *A nearly-m log n time solver for sdd linear systems*, in *FOCS*, (Washington, DC, USA), pp. 590–598, IEEE, 2011.

[99] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge University Press, New York, NY, USA, 1990.

[100] W. Anderson Jr and T. Morley, *Eigenvalues of the laplacian of a graph*, *Linear and multilinear algebra* **18** (1985), no. 2 141–145.

[101] A. Silva, A. Singh, and A. Swami, "Spectral algorithms for temporal graph cuts." http://arxiv.org/abs/1702.04746.pdf, 2017.

[102] A. Louis, P. Raghavendra, P. Tetali, and S. Vempala, *Many sparse cuts via higher eigenvalues*, in *STOC*, (New York, NY, USA), pp. 1131–1140, ACM, 2012.

[103] J. R. Lee, S. O. Gharan, and L. Trevisan, *Multiway spectral partitioning and higher-order cheeger inequalities*, *Journal of the ACM* **61** (2014), no. 6 1–30.

[104] I. Koutis, G. Miller, and R. Peng, "A generalized cheeger inequality." https://arxiv.org/abs/1412.6075, 2014.

[105] J. Kuczyński and H. Woźniakowski, *Estimating the largest eigenvalue by the power and lanczos algorithms with a random start*, *SIAM journal on matrix analysis and applications* **13** (1992), no. 4 1094–1122.

[106] L. R. Rabiner, *A tutorial on hidden markov models and selected applications in speech recognition*, *Proceedings of the IEEE* **77** (1989), no. 2 257–286.

[107] Z. Ghahramani and M. I. Jordan, *Factorial hidden markov models*, in *NIPS*, 1996.

[108] S. Fine, Y. Singer, and N. Tishby, *The hierarchical hidden markov model: Analysis and applications*, *Machine learning* **32** (1998), no. 1 41–62.

[109] N. Landwehr, *Modeling interleaved hidden processes*, in *ICML*, 2008.

[110] F. Wu, P. Anchuri, and Z. Li, *Structural event detection from log messages*, in *SIGKDD*, 2017.

[111] J.-G. Lou, Q. Fu, S. Yang, J. Li, and B. Wu, *Mining program workflow from interleaved traces*, in *SIGKDD*, 2010.

[112] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, *Detecting large-scale system problems by mining console logs*, in *SOSP*, 2009.

[113] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, *Performance debugging for distributed systems of black boxes*, in *SOSP*, 2003.

[114] E. Kim, S. Helal, and D. Cook, *Human activity recognition and pattern discovery*, *IEEE Pervasive Computing* **9** (2010), no. 1.

[115] C. B. Burge and S. Karlin, *Finding the genes in genomic dna*, *Current opinion in structural biology* **8** (1998), no. 3 346–354.

[116] A. Minot and Y. M. Lu, *Separation of interleaved markov chains*, in *ACSSC*, 2014.

[117] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, *Variational inference: A review for statisticians*, *Journal of the American Statistical Association* **112** (2017), no. 518 859–877.

[118] T. P. Minka, *Expectation propagation for approximate bayesian inference*, in *UAI*, 2001.

[119] W. K. Hastings, *Monte carlo sampling methods using markov chains and their applications*, *Biometrika* **57** (1970), no. 1 97–109.

[120] S. A. Terwijn, *On the learnability of hidden markov models*, in *International Colloquium on Grammatical Inference*, 2002.

[121] T. Petrie, *Probabilistic functions of finite state markov chains*, *The Annals of Mathematical Statistics* (1969) 97–115.

[122] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang, *Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs*, in *ASPLOS*, 2016.

[123] L. K. Saul and M. I. Jordan, *Mixed memory markov models: Decomposing complex stochastic processes as mixtures of simpler ones*, *Machine learning* **37** (1999), no. 1 75–87.

[124] R. M. Altman, *Mixed hidden markov models: an extension of the hidden markov model to the longitudinal data setting*, *Journal of the American Statistical Association* **102** (2007), no. 477 201–210.

[125] G. Seroussi, W. Szpankowski, and M. J. Weinberger, *Deinterleaving finite memory processes via penalized maximum likelihood*, *IEEE Transactions on Information Theory* **58** (2012), no. 12 7094–7109.

[126] D. Gillblad, R. Steinert, and D. R. Ferreira, *Estimating the parameters of randomly interleaved markov models*, in *ICDMW*, 2009.

[127] T. Batu, S. Guha, and S. Kannan, *Inferring mixtures of markov chains*, in *COLT*, 2004.

[128] C. P. Robert, *Monte carlo methods*. Wiley Online Library, 2004.

[129] K. P. Murphy, Y. Weiss, and M. I. Jordan, *Loopy belief propagation for approximate inference: An empirical study*, in *UAI*, 1999.

[130] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, *An introduction to variational methods for graphical models*, *Machine learning* **37** (1999), no. 2 183–233.

[131] D. J. Spiegelhalter, A. P. Dawid, S. L. Lauritzen, and R. G. Cowell, *Bayesian analysis in expert systems*, *Statistical science* (1993) 219–247.

[132] B.-H. Juang and L. R. Rabiner, *A probabilistic distance measure for hidden markov models*, *AT&T technical journal* **64** (1985), no. 2 391–408.

[133] D. Hsu, S. M. Kakade, and T. Zhang, *A spectral algorithm for learning hidden markov models*, *Journal of Computer and System Sciences* **78** (2012), no. 5 1460–1480.

[134] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky, *Tensor decompositions for learning latent variable models*, *The Journal of Machine Learning Research* **15** (2014), no. 1 2773–2832.

[135] A. Gupta and J. Könemann, *Approximation algorithms for network design: A survey*, *Surveys in Operations Research and Management Science* (2011) 3–20.

[136] S. Krumke, M. Marathe, H. Noltemeier, R. Ravi, and S. Ravi, *Approximation algorithms for certain network improvement problems*, *Journal of Combinatorial Optimization* **2** (1998) 257–288.

[137] Y. Lin and K. Mouratidis, *Best upgrade plans for single and multiple source-destination pairs*, *GeoInformatica* **19** (2015), no. 2 365–404.

[138] B. Dilkina, K. J. Lai, and C. P. Gomes, *Upgrading shortest paths in networks*, in *CPAIOR*, pp. 76–91, 2011.

[139] A. Meyerson and B. Tagiku, *Minimizing average shortest path distances via shortcut edge addition*, in *APPROX-RANDOM*, pp. 272–285, 2009.

[140] E. D. Demaine and M. Zadimoghaddam, *Minimizing the diameter of a network using shortcut edges*, *Lecture Notes in Computer Science* (2010) 420–431.

[141] E. B. Khalil, B. Dilkina, and L. Song, *Scalable diffusion-aware optimization of network topology*, in *SIGKDD international conference on Knowledge discovery and data mining*, pp. 1226–1235, ACM, 2014.

[142] H. Tong, B. A. Prakash, T. Eliassi-Rad, M. Faloutsos, and C. Faloutsos, *Gelling, and melting, large graphs by edge manipulation*, in *CIKM*, pp. 245–254, 2012.

[143] Y. Yoshida, *Almost linear-time algorithms for adaptive betweenness centrality using hypergraph sketches*, in *KDD*, pp. 1416–1425, 2014.

[144] A. Mahmoody, E. Charalampos, and E. Upfal, *Scalable betweenness centrality maximization via sampling*, in *KDD*, pp. 1765–1773, 2016.

[145] S. B. Seidman, *Network structure and minimum degree*, *Social networks* **5** (1983), no. 3 269–287.

[146] K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma, *Preventing unraveling in social networks: the anchored k-core problem*, *SIAM Journal on Discrete Mathematics* **29** (2015), no. 3 1452–1475.

[147] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse, *Identification of influential spreaders in complex networks*, *Nature physics* **6** (2010), no. 11 888.

[148] K. Shin, T. Eliassi-Rad, and C. Faloutsos, *Corescope: Graph mining using k-core analysis—patterns, anomalies and algorithms*, in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pp. 469–478, IEEE, 2016.

[149] C. Peng, T. G. Kolda, and A. Pinar, *Accelerating community detection by using k-core subgraphs*, *arXiv preprint arXiv:1403.2226* (2014).

[150] Z.-H. You, Y.-K. Lei, L. Zhu, J. Xia, and B. Wang, *Prediction of protein-protein interactions from amino acid sequences with ensemble extreme learning machines and principal component analysis*, .

[151] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani, *Large scale networks fingerprinting and visualization using the k-core decomposition*, in *Advances in neural information processing systems*, pp. 41–50, 2006.

[152] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir, *A model of internet topology using k-shell decomposition*, *Proceedings of the National Academy of Sciences* **104** (2007), no. 27 11150–11154.

[153] W. Zhu, C. Chen, X. Wang, and X. Lin, *K-core minimization: An edge manipulation approach*, in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 1667–1670, ACM, 2018.

[154] L. S. Shapley, *A value for n-person games*, *Contributions to the Theory of Games* **2** (1953), no. 28 307–317.

[155] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, *On the evolution of user interaction in facebook*, in *Proceedings of the 2nd ACM workshop on Online social networks*, pp. 37–42, ACM, 2009.

[156] S. Wuchty and E. Almaas, *Peeling the yeast protein network*, *Proteomics* **5** (2005), no. 2 444–449.

[157] M. Zitnik, R. Sosic, M. W. Feldman, and J. Leskovec, *Evolution of resilience in protein interactomes across the tree of life*, *bioRxiv* (2019).

[158] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, *Best algorithms for approximating the maximum of a submodular set function*, *Math. Oper. Res.* (1978) 177–188.

[159] C. Chekuri and A. Kumar, *Maximum coverage problem with group budget constraints and applications*, in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pp. 72–83. Springer, 2004.

[160] A. Goyal, F. Bonchi, and L. V. Lakshmanan, *Learning influence probabilities in social networks*, in *Proceedings of the third ACM international conference on Web search and data mining*, pp. 241–250, ACM, 2010.

[161] M. Kimura, K. Saito, and H. Motoda, *Minimizing the spread of contamination by blocking links in a network.*, in *AAAI*, 2008.

[162] L. Akoglu, H. Tong, and D. Koutra, *Graph based anomaly detection and description: a survey*, *DMKD* (2015).

[163] K. Henderson *et. al.*, *It's who you know: graph mining using recursive structural features*, in *KDD*, 2011.

[164] W. Eberle and L. B. Holder, *Discovering structural anomalies in graph-based data*, in *ICDM workshop*, 2007.

[165] P. Bogdanov *et. al.*, *Netspot: Spotting significant anomalous regions on dynamic networks*, in *SDM*, 2013.

[166] B. Micenková, R. T. Ng, X. H. Dang, and I. Assent, *Explaining outliers by subspace separability*, in *ICDM*, 2013.

[167] D. H. Ki *et. al.*, *Whole genome analysis for liver metastasis gene signatures in colorectal cancer*, *Int J Cancer* (2007).

[168] C. Hsieh *et. al.*, *A dual coordinate descent method for large-scale linear SVM*, in *ICML*, 2008.

[169] T. Hastie *et. al.*, *The Elements of Statistical Learning. Data Mining, Inference, and Prediction*. Springer, 2009.

[170] F. Keller, E. Müller, and K. Böhm, *Hics: High contrast subspaces for density-based outlier ranking*, in *ICDE*, 2012.

[171] H. Kriegel, M. Schubert, and A. Zimek, *Angle-based outlier detection in high-dimensional data*, in *SIGKDD*, 2008.

[172] A. Zimek *et. al.*, *A survey on unsupervised outlier detection in high-dimensional numerical data*, *Statistical Analysis and Data Mining* (2012).

[173] X. H. 0 *et. al.*, *Learning predictive substructures with regularization for network data*, in *ICDM*, 2015.

[174] R. Lambiotte, J.-C. Delvenne, and M. Barahona, *Random walks, markov processes and the multiscale modular organization of complex networks*, *IEEE Transactions on Network Science and Engineering* **1** (2014), no. 2 76–90.

[175] W. L. Hamilton, R. Ying, and J. Leskovec, *Representation learning on graphs: Methods and applications*, *arXiv preprint arXiv:1709.05584* (2017).

[176] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, *Convolutional networks on graphs for learning molecular fingerprints*, in *Advances in neural information processing systems*, pp. 2224–2232, 2015.

[177] Z. Li, D. Sun, R. Zhu, and Z. Lin, *Detecting event-related changes in organizational networks using optimized neural network models*, *PloS one* **12** (2017), no. 11 e0188733.

[178] C. C. Aggarwal and K. Subbian, *Event detection in social streams*, in *Proceedings of the 2012 SIAM international conference on data mining*, pp. 624–635, SIAM, 2012.

[179] B. Berendt and I. Subasic, *Measuring graph topology for interactive temporal event detection*, *Künstliche Intelligenz* **2** (2009), no. 2009 11–17.

[180] J. Rappaz, M. Catasta, R. West, and K. Aberer, *Latent structure in collaboration: the case of reddit r/place*, in *Twelfth International AAAI Conference on Web and Social Media*, 2018.

[181] P. F. Felzenszwalb and D. P. Huttenlocher, *Efficient graph-based image segmentation*, *International journal of computer vision* **59** (2004), no. 2 167–181.

[182] J. Long, E. Shelhamer, and T. Darrell, *Fully convolutional networks for semantic segmentation*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[183] J. Pearl, *Causality: models, reasoning and inference*, vol. 29. Springer, 2000.

[184] J. P. Hespanha, *Linear systems theory*. Princeton university press, 2018.

[185] F. Bullo, *Lectures on Network Systems*. CreateSpace, 2018.

[186] C. Dwork, *Differential privacy*, *Encyclopedia of Cryptography and Security* (2011) 338–340.

[187] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel, *Fairness through awareness*, in *Proceedings of the 3rd innovations in theoretical computer science conference*, pp. 214–226, ACM, 2012.

[188] N. Diakopoulos, *Accountability in algorithmic decision making*, *Communications of the ACM* **59** (2016), no. 2 56–62.