NONLINEAR DIFFUSIONS ON GRAPHS FOR CLUSTERING, SEMI-SUPERVISED LEARNING AND ANALYZING PREDICTIONS

by

Meng Liu

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Computer Science

West Lafayette, Indiana

December 2022

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. David F. Gleich, Chair

School of Computer Science

Dr. Petros Drineas

School of Computer Science

Dr. Ananth Y. Grama

School of Computer Science

Dr. Tamal K. Dey

School of Computer Science

Approved by:

Dr. Kihong Park

To my dad Changxin Liu and mom Yuanqiu Chen, and my beloved partner, Dr. Xu Zhang.

ACKNOWLEDGMENTS

It has been an honor to study for a Ph.D at Purdue University. And the journey of my Ph.D will never be successful without the help from many people.

First of all, I would like to express my deepest gratitude to my advisor, Dr. David F. Gleich, for all the inspirations of research ideas, patient guidance on designing experiments, writing papers and improving programming skills as well as continuous encouragement when I encounter academic difficulties. I would also like to thank my thesis committee members and other faculty, Dr. Petros Drineas, Dr. Ananth Y. Grama, Dr. Tamal K. Dey and Dr. Pan Li, for their valuable feedback to help improve my research. Thanks should also go to all my co-authors and collaborators for their knowledge, insights, guidance and hard working.

I am also thankful to all the former and present student colleagues in the research group, Nate Veldt, Nicole Eikmeier, Huda Nassar, Tao Wu, Yangyang Hou, Yanfei Ren, Charlie Colley, Rania Ibrahim, Omar Eldaghar, Yufan Huang and Disha Shur for their insightful discussions and unconditional help.

Last but not least, I would like to send my special thanks to my family. Especially I want to thank my parents, Changxin Liu, Yuanqiu Chen, for their endless love, encouragement and unconditional support to pursue my dream throughout my life. And I want to thank my wife Dr. Xu Zhang - no matter where I am and what decisions I make, you will always be there to support me. It is you that I share my happiness with when I accomplish something and cheer me up when I feel struggled and frustrated. Marrying you is the luckiest thing that has happened in my life!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Graph diffusion is the process of spreading information from one or few nodes to the rest of the graph through edges. The resulting distribution of the information often implies latent structure of the graph where nodes more densely connected can receive more signal. This makes graph diffusions a powerful tool for local clustering, which is the problem of finding a cluster or community of nodes around a given set of seeds. Most existing literatures on using graph diffusions for local graph clustering are linear diffusions as their dynamics can be fully interpreted through linear systems. They are also referred as eigenvector, spectral, or random walk based methods. While efficient, they often have difficulty capturing the correct boundary of a target label or target cluster. On the contrast, maxflow-mincut based methods that can be thought as 1-norm nonlinear variants of the linear diffusions seek to "improve" or "refine" a given cluster and can often capture the boundary correctly. However, there is a lack of literature to adopt them for problems such as community detection, local graph clustering, semi-supervised learning, etc. due to the complexity of their formulation. We addressed these issues by performing extensive numerical experiments to demonstrate the performance of flow-based methods in graphs from various sources. We also developed an efficient LocalGraphClustering Python Package that allows others to easily use these methods in their own problems. While studying these flow-based methods, we find that they cannot grow from small seed set. Although there are hybrid procedures that incorporate ideas from both linear diffusions and flow-based methods, they have many hard to set parameters. To tackle these issues, we propose a simple generalization of the objective function behind linear diffusion and flow-based methods which we call generalized local graph min-cut problem. We further show that by involving p-norm in this cut problem, we can develop a nonlinear diffusion procedure that can find local clusters from small seed set and capture the correct boundary simultaneously. Our method can be thought as a nonlinear generalization of the Anderson-Chung-Lang push procedure to approximate a personalized PageRank vector efficiently and is a strongly local algorithm-one whose runtime depends on the size of the output rather than the size of the graph. We also show that the p-norm cut functions improve on the standard Cheeger inequalities for linear diffusion methods. We further

extend our generalized local graph min-cut problem and the corresponding diffusion solver to hypergraph-based machine learning problems. Although many methods for local graph clustering exist, there are relatively few for localized clustering in hypergraphs. Moreover, those that exist often lack flexibility to model a general class of hypergraph cut functions or cannot scale to large problems. Our new hypergraph diffusion method on the other hand enables us to compute with a wide variety of cardinality-based hypergraph cut functions and still maintains the strongly local property. We also show that the clusters found by solving the new objective function satisfy a Cheeger-like quality guarantee.

Besides clustering, recent work on graph-based learning often focuses on node embeddings and graph neural networks. Although these GNN based methods can beat traditional ones especially when node attributes data is available, it is challenging to understand them because they are highly over-parameterized. To solve this issue, we propose a novel framework that combines topological data analysis and diffusion to transform the complex prediction space into human understandable pictures. The method can be applied to other datasets not in graph formats and scales up to large datasets across different domains and enable us to find many useful insights about the data and the model.

1. INTRODUCTION

Many datasets important to machine learning either start as a graph or have a simple translation into graph data. For instance, relational network data naturally starts as a graph. Arbitrary data vectors become graphs via nearest-neighbor constructions, among other choices. Graph structures naturally exist in many domains. To name a few, (1) in social graphs, nodes can be users and edges can represent friendship (2) in co-purchasing graphs, nodes can be products, edges can represent whether two products are commonly purchased together (3) in biology datasets, nodes can be disease or medicine and edges can be the relation that whether a medicine can cure a disease and many more. Consequently, understanding graph-based learning algorithms – those that learn from graphs – is a recurring problem.

Graph diffusion is the process of spreading the information from one or a few nodes to the rest of the graph via edges. By analyzing the distribution of where the information is information after many steps, insights can be obtained about the underlying structure of the graph. If the information is mostly concentrated nearby the starting nodes, it means there is a local cluster in the neighborhood where there are fewer edges leaving the cluster than edges within the cluster. If there is a homophily property for edges in the network, i.e. nodes connected by edges are more likely to be similar in terms of node attributes, the information spread is also important for semi-supervised learning. This is because nearby nodes with more information obtained imply better chances of sharing the same label as the starting nodes. Consequently, local clustering and semi-supervised learning are probably the two most common applications of graph diffusions (See Figure 1.1 as an example). Other useful applications of diffusions include searching and ranking over nodes [1], predicting missing links in Biological networks [2–4], generating low dimensional node embeddings for visualization [5, 6] and many more.

The existing literature on diffusions for local clustering are predominantly linear diffusions. They are called linear because the underlying dynamics is equivalent to solving a linear system defined over the adjacency matrix of the graph. They are often referred as eigenvector, spectral, random walk or PageRank based methods. Linear diffusions are pow-

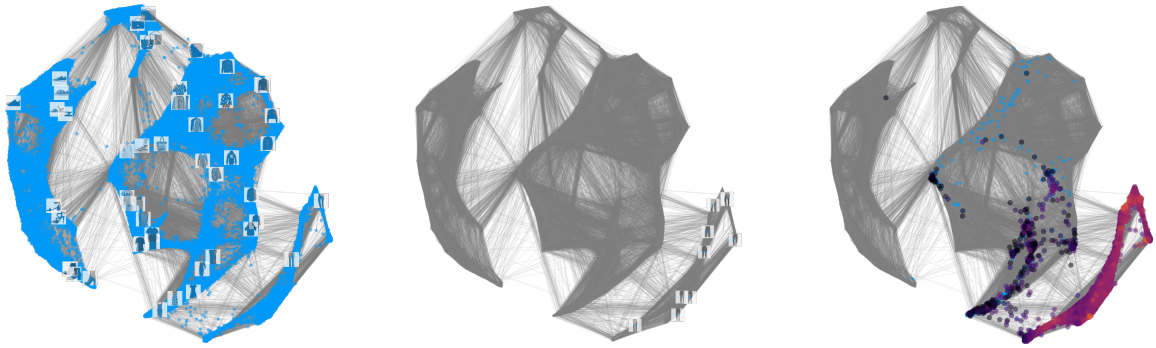
erful at finding local clusters as they can be efficiently computed over graphs in very large scale and they can do so by only visiting a small portion of the graph. Consequently, there exists *strongly local* algorithms for linear diffusions whose runtime only depends on the size of the target cluster instead of the size of the entire graph. More importantly, there is a theoretical guarantee over the quality of the local cluster, called the Cheeger inequality.

Comparing to linear diffusions, nonlinear diffusions are less widely used and studied. Some existing reseasrch simply replaces the linear equations by nonlinear functions without further understanding on which kind of optimization problems the new process is trying to solve [7, 8]. There is another line of research that uses maxflow-mincut computations to find local clusters by improving upon a given input cluster that presumes to have substantial overlap with the target cluster [9–11]. We categorize these flow-based methods as a special type of nonlinear diffusion because some existing work shows the optimizations problems they are trying to solve are a 1-norm variant of the optimization problems behind linear diffusions [12].

Although local clustering has been an active research topic for a long time, challenges remain with using these tools in diverse data. In our own study of the existing methods, we find they all have different drawbacks. Linear diffusions often have difficulty capturing the correct boundary of a target label or target cluster. In contrast, 1-norm or maxflow-mincut based methods capture the boundary, but cannot grow from small seed set. Some nonlinear diffusions that incorporate both have many hard to set parameters [8]. To address these issues, we propose a set of nonlinear diffusions that are based on generalized min-cut optimization problem. Especially, by involving p-norms, we can combine the advantages of linear diffusion and flow-based methods. More importantly, we can develop a strongly local nonlinear diffusion procedure to solve these problems which enable them to be used on large scale real world datasets in time comparable to linear diffusions.

Other than local clustering, recent work in graph-based learning has often focused on embeddings [13, 14] and graph neural networks [15–17]. These models are motivated by the huge success of using neural networks in tasks from other domains such as natural language processing and computer vision. Not surprisingly, they also perform well in graph-based learning problems and can easily beat traditional methods especially when there is a lot of

node features data as well besides the graph structure. However, it is very challenging to understand these algorithms because they are highly over-parameterized and behave largely like a black-box model. To tackle this issue, we propose a novel framework that combines topological data analysis and diffusion to analyze these complex predictions by transforming them into pictures representing a topological view. The result is a map of the predictions that enables inspection and can often provide human understandable insights over the model and the dataset. Other than graph-based learning, our method is general enough to be used for datasets in other formats like images or DNA sequences and can scale up easily.



(a) Fashion MNIST graph (b) Seed set (c) A local cluster

Figure 1.1. An example of local clustering on Fashion MNIST dataset. (a) The graph is constructed by connecting each image to its 5 nearest neighbors. (b) We select some "pants" nodes as seed nodes. (c) We run one of our local graph clustering algorithm [18] to find the other "pants" nodes. Blue nodes are the ground truth. Lighter nodes are more likely to be "pants" nodes.

In summary, this thesis explores new opportunities of using nonlinear diffusion on graphs for clustering, semi-supervised learning and analyzing predictions. The specific constructions are listed below:

- Chapter 3 introduces our generalized min-cut problem in formal as well as other related work in nonlinear diffusions [18].
- Chapter 4 will provide a comprehensive study on maxflow-mincut based local graph clustering algorithms which corresponds to a 1-norm variant of the linear diffusion [19].

- Chapter 5 will provide a new strongly local nonlinear diffusion that can solve a set of our generalized local min-cut problem as long as the cut functions satisfy a few criteria, such as p-norms or p-norm variants of Berhu or Huber functions [18].
- Chapter 6 will further extend our generalized local min-cut problem to hypergraphs that can model local hypergraph clustering problems associated with a broad family of hyperedge cut functions. It will also provide another strongly local diffusion algorithm to solve the generalized equation [20].
- Chapter 7 will introduce our novel framework that combines topological data analysis and diffusion to inspect the prediction space of complex models [21].
- Chapter 8 summarizes the results in this thesis and proposes a few directions that can further improve our nonlinear diffusion framework for clustering and the topology based method for analyzing predictions.

The research in this thesis has been joint work with my adviser David F. Gleich. The results in Chapter 4 are joint with Kimon Fountoulakis and Michael W. Mahoney. The ideas in Chapter 6 are also joint with Nate Veldt, Haoyu Song and Pan Li. The material in Chapter 7 is joint with Tamal K. Dey.

2. PRELIMINARY

This chapter will introduce some definitions that will be repeatedly used in the following chapters.

Graphs:

We consider graphs that are undirected, connected, and weighted with positive edge weights lower-bounded by 1. Let $G = (V, E, w)$ be such a graph, where $n = |V|$ and $m = |E|$. The adjacency matrix $\boldsymbol{A}$ has non-zero entries $w(i, j)$ for each edge (i, j), and all other entries are zero. This is symmetric because the graph is undirected. The degree vector $\mathbf{d}$ is defined as the row sum of $\boldsymbol{A}$ and $\boldsymbol{D}$ is a diagonal matrix defined as diag($\mathbf{d}$). The incidence matrix $\boldsymbol{B} \in \{0, -1, 1\}^{m \times n}$ measures the differences of adjacent nodes. The kth row of $\boldsymbol{B}$ represents the kth edge and each row has exactly two nonzero elements, i.e. 1 for start node of kth edge and -1 for end node of kth edge. For undirected graphs, either node can be the start node or end node and the order does not matter. We use $i \sim j$ to represent that node i and node j are adjacent.

Hypergraphs:

Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph where each hyperedge e $\in \mathcal{E}$ is a subset of V. Let $\zeta = \max_{\mathrm{e} \in \mathcal{E}} |\mathrm{e}|$ be the maximum hyperedge size. With each hyperedge, we associate a *splitting function* f_{e} that we use to assess an appropriate penalty for splitting the hyperedge among two labels or splitting the hyperedge between two clusters. Formally, let S be a cluster and let $A = \mathrm{e} \cap S$ be the hyperedge's nodes inside S, then $f_{\mathrm{e}}(A)$ penalizes splitting e.

Graph conductance:

Graph conductance is one of the common metrics that has been used to measure the quality of a cluster. Given a partition $(S, \bar{S})$, the *cut of the partition* is the sum of weights of edges between S and $\bar{S}$, which can be denoted by either

$$\mathbf{cut}(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}} A_{ij}, \quad \text{or} \quad \mathbf{cut}(S) = \sum_{i \in S, j \in \bar{S}} A_{ij}. \tag{2.1}$$

The graph conductance is defined as

$$\phi(S) = \frac{\text{cut}(S)}{\min(\text{vol}(S), \text{vol}(\bar{S}))} \tag{2.2}$$

Hypergraph conductance:

With a splitting function identified, the cut value of any given set S can be written as $\text{cut}_{\mathcal{H}}(S) = \sum_{\text{e} \in E} f_{\text{e}}(\text{e} \cap S)$. The node degree in this case can be defined as $d_i = \sum_{\text{e}: i \in \text{e}} f_{\text{e}}(\{i\})$ [22, 23], though other types of degree vectors can also be used in both the graph and hypergraph case. This gives rise to a definition of conductance on a hypergraph

$$\phi_{\mathcal{H}}(S) = \frac{\mathbf{cut}_{\mathcal{H}}(S)}{\min(\mathbf{vol}(S), \mathbf{vol}(\bar{S}))} \tag{2.3}$$

where $\mathbf{vol}(S) = \sum_{i \in S} d_i$. This reduces to the standard definition of graph conductance when each edge has only two nodes ($\zeta = 2$) and we use the *all-or-nothing penalty*.

Relative volume:

The *relative volume* of S with respect to R and κ is

$$\mathbf{rvol}(S; R, \kappa) = \mathbf{vol}(S \cap R) - \kappa \mathbf{vol}(S \cap \bar{R}). \tag{2.4}$$

Sweep cut:

The results of graph diffusion is usually a real-valued vector that induces a cluster. Given such a vector $\mathbf{x}$, a sweep cut process can convert $\mathbf{x}$ to a cluster by sorting $\mathbf{x}$ in decreasing order and evaluating the conductance of each prefix set $S_j = \{[1], [2], ..., [j]\}$ for each $j \in [n]$, where $[j]$ is the index of the j-th largest element. The set with the smallest conductance will be returned.

3. BACKGROUND

Diffusions are an intuitive procedure that have been widely used on various graph-based learning problems. These applications include: (1) detecting communities in graphs [24–27], (2) semi-supervised learning [28], (3) searching and ranking [1], (4) predicting protein functions in biology networks [2–4], (5) node embeddings for visualization [5, 6], and many more. However, most theoretical results about graph diffusions focus on finding clusters or communities in a graph with graph conductance as small as possible.

3.1 Linear graph diffusions

Most existing algorithms for graph diffusions are linear diffusions and they are closely related to the personalized PageRank problem as defined by the following equation

$$(\boldsymbol{I} - \alpha \boldsymbol{A}\boldsymbol{D}^{-1})\mathbf{x} = (1-\alpha)\mathbf{s}. \tag{3.1}$$

In this equation, $\boldsymbol{A}$ is the adjacency matrix of the graph, $\boldsymbol{D}$ is the diagonal degree matrix, $\boldsymbol{I}$ is the identity matrix, $\mathbf{s}$ is the initial distribution vector and $\mathbf{x}$ is the resulting diffusion vector. Also, $0 < \alpha < 1$ is a user defined teleportation parameter. The solution of this formulation is:

$$\mathbf{x} = (1-\alpha)\sum_{k=0}^{\infty} \alpha^k (\boldsymbol{A}\boldsymbol{D}^{-1})^k \mathbf{s} \tag{3.2}$$

It is also the same as the stationary distribution of a random walk with restart or lazy random walk [29].

This formulation can also be thought as a locally biased version of the global spectral optimization programs [30, 31].

Perhaps the most well known algorithm for approximating the results of equation 3.1 is Andersen-Chung-Lang *push* procedure [24], which is motivated by a local spectral partitioning algorithm called *Nibble* [32]. Given an initial distribution $\mathbf{s}$, the ACL algorithm maintains a residual vector for all nodes. Then at each iteration, it finds a node whose residual is larger than a given threshold and it pushes value into that node so that its residual

will be smaller than the threshold. This process continues until no residual is larger than the given threshold. This method is closely related to a coordinate descent method except that only a partial step is taken at each iteration [12]. A better formulation of this residual vector can be found in the next section. The most appealing part of ACL is speed, it can be proven that at each step, the sum of residual will at least decrease by a constant value, which then upper bounds the total number of iterations needed. Consequently, ACL is a strongly local method, one whose running time only depends on the size of the target cluster instead of the entire graph. Once the vector $\mathbf{x}$ has been computed or approximated, a sweepcut procedure can convert this vector into a cluster of the graph.

The global spectral method of partitioning a graph into two clusters comes with a theoretical guarantee called Cheeger inequality [33], which states that the global cluster obtained from spectral partitioning has a conductance not larger than a quadratic factor of the optimal value. Such theoretical guarantee also exists for local clusters found by ACL. Some other work [34] further improves such guarantee by connecting the conductance of the resulting cluster to the relative connectedness of the target cluster with respect to all the other clusters in this graph.

3.2 Nonlinear graph diffusions

Although most existing diffusions are linear, a recent work introduces nonlinearity in diffusions [7] and shows improvement comparing to linear diffusion in semi-supervised learning. The node feature updating rule of the increasingly popular graph neural network models can also be thought as nonlinear diffusions on graphs [14, 35]. Some other work in GNN have incorporated diffusion equations more directly and achieved good performance in graph based learning problems [36–38]. Other than diffusions on graphs, nonlinear diffusions have arisen in other domains like physics or ecology [39–42].

Another set of algorithms aim to improve an existing local cluster by solving a series of mincut-maxflow optimization problems [9–11]. These methods can find improved clusters with conductance within constant factor of the optimum assuming the input cluster has substantial overlap with the target cluster. Although their approach looks quite different

to the linear diffusions, some work shows that these maxflow-mincut based methods can actually be unified with linear diffusions involving 1-norm or 2-norm [12]. More details on this part can be found in the next section. Some other work [8] tries to combine linear diffusion and flow based cluster improving method into a single algorithm, which can be thought as nonlinear diffusion as well and can improve Cheeger inequality.

The ongoing p-Laplacian research [43–47] tries to generalize the global spectral partitioning into p-norms and doing so can improve the standard Cheeger inequality to the corresponding p-norm variant. A more recent work [48] also introduces p-norm into the mincut-maxflow based algorithms and shows that the resulting local clusters can improve the Cheeger inequality from linear diffusion approach as well. Using a simple nonlinearity on a Laplacian pseudoinverse is also competitive with complex embedding procedures [49].

3.3 Nonlinear diffusions as generalized min-cut optimization

This section will present a key result of this thesis, which is to formulate a set of nonlinear diffusion as a generalized min-cut optimization. Solving this optimization with different cut functions can lead to new strongly local solvers that find better local clusters both empirically and theoretically. This optimization problem originates from an important link between between linear diffusion methods and the mincut-maxflow computations, which is that they correspond to 1-norm and 2-norm variations on a general objective function (see [12]) that is defined on a localized cut graph. The localized cut graph is constructed by adding an extra source node s and an extra sink node t, and edges from s to seed nodes and from non-seed nodes to t. Formally, given a graph $G = (V, E)$ with adjacency matrix A, a seed set $R \subset V$ and two non-negative constants γ_s and γ_t, the adjacency matrix of the localized cut graph is:

$$\boldsymbol{A}_R = \begin{bmatrix} 0 & \gamma_s \mathbf{d}_R^T & 0 \\ \gamma_s \mathbf{d}_R & \boldsymbol{A} & \gamma_t \mathbf{d}_{\bar{R}} \\ 0 & \gamma_t \mathbf{d}_{\bar{R}}^T & 0 \end{bmatrix} \tag{3.3}$$

and a small illustration is:

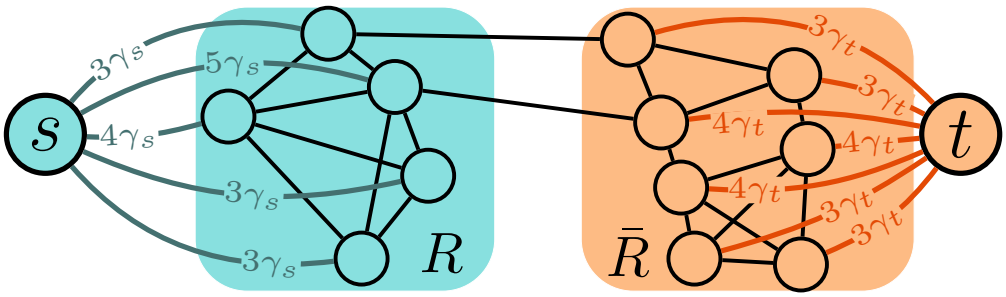


Figure 3.1. An example of localized cut graph.

Here $\mathbf{d}_R = \boldsymbol{D}\mathbf{e}_R$, $\mathbf{d}_{\bar{R}} = \boldsymbol{D}\mathbf{e}_{\bar{R}}$, and $\mathbf{e}_R$ is an indicator vector for R. Formally, the generalized graph cut problem is defined as:

Definition 3.3.1 (Generalized local graph cut)**.** Fix a set S of seeds and a value of γ. Let $\boldsymbol{B}$, $\mathbf{w}$ be the incidence matrix and weight vector of the localized cut graph. Then the generalized local graph cut problem is:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{w}^T \ell(\boldsymbol{B}\mathbf{x}) + \kappa\gamma \mathbf{d}^T \mathbf{x} = \textstyle\sum_{ij} w_{i,j} \ell(x_i - x_j) + \kappa\gamma \sum_i x_i d_i \\ \text{subject to} \quad & x_s = 1, x_t = 0, \mathbf{x} \geq 0. \end{aligned} \tag{3.4}$$

Here $\ell(\mathbf{x})$ is an element-wise cut function and $\kappa \geq 0$ is a sparsity-promoting term.

Linear diffusion or PageRank-based algorithm is equivalent to solve equation 3.4 with $\ell(x) = \frac{1}{2}x^2$ and flow-based algorithm will solve a series of equation 3.4 with $\ell(x) = |x|$. Moreover, suppose the objective function becomes $L(\mathbf{x})$ when $\ell(x) = \frac{1}{2}x^2$, then the residual vector of ACL algorithm is nothing but $-\frac{1}{\gamma} \cdot dL/d\mathbf{x}$. At the optimum, the KKT conditions require that $\mathbf{r} = 0$, however, the result of ACL only approximately satisfies this condition by requiring $0 < \mathbf{r} < \kappa\mathbf{d}$ along with some other approximations.

Most importantly, in chapter 5, we will show that we can design strongly local solver to approximately solve equation 3.4 as long as the cut functions satisfy a few properties (See Section 5.2 for more details). Our algorithm is closely related to ACL. We also show that for cut functions that are p-norms, i.e. $\ell(x) = \frac{1}{p}|x|^p$, where $1 < p < 2$. Solving equation 3.4 can produce better local clusters both empirically and theoretically. Our method is different from

the p-norm flow approach [48] as we directly solve the dual cut optimization problem. We also include the localizing set S in our nonlinear penalty. Also, our solver uses the cut values instead of the flow dual on the edges and our formulation can be easily adapted to solve other nonlinear cut functions beyond p-norm such as p-norm variants of Huber or Berhu functions. Moreover, a further generalized version of equation 3.4 can be used to solve local hypergraph clustering problems which will be discussed in detail in Chapter 6.

4. EMPIRICAL STUDY OF FLOW-BASED ALGORITHMS FOR IMPROVING LOCAL CLUSTERS

4.1 Chapter Overview and Motivation

In this chapter, we discuss empirical studies on maxflow-mincut based local graph clustering algorithms that can improve existing clusters. We see these algorithms as a special type of nonlinear "diffusion" on graphs as they are solving a series of equation 3.4 with $\ell(x) = |x|$ and other minor modifications. However, these algorithms indeed have some key difference comparing to other diffusion methods like PageRank as the results are no longer a real valued vector, but a binary vector with 1 indicating the node is included in the final cluster and 0 otherwise. Also, unlike other diffusion methods which usually grow from a seed set, these flow-based algorithms can both grow and shrink a seed set. As a result, the flow-based algorithms are usually phrased as cluster improving methods which are used to refine the outputs from other graph clustering approaches like PageRank-based methods. The resulting improved clusters often come with strong guarantee on conductance. For instance, MQI can find the subset of the input cluster with the minimum conductance [9]. The results and more details from this chapter can also be found in our paper [19].

Despite these flow-based algorithms are powerful both in theory and in practice. They are not widely adopted due to reasons including the lack of comprehensive comparison from the optimization point of view, the lack of examples demonstrating the power of these algorithms in various applications and the lack of user friendly software to deploy them. To resolve this, we first briefly talk about how the objective functions of various flow based cluster improvement algorithms can be unified. (Section 4.3) Then we provide detailed empirical studies on how to apply these methods to solve problems like (i) reducing conductance, (Section 4.3) (ii) growing or shrinking input sets to identify hidden target sets when seeded nearby by improving precision or recall, (Section 4.4) (iii) predicting labels of nodes when the nearby nodes share the same label and when given a set of true labels (Section 4.5) and (iv) generating locally-biased flow-based coordinates to highlight subtle hidden structure that is hidden from other visualization techniques like spectral methods. (Section 4.6) Finally, we

implement a user friendly open source software to facilitate future potential work on this topic of local graph clustering. (Section 4.7)

4.2 Unifying Objectives of Flow-based Algorithms

The three flow-based algorithms we will mainly discuss include MQI [9], FlowImprove [10] and LocalFlowImprove [11]. These three algorithms all seek to solve the following optimization problems subject to different parameter $\delta \geq 0$.

$$\begin{array}{ll} \underset{S \subset V}{\text{minimize}} & \dfrac{\mathbf{cut}(S)}{\mathbf{rvol}(S; R, \mathbf{vol}(R)/\mathbf{vol}(\bar{R}) + \delta)} \\ \text{subject to} & \mathbf{rvol}(S; \ldots) > 0 \end{array} \tag{4.1}$$

More specifically, MQI solves $\delta = \infty$ which implies that its solution set $S \subset R$. In another word, MQI can find the set S with the smallest conductance among all sets that are strictly within the seed set. FlowImprove solves $\delta = 0$ and LocalFlowImprove interpolates between FlowImprove and MQI by solving $\delta \geq 0$.

To understand better the connections between these three objectives, the following theorem states that conductance gets smaller, i.e., better, as we move from MQI to LocalFlowImprove to FlowImprove.

Theorem 4.2.1. *Let G be an undirected, connected graph with non-negative weights. Let $R \subset V$ have $\boldsymbol{vol}(R) \leq \boldsymbol{vol}(\bar{R})$, where $\bar{R}$ is the complement of R. Let S_{MQI}, S_{FI}, S_{LFI} be the optimal solution of the MQI, FlowImprove, and LocalFlowImprove(δ) objectives, respectively. If the solutions of FlowImprove and LocalFlowImprove satisfy $\boldsymbol{vol}(S_{FI}) \leq \boldsymbol{vol}(\bar{S}_{FI})$ and $\boldsymbol{vol}(S_{LFI}) \leq \boldsymbol{vol}(\bar{S}_{LFI})$ (that is, the solution set is on the small side of the cut), then for any $\delta > 0$, we have that*

$$\phi(S_{FI}) \leq \phi(S_{LFI}) \leq \phi(S_{MQI}).$$

Proof. The first piece, that $\phi(S_{\text{LFI}}) \leq \phi(S_{\text{MQI}})$ is a simple, useful exercise we briefly repeat from Theorem 4 of [11]. Note that if $S \subseteq R$ then $\phi(S) = \frac{\mathbf{cut}(S)}{\mathbf{rvol}(S;R,\kappa)}$ for any κ. Now, for any $\kappa \geq \mathbf{vol}(R)/\mathbf{vol}(\bar{R}) > 0$ we have

$$\phi(S_{\text{LFI}}) = \frac{\mathbf{cut}(S_{\text{LFI}})}{\mathbf{vol}(S_{\text{LFI}})} \leq \frac{\mathbf{cut}(S_{\text{LFI}})}{\mathbf{rvol}(S_{\text{LFI}}; R, \kappa)}.$$

Next, note that for the chosen setting of κ, we have that $\mathbf{rvol}(S; R, \kappa) > 0$ for all $S \subseteq R$. Thus, we have

$$\phi(S_{\text{LFI}}) \leq \min_{S \subseteq R} \frac{\mathbf{cut}(S)}{\mathbf{rvol}(S; R, \kappa)} \leq \min_{S \subseteq R} \phi(S) = \phi(S_{\text{MQI}}).$$

This shows that both LocalFlowImprove and FlowImprove give better conductance sets than MQI.

For the second piece, we use an alternative characterization of LocalFlowImprove as discussed in [50]. LocalFlowImprove(δ) is equivalent to solving the following optimization problem for some constant C:

$$\begin{array}{ll} \underset{S \subset V}{\text{minimize}} & \frac{\text{cut}(S)}{\text{rvol}(S; R, \text{vol}(R)/\text{vol}(\bar{R}))} \\ \text{subject to} & \frac{\text{vol}(S \cap R)}{\text{vol}(S)} \geq C, \mathbf{rvol}(S; \ldots) > 0 \end{array}$$

while FlowImprove solves the same problem without the constraint involving C. Then we have:

$$\frac{\text{cut}(S_{FI})}{\text{rvol}(S_{FI}; R, \text{vol}(R)/\text{vol}(\bar{R}))} \leq \frac{\text{cut}(S_{LFI})}{\text{rvol}(S_{LFI}; R, \text{vol}(R)/\text{vol}(\bar{R}))}$$

$$\frac{\text{cut}(S_{FI})}{\text{cut}(S_{LFI})} \leq \frac{\text{rvol}(S_{FI}; R, \text{vol}(R)/\text{vol}(\bar{R}))}{\text{rvol}(S_{LFI}; R, \text{vol}(R)/\text{vol}(\bar{R}))}.$$

If $\phi(S_{FI}) > \phi(S_{LFI})$, we have

$$\frac{\text{cut}(S_{FI})}{\text{cut}(S_{LFI})} > \frac{\text{vol}(S_{FI})}{\text{vol}(S_{LFI})}.$$

Thus,

$$\frac{\text{rvol}(S_{FI}; R, \text{vol}(R)/\text{vol}(\bar{R}))}{\text{rvol}(S_{LFI}; R, \text{vol}(R)/\text{vol}(\bar{R}))} \geq \frac{\text{cut}(S_{FI})}{\text{cut}(S_{LFI})} > \frac{\text{vol}(S_{FI})}{\text{vol}(S_{LFI})}.$$

By substituting the definition of rvol and $\text{vol}(S \cap \bar{R}) = \text{vol}(S) - \text{vol}(S \cap R)$,

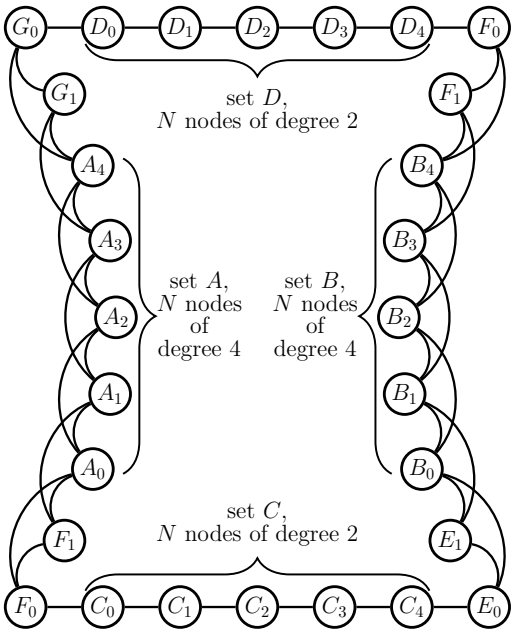
$$\frac{(1 + \text{vol}(R)/\text{vol}(\bar{R})) \cdot \text{vol}(S_{FI} \cap R) - \text{vol}(R)/\text{vol}(\bar{R}) \cdot \text{vol}(S_{FI})}{(1 + \text{vol}(R)/\text{vol}(\bar{R})) \cdot \text{vol}(S_{LFI} \cap R) - \text{vol}(R)/\text{vol}(\bar{R}) \cdot \text{vol}(S_{LFI})} > \frac{\text{vol}(S_{FI})}{\text{vol}(S_{LFI})}$$

$$\frac{\text{vol}(S_{FI} \cap R)}{\text{vol}(S_{FI})} > \frac{\text{vol}(S_{LFI} \cap R)}{\text{vol}(S_{LFI})} \geq C.$$

This means that S_{FI} also satisfies the additional constraint in the optimization problem of LFI. But S_{FI} has smaller objective value, which is a contradiction to the fact that S_{LFI} is the optimal solution of LFI optimization problem. □

Theorem 4.2.1 would suggest that one should always use FlowImprove to minimize the conductance around a reference set R. However, FlowImprove is not a strongly local algorithm while MQI and LocalFlowImprove are. Indeed, the following example shows that FlowImprove will return one fourth of the graph even when started with a set R that is a singleton.

Lemma 4.2.2. *Consider a cycle graph (illustrated on the right) with $4N + 8$ nodes in 4 major regions. Each set A and B has N nodes of degree 4 corresponding to a cycle graph with neighbors and neighbors of neighbors connected. Each set C and D has N degree 2 nodes. This introduces two extra nodes, of degree 3, between each pair of adjacent degree 2 and degree 4 regions. Consider using any node of degree 4 as the seed node to FlowImprove algorithm. Then, at optimality, FlowImprove will return a set with $N + 4$ nodes that is a continuous degree 4 region plus the four adjacent degree 3 nodes.*



Proof. Without loss of generality, suppose we seed on a node from set A. According to Lemma 7.2 of [19], when FlowImprove proceeds from iteration to iteration, it must return a set with a strictly smaller cut value or the seed set R was optimal. This means FlowImprove will only return one of the following sets. (Due to symmetry, there may be equivalent sets that we don't list.)

1. The seed node with cut 4.
2. A continuous subset of the A region, G_0, G_1, and a continuous subset of the set D, with cut 3.

3. All of the A region, two adjacent degree 3 nodes (without loss of generality, G_0 and G_1) on one end and one adjacency degree 3 node on the other edge (F_1), with cut 3.

4. All of the A region and all adjacency degree 3 nodes (G_0, G_1, F_0, F_1), with cut 2.

5. All of the A region and all adjacency degree 3 nodes (G_0, G_1, F_0, F_1 and additional nodes from sets C and D), with cut 2.

The goal is to show that case (4) is optimal, i.e., has the smallest objective value. Obviously, case (5) cannot be optimal since it has the same cut value as case (4) but smaller relative volume. Similarly, case (3) has the same cut value as case (2) but smaller relative volume. So case (3) won't be optimal either. So we only need to compare $\phi_R(S_1)$, $\phi_R(S_2)$ and $\phi_R(S_4)$. Observe that in this setting, $\theta = \frac{\text{vol}(R)}{\text{vol}(\bar{R})} = \frac{4}{(2N-1)4+2N\cdot 2+8\cdot 3} = \frac{1}{3N+5}$, so we can compute that

$$\phi_R(S_4) = \frac{2}{4-\theta(4(N-1)+3\cdot 4)} = \frac{3N+5}{4N+6} < 1 = \phi_R(S_1).$$

On the other hand, suppose in case (2), there are $1 \le k < N$ A nodes and $m \ge 0$ D nodes, then we can write

$$\phi_R(S_2) = \frac{3}{4-\theta(4(k-1)+3\cdot 2+2m)} \ge \frac{3}{4-6\theta} = \frac{9N+15}{12N+14} > \phi_R(S_4).$$

So case (4) is optimal. □

The connection to equation 3.4

The reason these algorithms are called "flow-based" algorithms is that their objectives can all be minimized by solving a series of maxflow-mincut subproblems. Specifically, they will all solve the following problem iteratively starting with $S_0 = R$ and $\theta_0 = \phi_R(R)$ until θ_0 no longer decreases, here $\sigma = \delta + \text{vol}(R)/\text{vol}(\bar{R})$.

$$S_{k+1} := \quad \text{argmin}_S \quad \mathbf{cut}(S) - \theta_k \left(\mathbf{vol}(S \cap R) - \sigma \mathbf{vol}(S \cap \bar{R})\right) \tag{4.2}$$

which is equivalent to the following optimization problem:

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \sum_{i\sim j} w_{ij}|x_i - x_j| + \delta(\mathbf{e}-\mathbf{x})^T\mathbf{d}_R + \sigma\theta_k\mathbf{x}^T\mathbf{d}_{\bar{R}} \\ \text{subject to} & \mathbf{x} \in \{0,1\}^n. \end{array} \tag{4.3}$$

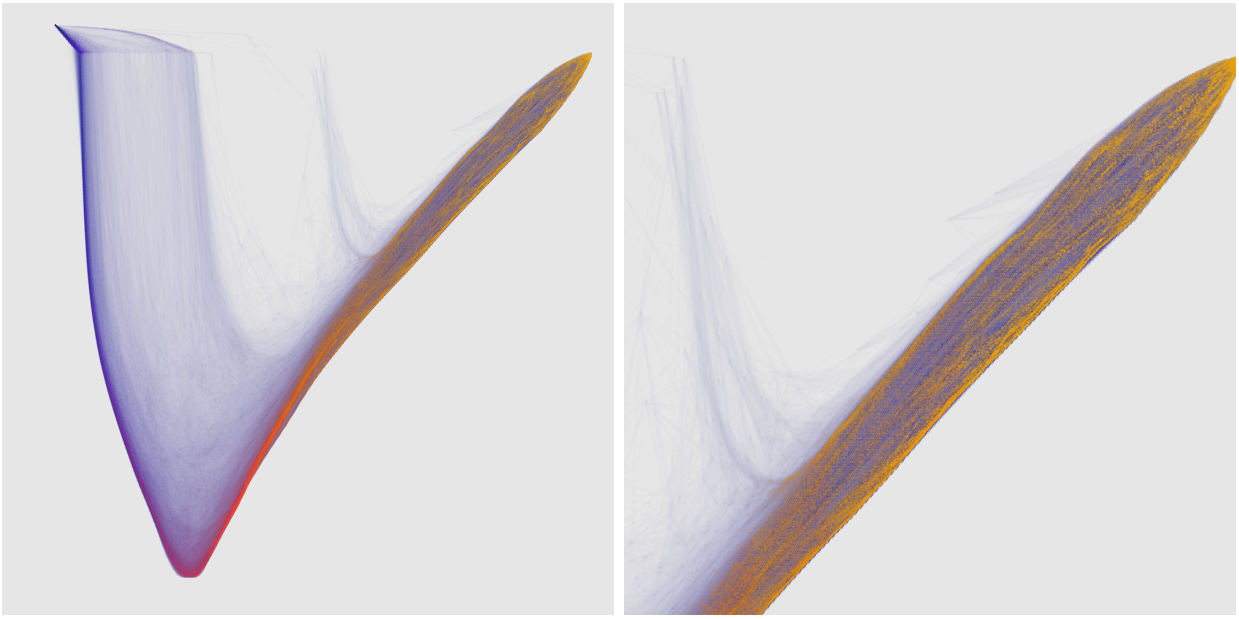
And problem 4.3 is equivalent to problem 3.4 with $\ell(x) = |x|$, $\gamma = \kappa = 0$, $\gamma_s = \delta$ and $\gamma_t = \sigma\theta_k$. Note that for LocalFlowImprove, to achieve strong locality, one must either construct this localized cut graph in an augmented way [11] or use a modified Dinic's maxflow algorithm [50] while for MQI and FlowImprove, any standard maxflow computations will work.

4.3 Flow-based Cluster Improvement Algorithms Reduce Conductance

In the first set of results, we will demonstrate how MQI, FlowImprove, and LocalFlowImprove can reduce the conductance of the input reference set in a real world graph. We will use FI and LFI-δ to denote FlowImprove and LocalFlowImprove. The dataset is a $k = 16$-nearest neighbor graph constructed on the Main Galaxy Sample (MGS) in SDSS Data Release 7. This data begins with the emission spectra of 517,182 galaxies in 3841 bands. We create a node for each galaxy and connect vertices if either is within the 16 closest vertices to the other based on a Euclidean distance-like measure). The graph is then weighted proportional to this distance. The result is a weighted undirected graph with 517,182 nodes and 15,856,315 edges (and 517,182 self-loops) representing nearest neighbor relationships among galaxy spectra. Figure 4.1 provides a visualization of a global Laplacian eigenvector embedding of this graph. For more details on this dataset, we refer readers to [51, 52].

Starting from a random node, we compute reference sets with 2-hop BFS. Then we run MQI, LFI-1, LFI-0.1, and LFI-0.01 on the results. We repeat this experiment 2526 times. The output to input conductance ratio is shown in Figure 4.3 with reference to the original reference conductance (Figure 4.2a) and also with reference to the MQI conductance (Figure 4.2b). From these results, we can see that reducing δ results in improved conductance for LFI and LFI always reduces the conductance more than MQI which verifies Theorem 4.2.1.

We repeat the experiment above, but in this time, we compute reference sets with seeded PageRank followed by a sweepcut procedure by [24] to locally optimize the conductance of



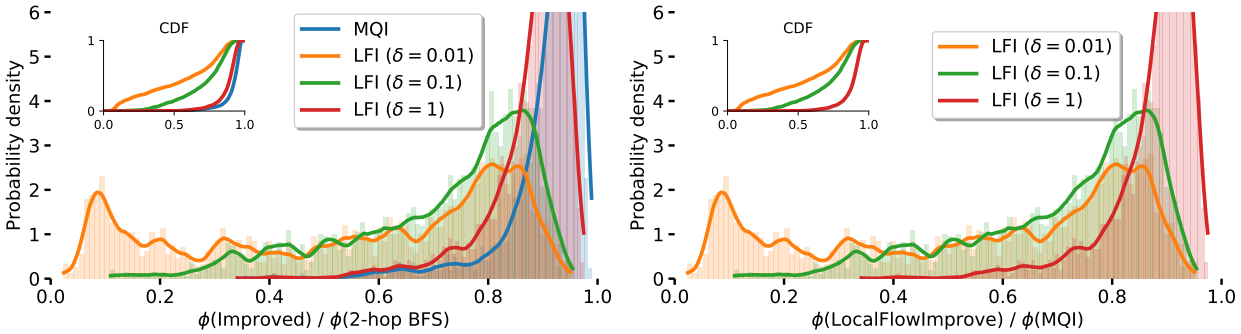
(a) The full graph (b) Zoom into top-right

Figure 4.1. The Main Galaxy Sample (MGS) dataset has 517,182 nodes and 32,229,812 edges. This display shows an eigenvector embedding of the graph along with edges shown in blue. The node color is determined by the horizontal coordinate, which will be re-used in plots in subsequent sections. The right part of the visualization (orange coordinates) hints at structure hidden within the upper band, which we will study in Section 4.6.

the result. Consequently, the reference sets we start with are already fairly high quality. The results with these new reference sets are in Figure 4.3a and Figure 4.3b. We can see that even when starting from reference sets that are already fairly high quality, MQI, FI and LFI can still improve their conductance.

4.4 Finding Nearby Targets by Growing and Shrinking

In this section, we will demonstrate cluster improvement methods can be used to recover a hidden target set of vertices from a nearby reference set, e.g. a coherent section of an image. The goal here is accuracy in returning the vertices of this set, and we can measure this in terms of precision and recall. Let T be a target set we seek to find and let S be the set returned by the algorithm. Then the precision score is $|T \cap S|/|S|$, which is the fraction



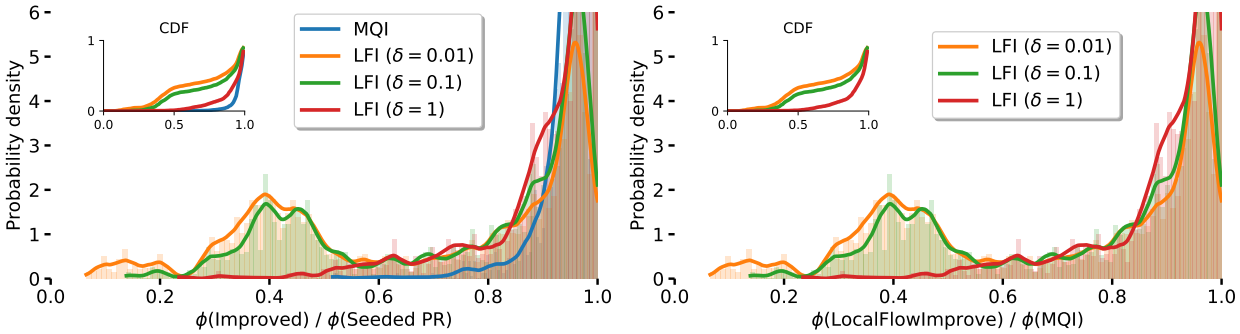
(a) Conductance improvement relative to 2-hop BFS (b) Conductance improvement relative to MQI

Figure 4.2. A summary of 2585 experiments in the `MGS` dataset that show (i) that reducing δ in LFI produces sets of smaller conductance, when the input set is a 2-hop BFS set, and also (ii) that LFI and FI always find smaller conductance sets than MQI.

of results that were correct, and the recall score is $|T \cap S|/|T|$, which is the fraction of all results that were obtained. The ideal scenario is that both precision and recall are near 1.

We begin by looking at the simple scenario when the initial reference R is entirely contained within T, and also a scenario when R is a strict superset of T. This setting allows us to see how the flow-based algorithms grow or shrink sets to find these targets T, and it gives us a useful comparison against simple greedy improvement algorithms as well as against spectral graph-based approaches. To view the results conveniently, we examine these algorithms on weighted graphs constructed from images. The construction of a graph based on an image is explained in Figure 4.4.

The results of the experiment are shown in Figure 4.5. We consider three distinct targets within a large image, as shown in Figure 4.5a and Figure 4.5b: the left dog, middle dog, and right person. In our first case, the reference is entirely contained within the target. In this case, we can use either FI or LFI to attempt to enlarge to the target. (Note that we cannot use MQI, as the target set is larger than the seed set.) For comparison, we use a seeded PageRank algorithm as well. We use two seeded PageRank scenarios that correspond to both FI and LFI, see Figure 4.5c to Figure 4.5f. ρ is a parameter to control the level of regularization similar to κ in equation 3.4. These show that spectral methods that grow tend either find



(a) Conductance improvement relative to seeded PageRank (b) Conductance improvement relative to MQI

Figure 4.3. A summary of 2526 experiments in the `MGS` dataset that show flow-based methods can still reduce conductance even when the input set is the result of another conductance minimizing procedure.

a region that is too big or fail to grow large enough to capture the entire region. This is quantified by a substantial drop in precision compared with the flow method. In neither case, spectral methods fail to capture the correct boundary of the target object. Second, we consider the case when the target is contained within the reference set. This corresponds to the MQI setting as well as a variation of spectral clustering that called Local Fiedler [53] (because it uses the eigenvector with minimal eigenvalue in a submatrix of the Laplacian). The results are in Figure 4.5g and Figure 4.5h, and they show a small precision advantage for the flow-based methods (see the text below each image). Finally, for reference, in Figure 4.5i and Figure 4.5j, we also include the results of a purely greedy strategy that grows or shrinks the reference set R to improve the conductance. This is able to find reasonably good results for only one of the test cases and shows that these sets are not overly *simple* to identify, e.g., since they cannot be detected by algorithms that trivially grow or expand the seed set.

4.5 Using Flow-based Algorithms for Semi-supervised Learning

In this section, we will evaluate the performance of flow-based algorithms in the semi-supervised learning setting. There are three datasets we use to evaluate the algorithm for

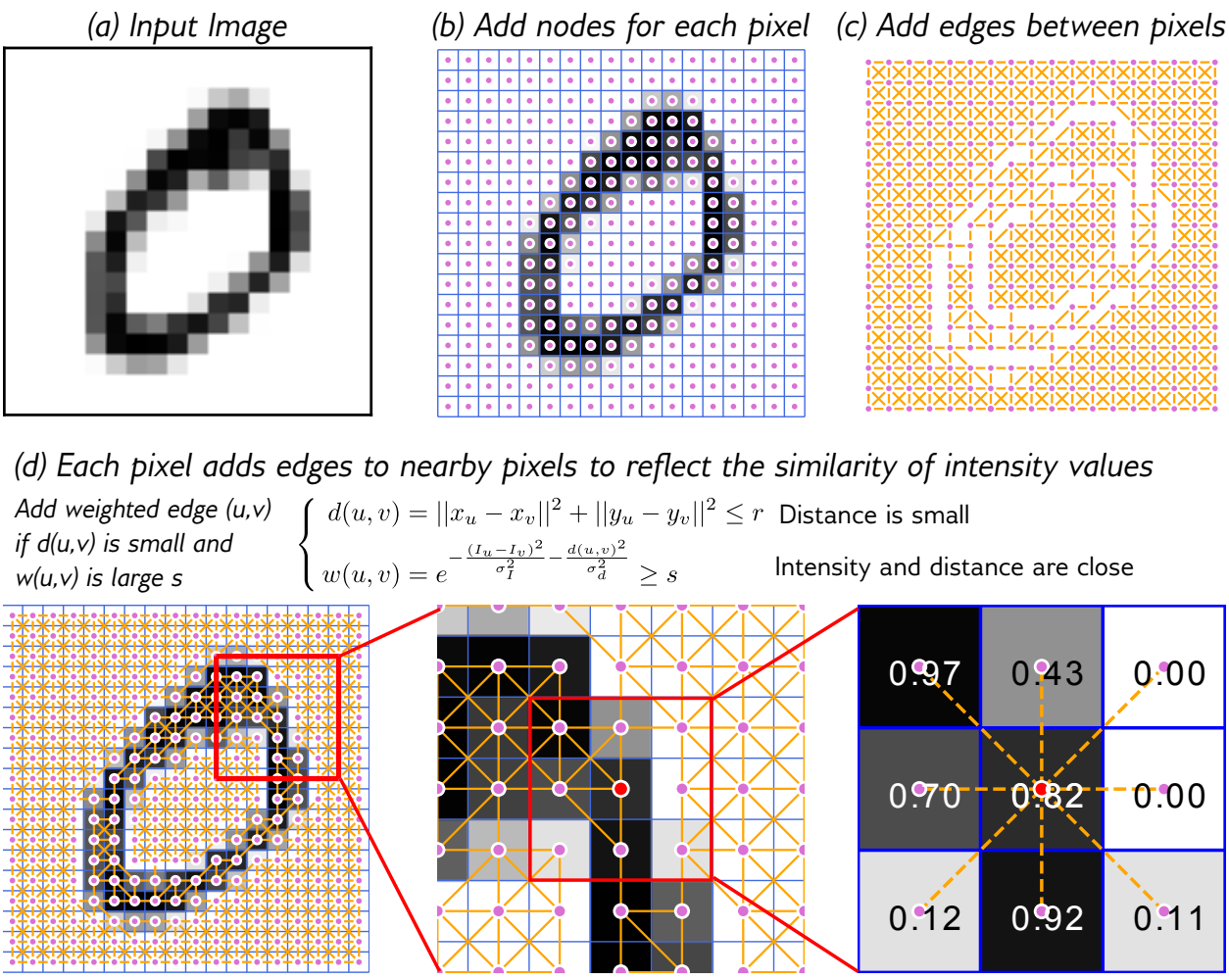


Figure 4.4. We turn an image into a graph by adding a node for every pixel (b). Then we connect the nodes if the associated pixels are close by (distance less than r) as well has have similar pixel values). We weight the edge by the degree of similarity. The resulting graph has small conductance sets when there are regions with similarly colored pixels.

semi-supervised learning: a synthetic stochastic block model, the MNIST digits data, and a citation network.

The experiment goes as follows: for each class, we randomly select a small subset of nodes, and we fix the labels of these nodes as known. We then run a spectral method or flow method where this set of nodes is the reference. We vary the number of labeled nodes included from 0.5% to 15% of the class size. For each fixed number of labeled nodes, we

(a) The full image (b) The targets

(c) FI (d) Seeded PageRank, $\rho = 10^{-12}$

(e) LFI-0.3 (f) Seeded PageRank, $\rho = 10^{-6}$

(g) MQI (h) LocalFiedler

(i) Greedy Grow (j) Greedy Shrink

Figure 4.5. Illustration of finding targets within an image (a) corresponding to the three low-conductance regions shown in (b). The reference sets given to MQI, FI, and LFI are given by the yellow regions, which either need to be grown or shrunk to find the target. For growing, we compare against seeded PageRank, which is a spectral analogue of FI and LFI; for shrinking, we compare against a local Fiedler vector, a spectral analogue of MQI, as well as simple greedy approaches for both. The flow-based methods capture the borders nicely and give high recall for growing and high precision for shrinking. Among other things, in this case, FI grows too large on the right person (c) whereas LFI (e) captures this target better. RC stands for recall and PR stands for precision.

repeat this 30 times to get a distribution of precision, recall, and $F1$ scores (where $F1$ is the harmonic mean of precision and recall), and we represent an aggregate view of this. For the flow methods, the output is a binary vector with 1 suggesting the node belongs to the class of reference nodes. Thus, it's possible that some nodes are classified into multiple classes, while some other nodes remain unclassified. We consider the first case as false positives and the second case as false negatives when computing precision and recall. For the spectral method, we use the real-valued solution vector to uniquely assign a node to a class. To robustify the process of rounding diffusion vector to class labels, we use a strategy from [28], which involves rounding to classes based on the node with the smallest rank in the ranked-list of each diffusion vector.

The results are in Figure 4.6 and show that the flow-based methods have uniformly high precision but extremely low recall when seed set is small. As the set of known labels increases, the recall increases, yielding a higher overall $F1$ score. Furthermore, the regularization in LFI-0.1 causes the set sizes to be smaller than FI, which manifests as a decrease in recall compared with FI. MNIST is the only exception where FI over-expands and classify one set of nodes into two labels.

The performance of FI and LFI in semi-supervised learning can be theoretically justified by the following lemma, which is a special case of Lemma 3.5 in our paper [19]:

Lemma 4.5.1. *If LFI or FI proceeds to iteration $k+1$ in equation 4.2, then it satisfies both $0 \leq \boldsymbol{rvol}(S_{k+1}) < \boldsymbol{rvol}(S_k)$ and $\boldsymbol{cut}(S_{k+1}) < \boldsymbol{cut}(S_k)$.*

Here $\mathbf{rvol}(S_{k+1}) \geq 0$ restricts that LFI cannot expand too far away from the seed set and $\mathbf{cut}(S_{k+1}) < \mathbf{cut}(S_k)$ restricts that FI won't stop expanding until it finds a set with smaller cut value.

4.6 Using Flow-based Methods for Local Coordinates

In this section, we investigate how flow-based methods can be used to compute real-valued coordinates that can show different types of structure within data compared with spectral methods.

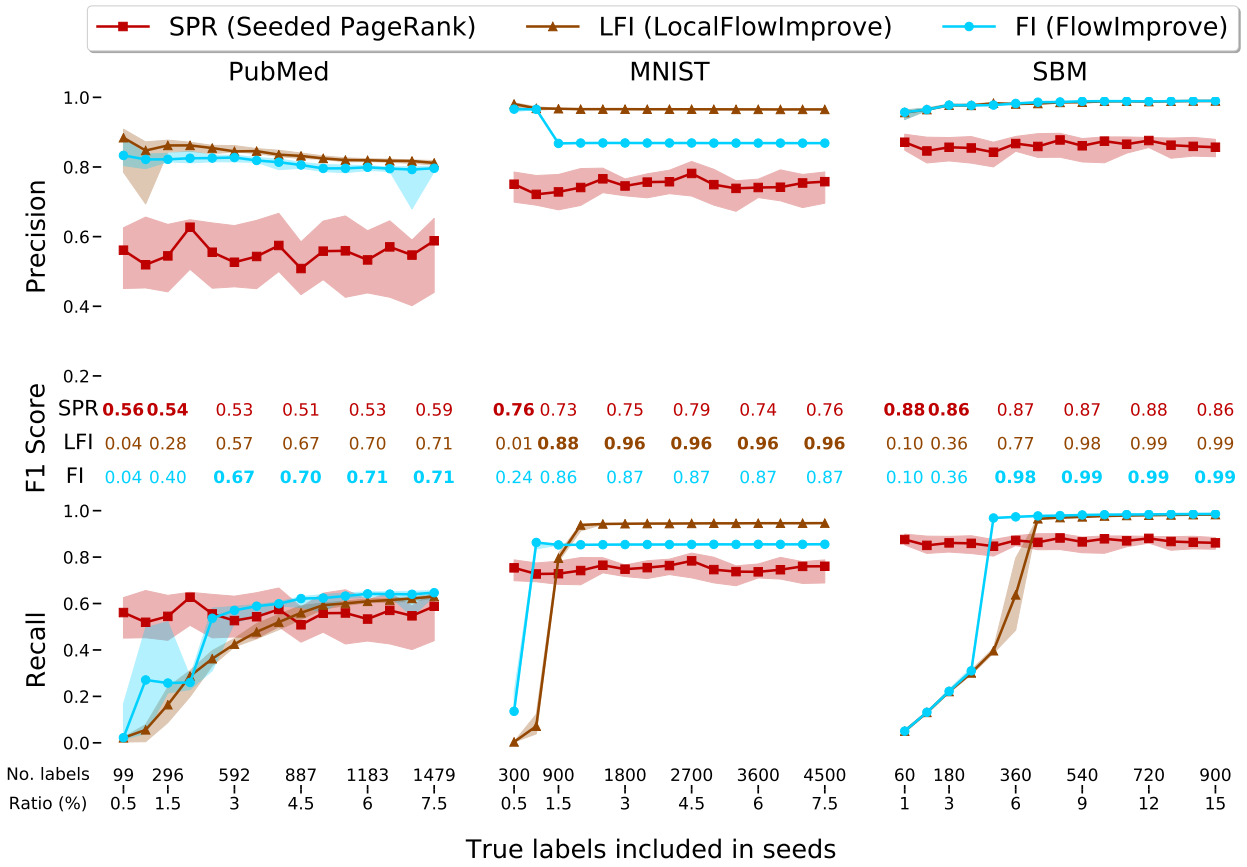


Figure 4.6. The results of the semi-supervised learning experiments show that the flow-based methods LFI-0.1 and FI are more sensitive to the number of known true labels included in the reference seed sets compared with seeded PageRank.

Given a reference set R, we randomly choose N subsets of R with exactly k entries; for each subset we add all nodes within distance d and call the resulting sets called $R_1, \ldots, R_N$. These serve as inputs to the flow algorithms. For each subset, we compute the result of a flow-based improvement algorithm, which gives us sets S_i. For each S_i, we form an indicator vector over the vertices, $\mathbf{x}_i$, where the entry is 1 if the vector is in the set and 0 otherwise. We assemble these vectors as columns of a matrix $\boldsymbol{X}$, and we use the coordinates of the dominant two left singular vectors as flow-based coordinates. This procedure is given as an algorithm in Algorithm 1. Note also that this procedure can be performed with spectral algorithms as well. The main intuition behind this procedure is that spectral method like

Algorithm 1 The local flow-based algorithm to generate flow-based coordinates.

Require: A graph G, a set R and parameters

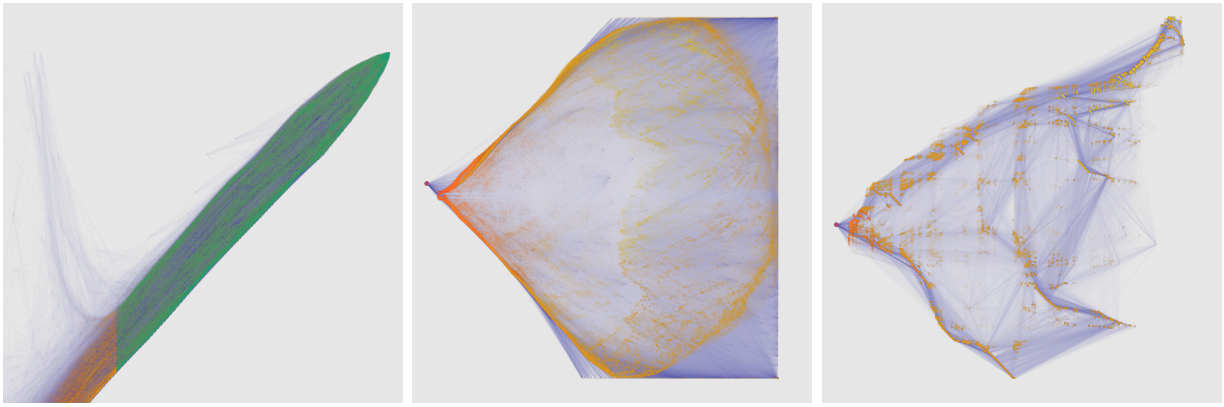
- N : the number of sets to sample
- k: the size of each subset
- d: the expansion distance
- c: the dimension of the final embedding
- `improve`: a cluster improvement algorithm

Ensure: An embedding of the graph into c coordinates for each node

Let n be the number of vertices.
Allocate $\boldsymbol{X}$, an n-by-N matrix of zeros.
for i in 1 to n **do**
    Let T be a sample of k entries from R at random without replacement
    Let R_i be the set of T and also all vertices within distance d from T
    Let S_i be the set that results from `improve`(G, R_i)
    Set $X[S_i, i] = 1$
end for
Compute the rank-c truncated SVD of $\boldsymbol{X}$ and let $\boldsymbol{U}$ be the left singular vectors.
Return $\boldsymbol{U}$, each row gives the c coordinates for a node

PageRank is solving a linear system. So the solution of a seeded PageRank can be thought as a stationary distribution of a lazy random walk with a uniform seed distribution over R. [24] Similarly, the first singular vector of $\boldsymbol{X}$ is also an approximation to the probability that one node shows up in the cluster of each iteration. To get the second coordinate, we want some orthogonal information, i.e. the second singular vector of $\boldsymbol{X}$.

We perform this analysis on the Main Galaxy Sample (MGS) dataset to highlight the local structure in a particularly dense region of the spectral embedding that was used for Figure 4.1. The seed region we use is shown in Figure 4.7a and has 201,252 vertices, which represents almost half the total graph. We use Algorithm 1 to get local spectral (Figure 4.7b) and local flow embeddings (Figure 4.7c). We find that the local flow embedding shows considerable substructure that is useful for future analysis.



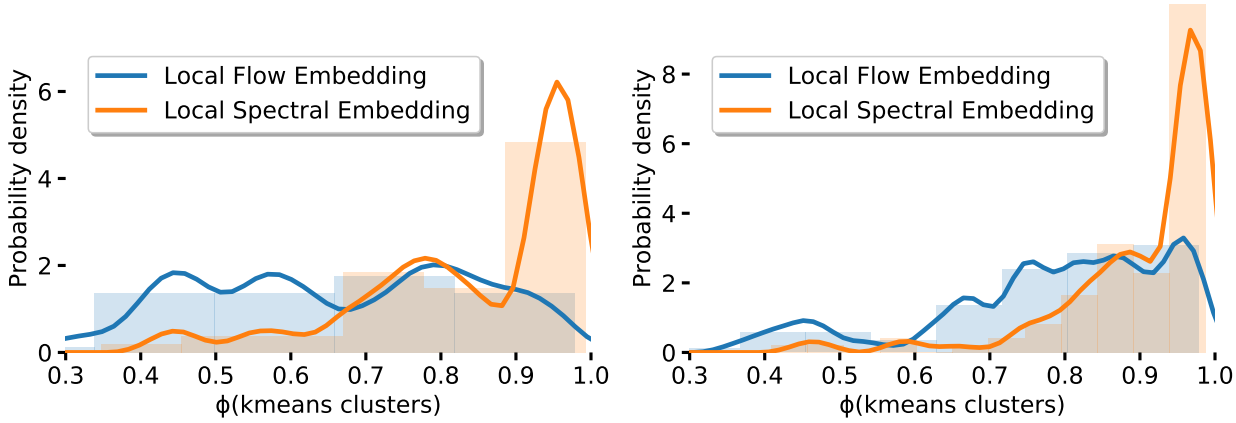
(a) The seed region (b) Local spectral embedding (c) Local flow embedding

Figure 4.7. Local spectral and local flow embeddings of the large, 201,252 node, seed region – shown in green in (a) – that is compressed in the global spectral embedding from Figure 4.1. In (b), the local spectral shows the nodes colored with the same color as in Figure 4.1. Nodes that were not touched by the local embedding are shown with the big node on the right hand side. In (c), the local flow embedding with the same color scheme and same big node on the right hand side. Note that the spectral embedding does not show any clear sub-structure besides a top-bottom split. In contrast, the flow embedding shows a number of pockets of structure indicative of small conductance subsets.

As a simple validation that this substructure is real, we use the 2d embedding coordinates as input to a k-means clustering procedure on both the local spectral and local flow coordinates. For each cluster that results from this procedure, we compute its conductance. Histograms of conductance values are shown in Figure 4.8 for $k = 50$ and $k = 100$. Both of these histograms show consistently smaller conductance values for the flow-based embedding.

4.7 Scalable Implementation of Local Graph Clustering Algorithms

In this section, we will introduce a open source framework *localgraphclustering*. The goal of this framework is:



(a) $k = 50$ (b) $k = 100$

Figure 4.8. A histogram of cluster conductance scores that come from using k-means on the two-dimensional local spectral and local flow embeddings from Figure 4.7.

Convenience:

It provides a unified and friendly interface in Python to compare different local graph clustering algorithms conveniently. This framework is used to perform all the analysis in this chapter. Currently, our framework includes global graph partitioning algorithms like computing Fiedler vector [53], PageRank-based local graph clustering algorithms like ACL [24] or l1-regularized PageRank [54], flow-based algorithms like MQI or LocalFlowImprove and some more advanced routines that combines ideas of PageRank and flow like CRD [55].

Scalability:

It provides scalable implementations for every local graph clustering algorithm. To avoid the overhead of Python, the algorithm is written in C++ and compiled into a shared object library which is then called by Python API. To reduce memory overhead, we represent the graph in its compressed sparse row format. And one can choose to store graph indices in 32 bit or 64 bit integers.

Exploration:

It provides convenient tools to explore characteristics of new datasets by generating its network community profile plot (NCP) [56]. Intuitively, NCP plot measures the quality of the best possible community at various community sizes. It can be formally defined as:

Definition 4.7.1. *Given a graph G with adjacency matrix $\boldsymbol{A}$, the NCP plot plots $\Phi(k)$ as a function of k, where*

$$\Phi(k) = \min_{S \subset V, f(S)=k} g(S) \tag{4.4}$$

Here $f(S)$ can be the effective size or volume of set S and $g(S)$ can be the conductance or expansion of set S.

To show the simplicity of our framework, in the following code segment, we use our framework to read a graph, perform ACL to find a local cluster and refine the local cluster with LFI.

```
import localgraphclustering as lgc
# Read a graph
g = lgc.GraphLocal('./datasets/JohnsHopkins.graphml','graphml','\t')
# Select seed nodes
ref_node = [2767]
# Run ACL
p = lgc.approximate_PageRank(g,ref_node,method="acl")
# Generate partition
partition = lgc.sweep_cut(g,p)[0]
# Compute statistics of this partition
g.set_scores(partition)
# Refine partition
refined_par = lgc.SimpleLocal(g,partition)
# Compute statistics of refined partition
g.set_scores(refined_par)
```

The following code segment shows how to generate NCP plot on the same dataset.

```
import localgraphclustering as lgc
# Read a graph
g = lgc.GraphLocal('./datasets/JohnsHopkins.graphml','graphml','\t')
# Create NCP data object
ncp_instance = lgc.NCPData(g)
# Call NCP with a method of choice
ncp_instance.approxPageRank(ratio=0.5,timeout=7200,nthreads=100)
# Check results in a data frame
ncp_instance.as_data_frame()
# Create NCP plot object
ncp_plots = lgc.NCPPlots(ncp_instance,method_name = "acl")
# Plot NCP results
fig, ax, min_tuples = ncp_plots.cond_by_size()
fig
```

The results of this code segment is shown in Figure 4.9. We use a hexagonal binning plot where the color of the points represent the number of experiments in that bin. If one area is brighter, it means more experiments lie in that area. The blue line marks the minimum y-axis value at each level of community size.

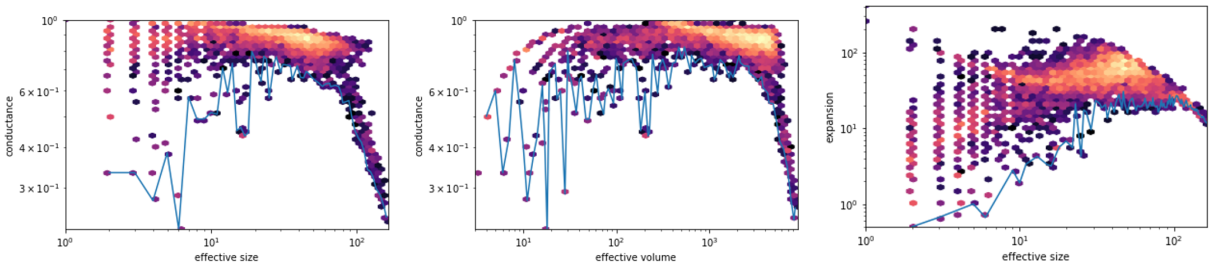


Figure 4.9. An example of NCP plot.

5. STRONGLY LOCAL P-NORM DIFFUSIONS ON GRAPHS FOR CLUSTERING AND SEMI-SUPERVISED LEARNING

5.1 Chapter Overview and Motivation

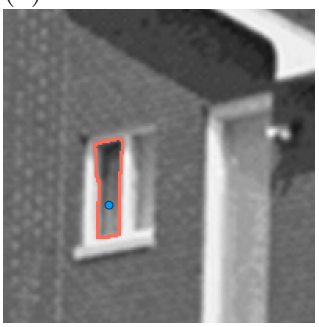
In this chapter, we will present a strongly local diffusion procedure to solve equation 3.4 with $\ell(x) = \frac{1}{p}|x|^p$ for any $p > 1$, especially $1 < p < 2$. The results and more details from this chapter can also be found in our paper [18].

The motivation of exploring these p-norm variants other than 1-norm or 2-norm which corresponds to flow-based or PageRank-based methods is that in our study of these two types of algorithms, we find that they often exhibit in some sense opposite behavior from each other. The flow based methods cannot grow from a small seed set. The linear diffusions or PageRank based methods on the other hand, often "expand" or "bleed out" over natural boundaries in the data. This can be seen in Figure 4.5 and Figure 4.6. This finding motivates us to look at p-norm variants of graph cut problems with $1 < p < 2$. Figure 5.1 is an example to show what these p-norm variants are capable of, where our p-norm variant can not only grow from a single seed but also capture the boundary of the target cluster nicely.

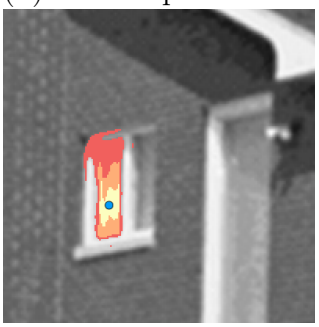
We are not the first to notice the usefulness or effects of p-norm as a solution. For instance, the p-Laplacian [43] and related ideas [45] has been widely studied as a way to improve results in spectral clustering [44] and semi-supervised learning [46]. This has recently been used to show the power of simple nonlinearities in diffusions for semi-supervised learning as well [7]. As the most related work, a similar p-norm variant of the flow dual problem is recently proposed [48] for local graph clustering. In Figure 5.2, we compare using power functions $\ell(x) = \frac{1}{p}|x|^p$ to a variety of other techniques for semi-supervised learning and local clustering. (For various reasons, we interchangbly switch between p-norm and q-norm in the subsequent sections of this chapter.)

The major rationale for our approach is that our algorithmic techniques are closely related to those used for 2-norm optimization. It remains the case that spectral (2-norm) approaches are far more widely used in practice, partly because they are simpler to implement and use, whereas the other approaches involve more delicate computations.

(a) Seed node and the target.



(b) 2-norm problem.



(c) 1.1-norm problem.

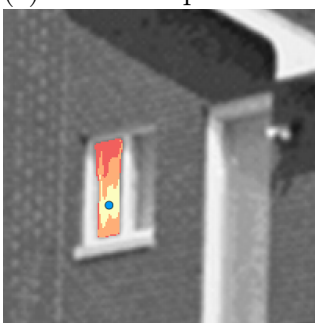
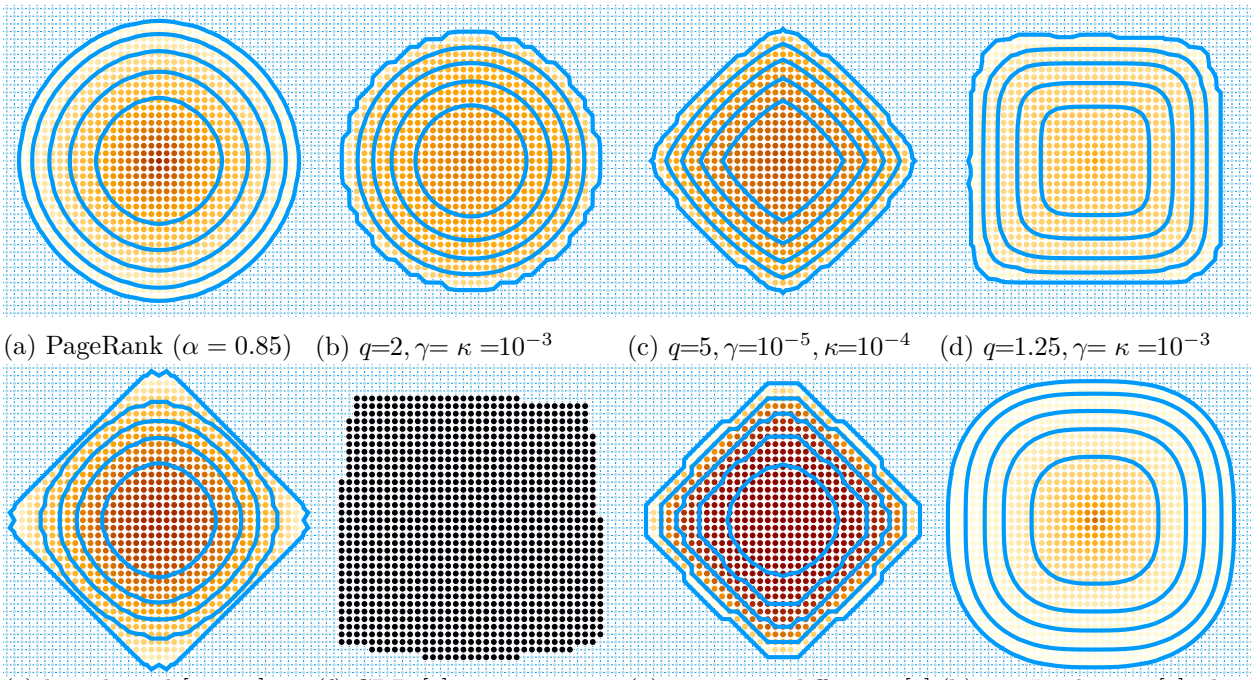


Figure 5.1. A simple illustration of the benefits of our p-norm methods. In this problem, we generate a graph from an image with weighted neighbors as described in [57]. We intentionally make this graph consider *large* regions, so each pixel is connected to all neighbors within 40 pixels away. (Full details in the supplement.) The target in this problem is the *cluster* defined by the interior of the window and we select a single pixel inside the window as the seed. The three colors (yellow, orange, red) show how the non-zero elements of the solution *fill-in* as we decrease a sparsity penalty in our formulation (yellow is sparsest, red is densest). The 2-norm result exhibits a typical phenomenon of over-expansion, whereas the 1.1-norm accurately captures the true boundary. We tried running various 1-norm methods, but they were unable to grow a single seed node, as has been observed in many past experiments and also theoretically justified in [19, Lemma 7.2].



(a) PageRank ($\alpha = 0.85$) (b) q=2, γ= κ =10^{-3} (c) q=5, γ=10^{-5}, κ=10^{-4} (d) q=1.25, γ= κ =10^{-3}

(e) heat kernel [26, 27] $t = 10, \varepsilon = 0.003$ (f) CRD [8] $U = 60, h = 60, w = 5$ (g) p = 1.5-diffusion [7], h=0.002, $k = 35000$ (h) 1.5-Laplacian [7], h = 0.0001, $n = 7500$

Figure 5.2. The graph is a 50-by-50 regular grid-graph with 4 axis-aligned neighbors, the seed is in the center. The diffusions localize before the boundary so we only show the relevant region and the quantile contours of the values. We selected the parameters to give similar-sized outputs. (Top row) At left (a), we have seeded PageRank; (b)-(d) show our q-norm objectives; (b) is a 2-norm which closely resembles PageRank; (c) is a 5-norm that has diamond-contours; and (d) is a 1.25-norm that has square contours. (Bottom row) Existing work with the (e) heat kernel diffusion [26, 27], (f) CRD [8], (g) nonlinear diffusions [7] (with a simple (g) p-norm nonlinearity in the diffusion or a (h) p-Laplacian) show that similar results are possible with existing methods, although they lack the simplicity of our optimization setup and often lack the strongly local algorithms.

In Section 5.3, we will show that our formulations are amenable to similar computation techniques as used for 2-norm problems, which makes them also easier to understand or implement and runs much faster than other nonlinear approaches 5.6. Moreover, our approach can solve other types of $\ell(x)$ functions as long as certain constraints are satisfied in Section 5.2. We also provide a theoretical analysis of finding planted target clusters

with our method and show that the p-norm cut functions improve on the standard Cheeger inequalities for random walk and spectral methods under certain assumptions 5.5.

5.2 Beyond P-norm Cut

If $\ell(x)$ is convex, then the problem 3.4 is convex and can be solved via general-purpose solvers such as CVX. An additional convex solver is SnapVX [58], which studied a general combination of convex functions on nodes and edges of a graph, although neither of these approaches scale to the large graphs we study in subsequent portions of this paper (65 million edges). To produce a specialized, strongly local solver, we found it necessary to restrict the class of functions $\ell(x)$ to have similar properties to the power function $\ell(x) = \frac{1}{p}|x|^p$ and its derivative $\ell'(x)$.

Definition 5.2.1. *In the* $[-1, 1]$ *domain, the loss function* $\ell(x)$ *should satisfy (1)* $\ell(x)$ *is convex; (2)* $\ell'(x)$ *is an increasing and anti-symmetric function; (3) For* $\Delta x > 0$, $\ell'(x)$ *should satisfy either of the following condition with constants* $k > 0$ *and* $c > 0$ *(3a)* $\ell'(x + \Delta x) \leq \ell'(x) + k\ell'(\Delta x)$ *and* $\ell''(x) > c$ *or (3b)* $\ell'(x)$ *is strictly increasing, c-Lipschitz continuous and* $\ell'(x + \Delta x) \geq \ell'(x) + k\ell'(\Delta x)$ *when* $x \geq 0$.

Remark. *If* $\ell'(x)$ *is Lipschitz continuous with Lipschitz constant to be* L *and* $\ell''(x) > c$*, then constraint 3(a) can be satisfied with* $k = L/c$*. However,* $\ell'(x)$ *can still satisfy 3(a) even if it is not Lipschitz continuous. A simple example is* $\ell(x) = |x|^{1.5}$, $-1 \leq x \leq 1$*. In this case,* $k = 1$ *but it is not Lipschitz continuous at* $x = 0$*. On the other hand, when* $\ell'(x)$ *is Lipschitz continuous, it can satisfy constraint 3(b) even if* $\ell''(x) = 0$*. An example is* $\ell(x) = |x|^{3.5}$, $-1 < x < 1$*. In this case* $\ell''(x) = 0$ *when* $x = 0$ *but* $\ell'(x)(x + \Delta x) \geq \ell'(x)(x) + \ell'(x)(\Delta x)$ *when* $x \geq 0$.

Lemma 5.2.1. *The power function* $\ell(x) = \frac{1}{p}|x|^p$, $-1 < x < 1$ *satisfies definition 5.2.1 for any* $p > 1$*. More specifically, when* $1 < p < 2$, $\ell(x)$ *satisfies 3(a) with* $c = p-1$ *and* $k = 2^{2-p}$*, when* $p \geq 2$, $\ell(x)$ *satisfies 3(b) with* $c = p - 1$ *and* $k = 1$.

Proof. First, we know $\ell'(x) = |x|^{p-1}\operatorname{sgn}(x)$ and $\ell''(x) = (p-1)|x|^{p-2}$. And we define $\ell''(0) = \infty$.

For 3(a), since $-1 < x < 1$, $1 < p < 2$, we have $\ell''(x) > (p-1)$. On the other hand

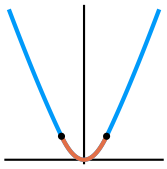
$$\frac{\ell'(x+\Delta x) - \ell'(x)}{\ell'(\Delta x)} = \left|\frac{x}{\Delta x} + 1\right|^{p-1} \operatorname{sgn}\left(\frac{x}{\Delta x} + 1\right) - \left|\frac{x}{\Delta x}\right|^{p-1} \operatorname{sgn}\left(\frac{x}{\Delta x}\right)$$

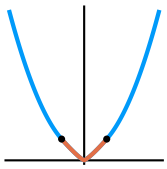
Define a new function $f(x) = |1+x|^{p-1}\operatorname{sgn}(1+x) - |x|^{p-1}\operatorname{sgn}(x)$. $f'(x) = |1+x|^{p-2} - |x|^{p-2}$. So the maximum of $f(x)$ is achived at $f(-0.5) = 2^{2-p}$.

For 3(b), since $-1 < x < 1$, $p > 2$, we have $\ell''(x) < (p-1)$. And when $x \geq 0$, $(x+\Delta x)^{p-1} \geq x^{p-1} + \Delta x^{p-1}$ is obvious. □

Note that the $\ell(x) = |x|$ does not satisfy either choice for property (3). Consequently, our theory will not apply to mincut problems. We note that p-norm generalizations of the Huber and Berhu loss functions [59] do satisfy these definitions.

Definition 5.2.2. *Given $1 < q < 2$ and $0 < \delta < 1$, the "q-Huber" and "Berq" function are*

q-Huber $\quad \ell(x) =$  $= \begin{cases} \frac{1}{2}\delta^{q-2}x^2 & \text{if } |x| \leq \delta \\ \frac{1}{q}|x|^q + (\frac{q-2}{2q})\delta^q & \text{otherwise} \end{cases}$

Berq $\quad \ell(x) =$  $= \begin{cases} \frac{1}{q}\delta^{2-q}|x|^q & \text{if } |x| \leq \delta \\ \frac{1}{2}x^2 + (\frac{2-q}{2q})\delta^2 & \text{otherwise.} \end{cases}$

Lemma 5.2.2. *When $-1 \leq x \leq 1$, both "q-Huber" and "Berq" satisfy Definition 5.2.1. The value of k for both is 2^{2-q}, the c for q-Huber is $q-1$ while the c for "Berq" is 1.*

Proof. Obviously, both condition (1) and (2) are satisfied for "q-Huber" and "Berq". Now we show 3(a) is also satisfied for "q-Huber" based on the proof of lemma 5.2.1. The proof of "Berq" is also similar.

When $\Delta x > \delta$ ($\Delta x \le \delta$ is similar)

$$k = \frac{\ell'(x+\Delta x) - \ell'(x)}{\Delta x^{q-1}}$$
$$= \begin{cases} \left|\frac{x}{\Delta x}+1\right|^{q-1}\operatorname{sgn}\left(\frac{x}{\Delta x}+1\right) - \left|\frac{x}{\Delta x}\right|^{q-1}\operatorname{sgn}\left(\frac{x}{\Delta x}\right) & , \quad |x|>\delta, |x+\Delta x|>\delta \\ \frac{\delta^{q-2}(x+\Delta x) - |x|^{q-1}\operatorname{sgn}(x)}{\Delta x^{q-1}} & , \quad |x|>\delta, |x+\Delta x|\le\delta \\ \frac{|x+\Delta x|^{q-1}\operatorname{sgn}(x+\Delta x) - \delta^{q-2}x}{\Delta x^{q-1}} & , \quad |x|\le\delta, |x+\Delta x|>\delta \\ \frac{\Delta x^{2-q}}{\delta^{2-q}} & , \quad |x|\le\delta, |x+\Delta x|\le\delta \end{cases}$$

Case 1:

Same as the proof of lemma 5.2.1.

Case 2:

In this case, x can only be negative, i.e. $x < -\delta$. After some simplification,

$$k = \left(\frac{\Delta x}{\delta}\right)^{2-q} - \left(\left(\frac{-x}{\delta}\right)^{2-q} - 1\right)\left(\frac{-x}{\Delta x}\right)^{q-1}$$

Note that the right hand side is an increasing function of Δx and $-\delta - x \le \Delta x \le \delta - x$. Replacing Δx by $-\delta - x$ yields

$$k = \frac{(-x)^{q-1} - \delta^{q-1}}{(-x-\delta)^{q-1}} > 0$$

Replacing Δx by $\delta - x$ yields

$$k = \frac{\delta^{q-1} + (-x)^{q-1}}{(\delta - x)^{q-1}} \le 2^{2-q}$$

Here the last inequality is due to Jensen's inequality.

Case 3:

Its proof is very similar to case 2.

Case 4:

Since $0 < \Delta x \le 2\delta$, $0 \le k \le 2^{2-q}$. □

We now state uniqueness.

Theorem 5.2.3. *Fix a set S, $\gamma > 0, \kappa > 0$. For any loss function satisfying Definition 5.2.1, then the solution $\mathbf{x}$ of (3.4) is unique. Moreover, define a residual function $\mathbf{r}(\mathbf{x}) = -\frac{1}{\gamma}\boldsymbol{B}^T diag(\ell'(\boldsymbol{B}\mathbf{x}))\mathbf{w}$. A necessary and sufficient condition to satisfy the KKT conditions is to find $\mathbf{x}^*$ where $\mathbf{x}^* \geq 0$, $\mathbf{r}(\mathbf{x}^*) = [r_s, \mathbf{g}^T, r_t]^T$ with $\mathbf{g} \leq \kappa\mathbf{d}$ (where $\mathbf{d}$ reflects the original graph), $\mathbf{k}^* = [0, \kappa\mathbf{d} - \mathbf{g}, 0]^T$ and $\mathbf{x}^T(\kappa\mathbf{d} - \mathbf{g}) = 0$.*

Proof. We first prove uniqueness. The Hessian of the objective in (3.4) is:

$$H(i,j) = \begin{cases} \ell''(x_i - (\mathbf{e}_S)_i) & \text{if } i = j \\ \ell''(x_i - x_j) & \text{if } i \sim j \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Thus $\mathbf{x}^T\boldsymbol{H}\mathbf{x} = \sum_{i\in V} x_i^2\ell''(x_i - (\mathbf{e}_S)_i) + \sum_{i,j,i\sim j} x_i x_j \ell'' x_i - x_j)$. If 3(a) is satisfied, we have $\ell''(x) > 0$ which means $\mathbf{x}^T\boldsymbol{H}\mathbf{x} > 0$. So the objective 3.4 is strictly convex and the uniqueness is guaranteed. When 3(b) is satisfied, $\ell'(x + \Delta x) \geq \ell'(x) + k\ell'(\Delta x)$ guarantees that $\ell''(x)$ can only become zero in a range around zero, i.e. $\ell'(x) = \ell''(x) = 0$ when $x \in [-\psi, \psi]$, where $0 \leq \psi \leq 1$. Then $\mathbf{x}^T\boldsymbol{H}\mathbf{x} = 0$ implies $x_i \geq 1 - \psi$ when $i \in S$, $x_i \leq \psi$ when $i \notin S$ and $-\psi \leq x_i - x_j \leq \psi$ or $x_i x_j = 0$. In this case, the uniqueness is implied by $\kappa\gamma\mathbf{d}$ in (3.4), i.e. each x_i will be the smallest feasible value.

Next, we will show the KKT condition of (3.4). If we translate problem (5.2.1) to add the constraint $\mathbf{u} = \boldsymbol{B}\mathbf{x}$, then the loss is $\ell(\mathbf{u})$. The Lagrangian is

$$\mathcal{L} = \mathbf{w}^T\ell(\mathbf{u}) + \kappa\gamma\mathbf{d}^T\mathbf{x} - \mathbf{f}^T(\boldsymbol{B}\mathbf{x} - \mathbf{u}) - \lambda_s(x_s - 1) - \lambda_t x_t - \mathbf{k}^T\mathbf{x}$$

Standard optimality results give the KKT of (5.2.1) as

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{x}} &= \kappa \mathbf{d} - \frac{1}{\gamma} \boldsymbol{B}^T \mathbf{f} - \lambda_s \mathbf{e}_s - \lambda_t \mathbf{e}_t - \mathbf{k} = 0 \\
\frac{\partial L}{\partial \mathbf{u}} &= \operatorname{diag}(\ell'(\mathbf{u}))\mathbf{w} + \mathbf{f} = 0 \\
\mathbf{k}^T \mathbf{x} &= 0 \\
\boldsymbol{B}\mathbf{x} &= \mathbf{u} \\
\mathbf{k} \geq 0, x_s &= 1, x_t = 0
\end{aligned} \tag{5.2}$$

Thus, combining the first and second equations, $\mathbf{r} = \frac{1}{\gamma} \boldsymbol{B}^T \mathbf{f}$. Since $\mathbf{k} \geq 0$, from the first equation, we have $\mathbf{g} \leq \kappa \mathbf{d}$. And from $\mathbf{k}^T \mathbf{x} = 0$, we have $\mathbf{x}^T(\kappa \mathbf{d} - \mathbf{g}) = 0$. □

5.3 Strongly Local Algorithms

In this section, we will provide a strongly local algorithm to approximately optimize equation (3.4) with $\ell(x)$ satisfying definition 5.2.1. The simplest way to understand this algorithms is as a nonlinear generalization of the Andersen-Chung-Lang *push* procedure for PageRank [24], which we call ACL. (The ACL procedure has strong relationships with Gauss-Seidel, coordinate solvers, and various other standard algorithms.) The overall algorithm is simple: find a vertex i where the KKT conditions from Theorem 5.2.3 are violated and increase x_i on that node until we approximately satisfy the KKT conditions. Update the residual, look for another violation, and repeat. The ACL algorithm targets $\ell(x) = \frac{1}{2}x^2$ case, which has a closed form update. We simply need to replace this with a binary search.

Algorithm 2 `nonlin-cut`$(\gamma, \kappa, \rho, \varepsilon)$ for set S and graph G where $0<\rho<1$ and $0<\varepsilon$ determine accuracy

1: Let $x(i) = 0$ except for $x_s = 1$ and set $\mathbf{r} = -\frac{1}{\gamma} \boldsymbol{B}^T \operatorname{diag}[\ell'(\boldsymbol{B}\mathbf{x})]\mathbf{w}$
2: While there is any vertex i where $r_i > \kappa d_i$, or stop if none exists *(find a KKT violation)*
3: Apply `nonlin-push` at vertex i, updating $\mathbf{x}$ and $\mathbf{r}$
4: Return $\mathbf{x}$

For $\rho < 1$, we only approximately satisfy the KKT conditions, as discussed further in the Section 5.4. We have the following strongly local runtime guarantee when 3(a) in

Algorithm 3 `nonlin-push`$(i, \gamma, \kappa, \mathbf{x}, \mathbf{r}, \rho, \varepsilon)$

1: Use binary search to find Δx_i such that the ith coordinate of the residual after adding Δx_i to x_i, $r_i' = \rho\kappa d_i$, the binary search stops when the range of Δx is smaller than ε *(satisfy KKT at i)*.
2: Change the following entries in $\mathbf{x}$ and $\mathbf{r}$ to update the solution and residual
3: (a) $x_i \leftarrow x_i + \Delta x_i$
4: (b) For each neighbor j in the original graph G, $r_j \leftarrow r_j + \frac{1}{\gamma} w_{i,j} \ell'(x_j - x_i) - \frac{1}{\gamma} w_{i,j} \ell'(x_j - x_i - \Delta x_i)$

definition 5.2.1 is satisfied. (This ignores binary search, but that only scales the runtime by $\log(1/\varepsilon)$ because the values are in $[0, 1]$.)

Theorem 5.3.1. *Let $\gamma > 0, \kappa > 0$ be fixed and let k and c be the parameters from 3(a) of Definition 5.2.1 for $\ell(x)$. For $0 < \rho < 1$, suppose* `nonlin-cut` *stops after K iterations, and d_i is the degree of node updated at the i-th iteration, then K must satisfy: $\sum_{i=1}^{K} d_i \leq vol(S)/c\ell'^{-1}\left(\gamma(1-\rho)\kappa/k(1+\gamma)\right) = O(vol(S))$.*

The notation ℓ'^{-1} refers to the inverse functions of $\ell'(x)$, This function must be invertible under the the definition of 3(a). Note that this sum of degrees bounds the total work because a *push* step at node i is $O(d_i)$ work (ignoring the binary search). Also note that if $\kappa = 0$, $\gamma = 0$, or $\rho = 1$, then this bound goes to ∞ and we lose our guarantee. *However, if these are not the case, then the bound shows that the algorithm will terminate in time that is independent of the size of the graph.* This is the type of guarantee provided by *strongly local* graph algorithms and has been extremely useful to scalable network analysis methods [11, 26, 60–62]. We also show that a similar runtime guarantee holds when $\ell(x)$ satisfies 3(b) of Definition 5.2.1. To prove this theorem, we first give the following lemmas.

Lemma 5.3.2. *During algorithm 2, for any $i \in \{V \backslash \{s, t\}\}$, g_i will stay nonnegative and $0 \leq x_i \leq 1$.*

Proof. We can show this by induction. At the initial step, for node $i \in S$, $g_i = d_i$, and for node $i \in \bar{S}$, $g_i = 0$. And after a nonlin-push step, every g_i will stay nonnegative.

To prove $0 \leq x_i \leq 1$, by expanding g_i, we have

$$g_i = -\frac{1}{\gamma} \sum_{j \sim i} w_i \ell'(x_i - x_j) - d_i \ell'(x_i - (\mathbf{e}_S)_i)$$

$x_i \geq 0$ because we only increase $\mathbf{x}$ and it starts at zero. Suppose x_i is the largest element of $\mathbf{x}$ and $x_i > 1$, then we will have $\ell'(x_i - x_j) \geq 0$ for $j \sim i$ and $\ell'(x_i - (\mathbf{e}_S)_i) > 0$. Then $g_i < 0$, which is a contradiction. □

Lemma 5.3.3. *When 3(a) is satisfied, after calling* `nonlin-push` *on node i, the decrease of $\|\mathbf{g}\|_1$ will be strictly larger than*

$$cd_i(\ell')^{-1}\left(\frac{\gamma(1-\rho)\kappa}{k(1+\gamma)}\right)$$

Proof. We use $\mathbf{g}'$ to denote $\mathbf{g}$ after calling `nonlin-push` on node i. At any intermediate step of `nonlin-cut` procedure,

$$\|\mathbf{g}\|_1 = \sum g_i = -\sum_{i\in S} d_i\ell'(x_i - 1) - \sum_{i\in\bar{S}} d_i\ell'(x_i)$$

This is because for any edge $(i,j) \in E$, g_i has a term $\frac{1}{\gamma}w(i,j)\ell'(x_i - x_j)$ while g_j has a term $\frac{1}{\gamma}w(j,i)\ell'(x_j - x_i)$. Since our graph is undirected, $w(i,j) = w(j,i)$, so these two terms will cancel out. What remains are the terms corresponding to the edges connecting to s or t. So after calling `nonlin-push` on node i,

$$\begin{aligned}\|\mathbf{g}\|_1 - \|\mathbf{g}'\|_1 &= d_i\ell'(x_i + \Delta x_i - (\mathbf{e}_S)_i) - d_i\ell'(x_i - (\mathbf{e}_S)_i)\\ &\geq d_i\min\{l''(x_i + \Delta x_i - (\mathbf{e}_S)_i), l''(x_i - (\mathbf{e}_S)_i)\}\Delta x_i\\ &\geq cd_i\Delta x_i\end{aligned}$$

On the other hand, we need to choose Δx_i such that $g_i' = \rho\kappa d_i$. We know

$$g_i' = -\frac{1}{\gamma}\sum_{j\sim i} w(i,j)\ell'(x_i + \Delta x_i - x_j) - d_i\ell'(x_i + \Delta x_i - (\mathbf{e}_S)_i)$$

is a decreasing function of Δx_i. And when $\Delta x_i = 0$, $g_i' = \kappa d_i > \rho\kappa d_i$, when $\Delta x_i = 1$, $g_i' < 0 < \rho\kappa d_i$, since $\ell'(x)$ is a strictly increasing function, there exists a unique Δx_i such that $g_i' = \rho\kappa d_i$. Moreover, we can lower bound Δx_i. To see that,

$$\begin{aligned}
g_i' &= \rho\kappa d_i \\
&= -\frac{1}{\gamma}\sum_{j\sim i} w(i,j)\ell'(x_i + \Delta x_i - x_j) - d_i\ell'(x_i + \Delta x_i - (\mathbf{e}_S)_i) \\
&\geq -\frac{1}{\gamma}\sum_{j\sim i} w(i,j)\ell'(x_i - x_j) - d_i\ell'(x_i - (\mathbf{e}_S)_i) - \frac{k(1+\gamma)}{\gamma} d_i\ell'(\Delta x_i) \\
&= g_i - \frac{k(1+\gamma)}{\gamma} d_i\ell'(\Delta x_i)
\end{aligned}$$

Thus, we have

$$\Delta x_i \geq (\ell')^{-1}\left(\frac{\gamma(g_i - \rho\kappa d_i)}{k(1+\gamma)d_i}\right) > (\ell')^{-1}\left(\frac{\gamma(1-\rho)\kappa}{k(1+\gamma)}\right)$$

which means

$$\left\|\mathbf{g}\right\|_1 - \left\|\mathbf{g}'\right\|_1 > cd_i(\ell')^{-1}\left(\frac{\gamma(1-\rho)\kappa}{k(1+\gamma)}\right).$$

□

The only step left to prove Theorem 5.3.1 is that at the beginning, we have $\left\|\mathbf{g}\right\|_1 = \mathrm{vol}(S)$. Then the theorem follows by Lemma 5.3.3.

The runtime bound when 3(b) holds is slightly different, see below.

Theorem 5.3.4. *Let $\gamma > 0, \kappa > 0$ be fixed and let k and c be the parameters from 3(b) of Definition 5.2.1 for $\ell(x)$. For $0 < \rho < 1$, suppose* **`nonlin-cut`** *stops after T iterations, and d_i is the degree of node updated at the i-th iteration, then T must satisfy: $\sum_{i=1}^{T} d_i \leq vol(S)/k\ell'\left(\gamma(1-\rho)\kappa/c(1+\gamma)\right) = O(vol(S))$.*

Note that in 3(b) of Definition 5.2.1, there is an extra strictly increasing condition so that $\ell'(\frac{\gamma(1-\rho)\kappa}{c(1+\gamma)})$ is positive. When ℓ' is not strictly increasing, i.e. $\ell'(x) = 0$ in a small range round 0, it is our conjecture that the algorithm will still finish in a strongly local time, although we have not yet proven that. Note that this strictly increasing criteria is true for all the loss

functions used in the experiments. Similar to the case of 3(a), we have the following lemma to lower bound the decrease of $\|\mathbf{g}\|_1$ after each `nonlin-push`.

Lemma 5.3.5. *When 3(b) is satisfied and $\ell'(x)$ is strictly increasing, then after calling `nonlin-push` on node i, the decrease of $\|\mathbf{g}\|_1$ will be strictly larger than*

$$kd_i\ell'\left(\frac{\gamma(1-\rho)\kappa}{c(1+\gamma)}\right)$$

Proof. Similarly to the proof of lemma 5.3.3, after calling `nonlin-push` on node i,

$$\begin{aligned}\|\mathbf{g}\|_1 - \|\mathbf{g}'\|_1 &= d_i\ell'(x_i + \Delta x_i - (\mathbf{e}_S)_i) - d_i\ell'(x_i - (\mathbf{e}_S)_i) \\ &\geq kd_i\ell'(\Delta x_i)\end{aligned}$$

On the other hand,

$$\begin{aligned}g_i' &= \rho\kappa d_i \\ &= -\frac{1}{\gamma}\sum_{j\sim i} w(i,j)\ell'(x_i + \Delta x_i - x_j) - d_i\ell'(x_i + \Delta x_i - (\mathbf{e}_S)_i) \\ &\geq -\frac{1}{\gamma}\sum_{j\sim i} w(i,j)\ell'(x_i - x_j) - d_i\ell'(x_i - (\mathbf{e}_S)_i) - \frac{c(1+\gamma)}{\gamma}d_i\Delta x_i \\ &= g_i - \frac{c(1+\gamma)}{\gamma}d_i\Delta x_i\end{aligned}$$

Thus, we have

$$\Delta x_i \geq \frac{\gamma(r_i - \rho\kappa d_i)}{c(1+\gamma)d_i} > \frac{\gamma(1-\rho)\kappa}{c(1+\gamma)}$$

which means

$$\|\mathbf{g}\|_1 - \|\mathbf{g}'\|_1 > kd_i\ell'\left(\frac{\gamma(1-\rho)\kappa}{c(1+\gamma)}\right).$$

□

Lemma 5.3.5 along with the same type of analysis as before give the following result when 3(b) is satisfied.

5.4 More details on ρ

When $\rho < 1$, then we only approximately satisfy the KKT conditions. Here, we do some quick analysis of the difference in the idealized slackness condition $\mathbf{k}^T\mathbf{x} = 0$ compared to what we get from our solver. Note that by choosing ρ close to 1, we do produce a fairly accurate solution when 3(a) is satisfied.

Lemma 5.4.1. *When Algorithm 4 returns, if $\ell(x)$ satisfies 3(a) we have*

$$\mathbf{k}^T\mathbf{x} \leq \frac{\kappa k \ell'(1)(1-\rho)vol(S)}{c}$$

Proof. We know $\mathbf{k} = [0, \kappa\mathbf{d} - \mathbf{r}, 0]^T$. Every time algorithm 3 is called at node i, it will set $g_i = \rho\kappa d_i$. In the following iterations, g_i can only increase until algorithm 3 is called at node i again. This means $\mathbf{k} \leq (1-\rho)\kappa\mathbf{d}$.

On the other hand, when 3(a) is satisfied, $\ell'(1-x_i) \leq -\ell'(x_i) + k\ell'(1)$

$$\|\mathbf{g}\|_1 = -\sum_{i\notin S} d_i\ell'(x_i) - \sum_{i\in S} d_i\ell'(x_i - 1) \leq -\sum_{i\in V} d_i\ell'(x_i) + k\ell'(1)\mathrm{vol}(S) \leq -c\mathbf{d}^T\mathbf{x} + k\ell'(1)\mathrm{vol}(S)$$

Thus

$$\mathbf{d}^T\mathbf{x} \leq \frac{k\ell'(1)}{c}\mathrm{vol}(S)$$

Combining the two inequality gives this lemma. □

When 3(b) is satisfied, it is easy to see $\mathbf{k}^T\mathbf{x} \leq (1-\rho)\kappa\mathbf{d}^T\mathbf{x}$, however, there isn't a closed form equation on the upper bound of $\mathbf{k}^T\mathbf{x}$ in terms of $\mathrm{vol}(S)$.

5.5 Cut Quality Analysis

A common use for the results of these localized cut solutions is as *localized Fiedler* vectors of a graph to induce a cluster [24, 30, 34, 60, 63]. And a real-valued "clustering hint" vector

$\mathbf{x}$ can be converted into clusters by a sweep cut process (see Chapter 2 for a definition). This computation is a key piece of Cheeger inequalities [64, 65].

In the following, we seek a slightly different type of guarantee. We posit the existence of a target cluster T and show that *if* T has useful clustering properties (small conductance, no good internal clusters), then a sweep cut over a q-norm or q-Huber localized cut vector seeded inside of T will accurately recover T. The key piece is understanding how the computation plays out with respect to T inside the graph and T as a graph by itself. We use $\mathrm{vol}_T(S)$, $\phi_T(S)$ to be the volume or conductance of set S in the subgraph induced by T and $\partial T \subset T$ to be the boundary set of T, i.e. nodes in ∂T has at least one edge connecting to $\bar{T}$. Quantities with tildes, e.g., $\tilde{d}$, reflect quantities in the subgraph induced by T. In the rest of this section, we assume $\kappa = 0$ and $\rho = 1$.

To begin with, we give the following observation. It is not directly related to the proof of main theorem, but we still find it useful in understanding the problem in general.

Lemma 5.5.1. *Suppose that $\kappa = 0$. When $\ell(x) = \frac{1}{p}|x|^p$, $1 < p < 2$, we can compute the exact solution of problem (3.4) under two extreme cases $\gamma \to \infty$ and $\gamma \to 0$,*

- *When $\gamma \to \infty$, $x_i = 1$ for $i \in S$ and $x_i = 0$ for $i \in \bar{S}$.*
- *When $\gamma \to 0$, $x_i \geq \frac{(vol(S))^{\frac{1}{p-1}}}{(vol(V))^{\frac{1}{p-1}}}$ for any $i \in V$.*

Proof. Considering two `nonlin-cut` processes P_1, P_2 using S_1 or S_2 as input correspondingly, suppose we set the initial vector of P_2 to be the solution of P_1, i.e. $\mathbf{x}_1$, then for nodes $i \notin S_2 \backslash S_1$, its residual stays zero, while for nodes $i \in S_2 \backslash S_1$, its residual becomes positive. This means P_2 needs more iterations to converge. And each iteration can only add nonnegative values to $\mathbf{x}_1$. Thus, $\mathbf{x}_1 \leq \mathbf{x}_2$. □

What this lemma suggests is that if the seed nodes are sampled from the target cluster, then for large γ, the solution stays mostly within the target set. But we would also like γ not to be too large, so that the value can be "mixed well" over all nodes. Otherwise, we will end up spreading almost no information which will result in a low recall. Formally, we will need the following two assumptions.

Assumption 1. *The seed set S satisfies $S \subseteq T$, $S \cap \partial T = \varnothing$ and $\sum_{i \in \partial T} (d_i - \tilde{d}_i) x_i^{q-1} \leq 2\phi(T) vol(S)$.*

We call this the leaking assumption, which roughly states that the solution with the set S stays mostly within the set T. As some quick justification for this assumption, we note that when when $q = 2$, [34] shows by a Markov bound that there exists T_g where $\mathrm{vol}(T_g) \geq \frac{1}{2}\mathrm{vol}(T)$ such that any node $i \in T_g$ satisfies $\sum_{i \in \partial T} (d_i - \tilde{d}_i) x_i \leq 2\phi(T) d_i$. So in that case, any seed sets $S \subseteq T_g$ meets our assumption. For $1 < q < 2$, it is straightforward to see any set S with $\mathrm{vol}(S) \geq \frac{1}{2}\mathrm{vol}(T)$ satisfies this assumption since the left hand side is always smaller than $\mathrm{cut}(T)$. However, such a strong assumption is not necessary for our approach. The above guarantee allows for a small $\mathrm{vol}(S)$ and we simply require Assumption 1 holds. We currently lack a detailed analysis of how many such seed sets there will be.

Our second assumption regards the behavior within only the set T compared with the entire graph. To state it, we wish to be precise. Consider the localized cut graph associated with the hidden target set T on the entire graph and let $\boldsymbol{B}, \mathbf{w}$ be the incidence and weights for this graph. We wish to understand how the solution $\mathbf{x}$ on this problem

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \mathbf{w}^T \ell(\boldsymbol{B}\mathbf{x}) \\ \text{subject to} & x_s = 1, x_t = 0, \mathbf{x} \geq 0 \end{array} \tag{5.3}$$

compares with one where we consider the problem *only* on the subgraph induced by T. Let $\tilde{\boldsymbol{B}}, \tilde{\mathbf{w}}$ be the incidence matrix of the localized cut graph on the vertex induced subgraph corresponding to T and seeded on T (so the tilde-problem is seeded on all nodes). So formally, we wish to understand how $\tilde{\mathbf{x}}$ in

$$\begin{array}{ll} \underset{\tilde{\mathbf{x}}}{\text{minimize}} & \tilde{\mathbf{w}}^T \ell(\tilde{\boldsymbol{B}}\tilde{\mathbf{x}}) \\ \text{subject to} & \tilde{x}_s = 1, \tilde{x}_t = 0, \tilde{\mathbf{x}} \geq 0 \end{array} \tag{5.4}$$

compares to $\mathbf{x}$. For these comparisons, we assume we are looking at values other than x_s, x_t and $\tilde{x}_s, \tilde{x}_t$.

Assumption 2. *A relatively small γ should be chosen such that the solution of localized q-norm cut problem in the subgraph induced by target cluster T can satisfy $min(\tilde{\mathbf{x}}_T) \geq \frac{(0.5vol_T(S))^{1/(q-1)}}{(vol_T(T))^{1/(q-1)}} = M$.*

The second part of Lemma 5.5.1 guarantees the existence of such γ. To better understand this assumption, when $\ell(x) = \frac{1}{q}|x|^q$ and $q = 2$, a solution of the `nonlin-cut` process (Algorithm 4) will be equivalent to a Markov process. In this case, one can lower bound $\min(\tilde{\mathbf{x}})$ by the well known infinity-norm mixing time of Markov chain. In fact, as shown in the proof of lemma 3.2 of [34], when $\gamma \leq O(\phi(T) \cdot \text{Gap})$, they show that $\min(\tilde{\mathbf{x}}_T) \geq \frac{0.8\text{vol}_T(S)}{\text{vol}_T(T)}$. Here Gap is defined as the ratio of internal connectivity and external connectivity and often assumed to be $\Omega(1)$. Formally:

Definition 5.5.1. *Given a target cluster T such that $vol(T) \leq \frac{1}{2}vol(V)$, $\phi(T) \leq \Psi$ and $min_{A \subset T}\phi_T(A) \geq \Phi$, the Gap is defined as:*

$$Gap = \frac{\Phi^2/logvol(T)}{\Psi}$$

[1] We refer to [34] for a detailed explanation of this. In the case of $q = 2$, by using the infinity-norm mixing time of a Markov chain, any $\gamma \leq O(\phi(T) \cdot \text{Gap})$ satisfies this assumption as shown in lemma 3.2 of [34]. For $1 < q < 2$, it will be more difficult to derive a closed form solution on how small γ needs to be. However, we can show that this assumption still holds for subgraphs with small diameters, i.e. $O(\log(|T|))$ (This is reasonable because we expect good clusters and good communities to have small diameters.).

Lemma 5.5.2. *Assume the subgraph induced by target cluster T has diameter $O(\log(|T|))$ and when we uniformly randomly sample points from T as seed sets, the expected largest*

[1]↑The proof of lemma 3.2 in [34] proves that the teleportation probability $\beta = 1 - \alpha$ needs to be smaller than $O(\phi(T) \cdot \text{Gap})$. When $q = 2$, as shown in [12], $\beta = \frac{\gamma_2}{1+\gamma_2}$, which means $\gamma_2 = \frac{\beta}{1-\beta}$. Since we assume $\gamma_2 < 1$, we have $\beta < \gamma_2 < 2\beta$. In other words, γ_2 and β are only different by a constant factor.

distance of any node in $\bar{S}$ *to* S *is* $O\left(\frac{\log(|T|)}{|S|}\right)$. *Also define* γ_2 *to be the largest* γ *such that assumption 2 is satisfied at* $q = 2$ *and assume* $\gamma_2 < 1$, *if we set* $\gamma = \gamma_2^{q-1}$ *for* $1 < q < 2$, *and*

$$\frac{vol_T(S)}{vol_T(T)} \leq 2\left(\frac{\gamma_2}{1+\gamma_2} \cdot \frac{1}{|T|^{\frac{1}{|S|}\log\left(1+l^{\frac{1}{q-1}}\right)}}\right)^{q-1}$$

where $l \leq (1+\gamma)max(\tilde{d}_i)$. *Then the solution of 5.4 can satisfy assumption 2.*

Proof. Given a seed set S, we can partition the $\bar{S}$ into disjoint subsets $L_1 \cup L_2 \cup L_3 \ldots \cup L_n$, where L_i contains nodes that are i distance away from S. For any node $i \in L_k$, we denote d_i^{out} to be

$$d_i^{out} = \sum_{j\sim i, j\in L_k \cup L_{k+1}} w(i,j)$$

And $d_i^{in} = \tilde{d}_i - d_i^{out}$. Also define $l = (1+\gamma)\frac{d_i^{out}}{d_i^{in}} \leq (1+\gamma)\max(\tilde{d}_i)$. Suppose $\tilde{x}_i \geq c$ for any node i with distance at most $k-1$, then we can show for node $i \in L_k$, $\tilde{x}_i \geq \frac{c}{1+l^{\frac{1}{q-1}}}$. To see this, if $\tilde{x}_i < c$, then by the KKT condition,

$$d_i^{in}(c - \tilde{x}_i)^{q-1} \leq d_i^{out} x_i^{q-1} + \gamma d_i x_i^{q-1}$$

Here for $j \sim i$, if j is closer to S, we set $\tilde{x}_j$ to be c, otherwise, we set $\tilde{x}_j$ to be 0. This means

$$\tilde{x}_i \geq \frac{c(d_i^{in})^{\frac{1}{q-1}}}{(d_i^{out} + \gamma d_i)^{\frac{1}{q-1}} + (d_i^{in})^{\frac{1}{q-1}}} \geq \frac{c}{l^{\frac{1}{q-1}} + 1}$$

Also, for node $i \in S$, the first iteration of q-norm process will add at least $\frac{\gamma^{\frac{1}{q-1}}}{1+\gamma^{\frac{1}{q-1}}}$ to $\tilde{x}_i$ (This follows from unrolling the first loop of our algorithm and checking that this satisfies the binary search criteria.), which means $\tilde{x}_i \geq \frac{\gamma^{\frac{1}{q-1}}}{1+\gamma^{\frac{1}{q-1}}}$. Thus, for node $i \in L_k$,

$$\tilde{x}_i \geq \frac{\gamma^{\frac{1}{q-1}}}{1+\gamma^{\frac{1}{q-1}}} \cdot \frac{1}{\left(1+l^{\frac{1}{q-1}}\right)^k} = \frac{\gamma_2}{1+\gamma_2} \cdot \frac{1}{\left(1+l^{\frac{1}{q-1}}\right)^k}$$

Since the subgraph induced by target cluster T has diameter $O(\log(|T|))$ and when we uniformly randomly sample points from T as seed sets, the expected largest distance r of any node in $\bar{S}$ to S is $O\left(\frac{\log(|T|)}{|S|}\right)$, we have $r = O\left(\frac{\log(|T|)}{|S|}\right)$, which means

$$\min(\tilde{\mathbf{x}}) \geq \frac{\gamma_2}{1+\gamma_2} \cdot \frac{1}{|T|^{\frac{1}{|S|}\log\left(1+l^{\frac{1}{q-1}}\right)}}$$

Assumption 2 requires $\min(\tilde{\mathbf{x}}) \geq \frac{(0.5\mathrm{vol}_T(S))^{\frac{1}{q-1}}}{(\mathrm{vol}_T(T))^{\frac{1}{q-1}}}$. So we just need

$$\frac{\mathrm{vol}_T(S)}{\mathrm{vol}_T(T)} \leq 2\left(\frac{\gamma_2}{1+\gamma_2} \cdot \frac{1}{|T|^{\frac{1}{|S|}\log\left(1+l^{\frac{1}{q-1}}\right)}}\right)^{q-1},$$

which was the final assumption. □

Lemma 5.5.3. *Under the previous assumptions, we can now define a sweep cut set S_c as $\left\{i \in V \mid x_i \geq \frac{c(0.5vol(S))^{\frac{1}{q-1}}}{(vol(T))^{\frac{1}{q-1}}}\right\}$, then for any $0 < c \leq \frac{1}{2}$,*

$$vol(S_c \backslash T) = O\left(\frac{\phi(T)}{\gamma c^{q-1}}\right) vol(T) \qquad vol(T \backslash S_c) = O\left(\frac{\phi(T)}{\gamma}\right) vol(T)$$

Proof. The proof is mostly a generalization to the proof of Lemma 3.4 in [34]. For any $i \in \bar{T}$, by the KKT condition and Assumption 1

$$\begin{aligned}
0 &= r_i(\mathbf{x}) \\
&= -\frac{1}{\gamma}\sum_{j\sim i} w(i,j)\ell'(x_i - x_j) - d_i x_i^{q-1} \\
&= -\frac{1}{\gamma}\sum_{j\sim i, j\in\bar{T}} w(i,j)\ell'(x_i - x_j) - \frac{1}{\gamma}\sum_{j\sim i, j\in T} w(i,j)\ell'(x_i - x_j) - d_i x_i^{q-1} \\
&= -\frac{1}{\gamma}\sum_{j\sim i, j\in\bar{T}} w(i,j)\ell'(x_i - x_j) + \frac{1}{\gamma}\sum_{j\sim i, j\in T} w(i,j)\ell'(x_j - x_i) - d_i x_i^{q-1} \\
&< -\frac{1}{\gamma}\sum_{j\sim i, j\in\bar{T}} w(i,j)\ell'(x_i - x_j) + \frac{1}{\gamma}\sum_{j\sim i, j\in T} w(i,j)\ell'(x_j) - d_i x_i^{q-1}.
\end{aligned}$$

By summing the inequality above over all nodes in $\bar{T}$, the first term will all cancel out, it yields that

$$\sum_{i\in\bar{T}} d_i x_i^{q-1} < \frac{1}{\gamma}\sum_{i\in\partial T}(d_i - \tilde{d}_i)x_i^{q-1} \le \frac{2\phi(T)\mathrm{vol}(S)}{\gamma}.$$

Now by the definition of our sweep cut set, we know that for $i \in S_c\backslash T$, $x_i^{q-1} \ge \frac{c^{q-1}u\mathrm{vol}(S)}{\mathrm{vol}(T)}$, thus

$$\frac{c^{q-1}\mathrm{vol}(S)}{2\mathrm{vol}(T)}\mathrm{vol}(S_c\backslash T) \le \sum_{i\in S_c\backslash T} d_i x_i^{q-1} \le \frac{2\phi(T)\mathrm{vol}(S)}{\gamma}$$

which means

$$\mathrm{vol}(S_c\backslash T) = O\left(\frac{\phi(T)}{\gamma c^{q-1}}\right)\mathrm{vol}(T).$$

In the following, we define $x_i = \tilde{x}_i + v_i$ and $\ell'(x_i - (\mathbf{e}_S)_i) = \ell'(\tilde{x}_i - (\mathbf{e}_S)_i) + k_i\ell'(v_i)$. For any node $i \in T$, by KKT condition,

$$\begin{aligned}
0 &= r_i(\mathbf{x}) \\
&= -\frac{1}{\gamma}\sum_{j\sim i} w(i,j)\ell'(x_i - x_j) - d_i\ell'(x_i - (\mathbf{e}_S)_i) \\
&= -\frac{1}{\gamma}\sum_{j\sim i, j\in T} w(i,j)\ell'(x_i - x_j) - \frac{1}{\gamma}\sum_{j\sim i, j\in\bar{T}} w(i,j)\ell'(x_i - x_j) - d_i\ell'(x_i - (\mathbf{e}_S)_i) \\
&> -\frac{1}{\gamma}\sum_{j\sim i, j\in T} w(i,j)\ell'(x_i - x_j) - \frac{1}{\gamma}\sum_{j\sim i, j\in\bar{T}} w(i,j)\ell'(x_i) - \tilde{d}_i\ell'(x_i - (\mathbf{e}_S)_i) - (d_i - \tilde{d}_i)\ell'(x_i) \\
&= -\frac{1}{\gamma}\sum_{j\sim i, j\in T} w(i,j)\ell'(x_i - x_j) - \tilde{d}_i\ell'(\tilde{x}_i - (\mathbf{e}_S)_i) - k_i d_i\ell'(v_i) - (1+\frac{1}{\gamma})(d_i - \tilde{d}_i)\ell'(x_i) \\
&= -\frac{1}{\gamma}\sum_{j\sim i, j\in T} w(i,j)\ell'(x_i - x_j) - \\
&\qquad \frac{1}{\gamma}\sum_{j\sim i, j\in T} w(i,j)\ell'(\tilde{x}_i - \tilde{x}_j) - k_i d_i\ell'(v_i) - (1+\frac{1}{\gamma})(d_i - \tilde{d}_i)\ell'(x_i).
\end{aligned}$$

By summing the inequality above over all nodes in T, the first and the second terms cancel out, so it yields:

$$\sum_{i\in T} k_i d_i \ell'(v_i) > -\frac{2(1+\gamma)}{\gamma}\phi(T)\mathrm{vol}(S).$$

For nodes $i \in T \backslash S_c$, $x_i < c\tilde{x}_i$, which means $v_i < (c-1)\tilde{x}_i$. And $\ell'(v_i) = -(-v_i)^{q-1} < -(1-c)^{q-1}\frac{0.5\mathrm{vol}_T(S)}{\mathrm{vol}_T(T)} \leq -(1-c)^{q-1}\frac{0.5\mathrm{vol}(S)}{\mathrm{vol}(T)}$. (Here we use the fact that $\mathrm{vol}_T(T) \leq \mathrm{vol}(T)$ and $S \cap \partial T = \emptyset$). From the proof of lemma 5.5.2, we know that S will be included in S_c. When $i \notin S$,

$$k_i = \left(-\frac{\tilde{x}_i}{v_i} + 1\right)^{q-1} - \left(-\frac{\tilde{x}_i}{v_i}\right)^{q-1} > \frac{(2-c)^{q-1} - 1}{(1-c)^{q-1}}.$$

Thus, we have

$$\mathrm{vol}(T \backslash S_c) = O\left(\frac{\phi(T)}{\gamma}\right)\mathrm{vol}(T).$$

□

Lemma 5.5.4. *Under the same assumptions as lemma 5.5.3, among sweep cut sets $S_c \in \{S_c \mid \frac{1}{4} \leq c \leq \frac{1}{2}\}$, there exsits one R such that $\phi(R) = O\left(\frac{\phi(T)^{\frac{1}{q}}}{Gap^{\frac{q-1}{2}}}\right)$.*

Proof. Our proof is mostly a generalization to the proof of Lemma 4.1 in [34]. If $\mathrm{cut}(S_c, \bar{S}_c) \geq E_0$ holds for all $\frac{1}{4} \leq c \leq \frac{1}{2}$, then we just need to upper bound E_0.

We introduce values $k(i,j)$ that allow us to break $\ell'(x_i - x_j)$ into $\ell'(x_i) - k(i,j)\ell'(x_j)$. The specific choice $k(i,j) > 0$ is uniquely determined by x_i and x_j. For any node $i \in S_c$, by KKT condition,

$$\begin{aligned} 0 &= \frac{1}{\gamma}\sum_{j\sim i} w(i,j)\ell'(x_i - x_j) + d_i\ell'(x_i - (\mathbf{e}_S)_i) \\ &= \frac{1}{\gamma}\sum_{j\sim i}(w(i,j)\ell'(x_i) - w(i,j)k(i,j)\ell'(x_j)) + d_i\ell'(x_i) - k_i d_i(\mathbf{e}_S)_i. \end{aligned}$$

Define $\boldsymbol{K}$ to be the matrix induced by $k(i,j)$. Rearranging the equation above yields:

$$(\boldsymbol{K} \circ \boldsymbol{A}\mathbf{x}^{q-1})_i = (1+\gamma)d_i x_i^{q-1} - \gamma k_i d_i(\mathbf{e}_S)_i.$$

Also for two adjacent nodes i, j that are both in S_c, we have

$$k(i,j)\ell'(x_j) + k(j,i)\ell'(x_i) = \ell'(x_i) + \ell'(x_j).$$

This is because $\ell'(x_i - x_j) + \ell'(x_j - x_i) = 0$. And for two adjacent nodes i, j such that $i \in S_c$ and $j \notin S_c$, $x_i > x_j$, $k(i,j) < 1$. Define a Lovasz-Simonovits curve y over $d_i x_i^{q-1}$, then we have

$$\begin{aligned}
\sum_{i \in S_c} (\boldsymbol{K} \circ \boldsymbol{A}\mathbf{x}^{q-1})_i + \sum_{i \in S_c} d_i x_i^{q-1} &= 2 \sum_{i \in S_c} \sum_{j \sim i, j \in S_c} w(i,j) x_j^{q-1} + \sum_{i \in S_c} \sum_{j \sim i, j \notin S_c} k(i,j) w(i,j) x_j^{q-1} \\
&< 2 \sum_{i \in S_c} \sum_{j \sim i, j \in S_c} w(i,j) x_j^{q-1} + \sum_{i \in S_c} \sum_{j \sim i, j \notin S_c} w(i,j) x_j^{q-1} \\
&\leq y[\mathrm{vol}(S) - \mathrm{cut}(S_c, \bar{S}_c)] + y[\mathrm{vol}(S) + \mathrm{cut}(S_c, \bar{S}_c)] \\
&\leq y[\mathrm{vol}(S) - E_0] + y[\mathrm{vol}(S) + E_0]
\end{aligned}$$

here the second inequality is due to the definition of Lovasz-Simonovits curve and the third inequality is due to $y(x)$ is concave. This means

$$\begin{aligned}
y[\mathrm{vol}(S) - E_0] + y[\mathrm{vol}(S) + E_0] &\geq \sum_{i \in S_c} (\boldsymbol{K} \circ \boldsymbol{A}\mathbf{x}^{q-1})_i + \sum_{i \in S_c} d_i x_i^{q-1} \\
&\geq (2 + \gamma) \sum_{i \in S_c} d_i x_i^{q-1} - \gamma \sum_{i \in S_c} k_i d_i (\mathbf{e}_S)_i \\
&\geq (2 + \gamma) \sum_{i \in S_c} d_i x_i^{q-1} - \gamma \sum_{i \in S} k_i d_i \\
&= (2 + \gamma) \sum_{i \in S_c} d_i x_i^{q-1} - \gamma \sum_{i \in V} d_i x_i^{q-1} \\
&= 2 \sum_{i \in S_c} d_i x_i^{q-1} - \gamma \sum_{i \notin S_c} d_i x_i^{q-1} \\
&\geq 2y[\mathrm{vol}(S_c)] - O(\phi(T)\mathrm{vol}(S)).
\end{aligned}$$

Thus,

$$y[\mathrm{vol}(S_c)] - y[\mathrm{vol}(S_c - E_0)] \leq y[\mathrm{vol}(S_c + E_0)] - y[\mathrm{vol}(S_c)] + O(\phi(T)\mathrm{vol}(S)).$$

Similarly to the proof of Lemma 4.1 in [34], we can then derive

$$\begin{aligned}\frac{0.5E_0\text{vol}(S)}{4^{q-1}\text{vol}(T)} &\leq y[\text{vol}(S_{1/4})] - y[\text{vol}(S_{1/4}) - E_0] \\ &\leq \frac{\text{vol}(S_{1/8}\backslash S_{1/4})}{E_0}O(\phi(T)\text{vol}(S)) + y[\text{vol}(S_{1/8})] - y[\text{vol}(S_{1/8}) - E_0] \\ &\leq \frac{\text{vol}(S_{1/8}\backslash T) + \text{vol}(T\backslash S_{1/4})}{E_0}O(\phi(T)\text{vol}(S)) + \frac{0.5E_0\text{vol}(S)}{8^{q-1}\text{vol}(T)} \\ &\leq \frac{O(\phi(T)/\gamma)\text{vol}(T)}{E_0}O(\phi(T)\text{vol}(S)) + \frac{0.5E_0\text{vol}(S)}{8^{q-1}\text{vol}(T)}.\end{aligned}$$

$$\text{Hence, } E_0 \leq O\left(\frac{\phi(T)}{\sqrt{\gamma}}\right)\text{vol}(T).$$

And from lemma 5.5.3, we know $\text{vol}(S_c) = 1 \pm O\left(\frac{\phi(T)}{\gamma}\right)\text{vol(T)}$, since we choose $\gamma = (\gamma_2)^{q-1}$ and $\gamma_2 = \Theta(\phi(T)\cdot\text{Gap})$, $\text{vol}(S_c) = \Theta(\text{vol}(T))$. So there exists R such that

$$\phi(R) = O\left(\frac{\phi(T)}{\sqrt{\gamma}}\right) = O\left(\frac{\phi(T)^{\frac{3-q}{2}}}{\text{Gap}^{(q-1)/2}}\right) \leq O\left(\frac{\phi(T)^{\frac{1}{q}}}{\text{Gap}^{(q-1)/2}}\right).$$

Here the last inequality uses the fact that $(3-q)/2 > 1/q$ when $1 < q < 2$. □

Combining all these lemmas will give us the following theorem.

Theorem 5.5.5. *Assume the subgraph induced by target cluster T has diameter $O(\log(|T|))$, when we uniformly randomly sample points from T as seed sets, the expected largest distance of any node in $\bar{S}$ to S is $O\left(\frac{\log(|T|)}{|S|}\right)$. Assume $\frac{vol_T(S)}{vol_T(T)} \leq 2\big((\frac{\gamma_2}{1+\gamma_2})/\,|T|^{\frac{1}{|S|}\log\left(1+l^{1/(q-1)}\right)}\big)^{q-1}$ where $l \leq (1+\gamma)max(\tilde{d}_i)$, then we can set $\gamma = \gamma_2^{q-1}$ to satisfy assumption 2 for $1 < q < 2$. Then a sweep cut over $\mathbf{x}$ will find a cluster R where $\phi(R) = O\big(\phi(T)^{\frac{1}{q}}/Gap^{\frac{q-1}{2}}\big)$.*

5.6 Experimental Results

In the experiments section, we will mainly show that our algorithm (i) outperforms flow-based methods especially when the seed set is small (ii) outperforms PageRank-based methods by producing higher accuracy (iii) outperforms other nonlinear methods by running much faster and providing more intuitive ways to tune parameters.

We perform three experiments that are designed to compare our method to others designed for similar problems. We call ours SLQ (strongly local q-norm) for $\ell(x) = (1/q)\,|x|^q$ with parameters γ for localization and κ for the sparsity. We call it SLQδ with the q-Huber loss. Existing solvers are (i) ACL [24], that computes a personalized PageRank vector approximately adapted with the same parameters [12]; (ii) CRD [8], which is hybrid of flow and spectral ideas; (iii) FS is FlowSeed [66], a 1-norm based method; (iv) HK is the push-based heat kernel [26]; (v) NLD is a recent nonlinear diffusion [7]; (vi) GCN is a graph convolutional network [67]. Parameters are chosen based on defaults or with slight variations designed to enhance the performance within a reasonable running time. All experiments in this section are performed on a server with Intel Xeon Platinum 8168 CPU and 5.9T RAM. (Nothing remotely used the full capacity of the system and these were run concurrently with other processes.) We evaluate the routines in terms of their recovery performance for planted sets and clusters. The bands reflect randomizing seeds choices in the target cluster.

The first experiment uses the LFR benchmark [68]. We vary the mixing parameter μ (where larger μ is more difficult) and provide 1% of a cluster as a seed, then we check how much of the cluster we recover after a conductance-based sweep cut over the solutions from various methods. Here, we use the $F1$ score (harmonic mean of precision and recall) and conductance value (cut to volume ratio) of the sets to evaluate the methods. The results are in Figure 5.3.

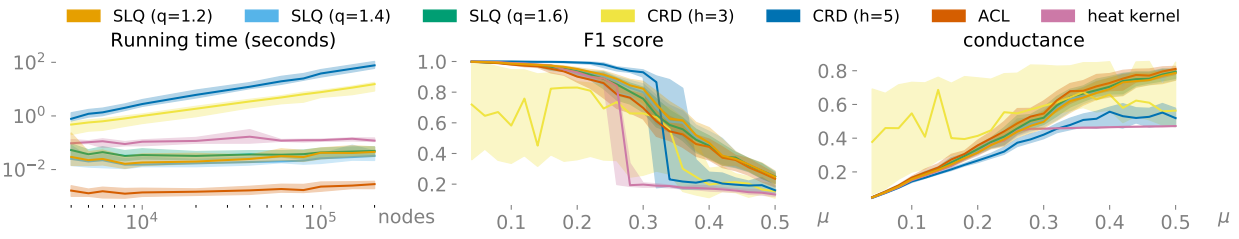


Figure 5.3. The left figure shows the median running time for the methods as we scale the graph size keeping the cluster sizes roughly the same. As we vary cluster mixing μ for a graph with $10,000$ nodes, the middle figure shows the median F1 score (higher is better) along with the 20-80% quantiles; the right figure shows the conductance values (lower is better). These results show SLQ is better than ACL and competitive with CRD while running much faster.

The second experiment uses the class-year metadata on Facebook [69], which is known to have good conductance structure for at least class year 2009 [70] that should be identifiable with many methods. The class year 2009 is the set of incoming students, which form better conductance groups because the students had not yet mixed with the other classes. Class year 2008 is already mixed and so the methods do not do as well there. Here, we use $F1$ values alone. We use 1% of the true set as seed. (For GCN, we also use the same number of negative nodes.) The results are in Table 5.1,5.2 and show SLQ is as good, or better than, CRD and much faster.

Table 5.1. Cluster recovery results from a set of 7 Facebook networks [69]. Students with a specific graduation class year are used as target cluster. We use a random set of 1% of the nodes identified with that class year as seeds. The class year 2009 is the set of incoming students, which form better conductance groups because the students had not yet mixed with the other classes. Class year 2008 is already mixed and so the methods do not do as well there. The values are median $F1$ and the violin plots show the distribution over choices of the seeds.

Year	Alg	UCLA F1 & Med.	MIT F1 & Med.	Duke F1 & Med.	UPenn F1 & Med.	Yale F1 & Med.	Cornell F1 & Med.	Stanford F1 & Med.
2009	SLQ	0.9	0.9	1.0	1.0	1.0	0.9	0.9
	SLQδ	0.9	0.8	1.0	0.9	0.9	0.9	0.9
	CRD-3	0.3	0.7	0.7	0.6	0.7	0.5	0.5
	CRD-5	0.9	0.9	1.0	1.0	1.0	0.9	0.9
	ACL	0.9	0.8	0.9	0.9	0.9	0.9	0.9
	FS	0.4	0.4	0.9	0.9	0.5	0.5	0.4
	HK	0.9	0.5	0.9	0.9	0.9	0.9	0.9
	NLD	0.2	0.2	0.3	0.3	0.3	0.3	0.3
	GCN	0.3	0.2	0.3	0.3	0.2	0.3	0.2
2008	SLQ	0.7	0.5	0.8	0.8	0.8	0.8	0.8
	SLQδ	0.6	0.5	0.7	0.7	0.7	0.7	0.7
	CRD-3	0.6	0.5	0.7	0.7	0.7	0.6	0.6
	CRD-5	0.5	0.5	0.5	0.5	0.7	0.6	0.5
	ACL	0.5	0.5	0.7	0.7	0.7	0.7	0.7
	FS	0.5	0.5	0.7	0.6	0.7	0.6	0.7
	HK	0.5	0.5	0.0	0.5	0.5	0.5	0.5
	NLD	0.3	0.3	0.3	0.3	0.3	0.3	0.2
	GCN	0.3	0.3	0.3	0.3	0.3	0.3	0.3

Table 5.2. Total running time of methods in this experiment.

Method	SLQ	SLQδ	CRD-3	CRD-5	ACL	FS	HK	NLD	GCN
Time (seconds)	123	80	3049	9378	12	1593	106	10375	16534

The final experiment evaluates a finding from [71] on the recall of seed-based community detection methods. For a group of communities with roughly the same size, we evaluate the recall of the largest k entries in a diffusion vector. Minimizing conductance is not an objective in this experiment. They found PageRank (ACL) outperformed many different methods. Also, ACL – with the standard degree normalization for conductance based sweepcuts performed worse than ACL without degree normalization in this particular setting, which is different from what conductance theory suggests. Here, with the flexibility of q, we see the same general result with respect to degree normalization and found that SLQ with $q > 2$ gives the best performance even though the conductance theory suggests $1 < q < 2$ for the best conductance bounds. (See Figure 5.4)

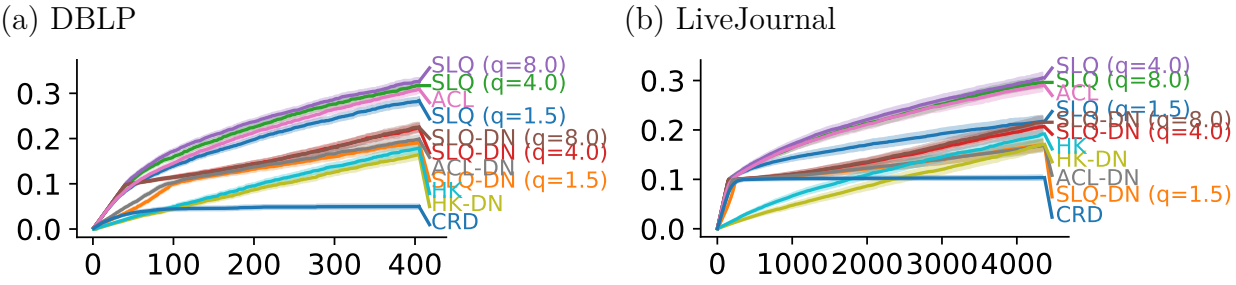


Figure 5.4. A replication of an experiment from [71] with SLQ on DBLP [72, 73] (with 1M edges) and edges LiveJournal [74] (with 65M edges). The plot shows median recall over 600 groups of roughly the same size as we look at the top k entries in the solution vector (x axis). The envelope represents 2 standard error. This shows SLQ with $q > 2$ gives better performance than ACL (PageRank), and all improve on the degree-normalized (DN) versions used for conductance-minimizing sweep cuts.

6. STRONGLY LOCAL HYPERGRAPH DIFFUSIONS FOR CLUSTERING AND SEMI-SUPERVISED LEARNING

6.1 Chapter Overview and Motivation

In this chapter, we will further extend our generalized local graph cut framework to hypergraphs. The results and more details from this chapter can also be found in our paper in our paper [20].

Hypergraphs, indeed, enable a flexible and rich data model that has the potential to capture subtle insights that are difficult or impossible to find with traditional graph-based analysis [23, 62, 75–77]. But, hypergraph generalizations of graph-based algorithms often struggle with scalability and interpretation [78, 79] with ongoing questions of whether particular models *capture* the higher-order information in hypergraphs. Existing techniques that rely on simple clique/star expansion or Lovász extension either only model one type of cut function or are not strongly local. [76, 77, 80–82]

To solve these problems, we will introduce, to our knowledge, the first local hypergraph clustering algorithm that includes all of the following features: it is (1) strongly-local, (2) can grow a cluster from a small seed set, (3) models flexible hyperedge cut penalties, and (4) comes with a conductance guarantee. Our algorithm starts by reducing hypergraphs to directed graphs using carefully constructed directed graph gadgets, along with a set of auxiliary nodes, to encode the properties of a general class of cardinality based hypergraph cut functions. Similar transformations have been adapted to generalize flow-based local graph clustering algorithms to hypergraphs [23] by solving a series of related problem of 3.4 with $\ell(x) = (x)_+ = \max(x, 0)$. Here the "+" sign is implied by edge directions. However, as stated earlier in previous sections, for local graph clustering, flow-based methods often have difficulty growing from small seed sets. This phenomenon still exists in local hypergraph clustering. This motivates us to extend this framework to incorporate other types of norms. (see Section 6.2 for a case study of comparing different strategies)

In this chapter, we first show that these transformations can not only preserve minimum cuts, but also preserve the hypergraph conductance values. (Section 6.3) Then we consider equation 3.4 with $\ell(x) = \frac{1}{2}(x)_+^2$ akin to personalized PageRank on graphs. Once we have

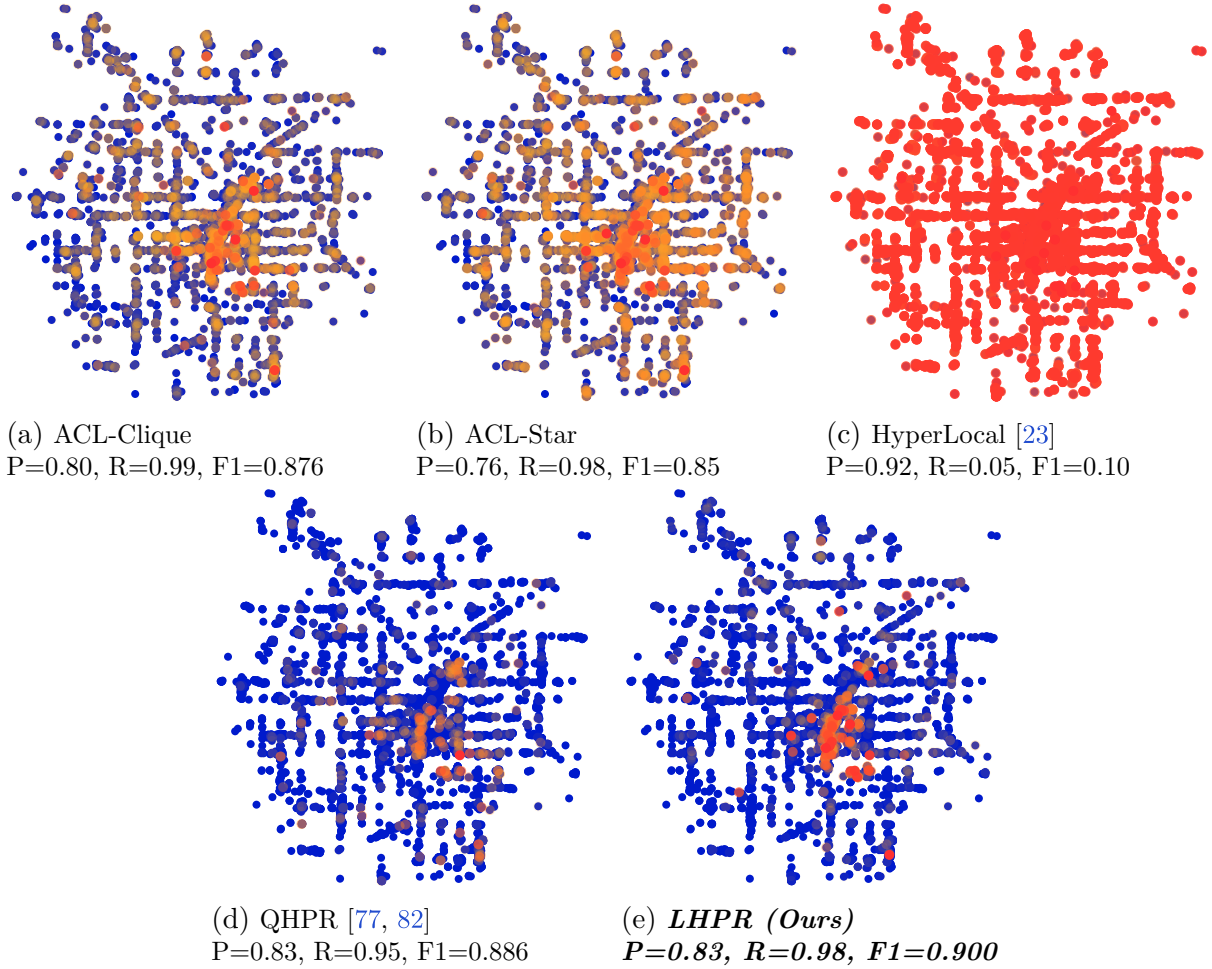
the framework in place, we are able to show that an adaptation of the push method for personalized PageRank will compute an approximate solution in strongly local time. (Section 6.5) We also prove that the clusters found by solving the new objective function satisfy a Cheeger-like quality guarantee. (Section 6.6) Similar to Chapter 5, the extended framework and the corresponding algorithm can also be generalized to p-norm, which can produce more accurate results in some cases. (Section 6.7) Experiments show that our new algorithm is not only efficient, but also produces results with much larger F1 scores than alternative methods. In particular, it is much faster and performs much better with extremely limited label information comparing to its flow-based alternative. (Section 6.8)

6.2 A Motivating Case Study with Yelp Reviews

We begin by illustrating the need and utility for the methods instead with a simple example of the benefit to these spectral or PageRank-style hypergraph approaches. For this purpose we consider a hypothetical use case with an answer that is easy to understand in order to compare our algorithm to a variety of other approaches. We build a hypergraph from the Yelp review dataset (https://www.yelp.com/dataset). Each restaurant is a vertex and each user is a hyperedge. This model enables users, i.e. hyperedges, to capture subtle socioeconomic status information as well as culinary preferences in terms of which types of restaurants they visit and review. The task we seek to understand is either an instance of local clustering or semi-supervised learning. Simply put, given a random sample of 10 restaurants in Las Vegas Nevada, we seek to find other restaurants in Las Vegas. The overall hypergraph has around 64k vertices and 616k hyperedges with a maximum hyperedge size of 2566. Las Vegas, with around 7.3k restaurants, constitutes a small localized cluster.

We investigate a series of different algorithms that will identify a cluster nearby a seed node in a hypergraph: (1) Andersen-Chung-Lang PageRank on the star and clique expansion of the hypergraph (ACL-Star, ACL-Clique, respectively), these algorithms are closely related to ideas proposed in [78, 83] (2). HyperLocal, a recent maximum flow-based hypergraph clustering algorithm [23], (3) quadratic hypergraph PageRank [77, 82] (which is also closely related to [79]), and (4) our Local Hypergraph-PageRank (LHPR). These are all

strongly local except for (3), which we include because our algorithm LHPR is essentially the strongly local analogue of (3). The results are shown in Figure 6.1. Our strongly local



(a) ACL-Clique
P=0.80, R=0.99, F1=0.876

(b) ACL-Star
P=0.76, R=0.98, F1=0.85

(c) HyperLocal [23]
P=0.92, R=0.05, F1=0.10

(d) QHPR [77, 82]
P=0.83, R=0.95, F1=0.886

(e) ***LHPR (Ours)***
P=0.83, R=0.98, F1=0.900

Figure 6.1. This figure shows locations of the ∼7,300 restaurants of Las Vegas that are reviewed on Yelp and how often algorithms recover them from a set of 10 random seeds; our hypergraph PageRank (LHPR) methods has the highest accuracy and finds the result by exploring only 10000 vertices total compared with a fully dense vector for QHPR giving a boost to scalability on larger graphs. The colors show the regions that are missed (red or orange) or found (blue) by each algorithm over 15 trials. HyperLocal is a flow-based method that is known to have trouble growing small seed sets as in this experiment. (The parameters for HyperLocal were chosen in consultation its authors; other parameters were hand tuned for best case performance.)

hypergraph PageRank (LHPR) slightly improves on the performance of a quadratic hyper-

graph PageRank (QHPR) that is not strongly local. In particular, it has 10k non-zero entries (of 64k) in its solution.

This experiment shows the opportunities with our approach for large hypergraphs. We are able to model a flexible family of hypergraph cut functions beyond those that use clique and star expansions and we equal or outperform all the other methods. (We also tried another recently proposed method [84], although this was unable to finish in a reasonable time (e.g. hours) on this hypergraph as it was designed for hypergraphs with small hyperedge size.)

6.3 Hypergraph-to-graph reduction

Minimizing conductance is NP-hard even in the case of simple graphs, though numerous techniques have been designed to approximate the objective in theory and practice [24, 64, 85]. A common strategy for searching for low-conductance sets in hypergraphs is to first reduce a hypergraph to a graph, and then apply existing graph-based techniques. This sounds "hacky" or least "ad-hoc" but this idea is both principled and rigorous. The most common approach is to apply a clique expansion [22, 75, 78, 83, 86], which explicitly models splitting functions of the form $f_{\mathrm{e}}(A) \propto |A|\,|\mathrm{e}\backslash A|$. For instance Benson et al. [75] showed that clique expansion can be used to convert a 3-uniform hypergraph into a graph that preserves the *all-or-nothing* conductance values. For larger hyperedge sizes, *all-or-nothing* conductance is preserved to within a distortion factor depending on the size of the hyperedge. Later, Li et al. [22] were the first to introduce more generalized notions of hyperedge splitting functions, focusing specifically on submodular functions.

Definition 6.3.1. *A splitting function f_{e} is submodular if*

$$f_{\mathrm{e}}(A) + f_{\mathrm{e}}(B) \geq f_{\mathrm{e}}(A \cup B) + f_{\mathrm{e}}(A \cap B) \quad \forall A, B \subseteq \mathrm{e}. \tag{6.1}$$

These authors showed that for this submodular case, clique expansion could be used to define a graph preserving conductance to within a factor $O(\zeta)$ (ζ is the largest hyperedge size).

More recently, Veldt et al. [81] introduced graph reduction techniques that *exactly* preserve submodular hypergraph *cut* functions which are cardinality-based.

Definition 6.3.2. *A splitting function f_e is cardinality-based if*

$$f_e(A) = f_e(B) \quad \textit{whenever } |A| = |B|. \tag{6.2}$$

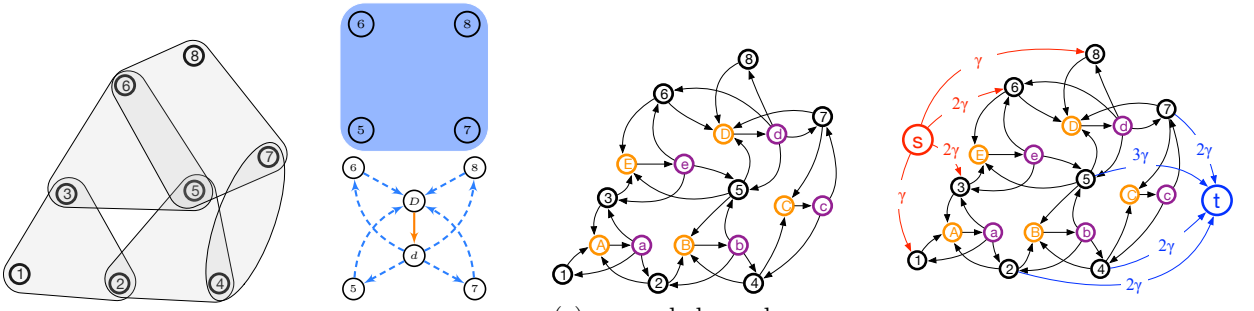
Cardinality-based splitting functions are a natural choice for many applications, since node identification is typically irrelevant in practice, and the cardinality-based model produces a cut function that is invariant to node permutation. Furthermore, most previous research on applying generalized hypergraph cut penalties implicitly focused on cut functions are that are naturally cardinality-based [75, 76, 79, 82, 86, 87]. Because of their ubiquity and flexibility, in this work we also focus on hypergraph cut functions that are submodular and cardinality-based. We briefly review the associated graph transformation and then we build on previous work by showing that these hypergraph reductions can be used to preserve the hypergraph *conductance* objective, and not just hypergraph cuts.

Reduction for Cardinality-Based Cuts

Veldt et al. [81] showed that the cut properties of a submodular, cardinality-based hypergraph could be preserved by replacing each hyperedge with a set of directed graph *gadgets*. Each gadget for a hyperedge e is constructed by introducing a pair of auxiliary nodes a and b, along with a directed edge (a, b) with weight $\delta_e > 0$. For each $v \in e$, two unit-weight directed edges are introduced: (v, a) and (b, v). The entire gadget is then scaled by a weight $c_e \geq 0$. The resulting gadget represents a simplified splitting function of the following form:

$$f_e(A) = c_e \cdot \min\{|A|, |e \backslash A|, \delta_e\}. \tag{6.3}$$

Figure 6.2(b) illustrates the process of replacing a hyperedge with a gadget. The cut properties of any submodular cardinality-based splitting function can be exactly modeled by introducing a set of $O(|e|)$ or fewer such splitting functions [81]. If an approximation suffices, only $O(\log |e|)$ gadgets are required [88].



(a) original hypergraph (b) single hyperedge reduction gadget (c) expanded graph (d) localized directed cut graph

Figure 6.2. A simple illustration of hypergraph reduction (Section 6.3) and localization (Section 6.4). (a) A hypergraph with 8 nodes and 5 hyperedges. (b) An illustration of the hyperedge transformation gadget for δ-linear splitting function. (c) The hypergraph is reduced to a directed graph by adding a pair of auxiliary nodes for each hyperedge and this preserves hypergraph conductance computations (Theorem 6.3.1). (d) The localized directed cut graph is created by adding a source node s, a sink node t and edges from s to hypergraph nodes or from hypergraph nodes to t to *localize* a solution.

An important consequence of these reduction results is that in order to develop reduction techniques for *any* submodular cardinality-based splitting functions, it suffices to consider hyperedges with splitting functions of the simplified form given in (6.3). In the remainder of the text, we focus on splitting functions of this form, with the understanding that all other cardinality-based submodular splitting functions can be modeled by introducing multiple hyperedges on the same set of nodes with different edge weights.

In Figure 6.2, we illustrate the procedure of reducing a small hypergraph to a directed graph, where we introduce a single gadget per hyperedge. Formally, for a hypergraph $\mathcal{H} = (V, E)$, this procedure produces a directed graph $G = (\hat{V}, \hat{E})$, with directed edge set $\hat{E}$, and node set $\hat{V} = V \cup V_a \cup V_b$, where V is the set of original hypergraph nodes. Sets V_a, V_b store auxiliary nodes, in such a way that for each pair of auxiliary nodes a, b where (a, b) is a directed edges, we have $a \in V_a$ and $b \in V_b$. This reduction technique was previously developed as a way of preserving minimum cuts and minimum s-t cuts for the original hypergraph. Here, we extend this result to show that for a certain choice for node degree, this reduction also preserves hypergraph conductance.

Theorem 6.3.1. *Define a degree vector* $\mathbf{d}$ *for the reduced graph* $G = (\hat{V}, \hat{E})$ *such that* $\mathbf{d}(v) = d_v$ *is the out-degree for each node* $v \in V$*, and* $\mathbf{d}(u) = d_u = 0$ *for every auxiliary node* $u \in V_a \cup V_b$*. If* T^* *is the minimum conductance set in* G *for this degree vector, then* $S^* = T^* \cap V$ *is the minimum hypergraph conductance set in* $\mathcal{H} = (V, E)$*.*

Proof. From previous work on these reduction techniques [81, 88], we know that the cut penalty for a set $S \subseteq V$ in $\mathcal{H}$ equals the cut penalty in the directed graph, as long as auxiliary nodes are arranged in a way that produces the smallest cut penalty subject to the choice of node set $S \subseteq V$. Formally, for $S \subseteq V$,

$$\mathbf{cut}_{\mathcal{H}}(S) = \underset{T \subset \hat{V} \colon S = T \cap V}{\text{minimize}} \mathbf{cut}_G(T), \tag{6.4}$$

where $\mathbf{cut}_G$ denotes the weight of directed out-edges originating inside S that are cut in G. By our choice of degree vector, the volume of nodes in G equals the volume of the non-auxiliary nodes in $\mathcal{H}$. That is, for all $T \subseteq \hat{V}$, $\mathbf{vol}_G(T) = \sum_{v \in V} d_v + \sum_{u \in V_a \cup V_b} d_u = \mathbf{vol}_G(T \cap V) = \mathbf{vol}_{\mathcal{H}}(T \cap V)$. Let $T^* \subseteq \hat{V}$ be the minimum conductance set in G, and $S^* = T^* \cap V$. Without loss of generality we can assume that $\mathbf{vol}_G(T^*) \leq \mathbf{vol}_G(\bar{T}^*)$. Since T^* minimizes conductance, and auxiliary nodes have no effect on the volume of this set, $\mathbf{cut}_G(T^*) = \text{minimize}_{T \subset \hat{V} \colon T \cap S^*} \mathbf{cut}_G(T) = \mathbf{cut}_{\mathcal{H}}(S^*)$, and so $\mathbf{cut}_G(T^*)/\mathbf{vol}_G(T^*) = \mathbf{cut}_{\mathcal{H}}(S^*)/\mathbf{vol}_{\mathcal{H}}(S^*)$. Thus, minimizing conductance in G minimizes conductance in $\mathcal{H}$. □

6.4 Localized Quadratic Hypergraph Diffusions

Having established a conductance-preserving reduction from a hypergraph to a directed graph, we will define a *localized directed cut graph* by adding a source and sink nodes and new weighted edges in the same way as Section 3.3. The key conceptual difference is that we apply this construction directly to the reduced graph G, which by Theorem 6.3.1 preserves conductance of the original hypergraph $\mathcal{H}$. Formally, we assume we are given a set of nodes $R \subseteq V$ around which we wish to find low-conductance clusters, and a parameter $\gamma > 0$. The localized directed cut graph is defined by applying the following steps to G:

- Add a source node s, and for each $r \in R$ define a directed edge (s, r) of weight γd_r.

- Add a sink node t, and for each $v \in \bar{R}$ define a directed edge (v, t) with weight γd_v.

We do not connect auxiliary nodes to the source or sink, which is consistent with the fact that their degree is defined to be zero in order for Theorem 6.3.1 to hold. We illustrate the construction of the localized directed cut graph in Figure 6.2(d). It is important to note that in practice we do not in fact form this graph and store it in memory. Rather, this provides a conceptual framework for finding localized low-conductance sets in G, which in turn correspond to good clusters in $\mathcal{H}$.

Definition: Local hypergraph quadratic diffusions.

Let $\boldsymbol{B}$ and $\mathbf{w}$ be the incidence matrix and edge weight vector of the localized directed cut graph with γ. The objective function for our hypergraph clustering diffusion, which we call *local hypergraph quadratic diffusion* or simply *local hypergraph PageRank*, is

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \frac{1}{2}\mathbf{w}^T(\boldsymbol{B}\mathbf{x})_+^2 + \kappa\gamma \sum_{i \in V} x_i d_i \\ \text{subject to} & x_s = 1, x_t = 0, \mathbf{x} \geq 0. \end{array} \tag{6.5}$$

We use the function $(x)_+ = \max\{x, 0\}$, applied element-wise to $\boldsymbol{B}\mathbf{x}$, to indicate we only keep the positive elements of this product. This is analogous to the fact that we only view a directed edge as being cut if it crosses from the source to the sink side; this is similar to previous directed cut minorants on graphs and hypergraphs [89].

6.5 A Strongly Local Solver for LHQD

Although Equation 6.5 looks very similar to Equation 3.4, Alogrithm 2 cannot be directly applied because: (1) $\ell(x) = \frac{1}{2}(x)_+^2$ doesn't satisfy Definition 5.2.1 (2) we set the degree of auxilary nodes to be zero, which will break the strong locality of Theorem 5.3.1 or Theorem 5.3.4. Thus, a new strongly local algorithm needs to be developed to solve Equation (6.5). We will first state the optimality conditions in Theorem 6.5.1, and then present the algorithm to solve them.

We begin with the optimality conditions for (6.5).

Theorem 6.5.1. *Fix a seed set R, $\gamma > 0$, $\kappa > 0$, define a residual function $\mathbf{r}(\mathbf{x}) = -\frac{1}{\gamma}\boldsymbol{B}^T diag((\boldsymbol{B}\mathbf{x})_+)\mathbf{w}$. A necessary and sufficient condition to satisfy the KKT conditions of (6.5) is to find $\mathbf{x}^*$ where $\mathbf{x}^* \geq 0$, $\mathbf{r}(\mathbf{x}^*) = [r_s, \mathbf{g}^T, r_t]^T$ with $g_i \leq \kappa d_i$ (where $\mathbf{d}$ reflects the graph before adding s and t but does include the 0 degree nodes), $(\kappa d_i - g_i)^T x_i^* = 0$ for $i \in V$ and $g_i = 0$ for all auxiliary nodes added.*

Proof. We will directly derive the KKT condition for the p-norm generalized version of equation (6.5), which is equation (6.19). Introduce a new set of variables $\mathbf{u} \in \mathbb{R}^{|\mathcal{E}|}$ and transform equation (6.19) into the following equivalent problem.

$$\begin{aligned} \underset{\mathbf{x},\mathbf{u}}{\text{minimize}} \quad & \mathbf{w}^T\ell(\mathbf{u}) + \kappa\gamma\textstyle\sum_{i\in V} x_i d_i \\ \text{subject to} \quad & x_s = 1, x_t = 0, \mathbf{x} \geq 0. \\ & \boldsymbol{B}\mathbf{x} \leq \mathbf{u}, \mathbf{u} \geq 0 \end{aligned} \tag{6.6}$$

Here $\boldsymbol{B}\mathbf{x} \leq \mathbf{u}$, $\mathbf{u} \geq 0$ and $\ell(x)$ is an increasing function imply that $\mathbf{u} = (\boldsymbol{B}\mathbf{x})_+$ at the optimality. The Lagrangian of equation (6.6) can written as

$$\mathcal{L} = \mathbf{w}^T\ell(\mathbf{u}) + \kappa\gamma\hat{\mathbf{d}}^T\mathbf{x} + \mathbf{f}^T(\boldsymbol{B}\mathbf{x} - \mathbf{u}) - \mathbf{z}^T\mathbf{u} + \lambda_s(x_s - 1) + \lambda_t x_t - \mathbf{k}^T\mathbf{x} \tag{6.7}$$

Here $\hat{d}_i = d_i$ when $i \in V$ and $\hat{d}_i = 0$ when $i \in V_a \cup V_b$. Standard optimality results show that the KKT conditions for equation (6.7) are:

$$\begin{aligned} & \frac{\partial\mathcal{L}}{\partial\mathbf{x}} = \kappa\gamma\hat{\mathbf{d}} + \boldsymbol{B}^T\mathbf{f} + \lambda_s\mathbf{e}_s + \lambda_t\mathbf{e}_t - \mathbf{k} = 0 \\ & \frac{\partial\mathcal{L}}{\partial\mathbf{u}} = \text{diag}(\ell'(\mathbf{u}))\mathbf{w} - \mathbf{f} - \mathbf{z} = 0 \\ & \boldsymbol{B}\mathbf{x} \leq \mathbf{u}, \mathbf{u} \geq 0 \\ & \mathbf{f}^T(\boldsymbol{B}\mathbf{x} - \mathbf{u}) = 0 \\ & \mathbf{z}^T\mathbf{u} = 0 \\ & \mathbf{k}^T\mathbf{x} = 0 \\ & \mathbf{k} \geq 0, x_s = 1, x_t = 0 \\ & \mathbf{f} \geq 0. \end{aligned}$$

Setting $\mathbf{u} = (\boldsymbol{B}\mathbf{x})_+$, which will be true at optimality, we get

$$\frac{\partial L}{\partial \mathbf{u}} = \operatorname{diag}(\ell'(\mathbf{u}))\mathbf{w} - \mathbf{f} - \mathbf{z} = 0 \implies \mathbf{f} = \operatorname{diag}\ell((\boldsymbol{B}\mathbf{x})_+)\mathbf{w} - \mathbf{z}$$

which means that for an edge $i \to j$,

$$f_{ij} = w_{ij} \cdot \ell((x_i - x_j)_+) - z_{ij}. \tag{6.8}$$

Since $\mathbf{f}^T(\boldsymbol{B}\mathbf{x} - \mathbf{u}) = 0$, we also have that for each edge $i \to j$,

$$f_{ij} \cdot ((x_i - x_j) - (x_i - x_j)_+) = 0 \tag{6.9}$$

$$\iff (-z_{ij} + w_{ij} \cdot \ell((x_i - x_j)_+)) \cdot ((x_i - x_j) - (x_i - x_j)_+) = 0. \tag{6.10}$$

Similarly, $\mathbf{z}^T\mathbf{u} = 0$ means that for edge $i \to j$,

$$z_{ij} \cdot (x_i - x_j)_+ = 0. \tag{6.11}$$

If $x_i \geq x_j$, then (6.10) holds automatically, but we need $z_{ij} = 0$ for (6.11) to hold. If $x_i < x_j$, then (6.11) holds automatically, but we need $z_{ij} = 0$ in order for (6.10) to hold. Either way, at optimality we simply have $\mathbf{z} = 0$. So line 2 of KKT condition can be simplified into $\mathbf{f} = \operatorname{diag}(\ell'(\mathbf{u}))\mathbf{w}$, which implies $\mathbf{f} \geq 0$. Then line 4 is automatically satisfied by combining $\mathbf{f} = \operatorname{diag}(\ell'(\mathbf{u}))\mathbf{w}$ and $\mathbf{u} = (\boldsymbol{B}\mathbf{x})_+$. Therefore, the simplified KKT condition can be written as

$$\begin{aligned}
&\kappa\gamma\hat{\mathbf{d}} + \boldsymbol{B}^T\operatorname{diag}(\ell'((\boldsymbol{B}\mathbf{x})_+))\mathbf{w} + \lambda_s\mathbf{e}_s + \lambda_t\mathbf{e}_t - \mathbf{k} = 0 \\
&\mathbf{k}^T\mathbf{x} = 0 \\
&\mathbf{k} \geq 0, x_s = 1, x_t = 0
\end{aligned}$$

Since the residual vector $\mathbf{r}$ is defined as $-\frac{1}{\gamma}\boldsymbol{B}^T\operatorname{diag}(\ell'((\boldsymbol{B}\mathbf{x})_+))\mathbf{w}$, from $\mathbf{k} \geq 0$ and $\hat{d}_i = 0$ for $i \in V_a \cup V_b$, $\hat{d}_i = d_i$ for $i \in V$, we have $\mathbf{g}_V \leq \kappa\mathbf{d}$ and $\mathbf{g}_{V_a \cup V_b} = 0$. Also for $i \in V_a \cup V_b$, from

$g_i = \hat{d}_i = 0$, we have $k_i = 0$. Thus $\mathbf{k}^T\mathbf{x} = 0$ can be reduced to $(\kappa\mathbf{d} - \mathbf{g}_V)^T\mathbf{x}_V = 0$. Now by replacing $\ell(x)$ with $x^2/2$, we can get theorem 6.5.1. □

We further note that solutions $\mathbf{x}^*$ are unique because the problem is strongly convex due to the quadratic.

In Section 6.3, we have shown that the reduction technique of any cardinality submodular-based splitting function suffices to introduce multiple directed graph gadgets with different δ_e and c_e. In order to simplify our exposition, we assume that each hyperedge has a δ-linear threshold splitting function [23] $f_e = \min\{|A|, |e\backslash A|, \delta\}$ with $\delta \geq 1$ to be a tunable parameter. This splitting function can be exactly modeled by replacing each hyperedge with one directed graph gadget with $c_e = 1$ and $\delta_e = \delta$. (This is what is illustrated in Figure 6.2.) Also when $\delta = 1$, it models the standard unweighted *all-or-nothing* cut [90–92] and when δ goes to infinity, it models star expansion [86]. Thus this splitting function can interpolate these two common cut objectives on hypergraphs by varying δ.

By assuming that we have a δ-linear threshold splitting function, this means we can associate exactly *two* auxiliary nodes with each hyperedge. We call these a and b for simplicity. We also let V_a be the set of all a auxiliary nodes and V_b be the set of all b nodes.

At a high level, the algorithm to solve this proceeds as follows: whenever there exists an graph node $i \in V$ that violates optimality, i.e. $r_i > \kappa d_i$, we first perform a `hyperpush` at i to increase x_i so that the optimality condition is approximately satisfied, i.e., $r_i = \rho\kappa d_i$ where $0 < \rho < 1$ is a given parameter that influences the approximation. This changes the solution $\mathbf{x}$ only at the current node i and residuals at adjacency auxiliary nodes. Then for adjacent auxiliary we immediately *push* on them, which means to increase their value so that the residuals remain zero. After pushing each pair (a, b) of associated auxiliary nodes, we then update residuals for *adjacent* nodes in V. Then we search for another optimality violation. (See Algorithm 4 for a formalization of this strategy.) When $\rho < 1$, we still only approximately satisfy the optimality conditions (see Section 5.4 for more explanation).

We have the following lemmas on $\mathbf{x}$ and $\mathbf{r}$.

Lemma 6.5.2. *At any iteration of Algorithm 4, for each pair of auxiliary nodes, $a \in V_a$, $b \in V_b$ and $a \to b$, $x_a \geq x_b$.*

Proof. If $x_a < x_b$, from $g_b = w_{ab}(x_a - x_b)_+ - \sum_{i \in V} w_{bi}(x_b - x_i)_+ = 0$ we have for any $i \in V$ where $b \to i$, $x_b \leq x_i$. This means $g_a = -w_{ab}(x_a - x_b)_+ + \sum_{i \in V} w_{ia}(x_i - x_a)_+ > 0$, which is a contradiction because g_a is zero for all iterations. □

Lemma 6.5.3. *At any iteration of Algorithm 4, for any $i \in V \cup V_a \cup V_b$, g_i will stay nonnegative and $0 \leq x_i \leq 1$.*

Proof. The nonnegativity of g_i is obvious. At any iteration of Algorithm 4, we either change g_i to $\rho\kappa d_i$ or add some nonnegative values to the residual of adjacent nodes or keep g_i as zero. For each original node $i \in V$, to prove $0 \leq x_i \leq 1$, recall $r_i = g_i$ expands to

$$g_i = \frac{1}{\gamma} \sum_{b \in V_b} w_{bi}(x_b - x_i)_+ - \frac{1}{\gamma} \sum_{a \in V_a} w_{ia}(x_i - x_a)_+ + d_i[\mathrm{Ind}(i \in R) - x_i]$$

Suppose x_i is the largest element and $x_i > 1$, then $\mathrm{Ind}(i \in R) - x_i < 0$. Since $g_i \geq 0$, this means there exists $b \in V_b$ such that $x_b > x_i > 1$. Assume $a \in V_a$ is adjacent to b, then

$$g_a = -w_{ab}(x_a - x_b)_+ + \textstyle\sum_{i \in V} w_{ia}(x_i - x_a)_+ = 0$$

$$g_b = w_{ab}(x_a - x_b)_+ - \textstyle\sum_{i \in V} w_{bi}(x_b - x_i)_+ = 0$$

Since x_i is the largest among all nodes $i \in V$, the only way to satisfy both equations is $x_a = x_b = x_i$. Thus, we have $x_i = x_b > x_i$, which is a contradiction. To prove $x_i \leq 1$ for $i \in V_a \cup V_b$, from Lemma 6.5.2, we only need to show $x_i \leq 1$ for $i \in V_a$. If there exists $a \in V_a$ such that $x_a > 1$, since $x_i \leq 1$ for any $i \in V$, then from $g_a = 0$, we have $x_a = x_b > 1$ where $a \to b$. This means $g_b < 0$, which is a contradiction. □

Notes on optimizing the procedure.

Algorithm 4 describes a general strategy to approximately solve these diffusions. We now note a number of optimizations that we have found to greatly accelerate this strategy. First, note that $\mathbf{x}$ and $\mathbf{r}$ can be kept as sparse vectors with only a small set of entries stored. Second, note that we can maintain a list of optimality violations because each update to $\mathbf{x}$

Algorithm 4 `LHQD`(γ, κ, ρ) for set R and hypergraph H with δ-linear penalty where $0 < \rho < 1$ determines accuracy

1: Let $\mathbf{x}=0$ except for $x_s=1$ and set $\mathbf{r}=-\gamma^{-1}\boldsymbol{B}^T\text{diag}((\boldsymbol{B}\mathbf{x})_+)\mathbf{w}(\delta,\gamma)$.
2: While there is any vertex $i \in V$ where $r_i > \kappa d_i$, or stop if none exists *(find an optimality violation)*
3: Perform `LHQD-hyperpush` at vertex i so that $r_i = \rho\kappa d_i$, updating $\mathbf{x}$ and $\mathbf{r}$. *(satisfy optimality at i)*
4: For each pair of adjacent auxiliary nodes a, b where $a \in V_a$, $b \in V_b$ and $a \to b$, perform `LHQD-auxpush` at a and b so that $r_a = r_b = 0$, then update $\mathbf{x}$ and $\mathbf{r}$ after each `auxpush`.
5: Return $\mathbf{x}$

only causes $\mathbf{r}$ to increase, so we can simply check if each coordinate increase creates a new violation and add it to a queue. Third, to find the value that needs to be "pushed" to each node, a general strategy is to use a binary search procedure as we will use for the p-norm generalization in Section 6.7. However, if the tolerance of the binary search is too small, it will slow down each iteration. If the tolerance is too large, the solution will be too far away from the true solution to be useful. In the remaining of this section, we will show that in the case of quadratic objective (6.5), we can (i) often avoid binary search and (ii) when it is still required, make the binary search procedure unrelated to the choice of tolerance in those iterations where we do need it. Note that these detailed techniques will not change the time complexity of the overall algorithm, but make a large difference in practice.

We will start by looking at the expanded formulations of the residual vector. When $i \in V$, r_i expands as:

$$r_i = \frac{1}{\gamma}\sum_{b\in V_b} w_{bi}(x_b - x_i)_+ - \frac{1}{\gamma}\sum_{a\in V_a} w_{ia}(x_i - x_a)_+ + d_i[\text{Ind}(i \in R) - x_i]. \tag{6.12}$$

Similarly, for each $a \in V_a$, $b \in V_b$ where $a \to b$, they will share the same set of original nodes and their residuals can be expanded as:

$$\begin{aligned} r_a &= -w_{ab}(x_a - x_b) + \textstyle\sum_{i\in V} w_{ia}(x_i - x_a)_+ \\ r_b &= w_{ab}(x_a - x_b) - \textstyle\sum_{i\in V} w_{bi}(x_b - x_i)_+ \end{aligned} \tag{6.13}$$

Note here we use lemma 6.5.2.

The goal in each hyperpush is to first find Δx_i such that $r'_i = \rho\kappa d_i$ and then in auxpush, for each pair of adjacent auxiliary nodes (a, b), find Δx_a and Δx_b such that r'_a and r'_b remain zero. (Δx_i, Δx_a and Δx_b are unique because the quadratic is strongly convex.) Observe that r_i, r_a and r_b are all piecewise linear functions, which means we can derive a closed form solution once the relative ordering of adjacent nodes is determined. Also, in most cases, the relative ordering won't change after a few initial iterations. So we can first reuse the ordering information from last iteration to directly solve Δx_i, Δx_a and Δx_b and then check if the ordering is changed.

Given these observations, we will record and update the following information for each pushed node. Again, this information can be recorded in a *sparse* fashion. When the pushed node i is a original node, for its adjacent $a \in V_a$ and $b \in V_b$, we record:

- $s_a^{(i)}$: the sum of edge weights w_{ia} where $x_a < x_i$
- $s_b^{(i)}$: the sum of edge weights w_{bi} where $x_b > x_i$
- $a_{min}^{(i)}$: the minimum x_a where $x_a \geq x_i$
- $b_{min}^{(i)}$: the minimum x_b where $x_b > x_i$

Now assume the ordering is the same, r'_i can be written as $r'_i = r_i - \frac{1}{\gamma}(s_a^{(i)} + s_b^{(i)})\Delta x_i = \rho\kappa d_i$, so

$$\Delta x_i = \gamma(r_i - \rho\kappa d_i)/(s_a^{(i)} + s_b^{(i)}). \tag{6.14}$$

Then we need to check whether the assumption holds by checking

$$x_i + \Delta x_i \leq \min\left(a_{min}^{(i)}, b_{min}^{(i)}\right) \tag{6.15}$$

If not, we need to use binary search to find the new location of $x_i + \Delta x_i$ (Note Δx_i here is the true value that is still unknown), update $s_a^{(i)}$, $s_b^{(i)}$, $a_{min}^{(i)}$ and $b_{min}^{(i)}$ and recompute Δx_i. This process is summarized in `LQHD-hyperpush`.

Similarly, when the pushed nodes $a \in V_a$, $b \in V_b$ where $a \to b$, are a pair of auxiliary nodes, for its adjacent nodes $i \in V$, we record:

Algorithm 5 `LQHD-hyperpush`$(i, \gamma, \kappa, \mathbf{x}, \mathbf{r}, \rho)$

1: Solve Δx_i with $s_a^{(i)}$, $s_b^{(i)}$, $a_{min}^{(i)}$ and $b_{min}^{(i)}$ using (6.14). *(assume the order of i doesn't change among its adjacent nodes)*
2: **if** (6.15) doesn't hold *(adding Δx_i changed the order of i)* **then**
3: Binary search on Δx_i until we find the smallest interval among all adjacent nodes of i that will include $x_i + \Delta x_i$, update $s_a^{(i)}$, $s_b^{(i)}$, $a_{min}^{(i)}$ and $b_{min}^{(i)}$.
4: Solve Δx_i with the found interval by setting $r_i = \rho\kappa d_i$ in (6.12).
5: **end if**
6: Update $\mathbf{x}$ and $\mathbf{r}$, $x_i \leftarrow x_i + \Delta x_i$, $r_i \leftarrow \rho\kappa d_i$

- z_a: the sum of edge weights w_{ia} where $x_a < x_i$
- z_b: the sum of edge weights w_{bi} where $x_b > x_i$
- $x_{min}^{(a)}$: the minimum x_i where $x_a < x_i$
- $x_{min}^{(b)}$: the minimum x_i where $x_b < x_i$

Then we solve Δx_a, Δx_b by solving the following linear system (here we assume $x_b \geq x_i$)

$$\begin{cases} -w_{ab}(\Delta x_a - \Delta x_b) + \dfrac{w_{ia}}{\gamma}((x_i' - x_a)_+ - (x_i - x_a)_+) - z_a \Delta x_a = 0 \\ \quad w_{ab}(\Delta x_a - \Delta x_b) - \dfrac{w_{bi}}{\gamma}((x_b - x_i')_+ - (x_b - x_i)_+) + z_b \Delta x_b = 0 \end{cases} \tag{6.16}$$

where x_i' refers to the updated x_i after applying `LQHD-hyperpush` at node i. And the assumption will hold if and only if the following inequalities are all satisfied:

$$x_i' \leq x_b, \qquad x_a + \Delta x_a \leq x_{min}^{(a)}, \qquad x_b + \Delta x_b \leq x_{min}^{(b)} \tag{6.17}$$

If not, we also need to use binary search to update the locations of $x_a + \Delta x_a$ and $x_b + \Delta x_b$, update z_a, z_b, $x_{min}^{(a)}$, $x_{min}^{(b)}$ and recompute Δx_a and Δx_b.

Algorithm 6 `LQHD-auxpush`$(i, a, b, \gamma, \mathbf{x}, \mathbf{r}, \Delta x_i)$

1: Solve Δx_a, Δx_b with z_a, z_b, $x_{min}^{(a)}$ and $x_{min}^{(b)}$ using (6.16).
2: **if** (6.17) doesn't hold. *(adding $\Delta x_a, \Delta x_b$ altered the order)* **then**
3: Binary search on Δx_a until we find the smallest interval among all adjacent original nodes of a that will include $x_a + \Delta x_a$, update z_a, $x_{min}^{(a)}$, similarly for z_b, $x_{min}^{(b)}$.
4: Solve $\Delta x_a, \Delta x_b$ with the found intervals by setting $r_a = r_b = 0$ in (6.13).
5: **end if**
6: Change the following entries in $\mathbf{x}$ and $\mathbf{r}$ to update the solution and the residual
7: (a) $x_a \leftarrow x_a + \Delta x_a$ and $x_b \leftarrow x_b + \Delta x_b$
8: (b) For each neighboring node $i \to a$ where $i \in V$, $r_i \leftarrow r_i + \frac{1}{\gamma} w_{ia}(x_i - x_a)_+ - \frac{1}{\gamma} w_{ia}(x_i - x_a - \Delta x_a)_+ - \frac{1}{\gamma} w_{bi}(x_b - x_i)_+ + \frac{1}{\gamma} w_{bi}(x_b + \Delta x_b - x_i)_+$

Establishing a runtime bound.

The key to understand the strong locality of the algorithm is that after each `LQHD-hyperpush`, the decrease of $\|\mathbf{g}\|_1$ can be lower bounded by some number that is independent of the total size of the hypergraph, while `LHQD-auxpush` doesn't change $\|\mathbf{g}\|_1$. Formally, we have the following theorem:

Theorem 6.5.4. *Given $\gamma > 0$, $\kappa > 0$, $\delta > 0$ and $0 < \rho < 1$. Suppose the splitting function f_e is submodular, cardinality-based and satisfies $1 \le f_e(\{i\}) \le \delta$ for any $i \in e$. Then calling* ***LQHD-auxpush*** *doesn't change $\|\mathbf{g}\|_1$ while calling* ***LQHD-hyperpush*** *on node $i \in V$ will decrease $\|\mathbf{g}\|_1$ by at least $\gamma\kappa(1-\rho)d_i/(\gamma\kappa + \delta)$.*

Suppose ***LHQD*** *stops after T iterations and d_i is the degree of the original node updated at the i-th iteration, then T must satisfy:*

$$\sum_{i=1}^{T} d_i \le (\gamma\kappa + \delta)vol(R)/\gamma\kappa(1-\rho) = O(vol(R)).$$

Proof. By using Lemma 6.5.3, $|\mathbf{g}|_1$ becomes

$$|\mathbf{g}|_1 = \sum_{i \in V \cup V_a \cup V_b} g_i = \sum_{i \in R} d_i(1 - x_i) - \sum_{i \in \bar{R}} d_i x_i$$

This implies that any change to the auxiliary nodes will not affect $\|\mathbf{g}\|_1$. Thus calling `LHQD-auxpush` doesn't change $\|\mathbf{g}\|_1$. When there exists $i \in V$ such that $g_i > \kappa d_i$, then `hyper-push` will find Δx_i such that $g_i' = \rho\kappa d_i$. Then the new g_i' can be written as

$$g_i' = \frac{1}{\gamma}\sum_{b\in V_b} w_{bi}(x_b - x_i - \Delta x_i)_+ - \frac{1}{\gamma}\sum_{a\in V_a} w_{ia}(x_i + \Delta x_i - x_a)_+ + \kappa d_i(\text{Ind}[i \in R] - x_i - \Delta x_i) \quad (6.18)$$

Note g_i' is a decreasing function of Δx_i and $g_i' > 0$ when $\Delta x_i = 0$, $g_i' < 0$ when $\Delta x_i = 1$ by using Lemma 6.5.3. This suggests that there exists a unique Δx_i that satisfies $g_i' = \rho\kappa d_i$. Moreover, $(x_b - x_i - \Delta x_i)_+ \geq (x_b - x_i)_+ - \Delta x_i$ and $(x_i + \Delta x_i - x_a)_+ \leq (x_i - x_a)_+ + \Delta x_i$, thus we have

$$\rho\kappa d_i = g_i' \geq g_i - \tfrac{1}{\gamma}(\textstyle\sum_{b\in V_b} w_{bi} + \sum_{a\in V_a} w_{ia})\Delta x_i - \kappa d_i \Delta x_i$$

From equation (4.9) of [81],

$$\textstyle\sum_{b\in V_b} w_{bi} = \sum_{a\in V_a} w_{ai} = \sum_{\mathrm{e}\in\mathcal{E}, i\in\mathrm{e}} f_{\mathrm{e}}(\{i\}) \leq \delta d_i$$

Thus, we have $d_i\Delta x_i \geq \frac{g_i - g_i'}{\kappa + \delta/\gamma} > \frac{\gamma\kappa(1-\rho)}{\gamma\kappa+\delta} d_i$ So the decrease of $\|\mathbf{g}\|_1$ will be at least $\gamma\kappa(1-\rho)d_i/(\gamma\kappa+\delta)$. Since $\|\mathbf{g}\|_1 = \mathbf{vol}(R)$ initially, we have $\sum_{i=1}^{T} d_i \leq (\gamma\kappa+\delta)\mathbf{vol}(R)/\gamma\kappa(1-\rho) = O(\mathbf{vol}(R))$. □

Similar to Theorem 5.3.1, this theorem only upper bounds the number of iterations Algorithm 4 requires. Each iteration of Algorithm 4 will also take $O(\sum_{\mathrm{e}\in\mathcal{E}, i\in\mathrm{e}} |\mathrm{e}|)$ amount of work. This ignores the binary search, which only scales it by $\log(\max\{d_i, \max_{\mathrm{e}\in\mathcal{E}, i\in\mathrm{e}}\{|\mathrm{e}|\}\})$ factor in the worst case. Putting these pieces together shows that if we have a hypergraph with *independently bounded* maximum hyperedge size, then we can treat this additional work as a constant. Consequently, our solver is strongly local for graphs with bounded maximum hyperedge size; this matches the interpretation in [23].

6.6 Local Conductance Approximation

We give a local conductance guarantee that results from solving (6.5). For simplicity, we focus on the case $\kappa = 0$. We prove that a sweepcut on the solution $\mathbf{x}$ of (6.5) leads to a Cheeger-type guarantee for conductance of the hypergraph $\mathcal{H}$ even when the seed-set size $|R|$ is 1. It is extremely difficult to guarantee a good approximation property with an arbitrary seed node, and so we first introduce a seed sampling strategy $\mathbb{P}$ with respect to a set S^* that we wish to find. Informally, the seed selection strategy says that the expected solution mass outside S^* is not too large, and more specifically, not too much larger than if you had seeded on the entire target set S^*.

Definition 6.6.1. *Denote $\mathbf{x}(\gamma, R)$ as the solution to (6.5) with $\kappa = 0$. A* good sampling *strategy $\mathbb{P}$ for a target set S^* is*

$$\mathbb{E}_{v\in\mathbb{P}}\left[\frac{1}{d_v}\sum_{u\in V\setminus S^*} d_u x_u(\gamma,\{v\})\right] \le \frac{c}{\boldsymbol{vol}(S^*)}\sum_{u\in V\setminus S^*} d_u x_u(\gamma, S^*)$$

for some positive constant c.

Note that $\mathbf{vol}(S^*)$ is just to normalize the effect of using different numbers of seeds. For an arbitrary S^*, a good sampling strategy $\mathbb{P}$ for the standard graph case with $c = 1$ is to sample nodes from S^* proportional to their degree. Now, we provide our main theorem. Its proof requires some understanding on Lovász-Simonovits Curve (LSC) as well as tons of analysis in linear algebra. We refer to our paper [20] for the full proof.

Theorem 6.6.1. *Given a set S^* of vertices s.t. $\boldsymbol{vol}(S^*) \le \frac{\boldsymbol{vol}(\mathcal{H})}{2}$ and $\phi_{\mathcal{H}}(S^*) \le \frac{\gamma}{8c}$ for some positive constant γ, c. If we have a seed sampling strategy $\mathbb{P}$ that satisfies Def. 6.6.1, then with probability at least $\frac{1}{2}$, sweepcut on (6.5) with find $S_{\mathbf{x}}$ with*

$$\phi(S_{\mathbf{x}}) \le \sqrt{32\gamma\bar{\delta}\ln\left(100\boldsymbol{vol}(S^*)/d_v\right)},$$

where $\bar{\delta} = \max_{\mathrm{e}\in\partial S_{\mathbf{x}}}\min\{\delta_{\mathrm{e}}, |\mathrm{e}|/2\}$ where $\partial S_{\mathbf{x}} = \{\mathrm{e}\in\mathcal{E} \mid \mathrm{e}\cap S_{\mathbf{x}} \neq \emptyset, \mathrm{e}\cap\bar{S}_{\mathbf{x}} \neq \emptyset\}$ and v is the seeded node.

This implies that for any set S^*, if we have a sampling strategy that matches S^* and tune γ, our method can find a node set with conductance $O(\sqrt{\phi_{\mathcal{H}}(S^*)\bar{\delta}\log(\mathbf{vol}(S^*))})$. The term $\bar{\delta}$ is the additional cost that we pay for performing graph reduction. The dependence on $\bar{\delta}$ essentially generalizes the previous works that analyzed the conductance with only *all-or-nothing* penalty [77, 82], as our result matches these when $\bar{\delta} = 1$. But our method gives the flexibility to choose other values δ_e and while $\bar{\delta}$ in the worst case could be as large as $|e|/2$, in practice, $\bar{\delta}$ can be chosen much smaller (See Section 6.8). Also, although we reduce $\mathcal{H}$ into a directed graph G, the previous conductance analysis for directed graphs [82, 89] is not applicable as we have degree zero nodes in G. Those degree zero nodes introduce challenges. The detailed proof can be found in our paper [20].

6.7 Generalization to P-norms

Similar to Chapter 5, our *hypergraph diffusion* framework can be easily generalized to p-norm.

Definition: p-norm local hypergraph diffusions.

Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$, seeds R, and values γ, κ. Let $\boldsymbol{B}, \mathbf{w}$ again be the incidence matrix and weight vector of the localized reduced directed cut graph. A p-norm local hypergraph diffusion is:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{w}^T \ell(\boldsymbol{B}\mathbf{x}) + \kappa\gamma \textstyle\sum_{i\in V} x_i d_i \\ \text{subject to} \quad & x_s = 1, x_t = 0, \mathbf{x} \geq 0. \end{aligned} \tag{6.19}$$

Here $\ell(x) = \frac{1}{p}(x)_+^p$, $1 < p \leq 2$.

The idea of solving (6.19) is similar to the quadratic case, where the goal is to iteratively push values to x_i as long as node i violates the optimality condition, i.e. $r_i > \kappa d_i$. The challenge of solving a more general p-norm cut objective is that we no longer have a closed form solution even if the ordering of adjacent nodes is known. Thus, we need to use binary search to find Δx_i, Δx_a and Δx_b up to ε accuracy at every iteration. This means that in

the worst case, the general push process can be slower than 2-norm based push process by a factor of $O(\log(1/\varepsilon))$.

Algorithm 7 `p-norm-hyperpush`$(i, \gamma, \kappa, \mathbf{x}, \mathbf{r}, \rho, \epsilon)$

1: Use binary search to find Δx_i such that the ith coordinate of residual after adding Δx_i to x_i, $r_i' = \rho\kappa d_i$, the binary search stops when the range of Δx_i is smaller than ε *(satisfy KKT at i)*.
2: Update $\mathbf{x}$ and $\mathbf{r}$, $x_i \leftarrow x_i + \Delta x_i$, $r_i \leftarrow \rho\kappa d_i$,
3: For each pair of adjacent auxiliary nodes $a \in V_a$, $b \in V_b$ where $a \to b$, update the corresponding residuals
$r_a \leftarrow r_a - \frac{1}{\gamma} w_{ia} \ell'((x_i - x_a)_+) + \frac{1}{\gamma} w_{ia} \ell'((x_i + \Delta x_i - x_a)_+)$ $r_b \leftarrow r_b + \frac{1}{\gamma} w_{bi} \ell'((x_b - x_i)_+) + \frac{1}{\gamma} w_{bi} \ell'((x_b - x_i - \Delta x_i)_+)$

Algorithm 8 `p-norm-auxpush`$(i, a, b, \gamma, \mathbf{x}, \mathbf{r}, \epsilon)$

1: Use binary search to find Δx_a, Δx_b up to accuracy ϵ such that $r_a' = 0$ and $r_b' = 0$ *(satisfy KKT at a, b)*.
2: $x_a \leftarrow x_a + \Delta x_a$ and $x_b \leftarrow x_b + \Delta x_b$
3: For each adjacent node i where $i \in V$, $r_i \leftarrow r_i + \frac{1}{\gamma} w_{ia} \ell'((x_i - x_a)_+) - \frac{1}{\gamma} w_{ia} \ell'((x_i - x_a - \Delta x_a)_+) - \frac{1}{\gamma} w_{bi} \ell'((x_b - x_i)_+) + \frac{1}{\gamma} w_{bi} \ell'((x_b + \Delta x_b - x_i)_+)$

Runtime guarantee.

Bounding the number of times to call `p-norm-hyperpush` (7) is similar. Whenever there exists $g_i' > \kappa d_i$, we need to find Δx_i such that $g_i' = \rho\kappa d_i$. And g_i' can be rewritten as:

$$g_i' = \frac{1}{\gamma} \sum_{b \in V_b} w_{bi} (x_b - x_i - \Delta x_i)_+^{p-1} - \frac{1}{\gamma} \sum_{a \in V_a} w_{ia} (x_i + \Delta x_i - x_a)_+^{p-1} + \kappa d_i \left| (\mathbf{1}_R)_i - x_i - \Delta x_i \right|^{p-1} \operatorname{sgn}((\mathbf{1}_R)_i - x_i - \Delta x_i) \quad (6.20)$$

Here $\mathbf{1}_R$ is a binary vector where $(\mathbf{1}_R)_i = 1$ if $i \in R$ and $(\mathbf{1}_R)_i = 0$ otherwise. For $1 < p < 2$, from lemma 4 of [18], we know $(x_b - x_i - \Delta x_i)_+^{p-1} \geq (x_b - x_i)_+^{p-1} - 2^{2-p} (\Delta x_i)^{p-1}$,

$(x_i + \Delta x_i - x_a)_+^{p-1} \leq (x_i - x_a)_+^{p-1} + 2^{2-p}(\Delta x_i)^{p-1}$ and $|(\mathbf{1}_R)_i - x_i - \Delta x_i|^{p-1} \operatorname{sgn}((\mathbf{1}_R)_i - x_i - \Delta x_i) \geq |(\mathbf{1}_R)_i - x_i|^{p-1} \operatorname{sgn}((\mathbf{1}_R)_i - x_i) - 2^{2-p}(\Delta x_i)^{p-1}$. Thus we have the following inequality:

$$g_i' \geq g_i - \frac{2^{2-p}}{\gamma}(\sum_{b \in V_b} w_{bi} + \sum_{a \in V_a} w_{ia})(\Delta x_i)^{p-1} - 2^{2-p}\kappa d_i (\Delta x_i)^{p-1}$$

which gives $\Delta x_i \geq (\gamma\kappa(1-\rho))^{1/(p-1)}/(\gamma\kappa + \delta)^{1/(p-1)}$. On the other hand, $\left\|\mathbf{g}\right\|_1$ now can be written as:

$$\left\|\mathbf{g}\right\|_1 = \sum_{i \in R} d_i (1 - x_i)^{p-1} - \sum_{i \in \bar{R}} d_i x_i^{p-1}$$

Also from lemma 4 of [18], $(1 - x_i - \Delta x_i)^{p-1} \leq (1 - x_i)^{p-1} - (p-1)\Delta x_i$ and $(x_i + \Delta x_i)^{p-1} \geq x_i^{p-1} + (p-1)\Delta x_i$ so the decrease of $\left\|\mathbf{g}\right\|_1$ will be at least $(p-1)d_i(\gamma\kappa(1-\rho))^{1/(p-1)}/(\gamma\kappa + \delta)^{1/(p-1)}$. If we need to call `p-norm-hyperpush` (7) T times, then

$$\sum_{i=1}^{T} d_i \leq \frac{(\gamma\kappa + \delta)^{1/(p-1)}}{(p-1)(\gamma\kappa(1-\rho))^{1/(p-1)}} \mathbf{vol}(R) = O(\mathbf{vol}(R))$$

6.8 Experimental Results

In the experiments, we will investigate both the LHQD (2-norm) and 1.4-norm cut objectives with the δ-linear threshold as the splitting function (more details about this function in Section 6.5). Our focus in this experimental investigation is on the use of the methods for semi-supervised learning. Consequently, we consider how well the algorithms identify "ground truth" clusters that represent various known labels in the datasets when given a small set of seeds. (We leave detailed comparisons of the conductances to a longer version.)

In the plots and tables, we use LH-2.0 to represent our LHQD or LHPR method and LH-1.4 to represent the 1.4 norm version from Section 6.7. The other methods we are comparing against include: (i) ACL [24], which is initially designed to compute approximated PageRank on graphs. Here we transform each hypergraph to a graph using three different techniques, which are star expansion (star+ACL), unweighted clique expansion (UCE+ACL) and weighted clique expansion (WCE+ACL) where a hyperedge e is replaced by a clique where each edge has weight $1/|e|$ [83]. We chose to run ACL on the expanded graphs

because ACL is known as one of the fastest and most successful local graph clustering algorithm in several benchmarks [11, 18] and has a similar quadratic guarantee on local graph clustering [24, 34]. (ii) flow [23], which is the maxflow-mincut based local method designed for hypergraphs. Since the flow method has difficulty growing from small seed set as illustrated in the yelp experiment in Section 6.2, we will first use the one hop neighborhood to grow the seed set. (OneHop+flow) To limit the number of neighbors included, we will order the neighbors using the *BestNeighbors* as introduced in [23] and only keep at most 1000 neighbors. (Given a seedset R, *BestNeighbors* orders nodes based on the fraction of hyperedges incident to v that are also incident to at least one node from R.) (iii) LH-2.0+flow, this is a simple combination of LH-2.0 and flow where we use the output of LH-2.0 as the input of the flow method to refine it. (iv) HGCRD [84], this is a hypergraph generalization of CRD [8], which is a hybrid diffusion and flow.[1]

In order to select an appropriate δ for different datasets, Veldt et al. found that the optimal δ is usually consistent among different clusters in the same dataset [23]. Thus, the optimal δ can be visually approximated by varying δ for a handful of clusters if one has access to a subset of ground truth clusters in a hypergraph. We adapt the same procedure in our experiments and report the results in Section 6.8.4. Other parameters are in the reproduction details footnote.[2]

6.8.1 Detecting Amazon Product Categories

In this experiment, we use different methods to detect Amazon product categories [93]. The hypergraph is constructed from Amazon product review data where each node represents a product and each hyperedge is set of products reviewed by the same person. It has 2,268,264

[1]↑Another highly active topic for clustering and semi-supervised learning involves graph neural networks (GNN). Prior comparisons between GNNs and diffusions shows mixed results in the *small seed set* regime we consider [7, 18] and complicates doing a fair comparison. As such, we focus on comparing with the most directly related work.

[2]↑*Reproduction details.* We fix the LH locality parameter γ to be 0.1, approximation parameter ρ to be 0.5 in all experiments. We set $\kappa = 0.00025$ for Amazon and $\kappa = 0.0025$ for Stack Overflow based on cluster size. For ACL, we use the same set of parameters as LH. For LH-2.0+flow, we set the flow method's locality parameter to be 0.1. For OneHop+flow, we set the locality parameter to be 0.05, 0.0025 on Amazon and Stack Overflow accordingly. For HGCRD, we set $U = 3$ (maximum flow that can be send out of a node), $h = 3$ (maximum flow that an edge can handle), $w = 2$ (multiplicative factor for increasing the capacity of the nodes at each iteration), $\alpha = 1$ (controls the eligibility of hyperedge), $\tau = 0.5$ and 6 maximum iterations.

nodes and 4,285,363 hyperedges. The average size of hyperedges is around 17. We select 6 different categories with size between 100 and 10000 as ground truth clusters used in [23]. The statistics of these 6 clusters are summarized in table 6.1 where the conductance is computed under the standard *all-or-nothing* penalty. We set $\delta = 1$ for this dataset (more details about this choice in §6.8.4). We select 1% nodes (at least 5) as seed set for each cluster and report median F1 scores and median runtime over 30 trials in Table 6.2 and 6.3. Overall, LH-1.4 has the best F1 scores and LH-2.0 has the second best F1. The two fastest methods are LH-2.0 and star+ACL. While achieving better F1 scores, LH-2.0 is 20x faster than HyperLocal (flow) and 2-5x faster than clique expansion based methods.

Table 6.1. Statistics on Amazon product categories

Cluster label	Cluster name	Size	Conductance
12	Gift Cards	148	0.132
18	Magazine Subscriptions	157	0.132
17	Luxury Beauty	1581	0.109
25	Software	802	0.137
15	Industrial & Scientific	5334	0.142
24	Prime Pantry	4970	0.097

Table 6.2. Median F1 scores on detecting Amazon product categories over 30 trials, the small violin plots show variance.

Alg	12 F1 & Med.	18 F1 & Med.	17 F1 & Med.	25 F1 & Med.	15 F1 & Med.	24 F1 & Med.
LH-2.0	0.77	0.65	0.25	0.19	0.22	0.62
LH-1.4	0.9	0.79	0.32	0.22	0.27	0.77
LH-2.0+flow	0.95	0.82	0.15	0.16	0.16	0.87
star+ACL	0.64	0.51	0.19	0.15	0.2	0.49
WCE+ACL	0.64	0.51	0.2	0.14	0.21	0.51
UCE+ACL	0.27	0.09	0.06	0.05	0.11	0.14
OneHop+flow	0.52	0.6	0.16	0.12	0.09	0.22
HGCRD	0.56	0.4	0.05	0.06	0.07	0.17

Table 6.3. Median runtime in seconds on detecting Amazon product categories

Alg	12	18	17	25	15	24
LH-2.0	0.9	0.7	2.8	1.0	5.6	13.3
LH-1.4	8.0	6.3	32.3	9.8	53.8	127.3
LH-2.0+flow	3.5	5.1	421.1	17.8	34.9	151.5
star+ACL	0.2	0.2	0.3	0.2	0.5	0.8
WCE+ACL	18.6	17.2	19.0	16.5	21.5	20.1
UCE+ACL	9.8	10.9	11.2	10.7	13.3	15.5
OneHop+flow	308.8	141.7	359.2	224.9	81.5	82.4
HGCRD	120.3	56.4	78.1	21.2	239.4	541.3

6.8.2 Detecting Stack Overflow Question Topics

In the Stack Overflow dataset, we have a hypergraph with each node representing a question on "stackoverflow.com" and each hyperedge representing questions answered by the same user [23]. Each question is associated with a set of tags. The goal is to find questions having the same tag when seeding on some nodes with a given target tag. This hypergraph is much larger with 15,211,989 nodes and 1,103,243 edges. The average hyperedge size is around 24. We select 40 clusters with 2,000 to 10,000 nodes and a conductance score below 0.2 using the *all-or-nothing* penalty. (There are 45 clusters satisfying these conditions, 5 of them are used to select δ in §6.8.4.) In this dataset, a large threshold can give better results. For diffusion based method, we set the δ-linear threshold to be 1000 (more details about this choice in §6.8.4), while for flow based method, we set $\delta = 5000$ based on Figure 3 of [23]. In Table 6.4, we summarize some recovery statistics of different methods on this dataset. In Figure 6.3, we report the performance of different methods on each cluster. Overall, LH-2.0 achieves the best balance between speed and accuracy, although all the diffusion based methods (LH, ACL) have extremely similar F1 scores (which is different from the last experiment). The flow based method still has difficulty growing from small seed set as we can see from the low recall in Table 6.4.

Table 6.4. This table summarizes the median of median runtimes in seconds for the Stack Overflow experiments as well as median Precision, Recall and F1 over the 40 clusters.

Alg.	LH2	LH1.4	LH2 +flow	ACL +star	ACL +WCE	ACL +UCE	Flow +1Hop	HG-CRD
Time	3.69	39.89	43.84	1.54	15.25	13.71	48.28	72.31
Pr	0.65	0.66	0.74	0.66	0.65	0.66	0.83	0.46
Rc	0.67	0.67	0.59	0.6	0.66	0.65	0.11	0.01
F1	0.66	0.66	0.66	0.63	0.65	0.65	0.19	0.02

6.8.3 Varying Number of Seeds

Comparing to the flow-based method, an advantage of our diffusion method is that it expands from a small seed set into a good enough cluster that detects many labels correctly. Now, we use the Amazon dataset to elaborate on this point. We vary the ratio of seed set from 0.1% to 10%. At each seed ratio, denoted as r, we set $\kappa = 0.025r$. And for each of the 6 clusters, we take the median F1 score over 10 trials. To have a global idea of how different methods perform on this dataset, we take another median over the 6 median F1 scores. For the flow-based method, we also consider removing the OneHop growing procedure. The results are summarized in Figure 6.4. We can see our hypergraph diffusion based method (LH-1.4, LH-2.0) performs better than alternatives for all seed sizes, although flow dramatically improves for large seed sizes.

6.8.4 Selecting δ

To select δ for each dataset, we run LH-2.0 on a handful of alternative clusters as we vary δ. In Figure 6.5, we show F1 scores on those clusters and pick $\delta = 1$ for Amazon and $\delta = 1000$ for Stack Overflow.

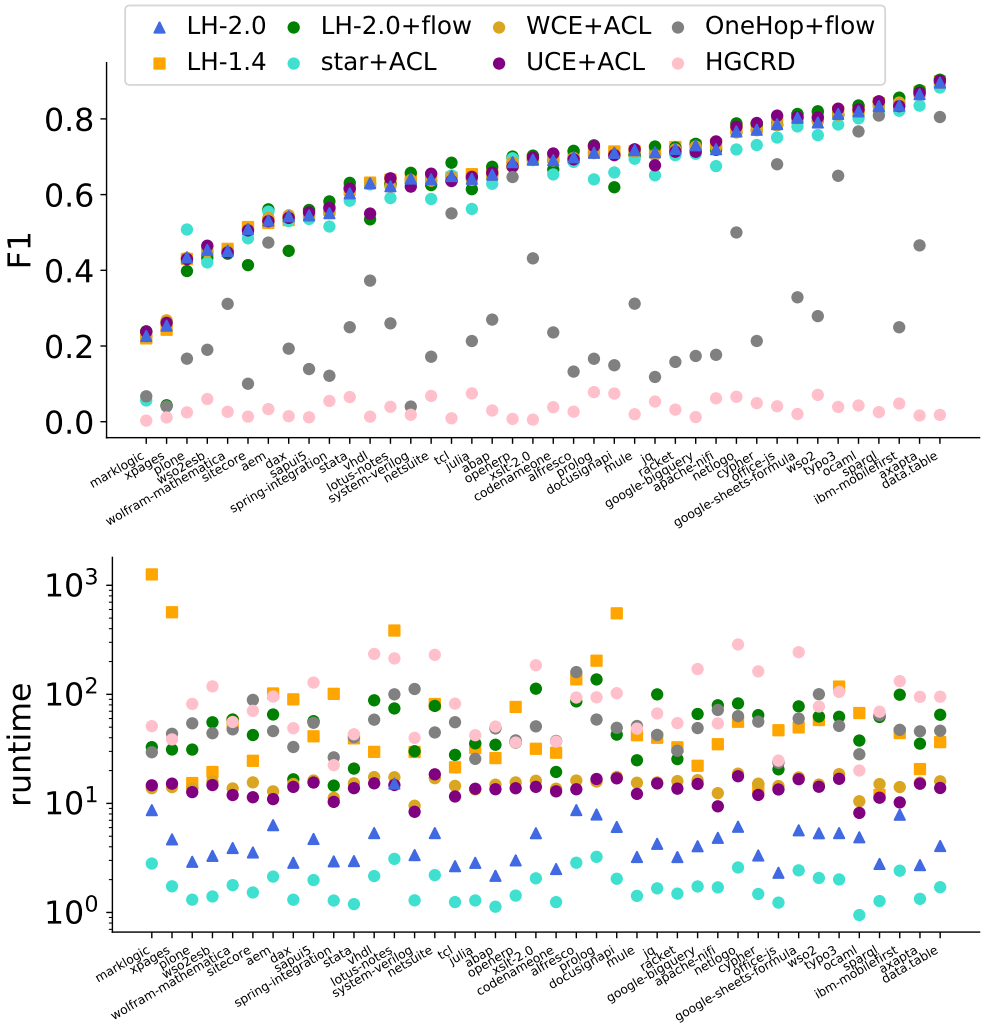


Figure 6.3. The upper plot shows median F1 scores of different methods over 40 clusters from the Stack Overflow dataset. The lower plot shows median running time. LH-2.0 achieves the best balance between speed and accuracy; LH-1.4 can sometimes be slower than the flow method when the target cluster contains many large hyperedges.

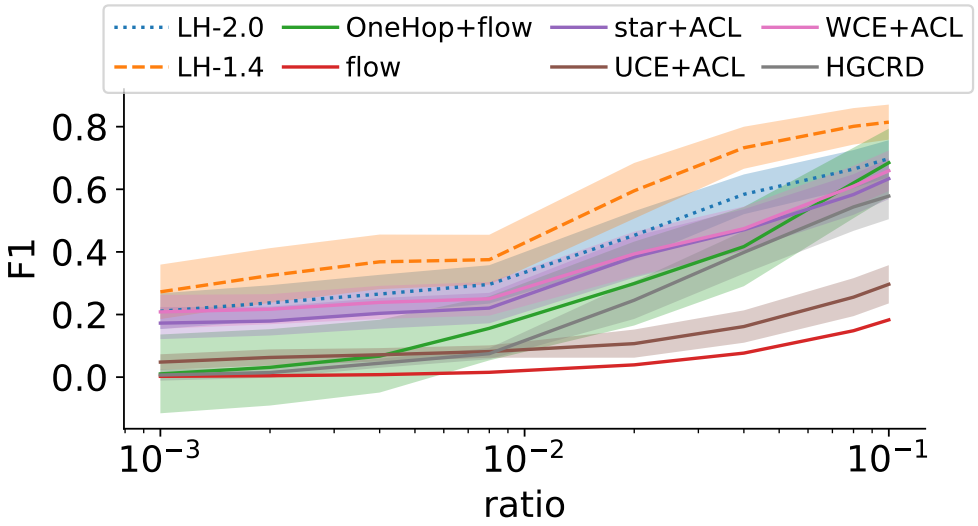
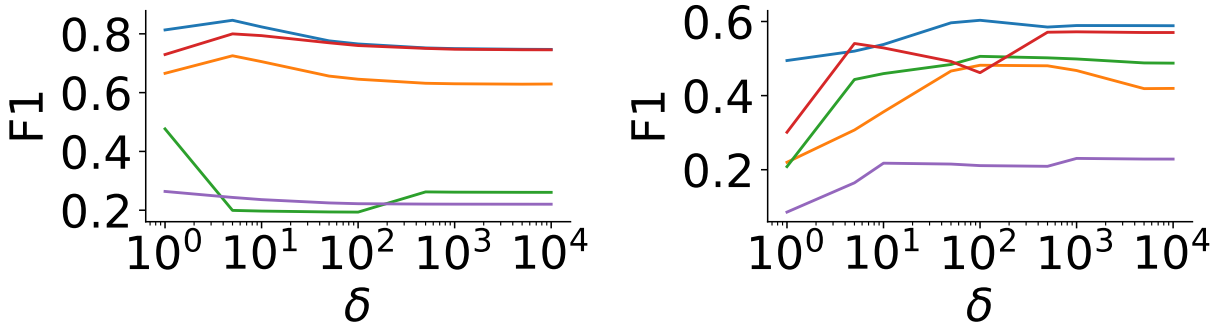


Figure 6.4. This plot shows the median of median F1 scores on detecting those 6 clusters in the Amazon data when varying the seed size. The envelope represents 1 standard error over the 6 median F1 scores. Without OneHop, the flow based method is not able to grow from seed set even for the largest seeds. Our hypergraph diffusion (LH) methods outperforms others, especially for small seeds.



(a) Amazon

(b) Stack Overflow

Figure 6.5. Selecting proper δ for Amazon and Stack Overflow.

7. COMBINING TOPOLOGICAL DATA ANALYSIS AND DIFFUSIONS FOR ANALYZING PREDICTIONS

7.1 Motivation and background on TDA

Deep learning is a successful strategy where a highly parameterized model makes human-like predictions across many fields [94–97].

Yet challenges in generalization may keep deep learning from use in practice [98, 99]. Detailed prediction mechanisms are also difficult to assess directly due to the large collection of model parameters.

Prior work on analyzing deep learning methods for errors focuses on a single result list [100]. Other research work seeks to explain model predictions by highlighting areas that make the most contribution to the final prediction [101–105] and the user needs to check if the highlighted areas make sense. Another set of studies take a different approach by training a simple and explainable model (i.e. a linear classifier) to mimic the prediction functions of the original model [106]. However, all these approaches can only explain the model's prediction on a single sample each time instead of model's prediction ability in the entire dataset. The dataset can have millions of training or testing samples, which makes checking the explanation or interpretation of all samples impossible. An alternate approach is to select representative samples from the entire dataset [106], but checking each selected sample is still required and the quality of the conclusion highly depends on how representative the selected samples are.

In this chapter, we attempt to address this issue by transforming the prediction space of complex models into a prediction map, represented in a graph. Subsequent analysis using diffusion on this graph can enable easy inspection over the original dataset. Before diving into our method, we will first introduce topological data analysis (TDA) and the *mapper* algorithm [107].

7.1.1 Background: Topological Data Analysis and the Mapper Algorithm

Topological methods of data analysis excel at distilling representation invariant information from large datasets [107–109]. Mapper builds a discrete approximation of a Reeb graph or Reeb space (see Figure 7.1). It begins with a set of datapoints $(x_1, \ldots, x_n)$, along with a single or multi-valued function sampled at each datapoint. The set of all these values $\{f_1, \ldots, f_n\}$ samples a map $f : X \to \mathbb{R}^k$ on a topological space X. The map f is called a *filter* or *lens*. The idea is that when f is single valued, a Reeb graph shows a quotient topology of X with respect to f and mapper discretizes this Reeb graph using the sampled values of f on points $x_1, \ldots, x_n$. Algorithmically, *mapper* consists of the steps:

- Sort the values f_i and split them into overlapping bins $B_1, \ldots, B_r$ of the same size.
- For each bin of values B_j, let S_j denote the set of datapoints with that same value and *cluster* the datapoints in each S_j independently. (That is, we run a clustering algorithm on each S_j as if it was the entire dataset.)
- For each cluster found in the previous step, create a node in the Reeb graph.
- Connect nodes of the Reeb graph if the clusters they represent share a common point.

The resulting graph **is a discrete approximation of the** *Reeb graph* and represents a compressed view of the shape underlying the original dataset.

In this chapter, we will propose a framework to extract a type of topological description similar to *mapper* to analyze the prediction space of complex models. Our framework can be easily adapted to models and datasets from various sources like images, DNA sequences or co-purchasing graphs etc. We will mainly use lenses that come from the prediction function and that are multi-valued, which we interpret as a collection of single-valued lenses (See Figure 7.2C as an example). The input to our method is no longer a point cloud, but a graph. Comparing to point cloud, graphs are even more general and datasets not in graph format like images or DNA sequences can be easily transformed into graphs first extracting intermediate outputs of the model as embeddings and then building a nearest neighbor graph from the embedding matrix. Another advantage is graph format facilitates easy clustering:

for each subset of points, we extract the subgraph induced by those points and then use a parameter-free connected components analysis to generate clusters. The resulting graph of our graph-based topological data analysis (GTDA), called *Reeb network*, is a discrete approximation of topological structures called Reeb spaces, which generalize Reeb graphs (See Figure 7.1 as an example).

7.1.2 Reeb graph vs. Reeb space vs. Reeb network

The main difference between a Reeb graph and Reeb network is the number of lenses used because the Reeb net involves a multivalued map which can be thought of as a collection of single valued maps. A demonstration to show this difference can be found in Figure 7.1.

Formally, let $f : X \to \mathbb{R}^k$ map a topological space X to a k-dimensional real space. Two points $x, y \in X$ are called equivalent if (i) $f(x) = f(y)$ and (ii) they belong to the same connected component of the level set $f^{-1}(f(x))$. Denoting this equivalence relation with $\sim$, we obtain the quotient space $R_X^f = X/\sim$. When the range of f is $\mathbb{R}$, R_X^f is a one-dimensional space called the Reeb graph of f. When f is multi-valued, that is, $k > 1$, R_X^f becomes a space called Reeb space. By choosing the bins in $\mathbb{R}^k$, we discretize this Reeb space with a graph which we call the *Reeb net* here. We choose the term Reeb net to distinguish it from discretized Reeb graph because both are graphs but one discretizes a one-dimensional space (a graph) and the other discretizes a quotient space that need not be one-dimensional.

7.1.3 Existing work of using topology in neural networks

Despite its success in extracting insights from the shape of complex data [107], topological data analysis (TDA) of complex predictive models such as deep learning remains in its infancy and lack comprehensive studies in real world problems. Naitzat et al. [111] studies how the topology of a two class classification dataset changes as it passes through different layers of a neural network and finds some activation functions can simplify the topology faster than others. TopoAct [112] studies the shape of activation space at a given layer of a neural network to provide insights on the learned representations and mainly focuses on images and NLP. Hence each sample in the space is just a randomly chosen activation

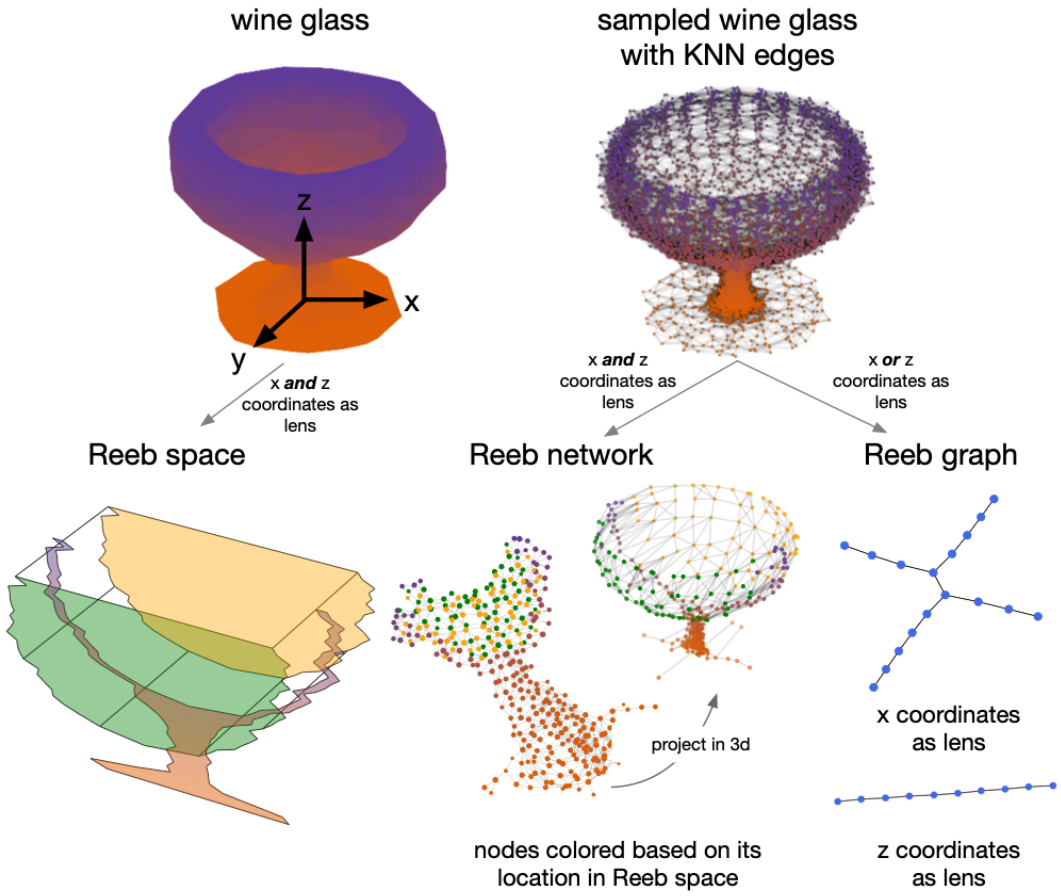


Figure 7.1. This illustrates the difference between a Reeb graph and a Reeb network on a topologically interesting object. The lenses we use here are the x and z coordinates. The inspiration for the object is [110].

vector that only represents a part of the input. Gabrielsson et al. [113] studies the topology of the learned weights in a neural network, especially convolutional neural networks, and shows that its topological structure strongly correlates with the models generalization ability on unseen data. Some other work like TDA-Net [114] or TCNN [115] tries to improve a models performance by incorporating the topological features into the training process of a traditional convolutional neural network.

7.1.4 Chapter Overview

In this chapter, we developed a graph-based topological data analysis (GTDA) framework to inspect the predictions of complex models (See Figure 7.2 as an example). The first step is to construct a Reeb net to visualize the interactions between predictions and data (Figure 7.2D), which is then followed by a diffusion process on the resulting Reeb net (Figure 7.2F). The framework has the following properties:

- it can produce a topological view of the original dataset through pictures
- the visualization can provide clues for any sample of interest to be inspected
- it is highly scalable and can process large datasets with thousands of classes
- it can provide intuitive insights and suggest places that are worth a further study for users without any prior knowledge on the model or the data

More specifically, this chapter is organized as the following sections:

- Section 7.2 presents the algorithmic details on how to build a Reeb network from a graph and a prediction function.
- Section 7.3 demonstrates how we use apply diffusion on the resulting Reeb net to highlight areas of the original dataset that are worth looking.
- Sections 7.4 to 7.8 provide comprehensive experiments by applying GTDA to multiple real world datasets and models. We show that our method can scale up to large dataset across different domains and enable us to detect labeling issues in training

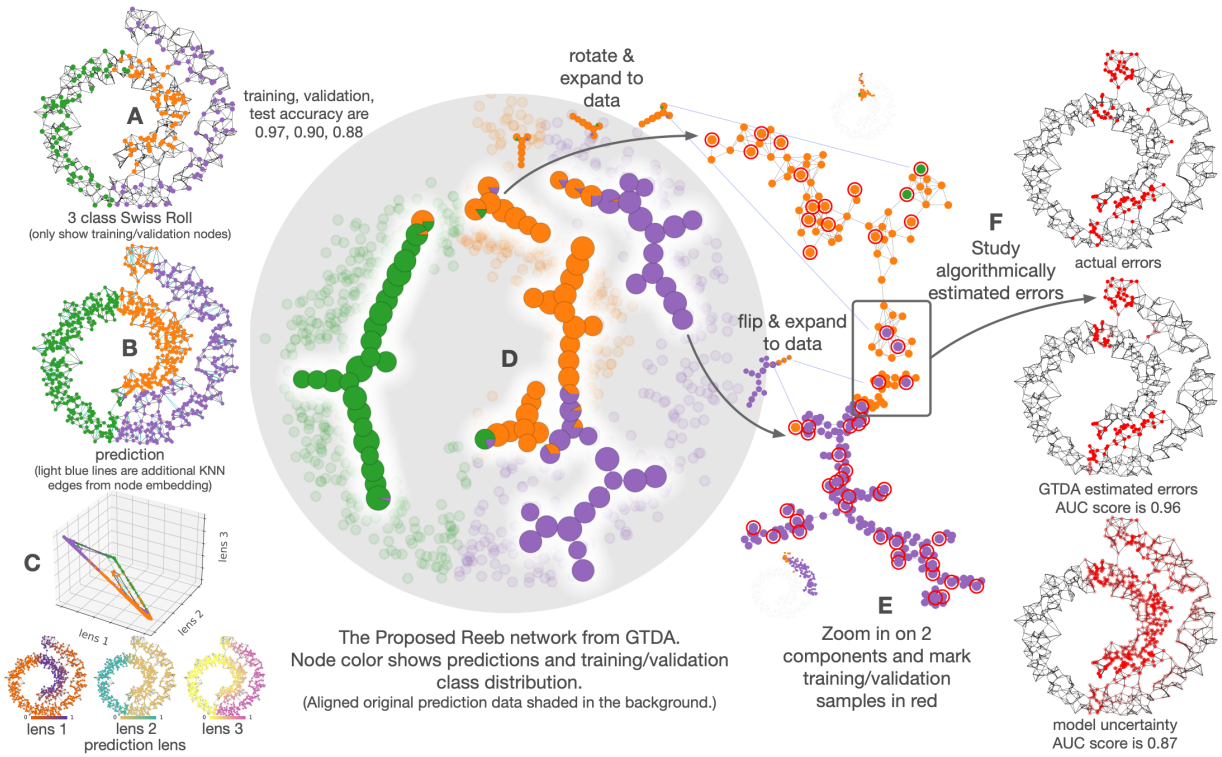


Figure 7.2. Consider a prediction scenario with three classes in a Swiss roll structure and an underlying graph (A). Graph neural network predictions show reasonable accuracy (B). The 3-dimensional prediction lens from the neural network is shown in (C) and gives a guide to class predictions. The Reeb network is shown in (D). Each node maps to a small cluster of similar values of the lens. Nodes are colored by the fraction of points in each predicted class. Regions with mixed predictions indicate potential ambiguities in the results. Further study of two such connected regions (E) shows many datapoints where there are training points with different labels closer to the given test points. This situation motivates an algorithmic error estimate for each datapoint without ground truth (F). This estimate is nevertheless highly correlated with true errors and better than class uncertainty estimates. The topological simplification highlights datapoints with confusing or ambiguous predictions given the totality of predictions.

data, understand generalization in image classification, inspect predictions of likely pathogenic mutations in the BRCA1 gene and many more.

- Section 7.9 discusses parameter selection of GTDA.
- Section 7.10 reports the runtime and scaling of GTDA on datasets in various sizes with different number of lenses.
- Section 7.11 compares GTDA to other visualization techniques including tSNE and UMAP. The results show that our framework can permit many types of analysis not clearly possible with tSNE and UMAP output and is faster.

7.2 The Reeb network construction on a prediction function using a graph

We take as input:

1. an n-node graph G
2. a set of m lenses based on a prediction model as an $n \times m$ matrix $\boldsymbol{P}$

The lenses we use are the prediction matrix $\boldsymbol{P}$ of a model where P_{ij} is the probability that sample i belongs to class j. Key differences from existing studies of TDA frameworks on graphs include using the connected components of each bin [116, 117] as clusters and also additional steps to improve the analysis of prediction functions by adding weak connections from a minimum spanning tree.

Problems with straightforward algorithmic adaptation

Mapper does extend to multidimensional lens functions by using a tensor product bin construction. We found issues with a straightforward adaptation of *mapper* to such multidimensional input for prediction functions. In our extensive trials, we found that most of the resulting Reeb networks end up with too many tiny components or even singletons where no prediction-specific insights were possible. This is especially so when the dataset has many classes, most multi-dimensional bins will just contain very few samples because the space

grows exponentially, limiting the potential of overlap to find relationships. Simply reducing the dimension of $\boldsymbol{P}$ with PCA will lose the interpretability of the lens. Moreover, classic *mapper* uses a fixed bin size and density-based or multi-scale alternatives [118] were unsuccessful in our investigations although they solve this problem from a theoretical perspective. (We note this is a potential area for followup work to better understand why.)

Preprocessing to smooth the predictions

As a preprocessing step, we apply a few steps (usually four or five) of the smoothing iteration: $\boldsymbol{P}^{(i+1)} = (1-\alpha)\boldsymbol{P} + \alpha \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{P}^{(i)}$. Here $\boldsymbol{P}^{(0)} = \boldsymbol{P}$, $\boldsymbol{A}$ is the adjacency matrix of the input graph, $\boldsymbol{D}$ is the diagonal degree matrix and $0 < \alpha < 1$. This helps to prevent prevent sharp changes between adjacent nodes. This equation is a diffusion-like equation closely related to the PageRank vector that is known to smooth data over graphs and has many uses [25]. The iteration keeps all the prediction data non-negative and the smoothed $\boldsymbol{P}$ will also be min-max column normalized so that each value is between 0 and 1. As is standard, this setup can use any weights associated with the adjacency matrix, or remove them and use an unweighted graph.

Our graph-based construction for a prediction function

Our GTDA approach uses a recursive splitting strategy to build the bins in the multi-dimensional space. For each subgroup of data, the idea is that we find the lens that has the maximum difference on those data. Then split the component by putting nodes into two approximately equal sized overlapping bins based on the values in this lens. Then if the resulting groups are big enough, we add them back as sets to consider splitting.

Detailed pseudo code can be found in Algorithm 9. We give a quick outline here. The recursive splitting starts with the set of connected components in the input graph. This is a set of sets: $\mathbb{S}$. The key step is when the algorithm takes a set $\mathbb{S}_i$ from $\mathbb{S}$, it splits $\mathbb{S}_i$ into new (possibly) overlapping sets $\mathbb{T}_1, \ldots, \mathbb{T}_h$ based on the lens with maximum difference in value on $\mathbb{S}_i$ and also connected components. Each $\mathbb{T}_i$ is then either added $\mathbb{S}$ if it is large enough

(with more than K vertices) and where there exists a lens with maximum difference larger than d. Otherwise, $\mathbb{T}_i$ goes into the set of finalized sets $\mathbb{F}$.

Once we have the final set of sets, $\mathbb{F}$, then we do have two final merging steps, along with building the Reeb net. The first is to merge sets in $\mathbb{F}$ if they are too small (Algorithm 10). The second is to add edges to the Reeb net to promote more connectivity (Algorithm 11).

In the first merging (Algorithm 10), which occurs before the Reeb net is constructed, we check and see if any set in $\mathbb{F}$ is too small (smaller or equal to s_1). If so, then we find nearby nodes based on the input graph G and based on a user-provided distance measure f and merge the small component with the closest component (giving preference to the smallest possible set to merge into). This could be a simple graph-distance measure (e.g. shortest path), something suggested by the domain, or a weight based on the prediction values (what we use). The algorithm is closely related to Borůvka's algorithm for a minimum spanning tree.

Next, we build the Reeb net $\hat{G}$ from this set of sets $\mathbb{F}$. Each group $\mathbb{F}_i$ becomes a node, and nodes are connected if they share any vertex.

In the second merging (Algorithm 11) we seek to improve the overall connectivity of the Reeb net by connecting small disconnected pieces of the Reeb net $\hat{G}$. This step is designed to enhance the ability to work with predictions by adding weaker connections to the more strongly connected topological pieces. It uses the same distance measure f to find components and uses a similar Borůvka-like strategy. We save the set of edges added at this step to study in the error estimation procedures noted below.

Choices for the parameters

As a result, GTDA has 8 parameters as in Table 7.1. Tuning of the parameters is straightforward, and we often use the default choice or values from a small set. The values K, d and s_1 provide direct control about the number of nodes in the final group visualization, while r and s_2 control how connected we want the visualization to be. In practice, we could first tune K and d to determine the number of nodes, then tune r so that no component in the Reeb net is too large and finally tune s_1, s_2 to remove any tiny nodes or components.

We leave the smoothing parameters fixed at $\alpha = 0.5$ and $S = 5$ or 10 (very smooth). A detailed discussion on these parameters can be found in Section 7.9.

Choice of distance function for merging

Possibly the hardest parameter to pick is the merging function f. The user can choose any distance metric f in the merging step, in our experiments, we use ℓ^∞ norm of the difference between rows of the preprocessed $\boldsymbol{P}$ as the distance between 2 samples, which roughly means how much larger the bin containing one of those 2 samples should be in order to include the other sample. Put another way, this choice makes us less sensitive to the exact choice for r because we will add small connections that would have been included in a slightly larger bin.

Drawing the graph

Unless otherwise specified, all coordinates of any layout we show are computed with Kamada-Kawai algorithm [119].

Showing the Reeb network and explorations

In the Reeb net of a prediction function, we draw each node as a small pie-chart. The size of the pie-chart represents the number of nodes. The pieces of the pie show the local prediction distribution. In some cases, we find it useful to study the predicted labels directly, such as when studying mechanisms underlying the prediction. In other cases, we find it useful to study predictions and training data, such as when studying possible errors. These visualizations facilitate exploring regions of the prediction landscape based on interactions among predicted values and training data. By mapping these *small* regions back to the original data, it suggests what the model is utilizing to make the predictions. Examples on this can be found in the experiments in the main text as well as in the supplemental information.

Table 7.1. List of parameters in GTDA.

parameter	description	suggested choices
K	component size threshold to stop splitting	5% of smallest class size
d	lens difference threshold to stop splitting	0 or 0.001
r	overlapping ratio	0.01
s_1	Reeb node size threshold	5
s_2	Reeb component size threshold	5
α	lens smoothing parameter	0.5 (used in all experiments)
S	lens smoothing steps	5 or 10
f	distance function in the merging step	ℓ^∞ difference of row i, j of $\boldsymbol{P}^{(S)}$

7.2.1 Demonstration of GTDA

We use a 3 class Swiss roll dataset to demonstrate each step of our GTDA framework (plot (A) of figure 7.3). For the GTDA parameters, we set $K = 20$, $d = 0$, $r = 0.1$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$, and $S = 5$. In (B), we show the three prediction lenses we use in the top plot as well as the predicted labels of the model we use. We also add additional edges based on nearest neighbors from node embeddings to take node features into account. This is standard practice in graph neural network methods. Details on the dataset and the model can be found in Section 7.2.2. Each lens is just the prediction probability of a class after smoothing and normalization. In (C), we pick the lens with the largest min-max difference and split it into 2 bins with 10% overlap (we pick the one with smaller index to break ties). This round of splitting finds 2 components. For each component found in the first iteration, we pick the lens with the largest min-max difference and split it again. In this case, the inner component is split along lens 3 while the outer component is split along lens 2. This round of splitting further divides the graph into 7 components. We repeat the splitting until no component has more than 20 vertices of the original graph.

In the end, we find 247 unique components. As noted above, we use a pie chart to represent each Reeb node and connect Reeb nodes with black lines if they have any samples in common to get the initial Reeb net, (D). Node size is proportional to the number of samples it represents, the pie chart shows the distribution over predicted values. This initial

Reeb net has many tiny components or even singletons that are a barrier to deeper insights; the merging steps address this issue. In (E), we use red dashed lines to mark how we will merge those small Reeb nodes so that all nodes will contain more than 5 samples. Similarly, we use red dashed lines to mark extra edges that will be added so that each connected component in the Reeb net will contain more than 5 Reeb nodes. The final Reeb net is shown in (F) with the original graph embedded in the background. We can see that all important structures found in (D) are also preserved in (F) such as the mixing of nodes from different classes. And what merging does is to estimate how the tiny nodes and components are connected in the original graph or via the prediction lens so that we have a clear view of predictions over the entire dataset. This supports an inspection of the model's prediction on any sample we want.

As a comparison, in plot (G), we show two Reeb nets that are constructed by the original *mapper* algorithm with different number of bins along each lens. These Reeb nets are not useful to understand the prediction structure. Most samples from the green class are grouped into a few nodes because prediction probability distribution on this class is more skewed, which makes the inspection hard.

7.2.2 Other details

Swiss Roll dataset construction

We use *scikit-learn* package to build the Swiss Roll dataset. We use 1000 samples in total and the noise parameter is set to be 1.2. The initial Swiss Roll dataset is a 1000-by-3 matrix X and a vector y which represents the position of each sample in the main dimension of the manifold. We only keep the first and the third columns of X and use them as node features. And we sort samples based on y and consider the first 33% samples as the first class, the second 33% samples as the second class and all the other samples as the third class. The graph is a nearest neighbor graph with each node connecting to its 5 closest neighbors using Euclidean metric on X. We use a random set of use 10% samples as training, another 10% samples as validation and all the other points as testing.

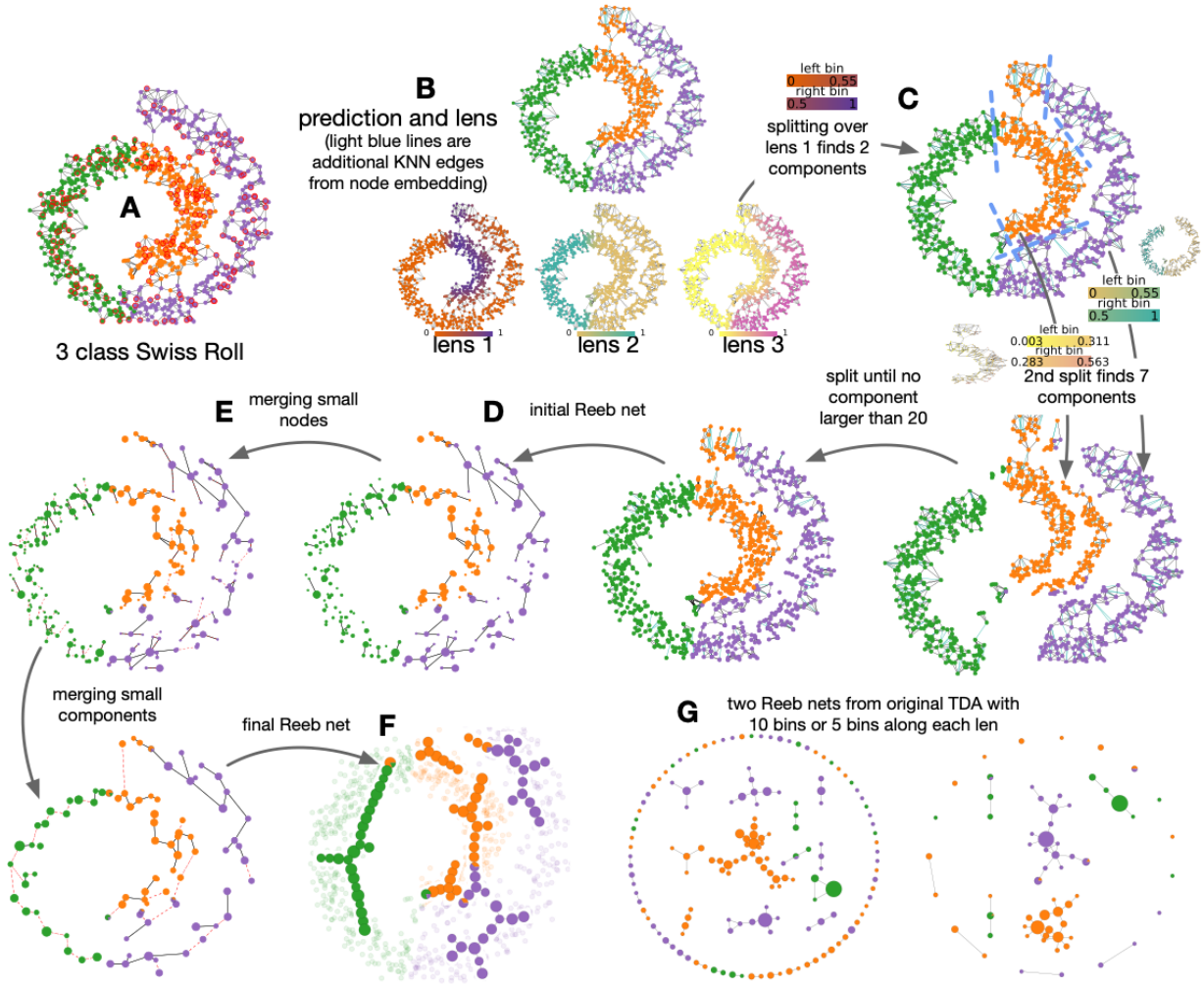


Figure 7.3. A detailed illustration of applying GTDA to build a Reeb net on a 3-class Swiss roll dataset. The original data graph and "ground truth" values are in (A). We show the model prediction for a simple GCN and the three prediction lenses (after smoothing) in (B). The first splitting iteration over lens 1 finds 2 components, (C). At the second split, for each component, we choose the lens with the largest difference, which means the outer ring is split over lens 2 and the inner ring is split over lens 3. The second splitting finds 7 components in total. We continue to split until no more components larger than 20 and get the initial Reeb net, (D). Then small nodes are merged to neighbors iteratively as shown by the red dashed lines in (E). Similarly, small components in the Reeb net are iteratively connected to get the final Reeb net in (F). As a comparison, two Reeb nets from the original *mapper* using 10 lens or 5 lens have many isolated nodes or components and are not suitable for the subsequent inspection. The figure (F) uses predicted classes for training and validation points instead of the actual training and validation classes as in fig. 7.2(D).

Algorithm 9 `GTDA`$(G, \boldsymbol{P}, K, d, r, s_1, s_2, \alpha, S, f)$ See Table 7.1 for parameters description

Smooth $\boldsymbol{P}$ for S steps with $\boldsymbol{P}^{(i+1)} = (1-\alpha)\boldsymbol{P} + \alpha \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{P}^{(i)}$ and $\boldsymbol{P}^{(0)} = \boldsymbol{P}$
Perform a min-max normalization along each column of $\boldsymbol{P}$
Find connected components in G and put all components with size larger than K and maximum lens difference larger than d in $\mathbb{S}$, otherwise in $\mathbb{F}$
while $\mathbb{S}$ is not empty **do**
  Let $\mathbb{S}^{(\text{iter})}$ be a copy of $\mathbb{S}$
  for each $\mathbb{S}_i$ in $\mathbb{S}^{(\text{iter})}$ **do**
    Remove $\mathbb{S}_i$ from $\mathbb{S}$
    Find column c_i (for a lens) such that $\boldsymbol{P}^{(S)}[\mathbb{S}_i, c_i]$ has the largest min-max difference
    Split interval $[\min(\boldsymbol{P}[\mathbb{S}_i, c_i]), \max(\boldsymbol{P}^{(S)}[\mathbb{S}_i, c_i])]$ into two halves of the same length and extend the left half by a ratio of r to give overlapping groups $\mathbb{R}_1$ and $\mathbb{R}_2$ based on which vertices had values in the left and right parts of the interval.
    Create sets $\mathbb{T}_1, \ldots, \mathbb{T}_h$ based on the connected components in $\mathbb{R}_1, \mathbb{R}_2$.
    for each $\mathbb{T}_i$ **do**
      If there are more than K vertices in $\mathbb{T}_i$ and if there is a lens with a maximum difference larger than d, then add $\mathbb{T}_i$ to $\mathbb{S}$. Otherwise, add $\mathbb{T}_i$ to $\mathbb{F}$.
    end for
  end for
end while
Run `node-merging`$(\mathbb{F}, G, s_1, f)$ to get the updated $\mathbb{F}$
Generate Reeb net $\hat{G}$ by considering each component of $\mathbb{F}$ as a Reeb net node and connecting two Reeb net nodes if their corresponding components have overlap
Run `component-merging`$(\mathbb{F}, G, \hat{G}, s_2, f)$ to get the updated $\hat{G}$ and the extra set of edges $\mathbb{E}$
Return $\hat{G}$, $\mathbb{E}$

Model and parameters

We use a standard 2-layer GCN model to predict labels of testing samples. The dimension of the hidden layer is 64, learning rate is 0.01 and weight decay is 10^{-5}. Once the model is trained, we use outputs of the first layer as node embeddings. The embedding matrix is reduced to 16 dimension using PCA with whitening and then each row is ℓ_2 normalized. We build another 2-NN graph using the preprocessed embedding matrix and cosine similarity to encode any information from node features. This graph is combined with the original graph. GTDA framework is then applied on the combined graph. For GTDA parameters, we set

Algorithm 10 `node-merging`$(\mathbb{F}, G, s_1, f)$

while there exists components in $\mathbb{F}$ with at most s_1 vertices **do**
  Set $\mathbb{C}$ to be empty.
  for each component $\mathbb{F}_i$ in $\mathbb{F}$ where $|\mathbb{F}_i| \leq s_1$ **do**
    for each edge (v_i, v_j) in G where $v_i \in \mathbb{F}_i$ and $v_j \in \mathbb{F}_j \neq \mathbb{F}_i$, compute $f(v_i, v_j)$
    Select the pair of nodes v_i, v_j with the smallest $f(v_i, v_j)$. Let $\mathbb{F}_j$ be the set associated with v_j and choose the smallest size F_j if v_j is in multiple such sets. Add $(\mathbb{F}_i, \mathbb{F}_j)$ to $\mathbb{C}$.
  end for
  View the choices in $\mathbb{C}$ as edges of an undirected graph H where vertices are $\mathbb{F}_i$.
  Compute connected components of H.
  for each connected component H_i of H of size larger than 1 **do**
    Let $\mathbb{F}_1, \ldots, \mathbb{F}_h$ be the underlying sets of H_i from $\mathbb{F}$. Remove each $\mathbb{F}_i$ from $\mathbb{F}$. Then add $\mathbb{F}_1 \cup \ldots \cup \mathbb{F}_h$ to $\mathbb{F}$.
  end for
end while
Return the updated $\mathbb{F}$

Algorithm 11 `component-merging`$(\mathbb{F}, G, \hat{G}, s_2, f)$

Initialize the set of extra edges $\mathbb{E}$ to be empty
Compute connected components of Reeb net $\hat{G}$
Let $\mathbb{D}$ be the set of connected components of $\hat{G}$.
while there exists any $\mathbb{D}_i \in \mathbb{D}$ where $\mathbb{D}_i$ has at most s_2 Reeb nodes **do**
  for each $\mathbb{D}_i \in \mathbb{D}$ where $\mathbb{D}_i$ has at most s_2 Reeb nodes **do**
    Let $\mathbb{C}$ be the union of vertices in G (not $\hat{G}$) for Reeb nodes in $\mathbb{D}_i$.
    For each edge $(v_i, v_j) \in G$ where $v_i \in \mathbb{C}$ and $v_j \notin \mathbb{C}$, compute $f(v_i, v_j)$.
    Select the pair of nodes v_i, v_j with the smallest $f(v_i, v_j)$. Let $\mathbb{F}_i$ and $\mathbb{F}_j$ be the Reeb nodes associated with v_i and v_j and choose the smallest size $\mathbb{F}_j$ if v_j is in multiple such sets. Pick an arbitrary $\mathbb{F}_i$ (we used smallest index in our data structure) if $\mathbb{F}_i$ in multiple such sets.
    Add $(\mathbb{F}_i, \mathbb{F}_j)$ to $\mathbb{E}$.
    Connect the Reeb nodes for $\mathbb{F}_i, \mathbb{F}_j$ in $\hat{G}$
  end for
  Recompute connected components analysis of $\hat{G}$ and update $\mathbb{D}$
end while
Return $\hat{G}$ and $\mathbb{E}$

$K = 20$, $d = 0$, $r = 0.1$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$ and $S = 5$. We use 10 steps of iterations for GTDA error estimation.

7.3 Error estimation using diffusion on the Reeb network

The model often highlights places where there is no reasonable basis for a prediction, e.g. where there is training data with a different label closer to a prediction. This scenario admits an estimate where the model will likely make mistakes by checking the relative locations between predictions and training data. In this section, we will use diffusion on the projected Reeb net to find places where prediction errors are the most likely to happen.

Using the Swiss roll example, in plot (A) of figure 7.4, we zoom in on two components GTDA. We then look at the induced subgraph of this region in a projection of the Reeb network. The Reeb network projection expands each Reeb node into the original set of input vertices with duplicated nodes merged and also adds in edges that we put into study predictions (the extra set $\mathbb{E}$ in the algorithms). A detailed projection procedure can be found in Algorithm 13.

Put formally: Given a set of Reeb network nodes in $\hat{G}$, find the union of all vertices in G these nodes represent and call that T. We look at the induced subgraph of T in the projection of the $\hat{G}$ from Algorithm 13.

To show these induced subgraphs, we can either use Kamada Kawai layout or, as an alternative to Kamada Kawai, we can also compute coordinates for each projected Reeb node and then combine different layouts using their relative coordinates in Reeb net.

Then we use red circles to mark training and validation data and color them with the true labels. Unknown data points are still colored with predicted labels.

One can immediately notice the problem: *There are some orange predictions in the grey box, but there is no orange training or validation data nearby to support them.* Thus, either the model or the dataset itself have issues with these prediction and merit a second look. In this case, it is just the model that cannot classify some parts of the graph correctly due to noisy links.

We developed an intuitive algorithm based on diffusion to automatically highlights which parts of the visualization will likely contain prediction errors, Algorithm 12. The core part of this algorithm is to perform a few steps of random walk starting from nodes with known labels. Predictions that are close to training data with the same labels in the Reeb net will

have higher scores in the column of predicted labels and hence have smaller error estimates. Although this algorithm is a linear diffusion over the projected Reeb net, it certainly can still be seen as a nonlinear diffusion over the original graph.

Applying this algorithm can successfully find other places where mistakes will happen (see plot (B) of figure 7.4). As a simple comparison, we also include another plot where we directly use model uncertainty (i.e. 1 minus model prediction probability) to estimate errors (see plot (C) of figure 7.4). This metric has been previously used to estimate uncertainty of dataset labels [120]. Clearly, GTDA is able to localize model errors much better and has a higher AUC score (0.95 vs 0.87). There always exists other methods [121] that can also give similar error estimations or even correct predicted labels. But explaining why those methods should work or be trusted to a user without background knowledge is a challenge, while our method offers a map-like justification that can give a rough rationale. Moreover, any results from Algorithm 12 can always be validated and supported through pictures similar to plot (A) of figure 7.4. Also, other than finding possible errors, as shown in the following experiments sections, we can often get many other insights about the model and the dataset by checking abnormal areas of GTDA visualization, ranging from labeling issues to strong correlation between model predictions and a particular dataset property. These are explored in subsequent case studies.

7.4 Demonstration in Graph-based prediction

In this section, we apply our GTDA framework on an Amazon co-purchase graph [122] constructed from Amazon reviews data [123]. Each node in this graph is a product, edges connect products that are purchased together and node features are bag-of-words from product reviews. The goal is to predict product category. The original dataset [122] that has been used in several GNN papers does not have information for each specific product. To better understand the visualization from GTDA, we build a similar dataset directly from the Amazon reviews data [123]. We use the 2014 year version of reviews data and extract products with the same set of labels as in the original [122].

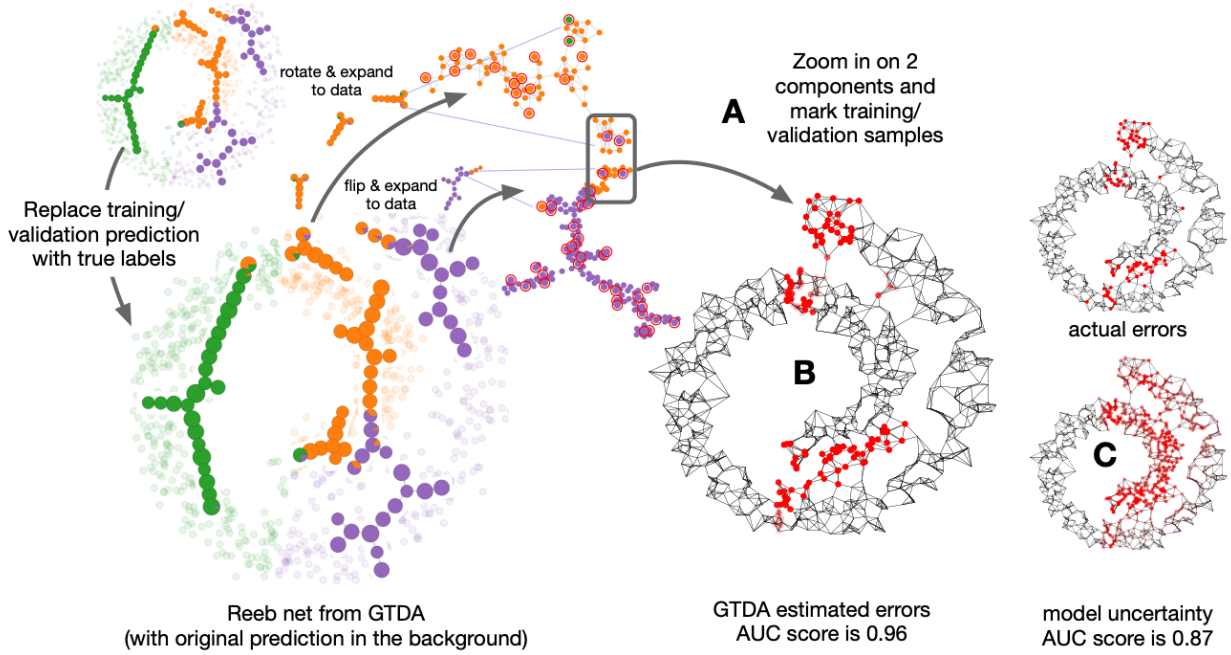


Figure 7.4. This figure demonstrates the procedure of estimating errors from the Reeb net produced by GTDA. In comparison with Figure 7.3, we show the training data labels in the pie charts instead of the predicted values. If we zoom in on two components and mark training and validation samples (red circles) with true labels, we see many orange predictions without any training or validation data nearby to support them (inset box nearby) (A), which suggests potential errors – note that the model may be using additional features to predict these values, but these examples do merit closer inspection. We develop an error estimation procedure in Algorithm 12 to automate this inspection. Overall, GTDA estimated errors have a AUC score of 0.95 with true errors (B), while using model uncertainty (one minus prediction probability) only has a AUC score of 0.87 (C).

Algorithm 12 `error_estimation`$(\hat{G}, \mathbb{E}, \ell, n, \alpha)$ where $\hat{G}$ and $\mathbb{E}$ is the Reeb net and extra set of edges from algorithm 9, ℓ are the original predicted labels, S is an integer for the number of steps (10, or 20 were used), and $0 < \alpha < 1$ (we use $\alpha = 0.5$ in all experiments).

Compute $G^{(R)}$, the projection of the Reeb net back to a graph from Algorithm 13.
Let $\boldsymbol{A}^{(R)}$ be the adjacency matrix of $G^{(R)}$
Compute a diagonal matrix $\boldsymbol{D}^{(R)}$ where $\boldsymbol{D}^{(R)}_{ii}$ is the degree of node i in $G^{(R)}$ and 0 elsewhere.
Initialize matrix $\hat{P}^{(0)}$ where $\hat{P}^{(0)}_{ij} = 1$ iff node i is a training node with label j, otherwise $\hat{P}^{(0)}_{ij} = 0$.
for $i = 1...S$ **do**
    $\hat{\boldsymbol{P}}^{(i)} = (1-\alpha)\hat{\boldsymbol{P}}^{(0)} + \alpha \boldsymbol{D}^{(R)^{-1}} \boldsymbol{A}^{(R)} \hat{\boldsymbol{P}}^{(i-1)}$
end for
Row normalize $\hat{\boldsymbol{P}}^{(S)}$ so that each row sums to 1.
Compute estimated prediction error for node i to be $\mathrm{e}_i = 1 - \hat{\boldsymbol{P}}^{(S)}[i, \ell_i]$
Return estimated errors $\mathbf{e}$.

Algorithm 13 `Reeb-graph-projection`$(\mathbb{F}, \mathbb{E}, G)$ where $\mathbb{F}$, $\mathbb{E}$ is the final set of components and extra set of edges from Algorithm 9 and G is the original graph

Initialize $G^{(R)}$ with the same dimension of G and no edges
for Each $\mathbb{F}_i$ of $\mathbb{F}$ **do**
    Add the set of edges of $\mathbb{F}_i$ from G to $G^{(R)}$
end for
Add edges in $\mathbb{E}$ to $G^{(R)}$
Return $G^{(R)}$

7.4.1 Central results

Our framework identifies a key ambiguity in product categories that limits prediction accuracy (Figure 7.5). Specifically, "Networking Products" and "Routers" overlap (a Router is a specific type of Network Product) and show high levels of confusion as do "Data Storage" and "Computer Components" (an internal data storage drive is a computer component). These results suggest that large improvements are unlikely with better algorithms and would require label improvements to differentiate categories or other divisions in a hierarchy [124]. This was verified by checking another graph neural network [38] with similar behavior (Section 7.4.3).

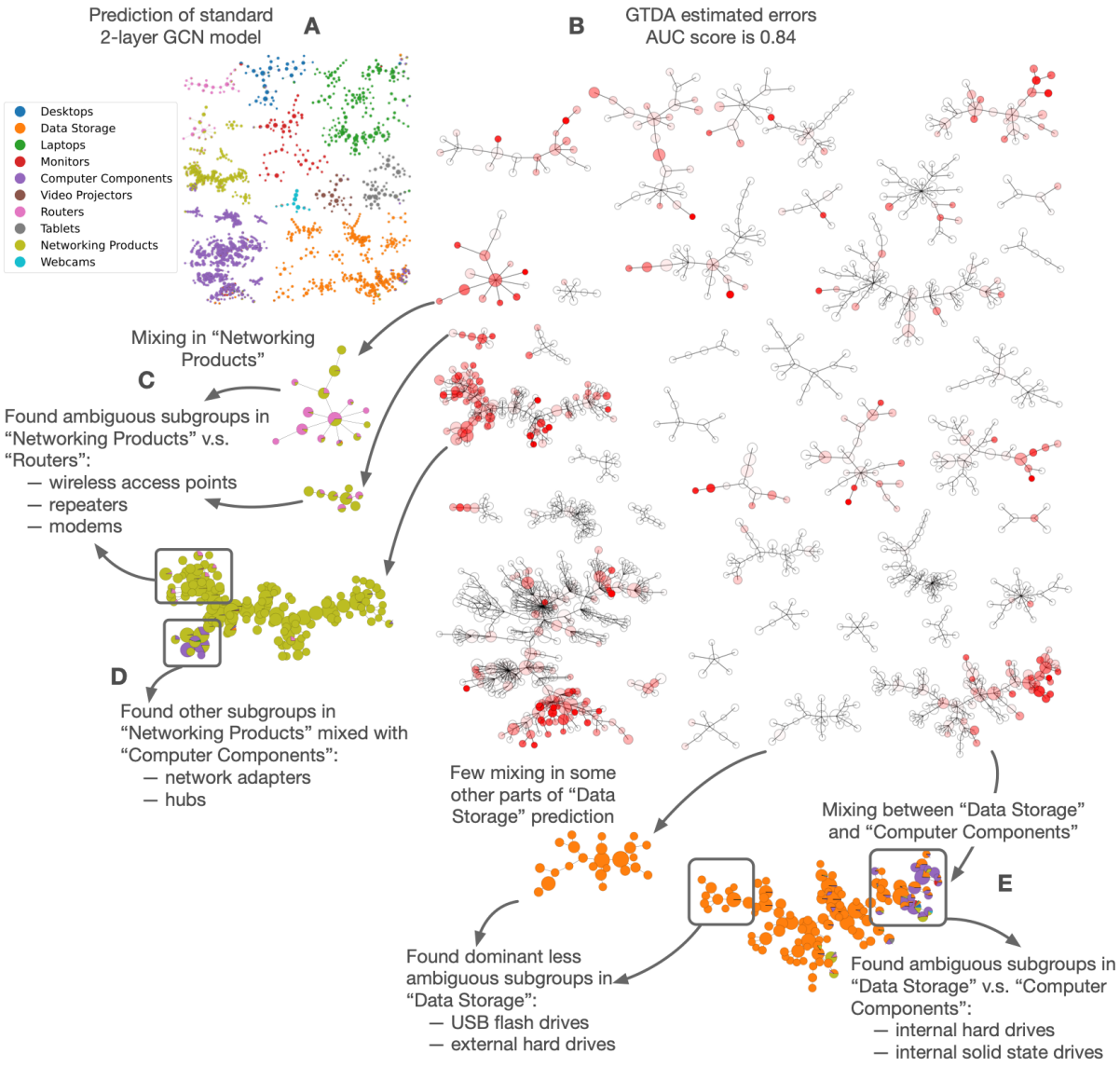


Figure 7.5. Reeb network of a standard 2-layer graph convolutional network model trained and validated on 10% labels of an Amazon co-purchase dataset (A) and estimated errors shown in red (B). The map highlights ambiguity between "Networking Products" and "Routers". Checking these products shows wireless access points, repeaters or modems as likely ambiguities (C). Additional label ambiguities involve "Networking Products" and "Computer Components" regarding network adapters (D); likewise "Data Storage" and "Computer Components" are ambiguous for internal hard drives (E). These findings suggest that the prediction quality is limited by arbitrary subgroups in the data, which Reeb networks helped locate quickly.

Table 7.2. Number of products for each category in our own version of Amazon Computers dataset.

category	number
Desktops	1,757
Data Storage	7,297
Laptops	4,590
Monitors	1,710
Computer Components	15,167
Video Projectors	804
Routers	1,086
Tablets	1,919
Networking Products	4,869
Webcams	548

7.4.2 Dataset and GNN model

Our own version of the Amazon co-purchase graph has the same set of the labels as the original one [122]. We download all products and reviews in the category of "Electronics" by following the link provided in [123]. We use the 2014 version as we can find the exact same set of labels in this version. In the Amazon reviews dataset, each product is associated with a list of categories. To assign labels, for each product, we check from the most general category (i.e. Electronics) to the most specific one (i.e. Routers). And if we find a match to the set of labels we choose, we directly assign the matched label to that product and ignore the other categories in the list. Two products will be connected if they are marked as "also bought", "bought together" or "buy after viewing". After we get the initial graph, we first make the graph undirected and then filter out components that are smaller than 100. We use bag-of-words node features with TF-IDF term weighting constructed from each product's review text. The final graph we get has 39,747 products and 798,820 edges. The number of products for each category is listed in table 7.2.

To get the prediction results used in Figure 2, we use the same 2-layer GCN model as the Swiss Roll experiment to predict product categories (Section 7.2.2). The dimension of the hidden layer is 64, learning rate is 0.01 and weight decay is 10^{-5}. We randomly use 10%

samples as training, another 10% samples as validation and all the other samples as testing. We extract the output of the first layer as node embeddings and we also build a 2-NN graph using cosine similarity to combine with the original graph. This will let GTDA show the impact of the feature similarity on the GNN. For GTDA parameters, we set $K = 100$, $d = 0$, $r = 0.01$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$ and $S = 5$. We use 20 steps of iterations for GTDA error estimation. For the more advanced GPRGNN model used below, we use the same set of parameters as suggested by its authors [38] and node embeddings are extracted from the first layer output as well. We also use the same GTDA parameters as GCN.

7.4.3 Inspecting another advanced model predictions with GTDA

In Figure 7.5, we found ambiguous subgroups inside "Data Storage" and "Networking Products" with the help of GTDA visualization. Similar ambiguities persist after switching to the more advanced GPRGNN model as shown in Figure 7.6. Here, we also notice many estimated errors in "Routers" and "Data Storage" as before. We show a detailed breakdown of products true categories for some components. For each component highlighted, we list top 2 most common categories. The other categories are put in "Others". For "Networking Products" and "Data Storage", we also list the top 3 most common subcategories. For the two "Routers" components in (A), we see many "Modems" or "Wireless Access Points" from "Networking Products". These should be frequently bought together, and "Routers" should be considered as another subcategory of "Networking Products". As a comparison, for the other "Networking Products" component that is less mixed (B), the most common subcategories are "Network Adapters" and "Hubs", which are more precise than the more ambiguous "Routers". Similarly, for the two "Data Storage" components in (C), the mixed one has many "Internal Drives" such as solid state drives (SSDs). These are essential parts of a PC and should be considered as a part of "Computer Components" as well. There are also a small portion of "Network Attached Storage", which may be confused with "Networking Products". On the contrary, the less mixed one mostly contains "External Drives" like USB drives, which are common additions to an already built PC. These results suggest that for this dataset, no matter which model we choose, the performance on some portion of the

dataset will always be limited by the same type of underlying labeling issues. GTDA helps capture those issues in both cases.

7.4.4 GTDA visualization on the original Amazon dataset

As a final check on our results, in Figure 7.7, we apply GTDA to inspect GPRGNN's prediction on the original Amazon dataset built by [122] with the same setting. We can observe similar behavior to Figure 7.6, that is "Routers" is mixed with "Networking Products" and components of "Data Storage" are mixed with "Computer Components".

7.5 Understanding image predictions

One of the most successful applications for complex neural network models is detecting objects in images. Image classifiers based on convolutional neural networks (CNN) can achieve extremely high accuracy, sometimes even higher than humans. What remains not entirely understood is how to explain a model's prediction and whether it will generalize well beyond the training scenario. In this section, we will use GTDA framework to tackle this issue by showing a visual taxonomy of images. Our hope is that by checking various local regions in this visualization, one should be able to find images with very similar visual properties, which then provides clues on the generalization ability of the model on other similar images it may see in the future. Comparing to other research work that tries to justify image predictions through saliency maps on each image [101–104], our approach can provide insights over the entire dataset efficiently. We note that our GTDA analysis could assist such efforts by studying the topology of the saliency maps, along with the predictions, although we have not pursued this direction. Also note that, since the image dataset is not in graph format, a KNN graph needs to be built first from the embedding matrix of the images. More details on this can be found in Section 7.5.2.

7.5.1 Central results

When our GTDA framework is applied to a pretrained ResNet50 model [125] on the Imagnette dataset [126], it produces a visual taxonomy of images suggesting *what* ResNet50

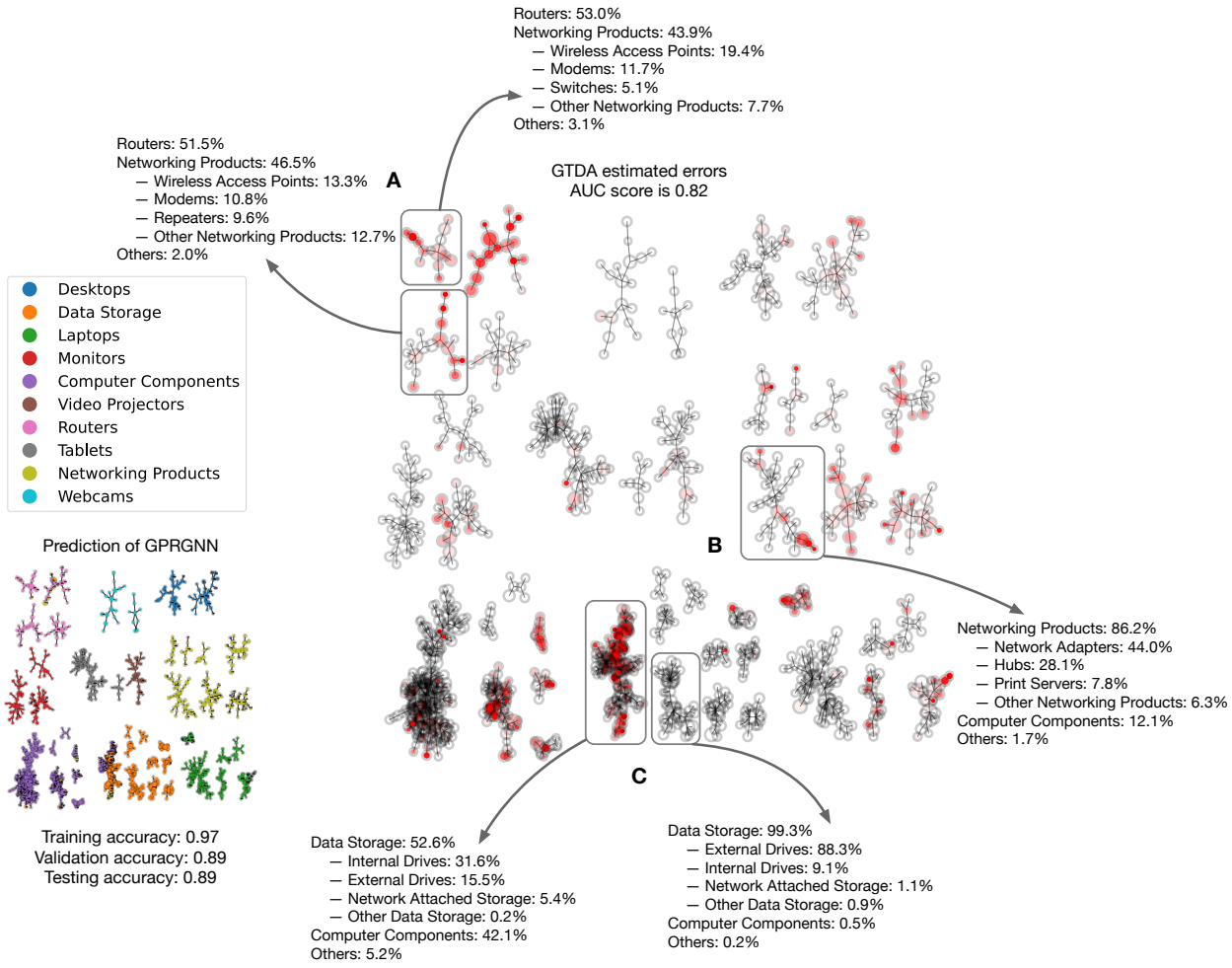
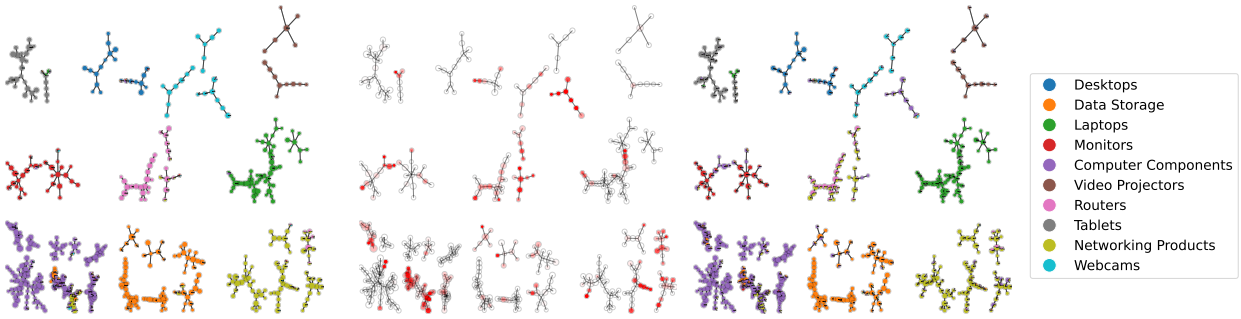


Figure 7.6. We provide GTDA results on inspecting the prediction on the GPRGNN method instead of the GCN used in Figure 7.5 in the main text. We list a detailed breakdown of categories and subcategories for a few components. For the two "Routers" components in (A), there are many estimated errors because of ambiguous subgroups of "Networking Products" like "Wireless Access Points", "Modems" or "Repeaters". The estimated errors are much less in (B) because "Networking Products" has dominant less ambiguous subgroups. Similarly, for two "Data Storage" components in (C), the one with more estimated errors has dominant ambiguous subgraphs like "Internal Drives" or "Network Attached Storage" which is confusing with "Computer Components" or "Networking Products".



(1) predicted labels (2) GTDA estimated errors (3) true labels

Figure 7.7. GTDA visualization of GPRGNN's prediction on the original Amazon Computers dataset [122]. Similar to Figure 7.6, "Routers" is mixed with "Networking Products" and some components of "Data Storage" are mixed with "Computer Components".

is using to categorize the images (Figure 7.8). This taxonomy is built by placing images directly on the layout of Reeb net. It was inspired by Tufte's work on image quilts and small multiples [127]. This example also highlights a region where the ground truth labels of the datapoints are incorrect and cars are erroneously labeled as "cassette player". We conjecture a car labeled as "cassette tape" may be due to images of cars listed for sale including the string "cassette tape player".

7.5.2 Dataset and CNN model

The dataset we use is Imagenette [126], which is a subset of the entire ImageNet containing 10 easily classified classes, "tench" (a type of fish), "English springer" (a type of dog), "cassette player", "chain saw", "church", "French horn", "garbage truck", "gas pump", "golf ball" and "parachute". This dataset can be directly downloaded from a Github repository [126]. The author uses a different training and testing split from the original ImageNet dataset so we first restore the original split before model training. This choice is because the pretrained model from the full ImageNet dataset may have had access to images in the Imagenette test set. The number of training and testing images for each class is shown in table 7.3.

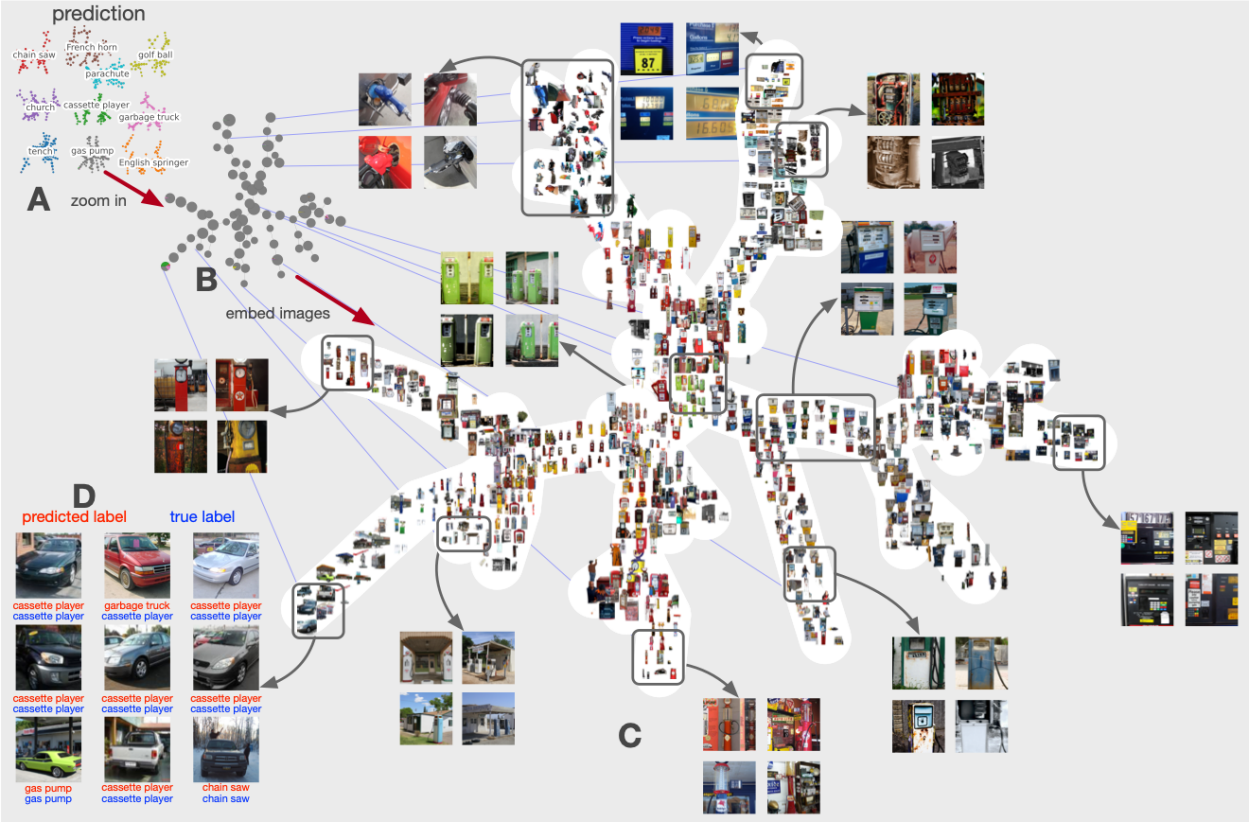


Figure 7.8. We take a pretrained ResNet50 model and retrain the last layer to predict 10 classes in Imagnette (A). In (B), we zoom into the Reeb network group of "gas pump" predictions and display images at different local regions (C). This shows gas pump images with distinct visual features. Examining these subgroups can provide a general idea on how the model will behave when predicting future images with similar features as well as help us quickly identify potential labeling issues in the dataset. For instance, we find a group of images in (D) whose true labels are "cassette player" even though they are really images of "cars".

Table 7.3. Number of training and testing images for each label.

label	training	testing
tench	1,300	50
English springer	1,300	50
cassette player	1,300	50
chain saw	1,194	50
church	1,300	50
French horn	1,300	50
garbage truck	1,300	50
gas pump	1,300	50
golf ball	1,300	50
parachute	1,300	50

We use a pretrained ResNet50 model that is included in the PyTorch package and retrain the last fully connected layer to make predictions on these 10 classes only. We use a batch size of 128, learning rate of 0.01 and run for 5 epochs. We also use the common image transform during training and testing. That is, each training image will be randomly cropped into 224-by-224, randomly horizontally flipped and normalized by the mean and standard deviation computed over the entire ImageNet dataset, while each testing image will be resized to 256 along the shorter edge, center cropped to 224-by-224 and then normalized. We modify the pooling of the last convolutional layer from average pooling to maximum pooling and extract its output as node embeddings. Similar techniques are used in the context of image retrieval [128]. Initially, the embedding dimension is 2048. We first PCA reduce the dimension to 128 with PCA whitening. Then each row is ℓ_2 normalized. A 5-NN graph is constructed on the preprocessed embedding matrix with cosine similarity. For GTDA parameters, we set $K = 25$, $d = 0.001$, $r = 0.01$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$ and $S = 10$. We use 10 steps of iterations for GTDA error estimation.

7.5.3 Details on selecting images to embed

We provide more details on how we embed images on a Reeb net component to get Figure 7.8. For each pair of adjacent Reeb net nodes, for each image in one end, we measure its smallest distance in the projected Reeb net to some node in the other end. Note some images can be duplicated in two ends, in such case, we consider the distance to be zero. If two images have the same distance, we include the one with larger degree in the projected Reeb net. Then we fill in the closest images to one half of the edge and vice versa. A simple demo can be found in Figure 7.9. We also apply a background removal algorithm [129] for each image we embed. After embedding selected images, we can then easily browse around different regions of the component to understand the model's behavior of predicting "gas pumps". Then we can simply select a few Reeb net nodes at different places and check them in detail by listing all images it contains to look for the most common patterns. Eventually, this can help us quickly identify 7 ambiguous "cassette player" images that are really just "cars".

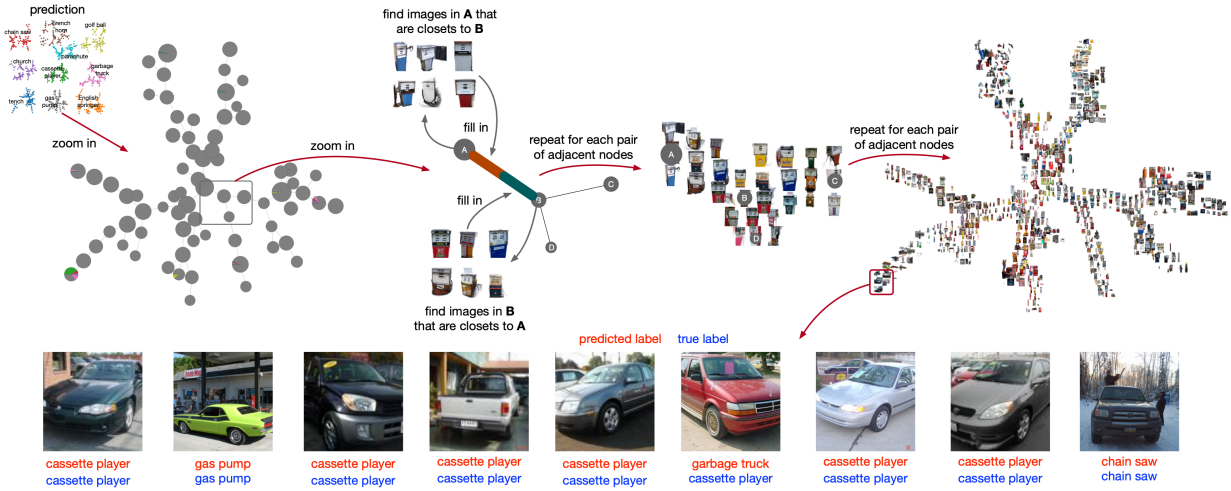


Figure 7.9. This figure demonstrates the procedure of embedding images on a Reeb net component. For each pair of adjacent nodes, we select images from one end that are closest to the other end and fill in those images in half of the edge and vice versa. Browsing around embedded images at different regions can help us quickly identify 7 ambiguous "cassette player" images that are really just "cars".

7.5.4 Statistical validation

Firstly, we verify that GTDA is stable in detecting those 7 confusing "cassette player" images as shown in Figure 7.8. We randomly train 100 models in the same way as described before and check the visualization using each of these 100 models. On average, only 1.3 of these 7 images are predicted wrong, which means simply iterating through all the prediction errors is not enough. We define that this labeling issue can be detected in a visualization if the following criteria can be met:

- All or most of these 7 images are in the same component
- Some neighbors of these images are from a different class
- These images are well localized in the component with small pairwise path length

In our results, we find the visualization from all 100 models can meet these criteria. More specifically, for 74 models, all 7 images can meet these 3 criteria. In the other 26 models, for 22 of them, 6 images can meet all 3 criteria, for 2 models, 5 images can meet and for the rest 2 models, 4 images can meet. Also the maximum pairwise path length for images meeting the criteria is 4 (for most models, this maximum length is 2). Secondly, we verify that a random group of 7 images will be very unlikely to satisfy these criteria. We pick one of the 100 models and randomly sample 7 images from each Reeb net component. We cannot find any randomly sampled group in 10000 Monte Carlo experiments that can satisfy these criteria simultaneously.

7.5.5 Comparing to influence functions

Influence functions [100] is a framework recently proposed to extract the most influential training samples on any specific testing sample. It can also be used to find adversarial or mislabeled training data. We used an existing implementation of influence functions from https://github.com/nimarb/pytorch_influence_functions to find ambiguous training samples of Imagenette. The biggest issue of influence functions is scalability. Computing influence for all 12,894 images will take almost 4 hours while our GTDA framework only

takes about 1 minute to process the entire dataset. Figure 7.10 compares the top 30 most confusing training images of "cassette player" from influence functions or GTDA. For GTDA, we directly take top 30 images with the largest estimated errors using Algorithm 12. Both methods find training images that indeed look confusing. However, another advantage of GTDA is we get more insights by grouping these ambiguous training images based on their locations in the visualization and checking nearby images in the visualization. For instance, we can conclude from Figure 7.9 that some "cassette player" images can be confused with "gas pump" or "chain saw" images with cars in them.

7.5.6 Understanding model generalization on other labels

Other than the detailed analysis for "gas pump" component, we provide similar figures (Figures 7.11 to 7.15) for components of other labels. We embed images on each component in the same way as above. GTDA can always find groups of images with different visual features. For instance, it can find "church" images that are either the inside decorations of a church or the outside landscapes in Figure 7.13. It can also find images that are ambiguous like group (C) in Figure 7.11 or group (D) in Figure 7.15. All these results can help us understand how the model is utilizing different features of an image to make the prediction and when it might make mistakes.

7.5.7 Comparing to a Reeb net from original TDA framework

Since the original format of the image representations is an embedding matrix, we get another Reeb net from the original TDA framework (i.e. *mapper*) without transforming the embedding matrix into a KNN graph. The embedding matrix is still PCA reduced to 128, whitened and ℓ_2 normalized. We also use the prediciton lens without softmax as the softmax function will make lens highly skewed, i.e. most lens will be close to 0 or 1. We split each lens into 10 bins with 10% overlap. Then we apply density based spatial clustering [130] for samples in each bin so that we don't need to select the number of clusters. This clustering scheme will consider some samples as noise and not clustering them. We set the maximum distance between two points to be in the same cluster as 3. The Reeb net is shown in

Figure 7.10. This figure compares the top 30 most confusing training images of "cassette player" from influence functions [100] or GTDA. Both method can find some common training images that are indeed ambiguous. However, it will take influence functions almost 4 hours to compute influence for all 12,894 training images while GTDA only takes about 1 minute to process the entire dataset.

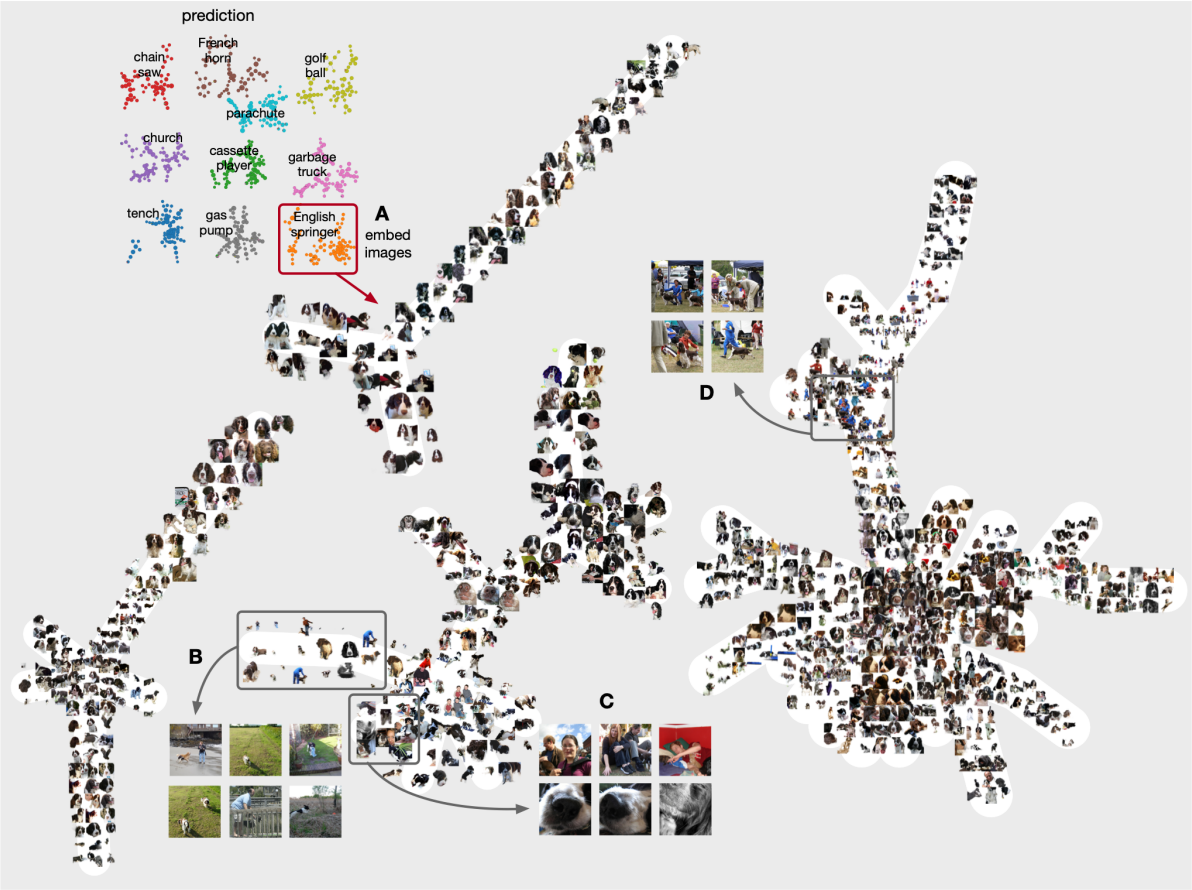


Figure 7.11. We embed images on components that are mostly "English Springer" predictions (A). While most "English Springer" images are easy to classify, we also find some groups where the background information is dominant in (B) and (D) or the images are ambiguous (C). Consider zooming in to see the micropictures.

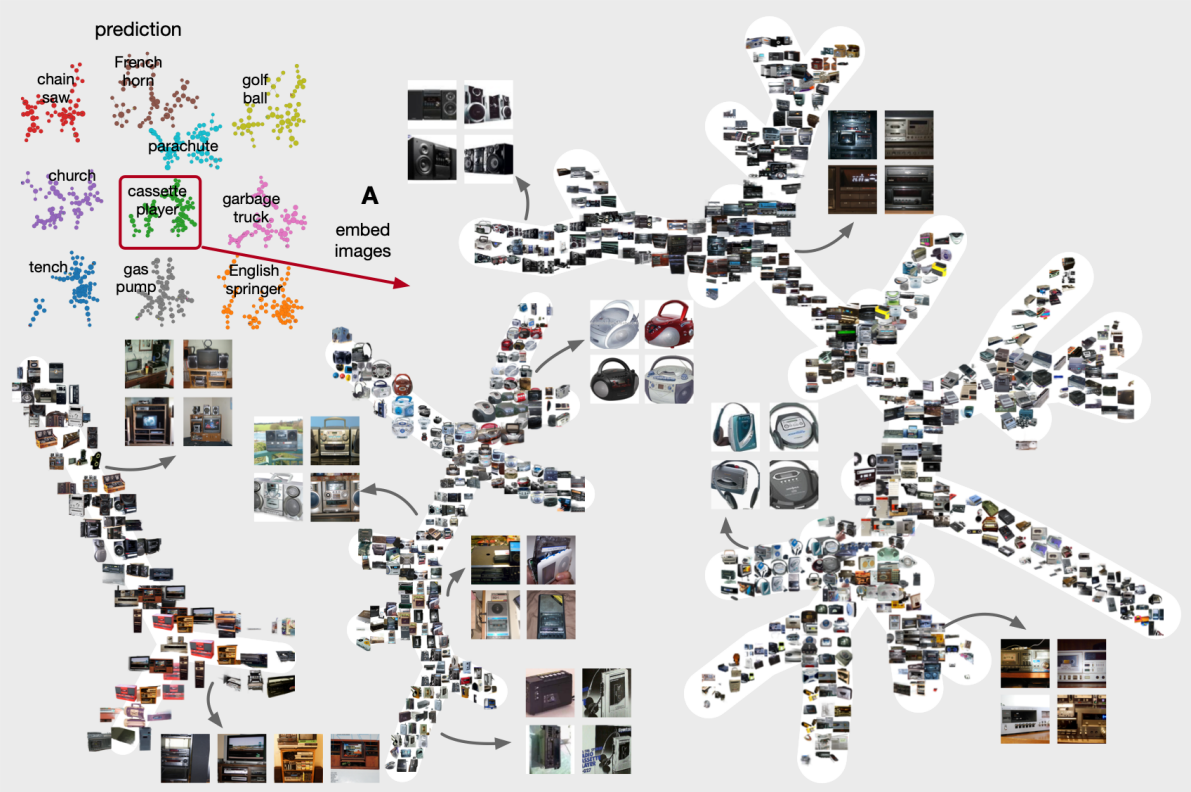


Figure 7.12. By embedding images on "cassette player" components (A) can help us find "cassette player" in various shapes.

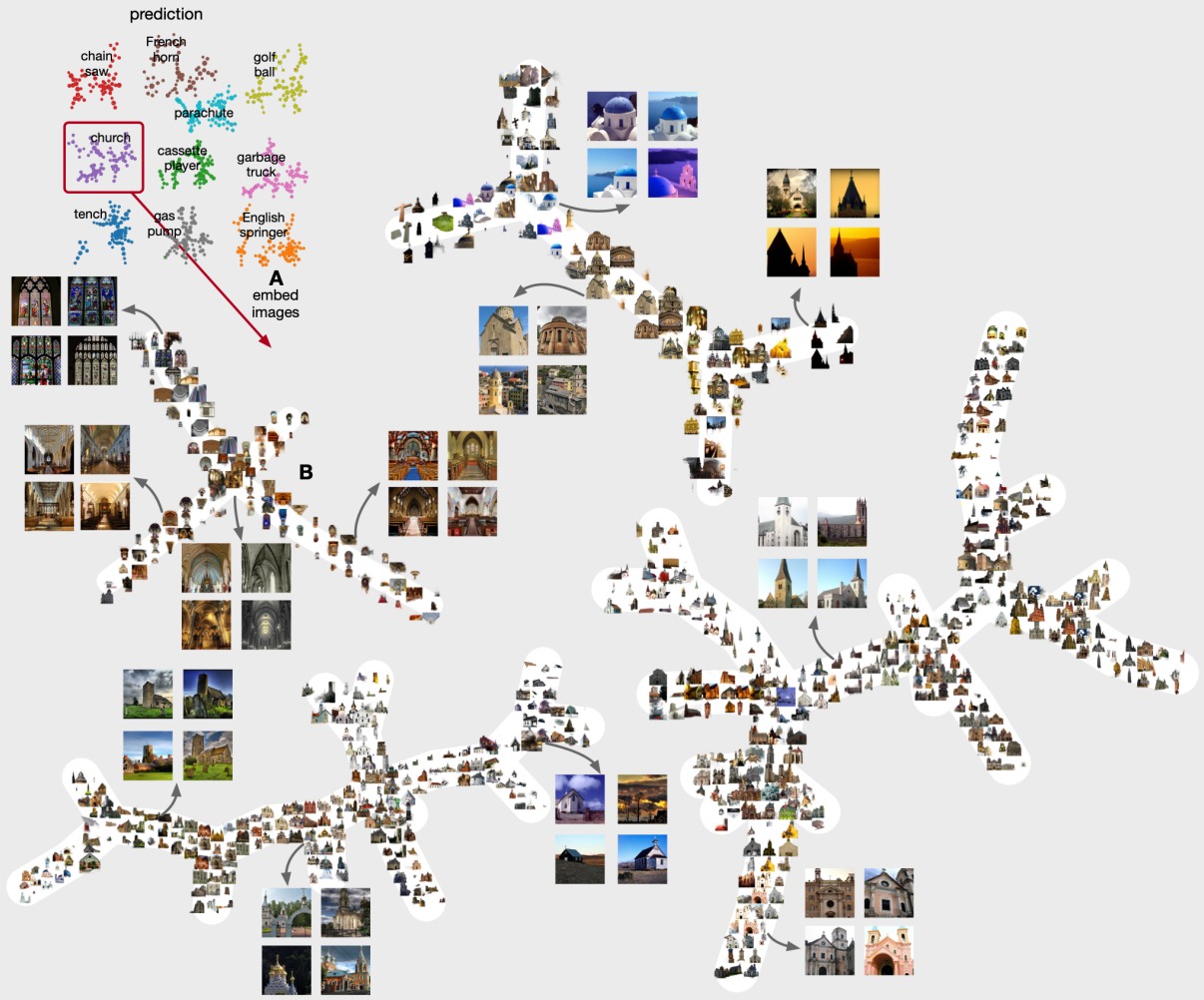


Figure 7.13. By embedding images on "church" components (A), we find one component has images that depicts the inside decorations of church (B) while the other components are images showing different outside landscapes of church.

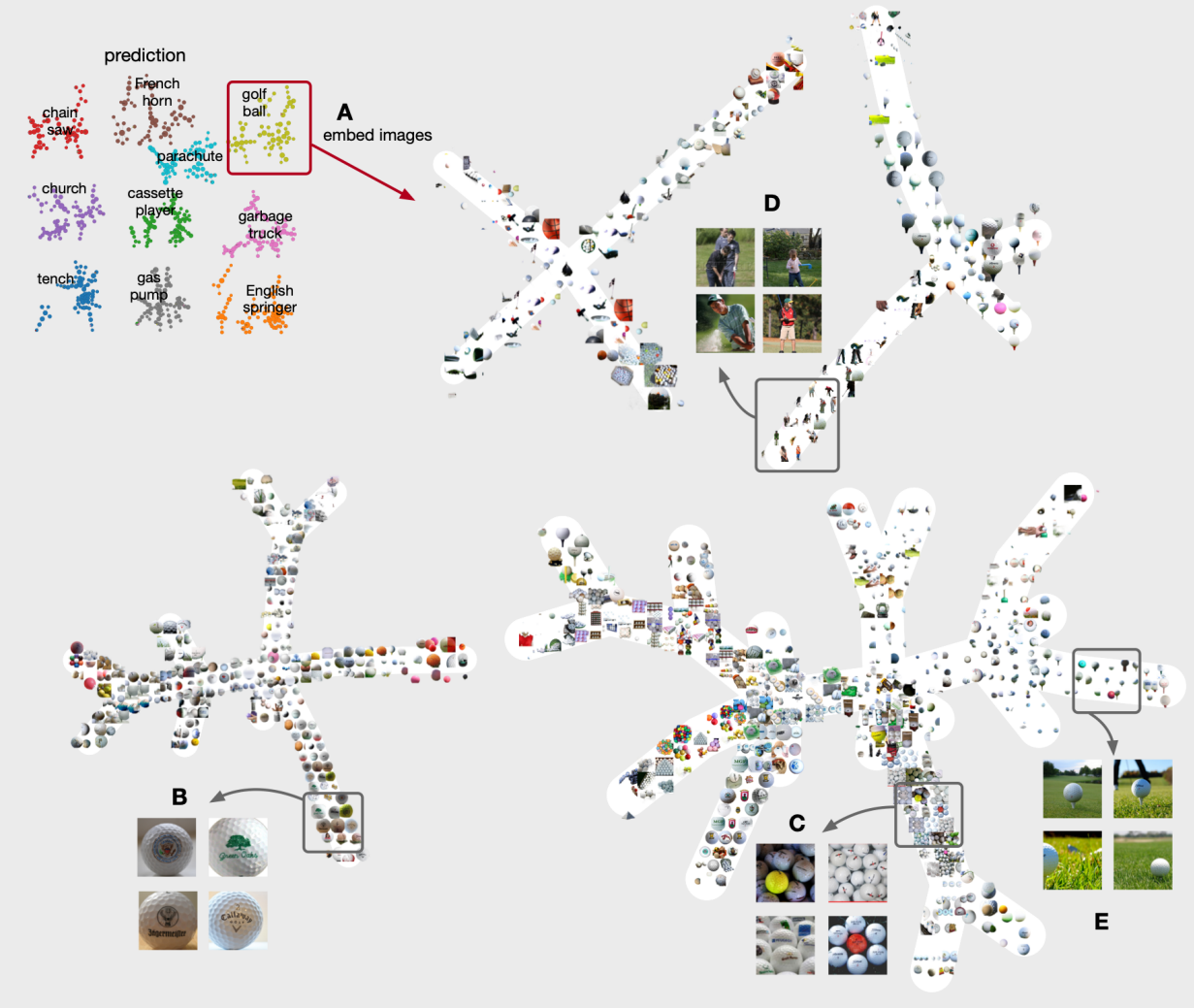


Figure 7.14. We embed images on "golf ball" components (A). We can find images with only one large golf ball (B), or images with lots of small golf balls (C), or images where a person is playing golf ball (D), or images with a golf ball placed on grass (E).

Figure 7.16, which doesn't show any obvious subgroups other than 10 major components representing 10 classes or any labeling issues previously discovered by GTDA. We also find that no information can be extracted at all for around 28% images as they are either in some very small Reeb net components or simply considered as noise by the clustering scheme.

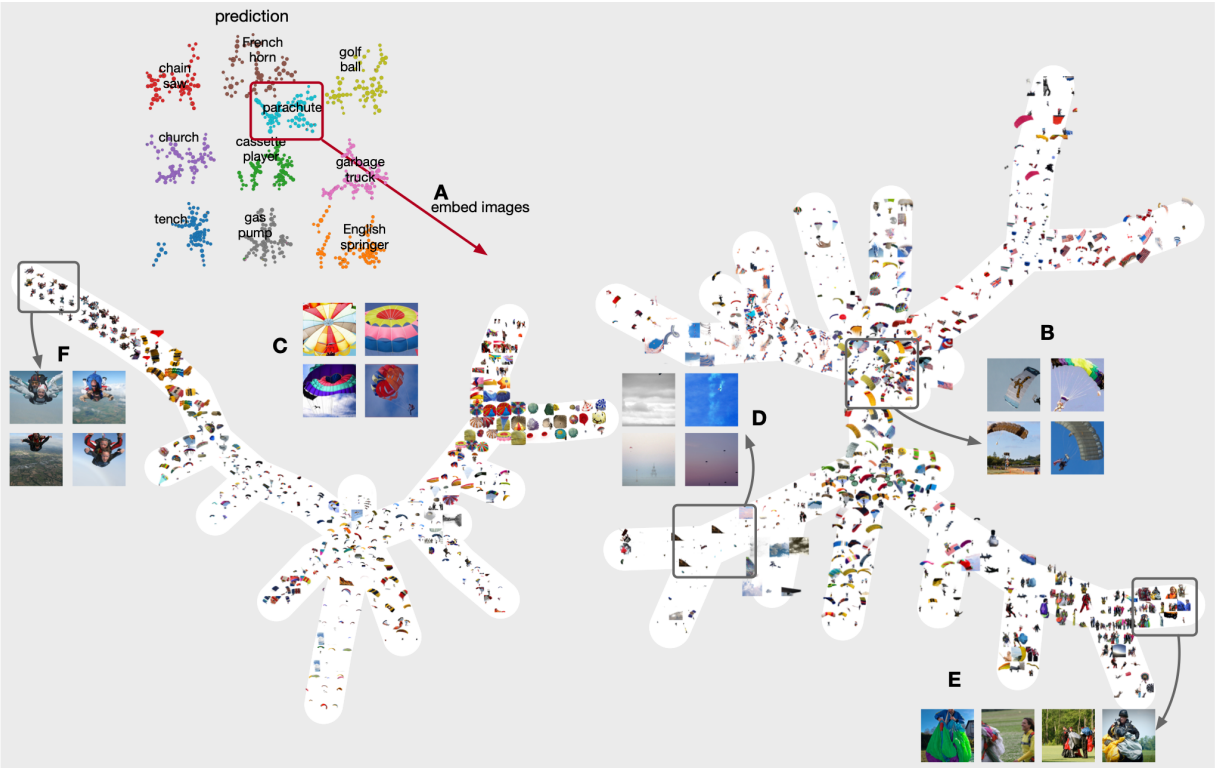


Figure 7.15. We embed images on "parachute" components (A). We can mainly see parachutes in two different shapes (B and C). Some images are ambiguous as they are really just "sky" (D). We also find images where a person is standing on the ground wearing a parachute (E) or a person that jumps into the sky (F).

7.6 Understanding Malignant Gene Mutation Predictions

In this section, we apply our method to inspect model predictions of gene sequence variants effects. A gene sequence variant means that some part of the DNA sequence for this gene is mutated compared with the reference sequence. Modifications include single nucleotide variation, deletion, duplication, etc.

7.6.1 Central results

Reeb networks (see Figure 7.18) from a proposed DNA prediction method [94], when applied to the BRCA1 gene, show Reeb components that are localized in the DNA sequence

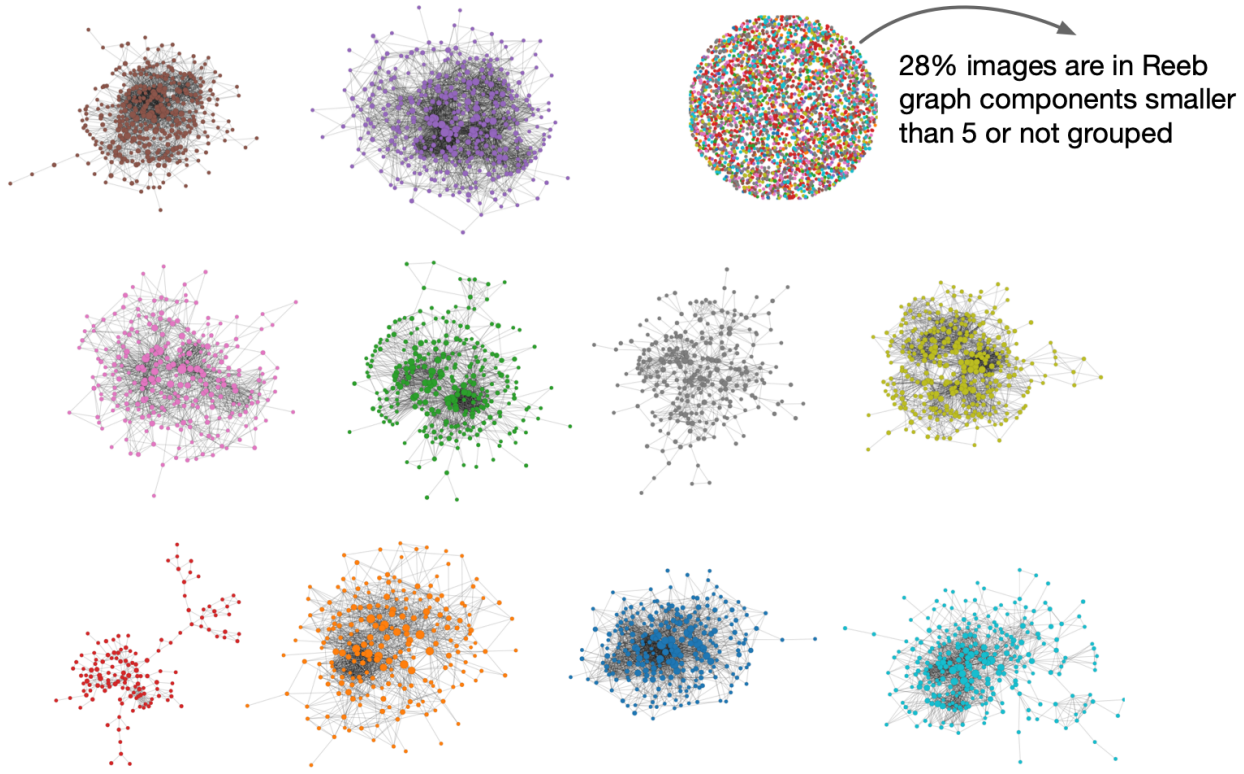


Figure 7.16. Reeb net on the 10 easy classes of ImageNet created by the original TDA framework. TDA is directly applied to the ResNet image embedding matrix without transforming into KNN graph. Unlike GTDA visualization, we cannot find any obvious subgroups other than 10 major components representing 10 classes or the labeling issues of some "cassette player" images. Moreover, no information can be extracted at all for around 28% images as they are either in some very small Reeb net components or simply considered as noise by the clustering scheme.

(Figure 7.17 A). Such location sensitivity is not very obvious in the visualization of other methods like mapper, tSNE or UMAP (Figure 7.17 B, C, D). We can also find known biological structures of DNA, like exons, are also localized in different Reeb net components (Figure 7.17 E). Such structures are also difficult to detect in the output of mapper, tSNE or UMAP ((Figure 7.17 F, G, H)).

When we examine one particular protein encoding region of BRCA1, i.e. in the 1JNX repeat region, we find that different Reeb net components in the result of GTDA also map

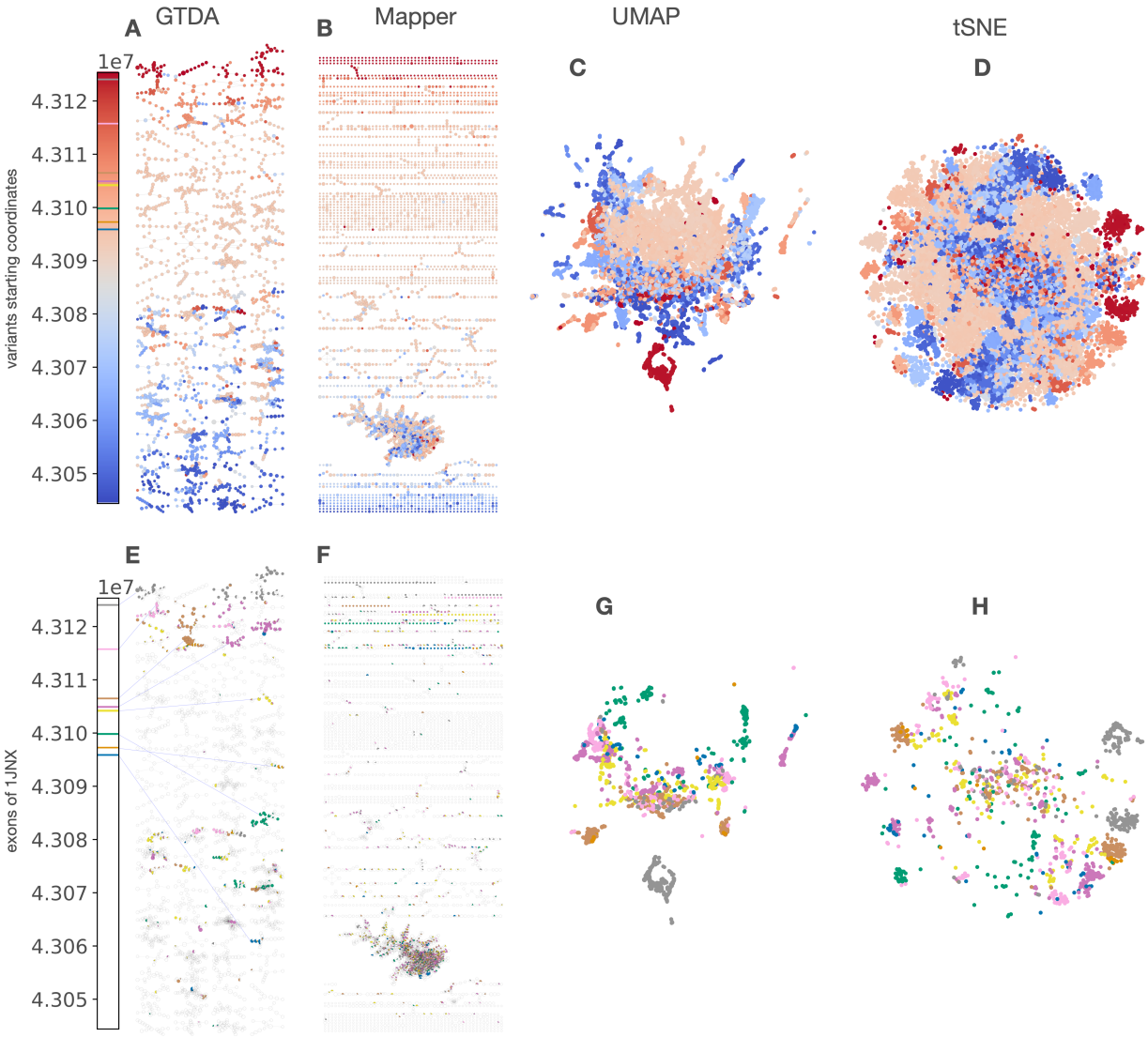


Figure 7.17. We find GTDA output is strongly correlated to the mutation starting coordinates. Such correlation is not immediately obvious in the visualization of other methods. We could find other known biological structures like exons are localized into different Reeb net components too, which is also weaker for other methods. In both cases, GTDA performs significantly better than other methods ($p < 0.001$, see Table 7.5) in two metrics we designed to measure such correlation.

to secondary structures (Figure 7.18 A). For one of the helix structures, this analysis shows regions where insertions and deletions are harmful (pathogenic) and single nucleotide variants lack evidence of harm (Figure 7.18 B). In an analysis of a component with many harmful predictions, these results show that places where the framework incorrectly predicts errors are strongly associated with insignificant results in the underlying data (Figure 7.18 C).

In the following subsections, we will provide details on this analysis, including the model and the dataset we use, how we extract and validate the insights from GTDA results and how we quantify the difference between GTDA and other methods.

7.6.2 Dataset and model

The model we use is recently proposed to predict gene expression from DNA sequence by integrating long-range interactions [94]. In this model, a consecutive DNA sequence of 196,608bp is used to predict 5,313 human genome tracks. For each gene variant, we follow the same steps as proposed by [94] to compute its embedding. First, we extract the reference and alternate DNA sequences from homo sapiens (human) genome assembly, either hg19 or hg38 as specified by the gene variant record. This gives a 393,216bp long DNA sequence with the centered on the VCF position (Variant Call Format). Note that for the alternate sequence, the gene variant is applied first before extracting the modified sequence. Then, we directly use the pretrained model from [94] to make predictions on the reference and alternate sequences. This model will aggregate the center 114,688bp into 128-bp bins of length 896. The prediction for each 128bp bin is a 5,313 vector, where each element represents the predicted gene expression in one of the 5,313 genome tracks for the human genome (including 2,131 transcription factor chromatin immunoprecipitation and sequencing tracks, 1,860 histone modification tracks, 684 DNase-seq or ATAC-seq tracks and 638 CAGE tracks). The prediction vector of the 4 128bp bins located in the center is then summed together to get a prediction vector for the reference or alternate sequence. After that, the elements in each prediction vector corresponding to the CAGE tracks is $\log(1+x)$ transformed. Finally, we compute the difference of preprocessed prediction vectors between reference and alternate sequences as the final embedding for the gene variant. In total, we

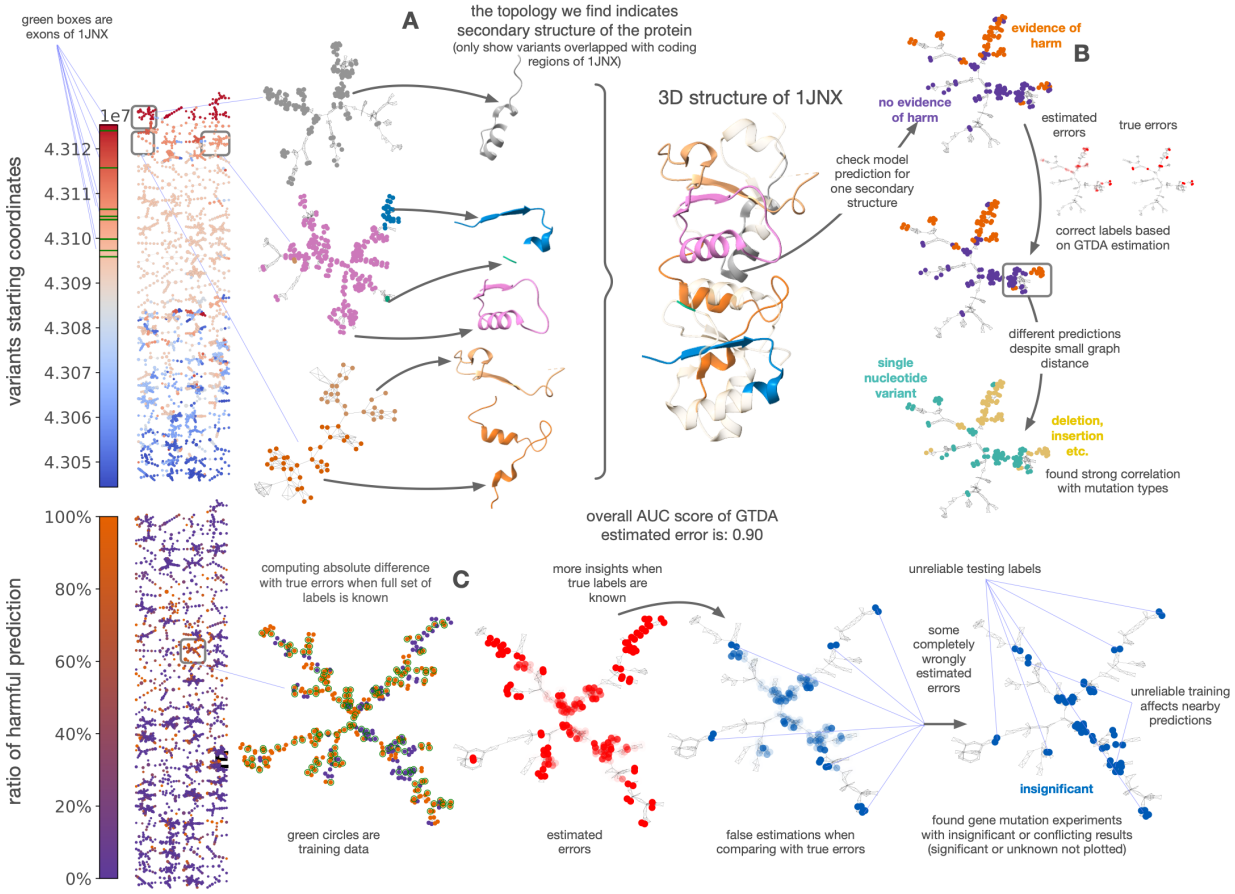


Figure 7.18. We use Reeb networks to visualize harmful (likely pathogenic) and potentially non-harmful (no evidence of pathogenicity) predictions of gene variants in BRCA1. Other than the strong location sensitivity, some Reeb net components also map to several secondary structures on part of the protein (1JNX) as shown in (A). We further check the model predictions on variants targeting one secondary structure (B). Our error estimate shows a number of likely erroneous predictions, and we flip these expected errors (a final analysis showed these errors were correctly identified). We continue to see variants with distinct prediction in a small region of a few amino acids. Close examination shows a strong association between mutation types and model predictions where deletion or insertion is more likely to be harmful than a single nucleotide variant. Additional insights when using the full label set show some estimated errors are completely wrong (C). These prediction mistakes involve gene mutation experiments with insignificant or conflicting results and indicate underlying uncertainty.

get a 23,376-by-5,313 embedding matrix for 23,376 gene variant records. Then, a linear classifier will be trained on this 5,313 difference vector to predict variants effects. The original paper uses the training and testing datasets from CAGI5 competition [131], where a Lasso regression is trained to predict a label of -1 (significant downregulating effect), 0 (very little to no effect on expression) or +1 (significant upregulating effect). We were not able to download the dataset from the official CAGI5 competition website. Therefore, we use similar procedure to predict harmful (label 1) v.s non-harmful (label 0) gene mutations from ClinVar. We download gene variants experiments from the official ClinVar website [132]. We choose all experiments that are targeting **BRCA1** as it is one of the genes with the most number of experiments and part of the protein it encodes has known 3D structures (i.e. **1JNX**). Gene variants without a valid VCF (variant call format) position are removed. As for the labels, we directly use the "ClinSigSimple" field as the label of each gene variant record. An integer 1 means at least one current record indicates "Likely pathogenic" or "Pathogenic", but doesn't necessarily mean this record includes assertion criteria or evidence. An integer 0 means there are no current records of "Likely pathogenic" or "Pathogenic". An integer -1 means no clinic significance and is replaced by label 0 in our experiments. And we use a logistic regression with L1 penalty since this is a binary prediction. We include 23,376 gene variants where 50% of them are used as training, and the other 50% are used as testing. To build the graph for GTDA, the embedding matrix is PCA reduced to 128 dimensions with PCA whitening and then each row is ℓ_2 normalized. A 5-NN graph is constructed on the preprocessed embedding matrix with cosine similarity. This 5-NN graph has some small components smaller than the threshold set by s_1 and s_2. As a result, 338 out of 23,376 gene variants ($\sim 1.4\%$) are not included in the final Reeb net; this is not expected to impact the results. We use 2 prediction lens and the first 2 PCA lens of the embedding matrix for GTDA analysis. For GTDA parameters, we set $K = 30$, $d = 0$, $r = 0.05$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$ and $S = 10$. We use 20 iterations for GTDA error estimation.

7.6.3 Validating GTDA visualization

The visualization we get from this dataset is shown in Figure 7.19. The first finding is that different components in this visualization are strongly related to different regions of the DNA sequence. Such a result is not surprising because this model aims to predict gene expressions from a long range of DNA sequence while most gene variants will only change one or two base pairs. Therefore, it is expected that gene variants close to each other in coordinates will also get similar embeddings. To further validate whether this visualization can capture finer 3D protein structures, we check the crystal structure of the BRCT repeat region (PDB id is **1JNX**), also shown in plot (C) of Figure 7.19. In total, **BRCA1** encodes a protein with 1863 amino acids. And **1JNX** covers amino acids from 1646 to 1849. In the color bar of Figure 7.19, we mark the protein coding regions (exons) of **1JNX** in green. In (B) of Figure 7.19, we check a few components in detail that contains gene mutation locations overlapped with the green area. Different node colors are assigned based on which exon they overlap with. We can find that different local structures of this crystal are also very well localized in our visualization. All these findings suggest that the model's embedding space has a strong correlation with VCF (variant call format) positions of gene variants and GTDA can capture such property successfully.

7.6.4 Estimating and correcting prediction errors

We apply Algorithm 12 to estimate errors of model prediction Figure 7.20. Overall, GTDA estimated errors (after normalizing to 0 to 1) achieve an AUC score of 0.90. In comparison, using model uncertainty gives an AUC score of 0.66. Since this is a binary classification, we can also flip predicted labels if they are more likely to be errors. Instead of setting a single threshold, we flip predicted labels when the estimated errors are larger than the probability of the current prediction. The corrected labels can improve training accuracy from 0.87 to 0.98 and testing accuracy from 0.78 to 0.86.

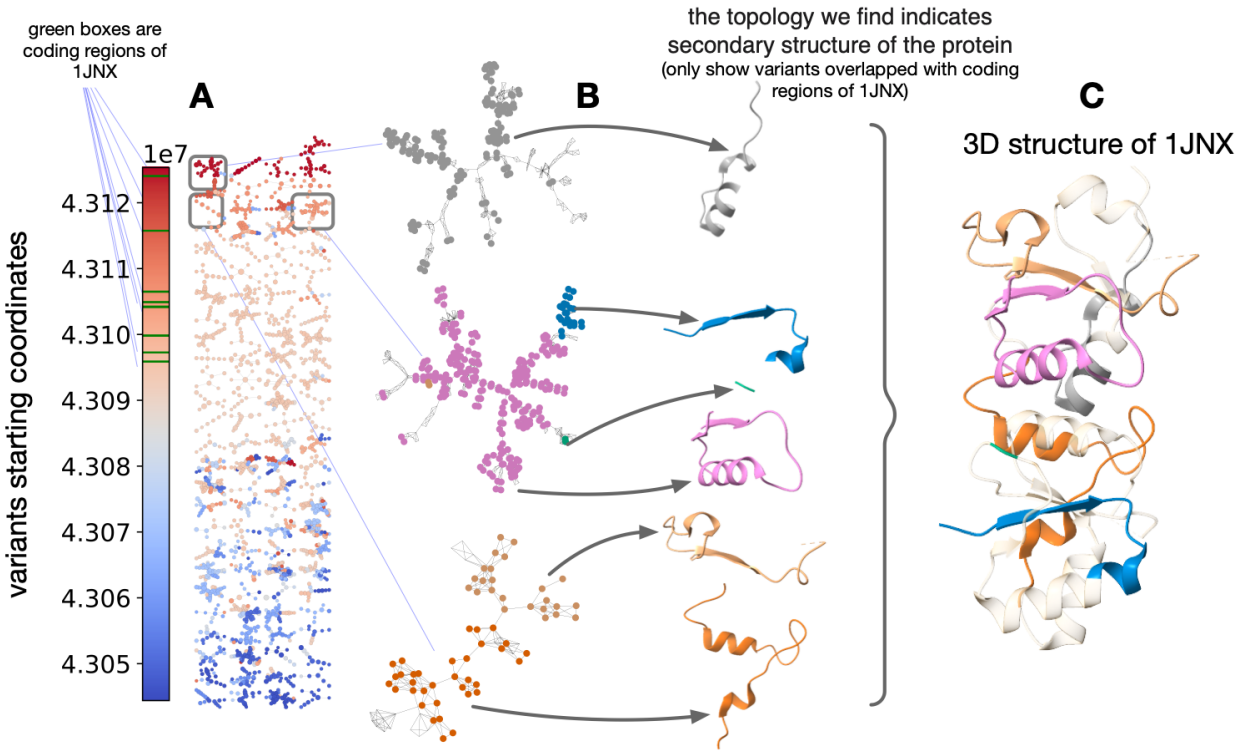


Figure 7.19. (A) shows components found by GTDA, where each node is colored by median hg38 coordinates of mutation starting positions. Different components are ordered by the averaged median coordinates in a zig-zag fashion from lower right to upper left. We zoom in a few components where the gene variants have the highest overlap ratio with the coding regions of **1JNX** (B). Different node colors are assigned based on which consecutive protein coding region they overlap with. Nodes for gene variants not in the coding regions of 1JNX are not plotted. We can find that different secondary structures of the crystal of **1JNX** (C) are also well separated in the GTDA visualization.

7.6.5 Extracting insights about mutation types and single nucleotide variants

As we explore model predictions for gene mutations happening inside protein encoding regions, i.e., green boxes in Figure 7.19, we find different predicted labels for mutations that target a small area of the protein structure. One such example is Figure 7.21, where records in the grey box happen in a small region of the protein structure with around 17 amino acids. So there should be other aspects that help the model make different predictions.

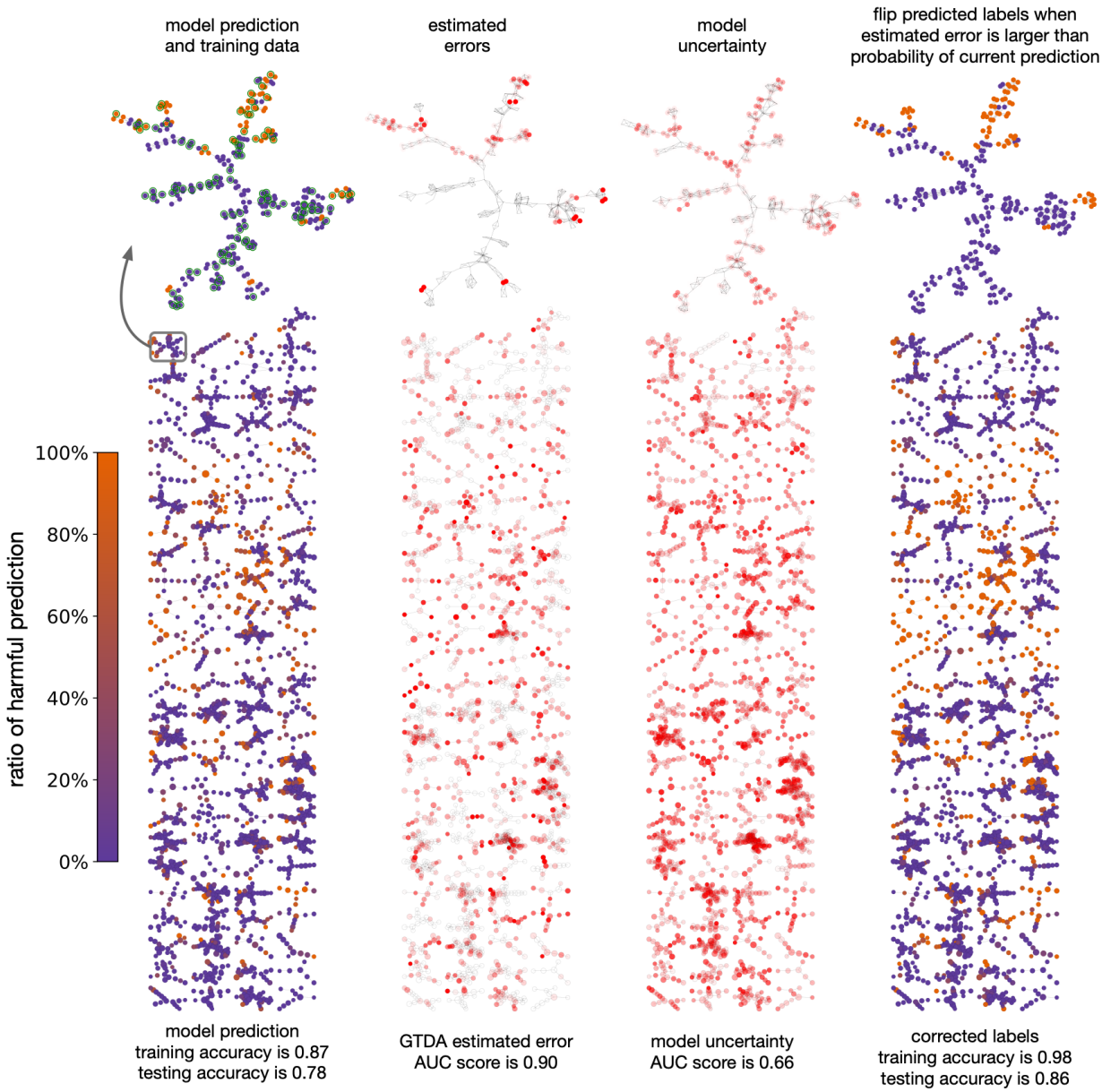


Figure 7.20. In the top part, we zoom in a component and mark training data using green circles. Then we show GTDA estimated errors and model uncertainty on this component. We flip predicted labels if the estimated error is larger than the prediction probability. In the lower part, we can see GTDA error estimation has much better overall AUC score and the corrected labels have higher training and testing accuracy.

By checking the actual mutation record, we find the non-harmful mutations are all single nucleotide variant (SNV), while harmful mutations are other types of mutations including deletion, insertion or duplication. This makes sense as the latter types will not only affect the current amino acid, but also the subsequent amino acids and hence cause more substantial changes to the final protein structure.

Overall, we find for gene mutations that are predicted harmful (after GTDA correction) and target gene encoding regions, 70% of them are mutations like deletion, insertion or duplication. For gene mutations that are predicted as non-harmful and target gene encoding regions, only 6% are mutations like deletion, insertion or duplication. When including gene mutations outside protein encoding regions as well, 72% of harmful predictions are mutations like deletion, insertion or duplication, while that number drops to 5% for non-harmful predictions.

We assess the statistical significance of the relationship between single nucleotide variants (SNV) and harmful predictions for each component GTDA identifies in Table 7.4. The associated Chi-square p-values highlight a few components where this association is missing, such as component 100 with 34 non-harmful non-SNV results throughout the entire BRCA1 structure (coding and non-coding regions), with a p value of 0.22. This suggests a difference in behavior for this component in comparison with the remainder of the components. Other large p-values include the nearby components 99 and 101, along with component 3, 26. Overall, this highlights another way these GTDA results could be used.

7.6.6 Incorrect GTDA error estimation implies unreliable labels

When we compared the GTDA error estimation with true errors, we found a few places where GTDA estimate is entirely wrong.

To understand this abnormality, in Figure 7.22 we zoom in a few components and use green circles to mark training and validation data. We show the GTDA estimated errors as well as the false estimations when comparing to the true errors. We can see a few false error estimates in each of these components. And on checking those false estimations, we

Table 7.4. For each component in the Reeb networks, 2 contingency tables are computed, where the left table only considers variants in the coding regions of 1JNX and the right table considers all variants. Only components where each cell of the right table has a count 3 or higher are included. Chi-square p-values are computed for tables where each cell has a count larger than 0.

	Prediction and Type (coding regions of 1JNX)					Prediction and Type (all)				
	Harmful	Harmful	non-Harm	non-Harm	Chi-square p-value	Harmful	Harmful	non-Harm	non-Harm	Chi-square p-value
Component	SNV	non-SNV	SNV	non-SNV		SNV	non-SNV	SNV	non-SNV	
0	11	6	83	4	5.1e-04	13	6	167	8	1.2e-04
2	0	0	0	0	-	17	264	49	16	1.2e-36
3	1	3	99	5	1.8e-05	12	3	230	9	2.5e-02
4	0	0	0	0	-	13	14	181	4	1.2e-16
6	0	10	10	2	-	24	38	298	20	2.7e-27
7	0	0	0	0	-	6	42	152	24	1.5e-22
8	0	4	0	0	-	16	82	96	22	6.2e-21
9	0	0	0	0	-	6	23	44	11	4.9e-07
10	0	0	0	0	-	17	102	129	22	1.0e-30
14	0	2	2	0	-	13	31	297	19	7.6e-30
15	6	2	40	0	-	25	146	437	24	8.7e-90
16	0	0	0	0	-	6	155	136	13	3.9e-53
17	5	14	49	4	6.5e-08	55	93	485	23	2.7e-59
18	0	0	2	0	-	9	7	115	3	7.5e-08
19	0	8	0	2	-	36	70	422	32	1.0e-44
21	0	6	64	2	-	20	31	376	17	4.7e-33
22	12	16	102	2	4.2e-13	32	20	188	6	1.1e-12
25	0	0	0	0	-	15	17	19	3	7.7e-03
26	0	0	0	0	-	34	4	256	12	2.4e-01
28	0	1	0	1	-	29	42	63	12	1.7e-07
29	3	6	193	18	8.1e-07	9	9	339	31	1.3e-07
30	0	0	0	0	-	19	57	93	9	6.4e-19
31	0	0	0	0	-	16	18	68	8	4.3e-06
32	0	4	2	0	-	5	23	51	5	1.0e-10
33	10	55	64	11	5.5e-16	16	55	204	23	1.1e-28
34	16	18	32	0	-	32	70	66	12	3.5e-12
36	2	4	250	6	1.8e-12	8	11	314	23	3.2e-12
37	0	0	0	0	-	40	76	26	14	1.5e-03
38	0	0	0	0	-	21	37	137	19	8.6e-14
39	0	0	0	0	-	14	198	24	14	3.4e-18
40	0	2	6	0	-	50	81	488	13	1.5e-63
41	0	0	2	0	-	16	14	158	6	1.1e-11
44	0	0	0	0	-	27	29	423	17	4.2e-30
46	0	16	30	2	-	19	36	51	10	2.0e-07
48	0	4	20	4	-	30	14	220	16	3.1e-06
50	0	0	62	0	-	4	36	262	10	2.2e-45
52	0	4	18	2	-	60	67	830	79	5.7e-40
53	0	0	0	0	-	15	27	303	25	2.7e-22
54	0	0	10	0	-	19	27	365	13	2.5e-32
55	0	0	0	0	-	21	19	109	3	2.8e-11
56	0	0	0	0	-	5	34	29	4	9.4e-10
57	0	0	0	0	-	40	18	78	6	4.5e-04
58	2	2	0	0	-	25	30	157	10	2.3e-15
59	0	0	0	0	-	17	44	163	6	6.6e-28
60	6	0	36	2	-	6	7	130	13	2.8e-05
62	2	33	12	7	1.9e-05	12	69	218	35	8.1e-33
64	2	25	114	7	5.0e-22	6	25	192	17	4.4e-22
66	0	4	24	0	-	27	23	165	9	1.9e-12
67	0	6	0	0	-	9	30	111	4	1.0e-20
68	21	13	87	9	3.3e-04	76	108	570	92	3.8e-36
69	2	3	78	7	4.4e-03	4	11	314	35	8.6e-12
70	0	0	0	0	-	40	48	6	6	1.0e+00
71	0	0	0	0	-	17	8	269	18	4.4e-05
72	0	0	12	0	-	6	5	318	21	1.7e-05
73	0	0	0	0	-	12	9	142	3	3.1e-10
74	0	0	0	0	-	19	11	119	15	1.5e-03
77	6	20	14	4	1.1e-03	13	33	213	11	6.0e-28
78	0	0	2	0	-	3	8	181	6	4.1e-16
79	10	4	52	0	-	33	10	203	6	3.3e-06
81	0	0	0	0	-	9	57	95	15	9.5e-21
82	0	0	0	0	-	8	4	212	6	1.5e-05
85	0	1	38	1	-	14	61	496	29	5.3e-65
88	0	0	0	0	-	3	34	269	16	6.8e-41
90	2	4	76	2	4.4e-07	10	4	162	4	1.0e-04
99	0	0	8	0	-	6	3	100	7	3.3e-02
100	0	0	2	0	-	4	6	64	34	2.2e-01
101	0	2	2	0	-	6	4	60	6	2.8e-02
102	0	0	0	0	-	7	9	109	5	5.3e-09
Overall	148	344	2114	122	2.9e-259	1506	3986	16208	1338	0.0e+00

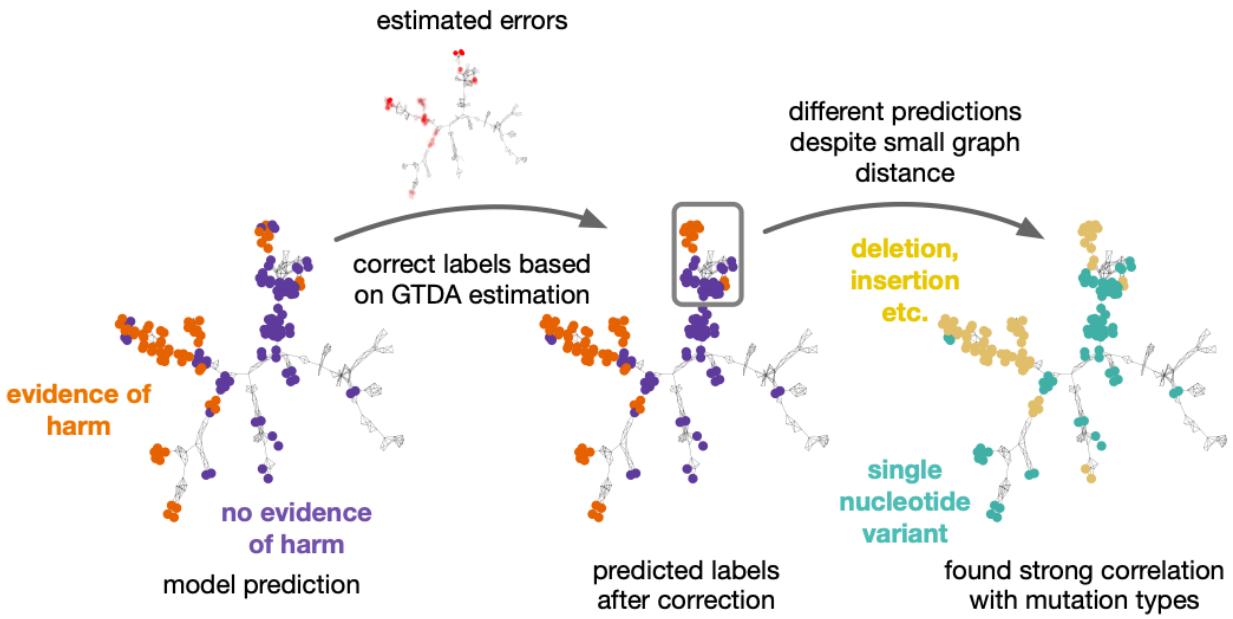


Figure 7.21. We zoom in one component GTDA finds and only show mutation records that happen in the protein coding regions (non-coding regions are not shown as colored dots, but do impact the Reeb net structure). After correcting prediction based on GTDA error estimation, we still see records that happen in a small region of the protein but still get different predictions. By checking these records, such difference can be well explained by different mutation types.

find they are either testing experiments with insignificant or conflicting results or affected by nearby insignificant training experiments.

To understand this effect across all components found by GTDA, we use the difference between the true presence of an error and our estimate. For instance, if GTDA estimation on whether a prediction is wrong is 0.3 and the prediction is indeed wrong based on its true label, such difference will be 1 minus 0.3. In total, we can find 2,031 GTDA error estimations where such difference is larger than 0.5. These are spread over 771 Reeb nodes. Since an error estimation being wrong can be due to either its own label being unreliable or training samples nearby have unreliable labels, we study how many of those 771 Reeb nodes have at least 1 insignificant or conflicting samples (either training or testing sample). We find 662

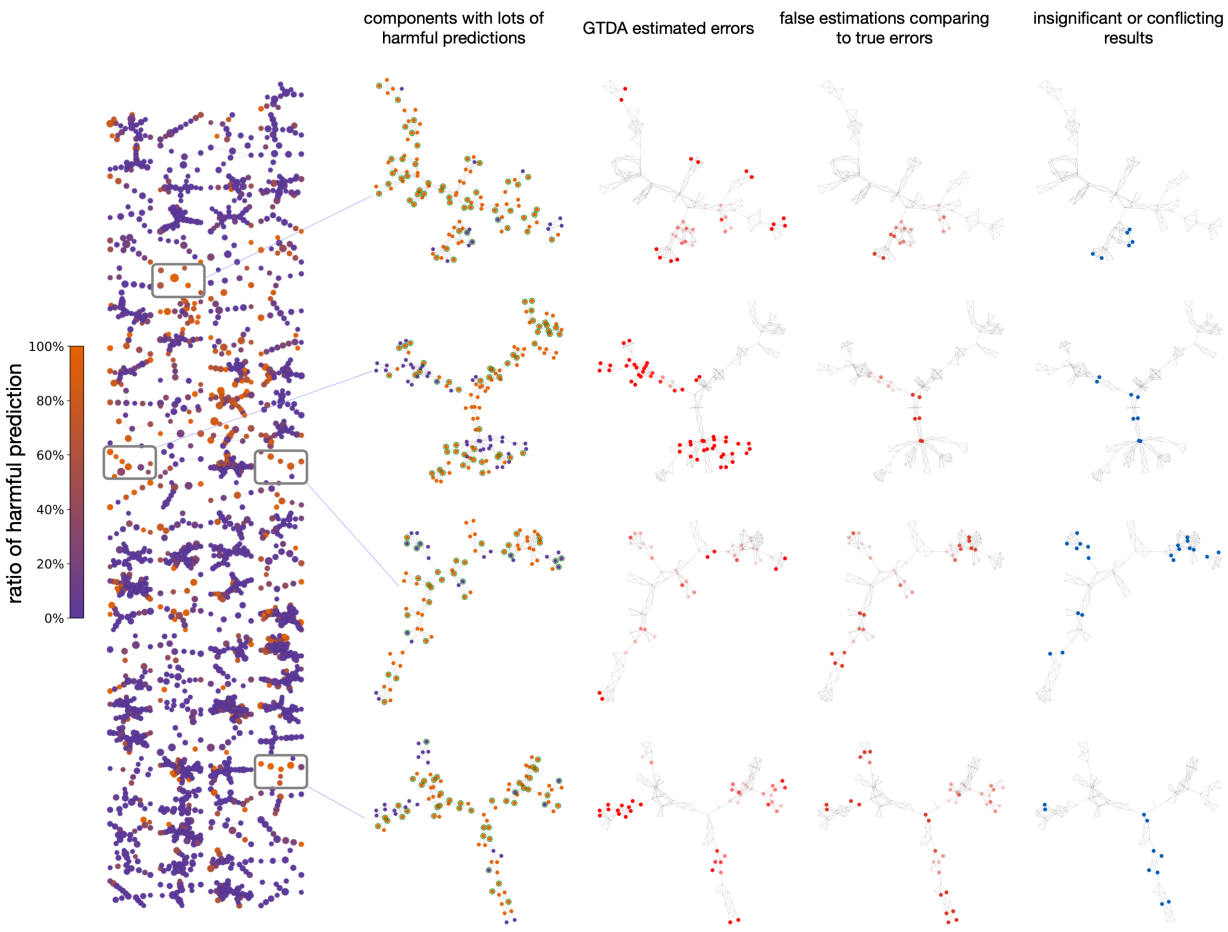


Figure 7.22. Checking false error estimations of GTDA in some components reveals that they are likely to be caused by variants experiments with insignificant or conflicting results.

of them (81%) have at least one problematic label. Consequently, the intuition from Figure 24 would hold across much of the dataset.

7.6.7 Comparison with other methods

To quantify the difference in performance between GTDA and other methods as shown in Figure 7.17, we first convert UMAP and tSNE visualizations into graphs by building a 5-NN graph on top of the 2 dimensional embedding. For GTDA and Mapper, we project each Reeb net node using 13 to get the corresponding graphs. We also add the original

Table 7.5. The ks statistics and p-value of the one tailed KolmogorovSmirnov test. The null assumption is that the ecdf of GTDA is larger than the ecdf of other methods at all locations.

	GTDA v.s. tSNE	GTDA v.s. UMPA	GTDA v.s. Mapper
ratio within the same exon	$(0.23, p < 10^{-10})$	$(0.35, p < 10^{-10})$	$(0.99, p < 10^{-10})$
ratio within a small range	$(0.33, p < 10^{-10})$	$(0.40, p < 10^{-10})$	$(0.97, p < 10^{-10})$

graph that has been used as input to GTDA and 100 random graphs by shuffling edges for comparison. Then we design the following metrics:

- ratio of samples within the same exon: in this metric, for each mutation sample that overlaps with an exon, we search the neighbors within 3 hops on each graph and compute the ratio of mutation samples that overlap with the same exon. Note that we only consider exons that encodes 1JNX.
- ratio of samples within a small range: in this metric, for each mutation sample, we search the neighbors within 3 hops on each graph and compute the ratio of mutation samples whose mutation starting coordinates are within 1000 base pairs of the starting coordinate of the selected mutation sample.

We also consider the corresponding ratio to be zero if the number of neighbors within 3 hops is smaller than 5. This is because the visualization of Mapper has too many single nodes or tiny components which could result in better metrics despite the visualization itself is much worse. In Figure 7.23, we compare the empirical cumulative distribution function of the ratio distributions. For each of the 100 random graphs, we compute the ecdf and report the average of the 100 ecdf curves. We can first notice that comparing to random graphs, the ratio in both metrics is much higher in the original graph, which means mutation samples are indeed significantly localized in the original graph. Also, GTDA performs the best on both metrics., which can be verified by the KolmogorovSmirnov test in Table 7.5.

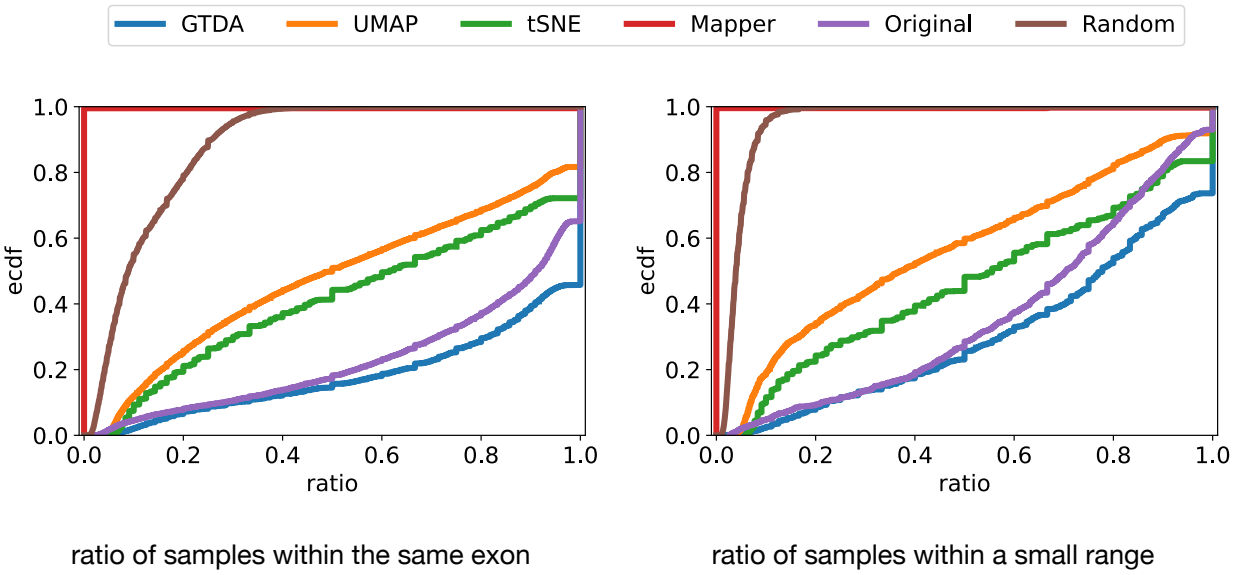


Figure 7.23. Overall GTDA performs the best on both metrics, while the other methods are not clearly better or even worse than the original graph. This suggests (1) the strong location sensitivity of mutation samples indeed exist in the original graph (2) GTDA can not only preserve and enhance such location sensitivity, but also visualize such property easily.

7.7 Comparing models on ImageNet-1k predictions

In this section, we apply GTDA framework on the entire ImageNet dataset with 1000 classes from 2012 [133] to compare performance between state of the art CNN models and historical models in any individual class. The results in the later sections show that GTDA can highlight which subgroups inside a class the more advanced models can have improved performance. It also shows how models predict when the image itself has confusing labels.

7.7.1 Dataset and CNN models

We use the training and validation images of entire ImageNet dataset with 1000 classes that was released in 2012 [133]. We use 3 different CNN models for comparison, AlexNet, ResNet-50 and VOLO. AlexNet is one of the historical CNN models, with around 60% top-1 testing accuracy. ResNet is one of the most widely used CNN models nowadays with

a better performance of about 75% top-1 testing accuracy. Finally, VOLO is one of the state of art CNN models that achieves about 87% top-1 testing accuracy without using any additional training data. Then, for each CNN model, we extract the prediction matrix and the image embeddings. For AlexNet and ResNet, the image embeddings are the outputs of the layer before final prediction layer. Similar to the previous section, we replace the average pooling by max pooling in the last convolutional layer. For VOLO, we directly used the dedicated feature forwarding function to get image embeddings. Similar to previous sections, all image embeddings are PCA reduced to 128 with whitening and normalization. For GTDA parameters, we set $K = 25$, $d = 0.001$, $r = 0.01$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$ and $S = 10$. We use 10 steps of iterations for GTDA error estimation.

7.7.2 Building graphs and initial results of GTDA

We first compare AlexNet and ResNet. To do so, we build a 5-NN graph using the image embeddings of ResNet only. Then we concatenate the prediction matrix of AlexNet and ResNet to get 2,000 lens. GTDA framework is then applied using the same set of parameters as Section 7.5. Similarly, to compare ResNet and VOLO, we build a 5-NN graph using the image embeddings of VOLO and concatenate the prediction matrix of ResNet and VOLO. In Table 7.6, we provide some initial statistics on the final Reeb nets. We can see that despite the Reeb net has tens of thousands of nodes, the maximum Reeb component size is just a few hundred of nodes, which guarantees that we can easily visualize any component of the Reeb net.

7.7.3 Highlighting subgroups where advanced models perform better

Figure 7.24 shows the results on one class, "screwdriver", from GTDA when comparing AlexNet with ResNet. AlexNet and ResNet have huge difference in terms of training or validation accuracy as shown in (A) of Figure 7.24. By embedding images on top of each component, we can find different subgroups inside the "screwdriver" class, where some groups like (B) or (C) can be predicted with high accuracy by both models, while for some other groups like (D), (E) or (F), only ResNet can maintain the high accuracy. By showing some

Table 7.6. Statistics on Reeb nets. Reeb node size is the number of samples represented in a Reeb net node. Average Reeb components for each class is the average number of Reeb net components where the most frequent predicted label (by one of the two models) is that class. The maximum Reeb component just has a few hundred of nodes, which guarantees that any component of the Reeb net can be easily visualized and analyzed.

	AlexNet v.s. ResNet	ResNet v.s. VOLO
original graph nodes	1,331,167	1,331,167
original graph edges	5,954,900	5,805,714
Reeb nodes	63,239	68,354
Reeb edges	59,881	64,360
Reeb components	3,395	4,046
max Reeb component size	169	79
max Reeb node size	330	643
average Reeb components for each class	3.5	4.0

example images from each group, we can see that in general, AlexNet can only find the screwdriver if both the handle and the tip are clear enough in the image. Showing only some part of the screwdriver or having a slightly complex background will likely cause AlexNet to fail. Similarly, Figure 7.25 compares with prediction of ResNet and VOLO on the "hook" class. We first find subgroups of images that shows a single hook where both model have high accuracy in group (B). Then we find ResNet model often prefers to predict chain instead of hook if they are both present in the image from group (C). ResNet model also has difficulty predicting hook if only part of the hook is shown (D), or the shape of the hook is not common (G) and (F), or there are lots of hooks in the image (E).

7.7.4 Understanding different models' predictions

In Figure 7.26, we compare the the predictions between ResNet and VOLO on "desktop computer". Both models have very similar training or validation accuracy on this class. But they make mistakes in different places. We highlight a few subgroups where we can see lots of difference in predicted labels. These subgroups contain images that are indeed confusing. For instance, images in group (D) clearly have a desk, a monitor and a desktop computer at the same time. We can see VOLO tends to predict all these confusing images as "desktop computer", even though the true labels for some of those images are different. This suggests the VOLO prediction of "desktop computer" is more robust, while the ResNet prediction is more likely to be affected by other objects in the image.

7.8 Inspecting chest X-ray images

In this section, we apply our GTDA framework to inspect the prediction of disease on 112,120 images of chest X-rays [134]. Each X-ray image might be either normal or indicating one or more diseases. Our results show that GTDA is very useful to help radiologists detect images with incorrect normal and abnormal labels.

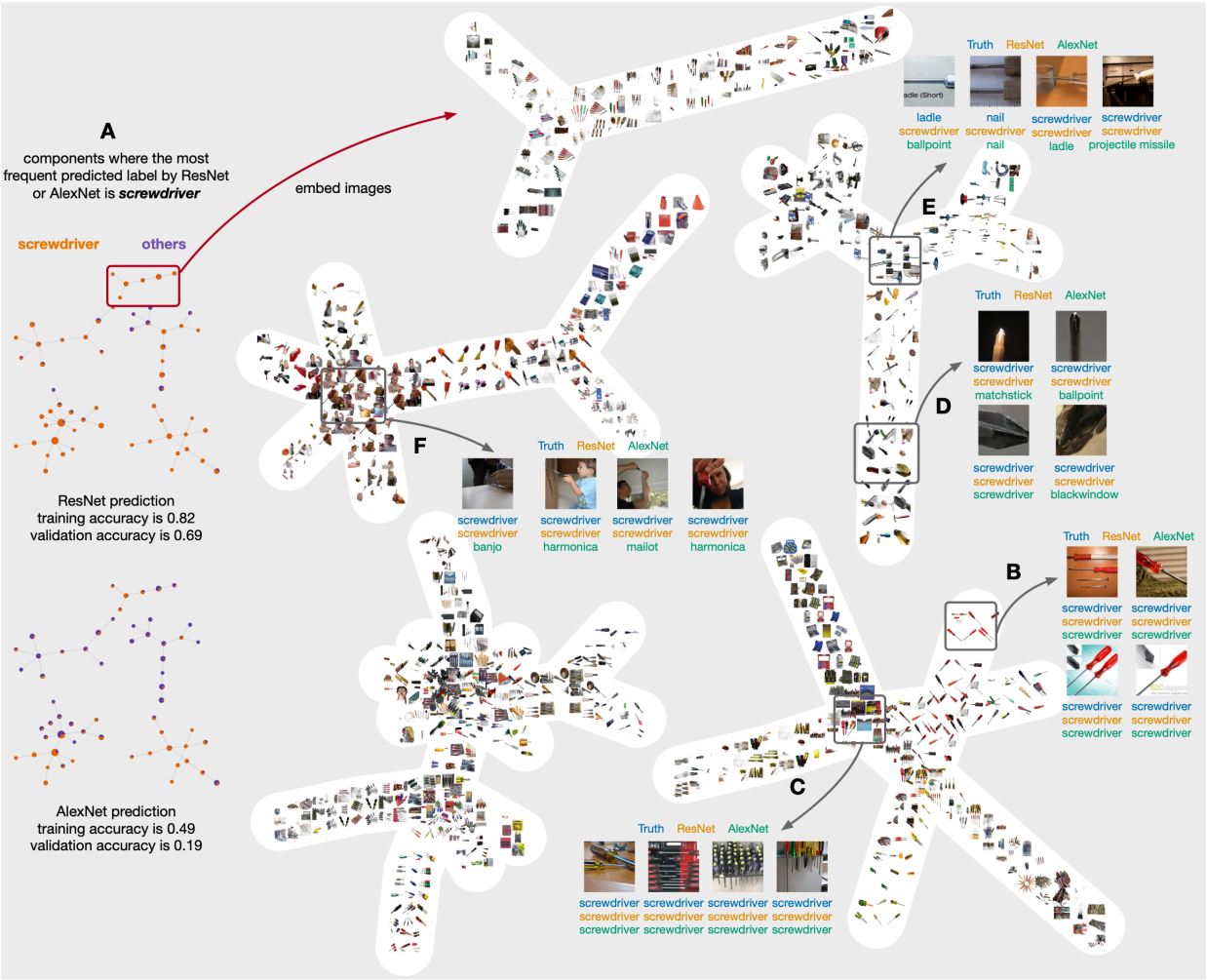


Figure 7.24. In this figure, we analyze the prediction of "screwdriver" from both ResNet and AlexNet. We can see AlexNet can only predict "screwdriver" with high accuracy if both handle and the tip are clearly visible in the image (see B and C). Otherwise, if only the tip (D) or a small part of the handle (E) is shown or the image is about a person using a screwdriver (F), AlexNet will likely fail while ResNet still maintains high accuracy.

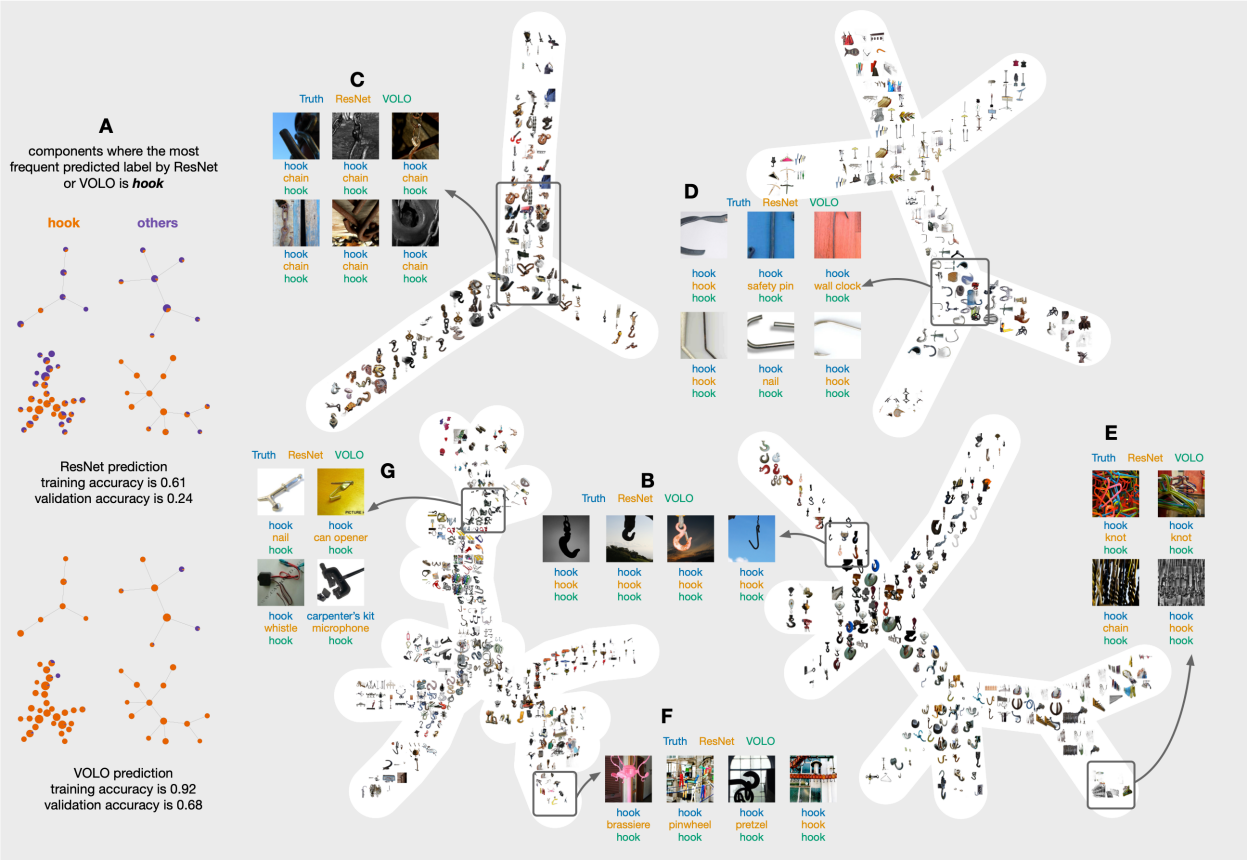


Figure 7.25. In this figure, we analyze the prediction of "hook" from both ResNet and VOLO. VOLO has much higher training and validation accuracy on this class than ResNet (A). We first find subgroups of images that shows a single hook where both model have high accuracy (B). Then we find ResNet model often prefers to predict chain instead of hook if they are both present in the image (C). ResNet model also has difficulty predicting hook if only part of the hook is shown (D), or the shape of the hook is not common (G) and (F), or there are lots of hooks in the image (E).

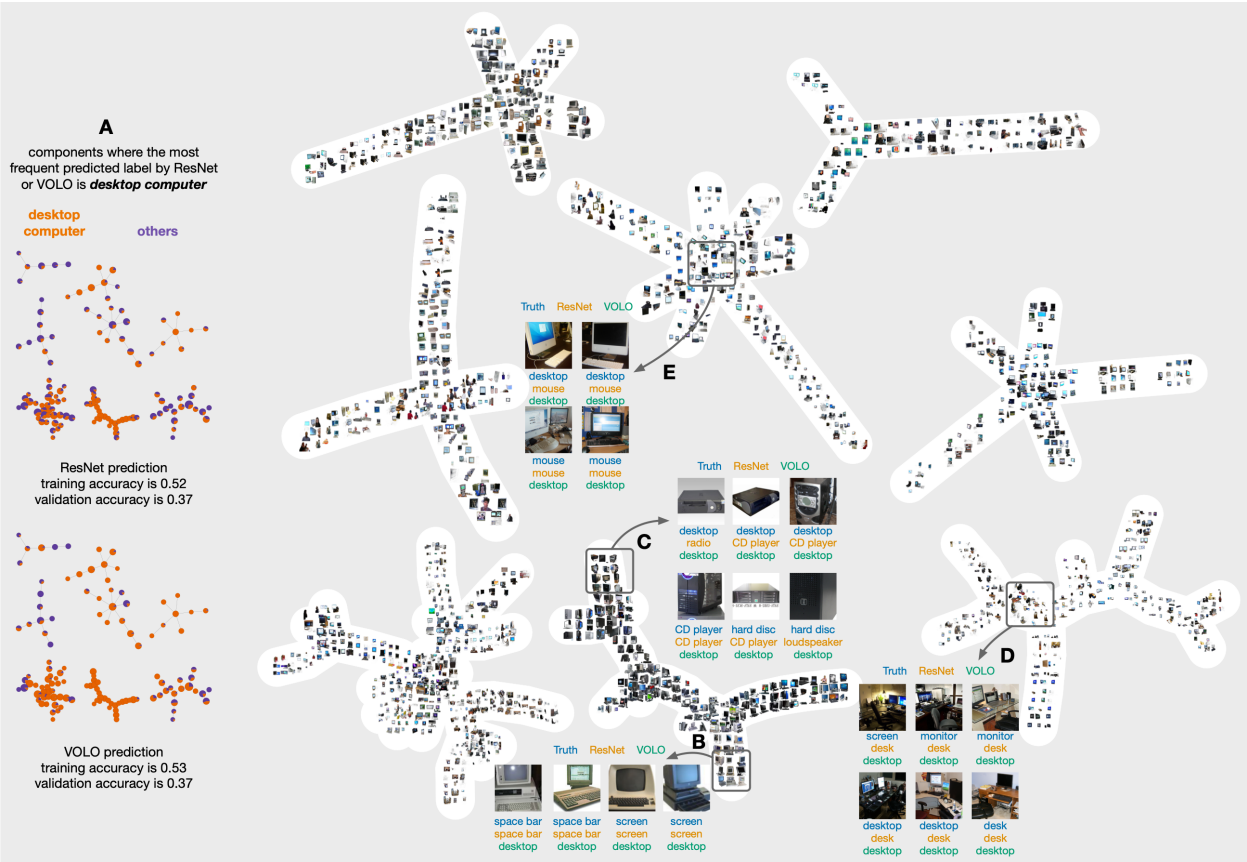


Figure 7.26. In this figure, we analyze the prediction of "desktop computer" from both ResNet and VOLO. In (A), we show all components GTDA has found where "desktop computer" is the most frequent predictions. ResNet and VOLO show very close training and validation accuracy on these components. By embedding images on them, we can first find subgroups of images that look confusing. For instance, some images in (B) have labels like "space bar" or "screen" despite they are just old fashioned desktop computers. Images in (C) show some "CD player" or "hard disc" that look very similar to PC chassis. Images in (D) have desk, desktop computer and monitor at the same time. And some images in (E) are labeled as "mouse" even if they also contain a monitor or a keyboard. We can also notice how ResNet and VOLO handle these confusing images differently. Overall, VOLO's prediction on "desktop computer" is more robust and less affected by other objects in the image or similar objects from other classes.

7.8.1 Dataset and model

The NIH ChestX-ray14 dataset we use comprises 112,120 de-identified frontal-view X-ray images of 30,805 unique patients [134]. Among these images, 86,524 images are used as training or validation and the others are used as testing. Images are split at the patient level, which means images belonging to the same patient will be put in the same group. Among the 86,524 images, we randomly choose 20% patients and use their associated images as validation data while images for the other patients are used as training data. In the original dataset, a text mining approach is used on the associated radiological reports to find the existence of 14 possible diseases and one image can have multiple disease labels. As a result, it is expected that many of the labels assigned are incorrect. In some other studies of these data, expert labels are solicited for 810 selected testing images from multiple experienced radiologists [135].

The model we use GTDA to study is called CheXNet [136] which is a 121-layer Dense Convolutional Network (DenseNet) [137]. When applying our GTDA framework, we first reduce the 14 disease predictions to a simple normal (label 0) vs abnormal (label 1) prediction. To do so, we first take a row wise maximum to reduce the prediction matrix for 14 disease into a vector v with values ranging 0 to 1. Then we consider each individual value as a threshold and generate predicted labels by treating values larger than this threshold as 1 or 0 otherwise. Then we compute the F1 score using the union of training and validation data. The threshold that gives the largest F1 score will be kept, denoted as t. Similar procedures have been used in other papers that predict ontological annotations [138, 139]. Finally, we transform each value of v using $v_i = \min(1, 0.5v_i/t)$. The transformed v also ranges from 0 to 1 and is considered as the probability of being abnormal. As a result, the row wise maximum column index of the new prediction matrix $\boldsymbol{P} = [1 - v, v]$ will give the same largest F1 score. Other than the abnormal vs normal lens, we also include the original disease prediction matrix as the extra lenses. This process gives 16 lenses in total. For GTDA parameters, we set $K = 50$, $d = 0$, $r = 0.005$, $s_1 = 5$, $s_2 = 5$, $\alpha = 0.5$ and $S = 10$. We use 10 iterations for GTDA error estimation.

7.8.2 GTDA finds incorrect normal vs abnormal labels

Out of the 810 images in the test set with expert labels, 222 images have incorrect normal vs abnormal labels. Our goal is to use the GTDA visualization to find images in this set (i.e. those that are more likely to an incorrect label). The procedure of finding those images is similar to find insignificant or conflicting gene mutation experiments from the previous section.

We first use GTDA to estimate prediction errors. The estimation is normalized to a number between 0 and 1. Then we use the original testing labels (i.e. without the correction from experts) to find which of these error estimates are wrong. We can then sort the test samples in the order of descending absolute difference between estimated error and true error.

For simplicity, images in the test set where such differences are larger than 0.5 are considered to have incorrect labels. A demonstration on this process can be found in Figure 7.27. Overall, out of the 810 testing images with expert labels, GTDA highlights 265 images are likely to have incorrect normal vs abnormal labels and 138 of them are confirmed by the expert labels, which gives a precision of 0.52 and a recall of 0.62. As a comparison, randomly sampling 265 images for experts to check can only find around 73 images with incorrect labels in average. More detailed results on each component are shown in Table 7.7. By testing multiple thresholds instead of 0.5, we get an AUC score of 0.75. As a comparison, using self confidence [120] gives an overall AUC score of 0.60.

7.9 Parameter selection of GTDA

In this section, we will discuss how to select parameters for our GTDA framework, especially the component size threshold and overlapping ratio in Algorithm 9. Currently, we manually focus the Reeb net's structure by varying these parameters. It remains an open question on how one might automatically select parameters for our GTDA framework as proposed for other TDA frameworks [140]. Although GTDA has 8 parameters (Table 7.1), the two most important are the component size threshold and the overlapping ratio.

Figure 7.27. We give a demonstration on how to use GTDA results to find which testing labels are likely to be problematic. We first zoom in a component found by GTDA and use green circles to mark testing images where we have expert labels (A). Then we use GTDA to estimate prediction errors on circled images (B). Comparing GTDA estimation with original testing labels can identify a few places with false estimations (C). We consider these false estimations are due to problematic testing labels and do a simple thresholding of 0.5, which flags 17 problematic testing labels in this component (D). Comparing to expert labels can find 14 true positives with a precision of 0.82 and a recall of 0.78 (E).

Table 7.7. Detailed precision and recall on different components when using GTDA to find likely incorrect testing labels of ChestX-ray14 dataset. Components are ordered by decreasing number of incorrect labels identified by experts they contain. Results for components with less than 3 incorrect labels are reported together.

Type	Expert Labels in Component	Incorrect by Experts	Flagged as Problematic	Precision	Recall
Single Component	53	18	17	0.82	0.78
Single Component	10	5	5	1.0	1.0
Single Component	9	5	4	0.25	0.2
Single Component	19	4	7	0.57	1.0
Single Component	9	4	5	0.8	1.0
Single Component	10	4	3	0.33	0.25
Single Component	7	4	2	1.0	0.5
Single Component	8	4	5	0.6	0.75
Single Component	14	4	4	1.0	1.0
Single Component	4	4	2	1.0	0.5
Single Component	7	4	3	0.33	0.25
Single Component	10	3	2	0.0	0.0
Single Component	6	3	1	0.0	0.0
Single Component	4	3	2	0.5	0.33
Single Component	6	3	3	0.33	0.33
Single Component	3	3	2	1.0	0.67
Single Component	5	3	3	1.0	1.0
Single Component	5	3	2	0.5	0.33
Single Component	8	3	5	0.4	0.67
Single Component	7	3	4	0.5	0.67
Single Component	19	3	8	0.25	0.67
Single Component	9	3	8	0.38	1.0
Single Component	8	3	3	0.33	0.33
Single Component	8	3	4	0.5	0.67
Components with 2 incorrect labels	135	56	50	0.74	0.66
Components with 1 incorrect label	219	67	78	0.5	0.58
Components with 0 incorrect label	208	0	33	0.0	NaN
Overall	810	222	265	0.52	0.62

7.9.1 Selecting component size threshold

Recall that the component size threshold is the smallest component where we stop splitting. Choosing a good component size threshold depends on the dataset we want to analyze. If the threshold is too small, we will end up with too many nodes to make the subsequent visualization and analysis difficult. On the other hand, if the threshold is too large, the

topological structure of some small classes might be over simplified and components from different classes can be mixed. Figure 7.28 shows how the Reeb net will change as we vary component size threshold. In general, we start from a larger component size threshold and then check the Reeb net we get, especially the size of the largest Reeb net component as well as whether different classes are mixed. If we have a component that is too large to be easily visualized or different classes are clearly mixed, we reduce this value. If the class sizes are highly skewed, we usually choose the threshold based on the smallest class. In this case, we the lower bound on the absolute difference of the lens parameter becomes useful. This is to avoid oversplitting class with large size, i.e., if the difference is smaller than the lower bound, we stop splitting as well.

We find the results are stable to the choices and in particular, for uses to find clues of possible predicting errors or labeling issues from the visualization. As we can see in Figure 7.28, choosing a threshold between 100 and 200 or choosing an overlapping ratio between 0.5% and 1.5% can all show the ambiguity in "Networking Products" v.s. "Routers" and some part of "Data Storage" v.s. "Computer Components" allowing human insight into the predictions.

7.9.2 Select overlapping ratio

The selection of overlapping ratio is similar to select component size threshold, we can start from a larger ratio like 10% and then check the Reeb net to see if there is any component that is too large or too mixed. If so, we need to gradually reduce the ratio until every component can be properly visualized by a simple layout algorithm like spring layout [141] or Kamada Kawai algorithm [119]. Figure 7.29 shows different Reeb nets as we vary overlapping ratio.

7.9.3 Notes on other parameters

Other than component size threshold and overlapping ratio, Algorithm 9 has several other parameters. Two important ones are the smallest node size and the smallest component size.

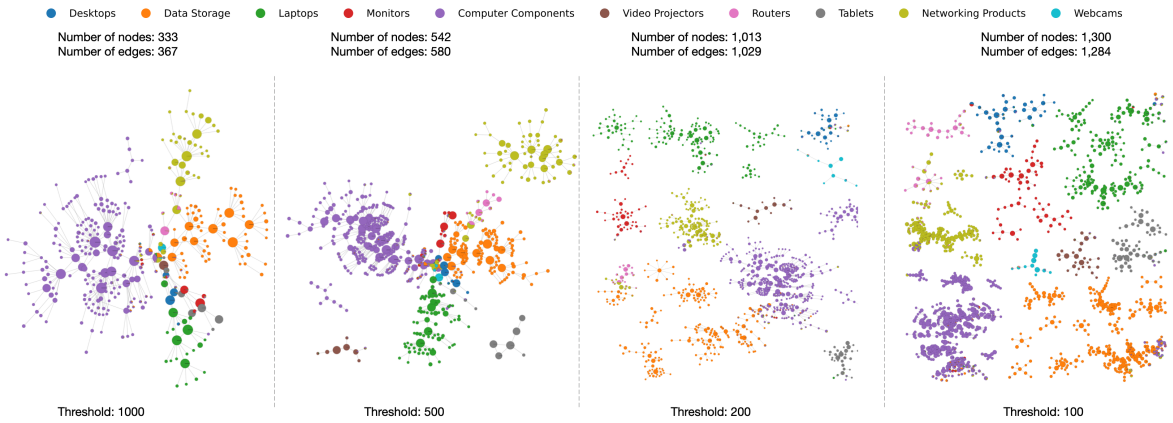


Figure 7.28. We show different GTDA visualizations as we vary the component size threshold. The overlapping ratio is fixed as 1%. Using a large threshold will cause different classes to be mixed together and the structure of small class like "Routers" or "Webcams" will be over simplified. As we gradually reduce the thresholds, the number of nodes and edges in the visualization will increase as well and different classes will be separated into several components. The results look similar between 100 and 200, which suggests GTDA structure are stable with respect to small change in parameters.

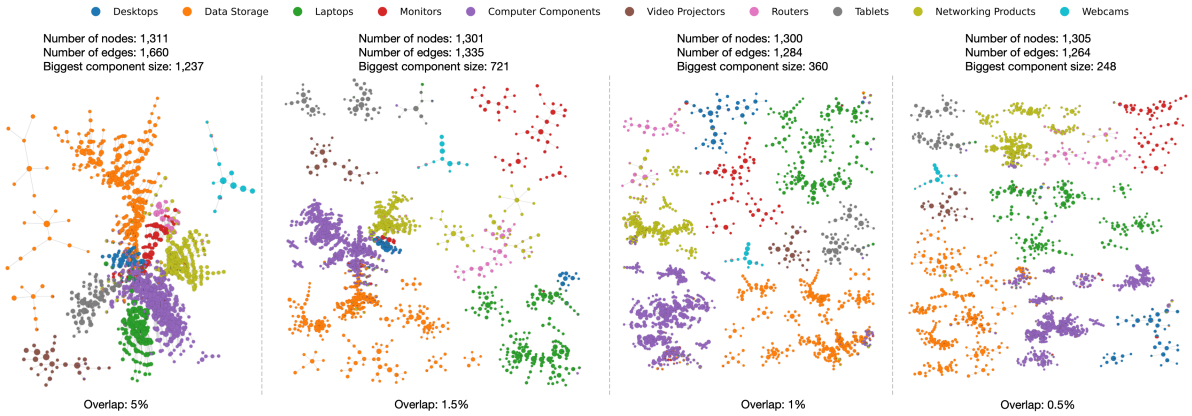


Figure 7.29. We show different GTDA visualizations as we vary the overlapping ratio. The component size threshold is fixed as 100. Using a large overlapping ratio will cause different classes to be mixed together and some components too large to be properly visualized. As we gradually reduce the overlapping ratio, different classes will be separated into several components with each one easier to be plotted. Similar ambiguity in "Networking Products" v.s. "Routers" and some part of "Data Storage" v.s. "Computer Components" can be observed for overlapping ratio between 0.5% and 1.5%.

In our experiments, we can get consistently good visualizations by requiring the size of any Reeb net node or Reeb net component larger than 5.

7.10 Performance and scaling

Our GTDA framework scales to predictions with thousands of classes and millions of datapoints. We only split along the lens with the maximum difference at each iteration, which can be easily recomputed in linear time in the data or even more efficiently updated. After each split, we immediately check all the connected components we have found, which can be done in $O(N + M)$ where N is the number of nodes and M is the number of edges.

It is difficult to estimate how many splitting iterations are needed. Assuming we have L lenses, initially the min-max difference across all lenses is 1 and the overlapping ratio is 0, then we will need at most L iterations before the largest min-max difference across all components is reduced to 0.5, which means at most $O(tL)$ iterations are needed to reduce such difference below 2^{-t}. If after a sufficient number of iterations, we still see large components with size bigger than K, it means new lenses are needed to further distinguish those nodes or a lower bound on the difference is needed to stop the splitting early.

Another step is to find out which pairs of components have overlap. This can be easily done in the original *mapper* algorithm by checking the adjacent bins of each bin. In our GTDA framework, we first build a bipartite graph with all component indices on one side, all samples on the other side and connecting each component index to all samples it includes. Then identifying the overlapping components is equivalent to find 2-hop neighbors of each component index, which can also be done in $O(N + M)$. Finally, for the merging step, since the size of each super node or the size of Reeb net component will be at least doubled, it needs at most $O(M(\log(s_1 s_2)))$ time. Also note that, many steps of our GTDA algorithm can be easily parallelized. In our code, we mainly parallelize the merging steps using 10 cores, which has already given reasonable running time on graphs with millions of nodes and edges.

Detailed running time for all datasets we have tested can be found in the table 7.8. All running time are reported on a server with 2 AMD EPYC 7532 processors (128 cores in total), 512 GB memory and one A100 GPU.

Table 7.8. Statistics on datasets and running time in seconds. Predicting and embedding represents the time used to generate prediction and extract embedding for all samples from a trained model. Preprocessing time includes PCA, normalization as well as building a KNN graph if the original dataset is not in graph format. GTDA time is the time to compute Reeb network given the input graph and the lens.

dataset	nodes	edges	classes	lens	predicting & embedding (s)	preprocessing (s)	GTDA time (s)
Swiss Roll	1,000	3,501	3	3	0.003	0.3	1
Amazon Computers	39,747	399,410	10	10	0.17	7	10
Subset of ImageNet	13,394	51,520	10	10	27	5	7
ImageNet-1k (ResNet vs AlexNet)	1,331,167	5,954,900	1,000	2,000	2,379	717	26,036
ImageNet-1k (VOLO vs ResNet)	1,331,167	5,805,714	1,000	2,000	13,426	617	18,894
BRCA1 Gene Variants	23,376	83,096	2	4	18,583	21	3
Chest X-rays	112,120	431,893	2	16	821	35	26

7.11 Comparing to tSNE and UMAP

The goals of the Reeb net analysis from GTDA are distinct from the goals of dimension reduction techniques such as tSNE and UMAP. We seek the topological information identified by the Reeb net. The Reeb net is both useful for generating pictures or maps of the data as well as the algorithmic error estimate. We use the Kamada-Kawai [119] method to compute a visualization of the Reeb net, which does have many similarities with summary pictures from tSNE and UMAP. We compare here GTDA results with visualization from tSNE [142] and UMAP [143, 144] on all 4 datasets of the main text. For tSNE, we directly use the implementation from *scikit-learn*. For UMAP, we use the implementation from https://umap-learn.readthedocs.io. The inputs to tSNE and UMAP are the concatenation of neural model embedding and prediction probability. We keep all parameters as default except setting the number of final dimension as 2. The results are shown in Figure 7.30.

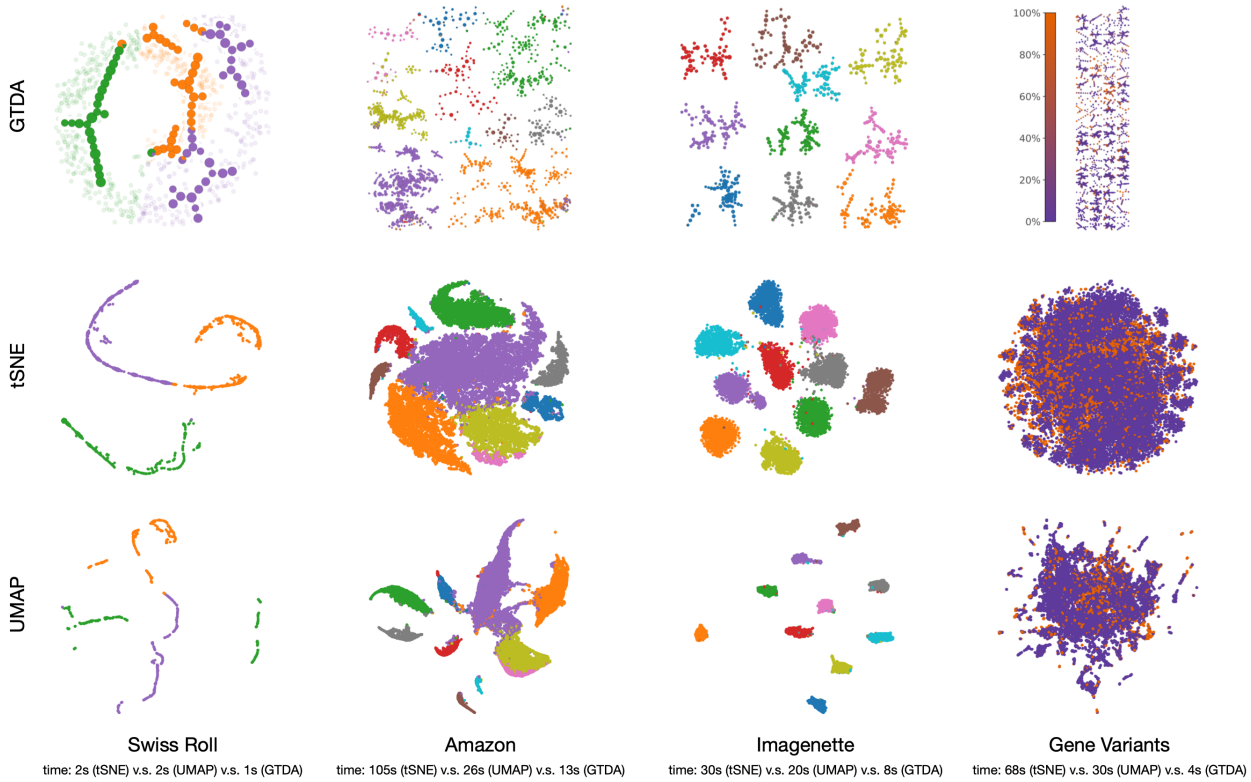


Figure 7.30. Comparing the results of the dimension reduction techniques tSNE and UMAP on 4 datasets to the topological Reeb net structure from GTDA shows similarities and differences among summary pictures generated by these methods. The graph created by GTDA permits many types of analysis not clearly possible with tSNE and UMAP output. For running time comparison, since we also need to extract model embeddings and predictions just like GTDA, we exclude such time and only report the time of the actual execution of tSNE or UMAP or GTDA (including Kamada-Kawai).

These pictures support different uses and purposes. Reeb nets from GTDA offer a number of compelling advantages as described throughout the main text and supplement. Among others, note that GTDA is faster than tSNE (2 to 15 times faster) and UMAP (2 to 8 times faster) in all 4 datasets. It also scales easily to datasets with millions of datapoints.

8. SUMMARY AND FUTURE DIRECTIONS

This thesis explores nonlinear graph diffusions for clustering, semi-supervised learning and analyzing complex predictions.

8.1 Conclusions in using nonlinear diffusions for local clustering

In the first part,we propose a generalized min-cut optimization problem 3.4 that unifies the objective functions behind some commonly used local graph clustering algorithms. When $\ell(x) = |x|$, this relates to flow-based algorithms for improving clusters which can also be though as the 1-norm variant of linear diffusions. The goal is to "refine" or "improve" local clusters assuming the target cluster has a substantial overlap with the input cluster. The objectives of the 3 specific flow-based algorithms (MQI, FLowImprove and LocalFlowImprove) can be further unified by equation 4.1 under different δ. We then illustrated how these flow-based algorithms can use applied in reducing conductance, semi-supervised learning or generating local coordinates to highlight local structures of graphs. We have also shown that these algorithms can scale to very large graphs as they are designed as outputting clusters without even touching most of the graph. These appealing features of flow-based algorithms motivate us to develop a software package to open the door for novel analyses of large graphs. This software can be freely obtained from https://github.com/kfoynt/LocalGraphClustering.

Although flow-based algorithms have very good theoretical guarantee in terms of cluster conductance, in practice, they often suffer from the fact that they cannot grow from small seed sets and can only work well when the input cluster has already had substantial overlap with target cluster. Linear diffusions or PageRank-based method can grow, but have difficulty finding the correct boundaries. These motivate us to design an algorithm that sits between flow-based methods and PageRank based methods. We have shown that by solving equation 3.4 with $\ell(x) = \frac{1}{p}|x|^p$, where $1 < p < 2$, we can often get clusters from small seed set that can capture the boundaries accurately as well. We then developed a strongly local nonlinear diffusion procedure based on Andersen-Chung-Lang *push* procedure for PageRank to approximately solve the new p-norm cut objective. Our nonlinear diffusion procedure is general enough that can be used to solve other types of cut functions like *q-Huber* or

Berq. We have then demonstrated that the new algorithm is every efficient and can achieve better performance in both local graph clustering and semi-supervised learning in real world datasets.

Comparing to graphs, hypergraphs provide more information. But it is not easy to directly adapt local graph clustering algorithms to the context of hypergraphs due to the complexities of possible hypergraph cut functions. Following some recent work [23] that generalizes flow-based local graph clustering algorithms to hypergraphs by reducing them to directed graphs, we have proposed a hypergraph version of the quadratic graph cut objective (equation 6.5). We have developed another strongly local diffusion procedure that can solve the new objective for all cardinality based hypergraph cut functions. This algorithm shares some similarity to the nonlinear diffusion process that is used for solving p-norm graph cut objective and can also grow a cluster from a small seed set and come with a conductance guarantee.

8.1.1 Future opportunities in local clustering

Although our nonlinear diffusions for solving p-norm cut objective has already achieved strongly local runtime guarantee, there are faster converging (in theory) solvers using different optimization procedures [145] for 2-norm problems as well as parallelization strategies [146]. It will be also interesting to further extend our hypergraph algorithm to solve objectives with hypergraph cut functions that are not cardinality based. As we were writing this thesis, some progress has already been made along this line [147].

8.2 Conclusions in using diffusions for analyzing predictions

It is always difficult to diagnose a machine learning model especially when the model is highly parameterized like the recent deep learning models. Prior work on analyzing deep learning methods for errors focuses on a single result list [100], without the associated topological structure provided by Reeb nets. Our graph-based topological data analysis framework that combines TDA with diffusion has made some progress towards this direction by transforming the complex predictions into a comprehensible prediction map to aid naviga-

tion of a large space of predictions to those most interesting areas. Beyond identifying that there is a problem, the insights from the topology suggest relationships to nearby data and thereby suggest mechanisms that could be addressed through future improvements. Considering the ability of these topological inspection techniques to translate prediction models into actionable human level insights from label issues to protein structure we expect them to be applicable to new models and predictions, broadly, as they are created and to be a critical early diagnostic of prediction models. The interaction of topology and prediction may provide a fertile ground for future improvements in prediction methods.

8.2.1 Future opportunities in GTDA

Other than the case studies we have shown, there are multiple variations that would be easy to adapt. For instance, we could easily combine multiple graphs from different sources to reveal potential errors that might hidden in a single source.

Areas for future improvement

Our current GTDA framework does rely on some tuning of parameters and manually finding any interesting local structures in the visualization, especially the component size threshold, which behaves similarly to bin size in the original TDA algorithm. While we designed the algorithm to be as robust as possible, it remains an open question on whether we can automatically select a good set of parameters and identify structures worth looking at. Existing work selects parameters for the original TDA framework based on statistical analysis [140]. But it is not clear how to extend such technique to our GTDA framework.

Areas for additional topology

Another direction is to study the outputs of GTDA under perturbations or filtrations over parameters. Alternatively, there are opportunities to utilize additional topological insights to improve the graph drawing. Consider that a study of persistence of structures in the graph should suggest their placement, i.e. two components that will be connected more easily by

perturbing parameters should be put closer. This can then lead to a better overall view of the entire dataset.

REFERENCES

[1] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[2] A. M. Lisewski *et al.*, "Supergenomic network compression and the discovery of exp1 as a glutathione transferase inhibited by artesunate," *Cell*, vol. 158, no. 4, pp. 916–928, 2014.

[3] B. Jiang, K. Kloster, D. F. Gleich, and M. Gribskov, "Aptrank: An adaptive pagerank model for protein function prediction on bi-relational graphs," *Bioinformatics*, vol. 33, no. 12, pp. 1829–1836, 2017.

[4] C.-H. Lin *et al.*, "Multimodal network diffusion predicts future disease-gene-chemical associations," *Bioinformatics*, J. Wren, Ed., bty858, Oct. 2018. DOI: 10.1093/bioinformatics/bty858.

[5] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, Jun. 2003. DOI: 10.1162/089976603321780317. eprint: http://www.mitpressjournals.org/doi/pdf/10.1162/089976603321780317.

[6] R. R. Coifman and S. Lafon, "Diffusion maps," *Applied and computational harmonic analysis*, vol. 21, no. 1, pp. 5–30, 2006.

[7] R. Ibrahim and D. F. Gleich, "Nonlinear diffusion for community detection and semi-supervised learning," in *The World Wide Web Conference*, ser. WWW '19, San Francisco, CA, USA: ACM, 2019, pp. 739–750, ISBN: 978-1-4503-6674-8. DOI: 10.1145/3308558.3313483.

[8] D. Wang, K. Fountoulakis, M. Henzinger, M. W. Mahoney, and S. Rao, "Capacity releasing diffusion for speed and locality," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 3598–3607.

[9] K. Lang and S. Rao, "A flow-based method for improving the expansion or conductance of graph cuts," in *IPCO 2004: Integer Programming and Combinatorial Optimization*, 2004, pp. 325–337.

[10] R. Andersen and K. J. Lang, "An algorithm for improving graph partitions," in *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, 2008, pp. 651–660.

[11] L. N. Veldt, D. F. Gleich, and M. W. Mahoney, "A simple and strongly-local flow-based method for cut improvement," in *International Conference on Machine Learning*, 2016, pp. 1938–1947. [Online]. Available: http://jmlr.org/proceedings/papers/v48/veldt16.html.

[12] D. Gleich and M. Mahoney, "Anti-differentiating approximation algorithms: A case study with min-cuts, spectral, and flow," in *International Conference on Machine Learning*, 2014, pp. 1018–1025.

[13] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14, New York, New York, USA: ACM, 2014, pp. 701–710, ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623732.

[14] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, San Francisco, California, USA: ACM, 2016, pp. 855–864, ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939754.

[15] N. Yadati, M. R. Nimishakavi, P. Yadav, V. Nitin, A. Louis, and P. Talukdar, "Hypergcn: A new method for training graph convolutional networks on hypergraphs," in *NeurIPS*, 2019.

[16] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *International Conference on Learning Representations (ICLR)*, 2019.

[17] Q. Li, X.-M. Wu, H. Liu, X. Zhang, and Z. Guan, "Label efficient semi-supervised learning via graph filtering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9582–9591.

[18] M. Liu and D. F. Gleich, "Strongly local p-norm-cut algorithms for semi-supervised learning and local graph clustering," in *Proceedings of NeurIPS*, Accepted, 2020.

[19] K. Fountoulakis, M. Liu, D. F. Gleich, and M. W. Mahoney, "Flow-based algorithms for improving clusters: A unifying framework, software, and performance," *arXiv*, vol. cs.LG, p. 2004.09608, 2020.

[20] M. Liu, N. Veldt, H. Song, P. Li, and D. F. Gleich, "Strongly local hypergraph diffusions for clustering and semi-supervised learning," in *TheWebConf 2021*, vol. cs.SI, 2021.

[21] M. Liu, T. K. Dey, and D. F. Gleich, "Topological structure of complex predictions," *arXiv preprint arXiv:2207.14358*, 2022.

[22] P. Li and O. Milenkovic, "Inhomogeneous hypergraph clustering with applications," in *NeurIPS*, 2017, pp. 2308–2318. [Online]. Available: http://papers.nips.cc/paper/6825-inhomogeneous-hypergraph-clustering-with-applications.pdf.

[23] N. Veldt, A. R. Benson, and J. Kleinberg, "Minimizing localized ratio cut objectives in hypergraphs," in *KDD*, 2020, pp. 1708–1718.

[24] R. Andersen, F. Chung, and K. Lang, "Local graph partitioning using pagerank vectors," in *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, IEEE, 2006, pp. 475–486.

[25] D. F. Gleich, "Pagerank beyond the web," *SIAM Review*, vol. 57, no. 3, pp. 321–363, 2015.

[26] K. Kloster and D. F. Gleich, "Heat kernel based community detection," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14, New York, NY, USA: ACM, 2014, pp. 1386–1395, ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623706.

[27] F. Chung, "The heat kernel as the PageRank of a graph," *Proceedings of the National Academy of Sciences*, vol. 104, no. 50, pp. 19 735–19 740, Dec. 2007. DOI: 10.1073/pnas.0708838104.

[28] D. F. Gleich and M. W. Mahoney, "Using local spectral methods to robustify graph-based learning algorithms," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15, Sydney, NSW, Australia: ACM, 2015, pp. 359–368, ISBN: 978-1-4503-3664-2. DOI: 10.1145/2783258.2783376.

[29] A. N. Langville and C. D. Meyer, "Google's pagerank and beyond," in *Google's PageRank and Beyond*, Princeton university press, 2011.

[30] M. W. Mahoney, L. Orecchia, and N. K. Vishnoi, "A local spectral method for graphs: With applications to improving graph partitions and exploring data graphs locally," *Journal of Machine Learning Research*, vol. 13, pp. 2339–2365, Aug. 2012. [Online]. Available: http://www.jmlr.org/papers/volume13/mahoney12a/mahoney12a.pdf.

[31] D. F. Gleich, "PageRank beyond the web," *SIAM Review*, vol. 57, no. 3, pp. 321–363, Aug. 2015. DOI: 10.1137/140976649.

[32] D. A. Spielman and S.-H. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, 2004, pp. 81–90.

[33] F. R. Chung, *Spectral graph theory*. American Mathematical Soc., 1997, vol. 92.

[34] Z. A. Zhu, S. Lattanzi, and V. S. Mirrokni, "A local algorithm for finding well-connected clusters.," in *ICML (3)*, 2013, pp. 396–404.

[35] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2014, pp. 701–710.

[36] J. Klicpera, S. WeiSSenberger, and S. Günnemann, "Diffusion improves graph learning," *arXiv preprint arXiv:1911.05485*, 2019.

[37] J. Zhao, Y. Dong, M. Ding, E. Kharlamov, and J. Tang, "Adaptive diffusion in graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 23 321–23 333, 2021.

[38] E. Chien, J. Peng, P. Li, and O. Milenkovic, "Adaptive universal generalized pagerank graph neural network," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=n6jl7fLxrP.

[39] F. Henderson and R. Wooding, "Overland flow and groundwater flow from a steady rainfall of finite duration," *Journal of Geophysical Research*, vol. 69, no. 8, pp. 1531–1540, 1964.

[40] J. L. Vázquez, "Perspectives in nonlinear diffusion: Between analysis, physics and geometry," in *Volumes II and III were printed before the Congress and distributed to the partici-pants in Madrid. They gather the articles written by the invited speakers in the different scientific sections of the Congress. The on-line version of these volumes is accessible at the address http://www. icm2006. org/proceedings*, 2007, p. 609.

[41] J. G. Berryman and C. J. Holland, "Nonlinear diffusion problem arising in plasma physics," *Physical Review Letters*, vol. 40, no. 26, p. 1720, 1978.

[42] J. King, "Extremely high concentration dopant diffusion in silicon," *IMA journal of applied mathematics*, vol. 40, no. 3, pp. 163–181, 1988.

[43] S. Amghibech, "Eigenvalues of the discrete p-laplacian for graphs," *Ars Comb.*, vol. 67, 2003.

[44] T. Bühler and M. Hein, "Spectral clustering based on the graph p-laplacian," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 81–88.

[45] M. Alamgir and U. V. Luxburg, "Phase transition in the family of p-resistances," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2011, pp. 379–387. [Online]. Available: http://papers.nips.cc/paper/4185-phase-transition-in-the-family-of-p-resistances.pdf.

[46] N. Brindle and X. Zhu, *P-voltages: Laplacian regularization for semi-supervised learning on high-dimensional data*. Workshop on Mining and Learning with Graphs, 2013. [Online]. Available: http://snap.stanford.edu/mlg2013/submissions/mlg2013_submission_6.pdf.

[47] P. Li and O. Milenkovic, "Submodular hypergraphs: P-laplacians, Cheeger inequalities and spectral clustering," in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, Stockholm Sweden: PMLR, Jul. 2018, pp. 3014–3023. [Online]. Available: http://proceedings.mlr.press/v80/li18e.html.

[48] K. Fountoulakis, D. Wang, and S. Yang, "P-norm flow diffusion for local graph clustering," in *Proceedings of the International Conference on Machine Learning*, 2020, pp. 5619–5629.

[49] S. Chanpuriya and C. Musco, *Infinitewalk: Deep network embeddings as laplacian embeddings with a nonlinearity*, 2020. arXiv: 2006.00094 [cs.LG].

[50] L. Orecchia and Z. A. Zhu, "Flow-based algorithms for local graph clustering," in *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2014, pp. 1267–1286.

[51] D. Lawlor, T. Budavári, and M. W. Mahoney, "Mapping the similarities of spectra: Global and locally-biased approaches to SDSS galaxy data," arXiv, Tech. Rep. 1609.03932, 2016, Preprint with expanded information.

[52] D. Lawlor, T. Budavári, and M. W. Mahoney, "Mapping the similarities of spectra: Global and locally-biased approaches to SDSS galaxies," *The Astrophysical Journal*, vol. 833, no. 1, p. 26, 2016.

[53] F. Chung, "Random walks and local cuts in graphs," *Linear Algebra and its Applications*, vol. 423, pp. 22–32, 2007.

[54] K. Fountoulakis, F. Roosta-Khorasani, J. Shun, X. Cheng, and M. W. Mahoney, "Variational perspective on local graph clustering," *Mathematical Programming B*, pp. 1–21, 2017.

[55] D. Wang, K. Fountoulakis, M. Henzinger, M. W. Mahoney, and S. Rao, "Capacity releasing diffusion for speed and locality," *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 3607–2017, 2017.

[56] J. Leskovec, K.J. Lang, and M.W. Mahoney, "Empirical comparison of algorithms for network community detection," in *WWW '10: Proceedings of the 19th International Conference on World Wide Web*, 2010, pp. 631–640.

[57] J. Shi and J. Malik, "Normalized cuts and image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 8, pp. 888–905, Aug. 2000, ISSN: 0162-8828. DOI: 10.1109/34.868688.

[58] D. Hallac *et al.*, "Snapvx: A network-based convex optimization solver," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 110–114, 2017.

[59] A. B. Owen, "A robust hybrid of lasso and ridge regression," *Contemporary Mathematics*, vol. 443, no. 7, pp. 59–72, 2007.

[60] 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*, vol. 6, no. 1, pp. 29–123, Sep. 2009. DOI: 10.1080/15427951.2009.10129177.

[61] L. G. S. Jeub, P. Balachandran, M. A. Porter, P. J. Mucha, and M. W. Mahoney, "Think locally, act locally: Detection of small, medium-sized, and large communities in large networks," *Phys. Rev. E*, vol. 91, p. 012 821, 1 Jan. 2015. DOI: 10.1103/PhysRevE.91.012821.

[62] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich, "Local higher-order graph clustering," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17, Halifax, NS, Canada: ACM, 2017, pp. 555–564, ISBN: 978-1-4503-4887-4. DOI: 10.1145/3097983.3098069.

[63] L. Orecchia and Z. A. Zhu, "Flow-based algorithms for local graph clustering," in *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, 2014, pp. 1267–1286.

[64] F. R. L. Chung, *Spectral Graph Theory*. American Mathematical Society, 1992.

[65] M. Mihail, "Conductance and convergence of markov chains-a combinatorial treatment of expanders," in *Foundations of Computer Science, 1989., 30th Annual Symposium on*, Oct. 1989, pp. 526–531. DOI: 10.1109/SFCS.1989.63529.

[66] N. Veldt, C. Klymko, and D. F. Gleich, "Flow-based local graph clustering with better seed set inclusion," in *Proceedings of the SIAM International Conference on Data Mining*, 2019, pp. 378–386. DOI: 10.1137/1.9781611975673.43.

[67] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[68] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Phys. Rev. E*, vol. 78, p. 046 110, 4 Oct. 2008. DOI: 10.1103/PhysRevE.78.046110.

[69] A. L. Traud, P. J. Mucha, and M. A. Porter, "Social structure of facebook networks," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 16, pp. 4165–4180, 2012, ISSN: 0378-4371. DOI: 10.1016/j.physa.2011.12.021.

[70] N. Veldt, A. Wirth, and D. F. Gleich, "Learning resolution parameters for graph clustering," in *The World Wide Web Conference*, ser. WWW '19, San Francisco, CA, USA: ACM, 2019, pp. 1909–1919, ISBN: 978-1-4503-6674-8. DOI: 10.1145/3308558.3313471.

[71] I. M. Kloumann and J. M. Kleinberg, "Community membership identification from small seed sets," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14, New York, New York, USA: ACM, 2014, pp. 1366–1375, ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623621. [Online]. Available: http://doi.acm.org/10.1145/2623330.2623621.

[72] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, "Group formation in large social networks: Membership, growth, and evolution," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '06, Philadelphia, PA, USA: ACM, 2006, pp. 44–54, ISBN: 1-59593-339-5. DOI: 10.1145/1150402.1150412. [Online]. Available: http://doi.acm.org/10.1145/1150402.1150412.

[73] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, Dec. 2012, pp. 745–754. DOI: 10.1109/ICDM.2012.138.

[74] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '07, San Diego, California, USA: ACM, 2007, pp. 29–42, ISBN: 978-1-59593-908-1. DOI: 10.1145/1298306.1298311.

[75] A. Benson, D. F. Gleich, and J. Leskovec, "Higher-order organization of complex networks," *Science*, vol. 353, no. 6295, pp. 163–166, 2016. DOI: 10.1126/science.aad9029.

[76] P. Li and O. Milenkovic, "Submodular hypergraphs: P-laplacians, Cheeger inequalities and spectral clustering," in *ICML*, vol. 80, 2018, pp. 3014–3023. [Online]. Available: http://proceedings.mlr.press/v80/li18e.html.

[77] Y. Takai, A. Miyauchi, M. Ikeda, and Y. Yoshida, "Hypergraph clustering based on pagerank," in *KDD*, 2020, pp. 1970–1978.

[78] S. Agarwal, K. Branson, and S. Belongie, "Higher order learning with graphs," in *ICML*, 2006, pp. 17–24, ISBN: 1-59593-383-2. DOI: 10.1145/1143844.1143847. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143847.

[79] M. Hein, S. Setzer, L. Jost, and S. S. Rangapuram, "The total variation on hypergraphs - learning on hypergraphs revisited," in *NeurIPS*, 2013, pp. 2427–2435. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999792.2999883.

[80] S. Agarwal, J. Lim, L. Zelnik-Manor, P. Perona, D. Kriegman, and S. Belongie, "Beyond pairwise clustering," in *CVPR*, 2005, pp. 838–845, ISBN: 0-7695-2372-2. DOI: 10.1109/CVPR.2005.89. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2005.89.

[81] N. Veldt, A. R. Benson, and J. Kleinberg, *Hypergraph cuts with general splitting functions*, 2020. arXiv: 2001.02817 [cs.DS].

[82] P. Li, N. He, and O. Milenkovic, "Quadratic decomposable submodular function minimization: Theory and practice," *JMLR*, vol. 21, pp. 1–49, 2020. [Online]. Available: http://jmlr.org/papers/v21/18-790.html.

[83] D. Zhou, J. Huang, and B. Schölkopf, "Learning with hypergraphs: Clustering, classification, and embedding," in *NeurIPS*, 2006, pp. 1601–1608. [Online]. Available: http://dl.acm.org/citation.cfm?id=2976456.2976657.

[84] R. Ibrahim and D. F. Gleich, "Local hypergraph clustering using capacity releasing diffusion," *arXiv*, vol. cs.SI, p. 2003.04213, 2020.

[85] R. Andersen and K. J. Lang, "An algorithm for improving graph partitions," in *SODA*, 2008, pp. 651–660. [Online]. Available: http://dl.acm.org/citation.cfm?id=1347082.1347154.

[86] J. Y. Zien, M. D. F. Schlag, and P. K. Chan, "Multilevel spectral hypergraph partitioning with arbitrary vertex sizes," *IEEE TCAD*, vol. 18, no. 9, pp. 1389–1399, 1999. DOI: 10.1109/43.784130.

[87] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in vlsi domain," *VLSI*, vol. 7, no. 1, pp. 69–79, Mar. 1999, ISSN: 1063-8210. DOI: 10.1109/92.748202.

[88] A. R. Benson, J. Kleinberg, and N. Veldt, "Augmented sparsifiers for generalized hypergraph cuts," *arXiv preprint arXiv:2007.08075*, 2020.

[89] Y. Yoshida, "Nonlinear laplacian for digraphs and its applications to network analysis," in *WSDM*, 2016, pp. 483–492.

[90] S. W. Hadley, "Approximation techniques for hypergraph partitioning problems," *Discrete Applied Mathematics*, vol. 59, no. 2, pp. 115–127, 1995, ISSN: 0166-218X. DOI: https://doi.org/10.1016/0166-218X(93)E0166-V. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0166218X93E0166V.

[91] E. Ihler, D. Wagner, and F. Wagner, "Modeling hypergraphs by graphs with the same mincut properties," *Information Processing Letters*, vol. 45, pp. 171–175, 1993, ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(93)90115-P. [Online]. Available: http://www.sciencedirect.com/science/article/pii/002001909390115P.

[92] E. L. Lawler, "Cutsets and partitions of hypergraphs," *Networks*, vol. 3, no. 3, pp. 275–285, 1973. DOI: 10.1002/net.3230030306.

[93] J. Ni, J. Li, and J. McAuley, "Justifying recommendations using distantly-labeled reviews and fine-grained aspects," in *EMNLP-IJCNLP*, 2019.

[94] . Avsec *et al.*, "Effective gene expression prediction from sequence by integrating long-range interactions," *Nature methods*, vol. 18, no. 10, pp. 1196–1203, 2021.

[95] A. Esteva *et al.*, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, Jan. 2017. DOI: 10.1038/nature21056. [Online]. Available: https://doi.org/10.1038/nature21056.

[96] M. Reichstein *et al.*, "Deep learning and process understanding for data-driven earth system science," *Nature*, vol. 566, no. 7743, pp. 195–204, Feb. 2019. DOI: 10.1038/s41586-019-0912-1. [Online]. Available: https://doi.org/10.1038/s41586-019-0912-1.

[97] R. J. L. Townshend *et al.*, "Geometric deep learning of RNA structure," *Science*, vol. 373, no. 6558, pp. 1047–1051, Aug. 2021. DOI: 10.1126/science.abe5650. [Online]. Available: https://doi.org/10.1126/science.abe5650.

[98] J. R. Zech, M. A. Badgeley, M. Liu, A. B. Costa, J. J. Titano, and E. K. Oermann, "Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study," *PLOS Medicine*, vol. 15, no. 11, A. Sheikh, Ed., e1002683, Nov. 2018. DOI: 10.1371/journal.pmed.1002683. [Online]. Available: https://doi.org/10.1371/journal.pmed.1002683.

[99] L. Oakden-Rayner *et al.*, "Validation and algorithmic audit of a deep learning system for the detection of proximal femoral fractures in patients in the emergency department: A diagnostic accuracy study," *The Lancet Digital Health*, vol. 4, no. 5, e351–e358, May 2022. DOI: 10.1016/s2589-7500(22)00004-8. [Online]. Available: https://doi.org/10.1016/s2589-7500(22)00004-8.

[100] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *International conference on machine learning*, PMLR, 2017, pp. 1885–1894.

[101] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *ICLR (workshop track)*, 2015. [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2015/DB15a.

[102] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2921–2929.

[103] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 618–626.

[104] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," in *ICLR (workshop track)*, Y. Bengio and Y. LeCun, Eds., 2014. [Online]. Available: http://arxiv.org/abs/1312.6034.

[105] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[106] M. T. Ribeiro, S. Singh, and C. Guestrin, """ why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[107] P. Y. Lum *et al.*, "Extracting insights from the shape of complex data using topology," *Scientific reports*, vol. 3, no. 1, pp. 1–8, 2013.

[108] G. Singh, F. Mémoli, and G. E. Carlsson, "Topological methods for the analysis of high dimensional data sets and 3d object recognition.," *SPBG*, vol. 91, p. 100, 2007.

[109] M. Nicolau, A. J. Levine, and G. Carlsson, "Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival," *Proc. Natl. Acad. Sci.*, vol. 108, no. 17, pp. 7265–7270, Apr. 2011. DOI: 10.1073/pnas.1102826108. [Online]. Available: https://doi.org/10.1073/pnas.1102826108.

[110] B. Strodthoff and B. Jüttler, "Layered reeb graphs for three-dimensional manifolds in boundary representation," *Computers & Graphics*, vol. 46, pp. 186–197, 2015, Shape Modeling International 2014, ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2014.09.026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0097849314001149.

[111] G. Naitzat, A. Zhitnikov, and L. Lim, "Topology of deep neural networks," *J. Mach. Learn. Res.*, vol. 21, 184:1–184:40, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-345.html.

[112] A. Rathore, N. Chalapathi, S. Palande, and B. Wang, "Topoact: Visually exploring the shape of activations in deep learning," in *Computer Graphics Forum*, Wiley Online Library, vol. 40, 2021, pp. 382–397.

[113] R. B. Gabrielsson and G. Carlsson, "Exposition and interpretation of the topology of neural networks," in *2019 18th ieee international conference on machine learning and applications (icmla)*, IEEE, 2019, pp. 1069–1076.

[114] M. Hajij, G. Zamzmi, and F. Batayneh, "Tda-net: Fusion of persistent homology and deep learning features for covid-19 detection from chest x-ray images," in *2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, IEEE, 2021, pp. 4115–4119.

[115] E. R. Love, B. Filippenko, V. Maroulas, and G. Carlsson, "Topological deep learning," *arXiv preprint arXiv:2101.05778*, 2021.

[116] C. Bodnar, C. Cangea, and P. Liò, "Deep graph mapper: Seeing graphs through the neural lens," *Frontiers in big Data*, vol. 4, 2021.

[117] M. Hajij, P. Rosen, and B. Wang, "Mapper on graphs for network visualization," *arXiv preprint arXiv:1804.11242*, 2018.

[118] T. K. Dey, F. Mémoli, and Y. Wang, "Multiscale mapper: Topological summarization via codomain covers," in *Proceedings of the twenty-seventh annual acm-siam symposium on discrete algorithms*, SIAM, 2016, pp. 997–1013.

[119] T. Kamada, S. Kawai, *et al.*, "An algorithm for drawing general undirected graphs," *Information processing letters*, vol. 31, no. 1, pp. 7–15, 1989.

[120] C. G. Northcutt, L. Jiang, and I. L. Chuang, "Confident learning: Estimating uncertainty in dataset labels," *Journal of Artificial Intelligence Research (JAIR)*, vol. 70, pp. 1373–1411, 2021.

[121] Q. Huang, H. He, A. Singh, S.-N. Lim, and A. Benson, "Combining label propagation and simple models out-performs graph neural networks," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=8E1-f3VhX1o.

[122] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[123] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, 2015, pp. 43–52.

[124] B. Zhao, F. Li, and E. Xing, "Large-scale category structure aware image categorization," in *Advances in Neural Information Processing Systems*, vol. 24, 2011. [Online]. Available: https://proceedings.neurips.cc/paper/2011/file/d5cfead94f5350c12c322b5b664544c1-Paper.pdf.

[125] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2016. DOI: 10.1109/cvpr.2016.90. [Online]. Available: https://doi.org/10.1109/cvpr.2016.90.

[126] J. Howard, *Imagenette dataset*, https://github.com/fastai/imagenette, 2021.

[127] E. Tufte, *Seeing with fresh eyes: Meaning, Space, Data, Truth*. Graphics Press, 2020.

[128] A. S. Razavian, J. Sullivan, S. Carlsson, and A. Maki, "Visual instance retrieval with deep convolutional networks," *ITE Transactions on Media Technology and Applications*, vol. 4, no. 3, pp. 251–258, 2016.

[129] X. Qin, Z. Zhang, C. Huang, M. Dehghan, O. R. Zaiane, and M. Jagersand, "U2-net: Going deeper with nested u-structure for salient object detection," *Pattern Recognition*, vol. 106, p. 107 404, 2020.

[130] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *KDD*, vol. 96, 1996, pp. 226–231.

[131] D. Shigaki *et al.*, "Integration of multiple epigenomic marks improves prediction of variant impact in saturation mutagenesis reporter assay," *Human mutation*, vol. 40, no. 9, pp. 1280–1291, 2019.

[132] M. J. Landrum *et al.*, "Clinvar: Improving access to variant interpretations and supporting evidence," *Nucleic acids research*, vol. 46, no. D1, pp. D1062–D1067, 2018.

[133] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[134] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers, "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2097–2106.

[135] Z. Nabulsi *et al.*, "Deep learning for distinguishing normal versus abnormal chest radiographs and generalization to two unseen diseases tuberculosis and covid-19," *Scientific reports*, vol. 11, no. 1, pp. 1–15, 2021.

[136] P. Rajpurkar *et al.*, "Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning," *arXiv preprint arXiv:1711.05225*, 2017.

[137] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017, pp. 4700–4708.

[138] W. T. Clark and P. Radivojac, "Information-theoretic evaluation of predicted ontological annotations," *Bioinformatics*, vol. 29, no. 13, pp. i53–i61, 2013.

[139] M. Kulmanov and R. Hoehndorf, "Deepgoplus: Improved protein function prediction from sequence," *Bioinformatics*, vol. 36, no. 2, pp. 422–429, 2020.

[140] M. Carriere, B. Michel, and S. Oudot, "Statistical analysis and parameter selection for mapper," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 478–516, 2018.

[141] W. T. Tutte, "How to draw a graph," *Proceedings of the London Mathematical Society*, vol. 3, no. 1, pp. 743–767, 1963.

[142] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html.

[143] E. Becht *et al.*, "Dimensionality reduction for visualizing single-cell data using UMAP," *Nature Biotechnology*, vol. 37, no. 1, pp. 38–44, Dec. 2018. DOI: 10.1038/nbt.4314. [Online]. Available: https://doi.org/10.1038/nbt.4314.

[144] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv*, vol. stat.ML, p. 1802.03426, 2018. DOI: 10.48550/ARXIV.1802.03426.

[145] K. Fountoulakis, F. Roosta-Khorasani, J. Shun, X. Cheng, and M. W. Mahoney, "Variational perspective on local graph clustering," *Mathematical Programming*, Dec. 2017, ISSN: 1436-4646. DOI: 10.1007/s10107-017-1214-8.

[146] J. Shun, F. Roosta-Khorasani, K. Fountoulakis, and M. W. Mahoney, "Parallel local graph clustering," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 1041–1052, 2016.

[147] N. Veldt, A. R. Benson, and J. Kleinberg, "Hypergraph cuts with general splitting functions," *SIAM Review*, vol. 64, no. 3, pp. 650–685, 2022.