ABSTRACT

Title of Dissertation: Structured discovery in graphs:
Recommender systems and temporal graph analysis

Sheyda Peyman
Doctor of Philosophy, 2024

Dissertation Directed by: Professor Vince Lyzinski
Department of Mathematics

Graph-valued data arises in numerous diverse scientific fields ranging from sociology, epidemiology and genomics to neuroscience and economics. For example, sociologists have used graphs to examine the roles of user attributes (gender, class, year) at American colleges and universities through the study of Facebook friendship networks and have studied segregation and homophily in social networks; epidemiologists have recently modeled Human-nCov protein-protein interactions via graphs, and neuroscientists have used graphs to model neuronal connectomes.

The structure of graphs, including latent features, relationships between the vertex and importance of each vertex are all highly important graph properties that are main aspects of graph analysis/inference. While it is common to imbue nodes and/or edges with implicitly observed numeric or qualitative features, in this work we will consider latent network features that must be estimated from the network topology. The main focus of this text is to find ways of extracting the latent structure in the presence of network anomalies. These anomalies occur in different

scenarios: including cases when the graph is subject to an adversarial attack and the anomaly is inhibiting inference, and in the scenario when detecting the anomaly is the key inference task. The former case is explored in the context of vertex nomination information retrieval, where we consider both analytic methods for countering the adversarial noise and also the addition of a user-in-the-loop in the retrieval algorithm to counter potential adversarial noise. In the latter case we use graph embedding methods to discover sequential anomalies in network time series.

Structured discovery in graphs: Recommender systems and temporal graph analysis

by

Sheyda Peyman

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Advisory Committee:
Professor Vince Lyzinski, Chair/Advisor
Professor Lizhen Lin
Professor Wojciech Czaja
Professor Tracy Sweet
Professor Radu Balan

Dedication

I dedicate this dissertation to our one year old son, Matisse Olinga in the hope that one day he will enjoy the elegant truths of mathematical proofs the same way that his mother did. :)

Acknowledgments

It is said that most important thing in the journey of pursuing a PhD is your advisor. I was the luckiest person, as Dr. Vince Lyzinski was the best advisor one could wish for. His incredible knowledge, creativity of thought, constant support, patience, sense of humor and love for coffee made him the perfect advisor. Thank you Dr. Vince Lyzinski for making my PhD experience a positively memorable one and one that I already miss!

I would also like to thank my dissertation committee, namely Professors Lizhen Lin, Wojciech Czaja, Tracy Sweet and Professor Radu Balan for agreeing to serve on the committee and for their suggestions of research questions to further explore.

I've had the pleasure of working with collaborators Professors William Frost (Rosalind Franklin University of Medicine and Science) and Jeffrey Brown (Penn State) on our work related to network time series (See Chapter 4) and also worked closely with Hayden Helm (Microsoft) and Ben Johnson (Jataware), all of whom I'd like to acknowledge here.

Furthermore, Professor Doron Levy and Professor Konstantina Trivisa have both played a huge part in supporting and encouraging me throughout my PhD. They were both present in my academic journey from the first day and I consider them huge mentors in my career so far, so thank you both!

And finally, none of my work would have been possible without the constant support, encouragement and patience of my loving husband, Shamim, and our sweet little 1 year old

boy, Matisse Olinga. Their support, together with that of my precious brother, parents and grandparents has been the fuel that has driven me throughout this journey! The friendship and prayers that have surrounded me through the Baha'i Community of Washington D.C. are also something I cherish deeply!

Table of Contents

List of Figures

List of Abbreviations

ALP	"approximate" Integer Linear Program Vertex Nomination framework
ASE	Adjacency Spectral Embedding
DCRDPG	Degree correlated Random Dot Product Graph
ER	Erdős–Rényi
GRDPG	Generalized Random Dot Product Graph
ILP	Integer Linear Program Vertex Nomination framework
LAP	Linear Assignment Problem
MMSBM	Mixed Membership Stochastic Block Model
MQ-ILP	Multiple Query Integer Linear Program Vertex Nomination framework
RDPG	Random Dot Product Graph
SBM	Stochastic Block Model
SVC	Support Vector Classifier (alternate name for an Support Vector Machine)
SVM	Support Vector Machine
VOI	Vertex of Interest
VN	Vertex Nomination

Chapter 1: Introduction and Background

Graph-valued data arises in numerous diverse scientific fields ranging from sociology, epidemiology and genomics to neuroscience and economics. For example, sociologists have used graphs to examine the roles of user attributes (gender, class, year) at American colleges and universities through the study of Facebook friendship networks [5] and have studied segregation and homophily in social networks [6]; epidemiologists have recently used networks to model the spread of Covid-19 in communities under different intervention strategies [1] (see Figure 1.1); and neuroscientists have used graphs to model neuronal connectomes [7].

In this work, a graph is an ordered pair $G = (V, E)$ where V is the set of vertices of the graph and E is the set of edges. A network (or a graph: for our purposes, these words are used interchangeably) can be thought of as a set of objects (V) and interactions between them (E). The objects can be anything from people, proteins, to neurons and the interactions can be anything from phone calls (emails sent) between individuals, to being friends on social media, to synapses between neurons or protein-protein interactions. A *directed* graph is one in which the interactions are only "one-way" (see 1.2 RHS for an example), so that $E \subset V \times V$. For example, the synapses between a set of neurons can be modeled by a directed graph, since each synapse starts at one neuron and ends at another. An *undirected* graph, on the other hand, is a graph in which the interactions are "two-way" (see 1.2 LHS for an example). In the case of undirected

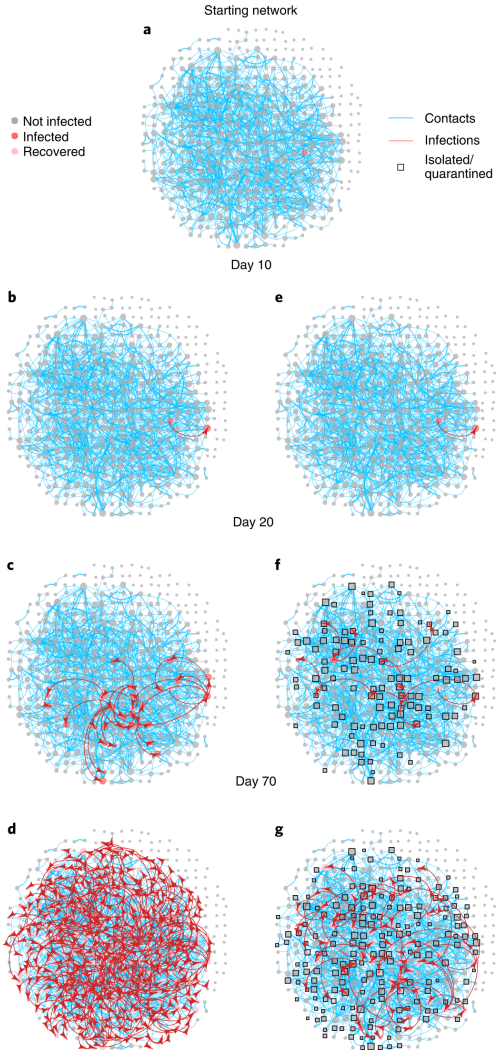

Figure 1.1: Figure from [1]. A social network with vertices representing residents of the town of Haslemere, UK. Here there are 468 individuals (gray nodes) with 1,257 social links (blue edges) weighted by 1,616 daily contacts (edge thickness) and a single starting infector (red). In these figures one can see the progression of the COVID-19 epidemic under the no-intervention (b–d) and secondary contact tracing (e–g) scenarios. Red arrows indicate an infection route, and squares highlight isolated or quarantined individuals (see [1] for more details).

graphs, $E \subset \binom{V}{2}$. Most graphs under consideration in this dissertation will be undirected graphs.

The structure of graphs, including latent features, relationships between the vertex and importance of each vertex are all highly important graph properties that are main aspects of graph analysis/inference [8, 9]. While it is common to imbue nodes and/or edges with implicitly observed numeric or qualitative features, in this work we will consider latent network features

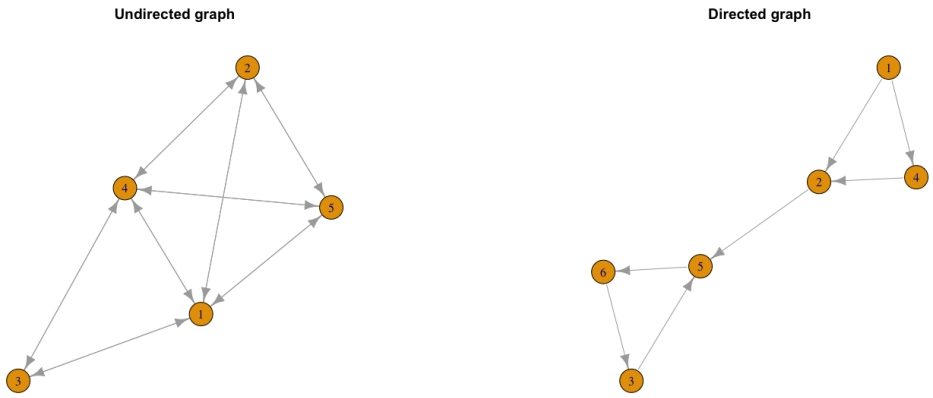

Figure 1.2: Examples of a directed graph (R) and undirected graph (L)

that must be estimated from the network topology. The main focus of this text is to find ways of extracting the latent structure in the presence of network anomalies. These anomalies occur in different scenarios: including cases when the graph is subject to an adversarial attack and the anomaly is inhibiting inference (Chapter 2-3), and in the scenario when detecting the anomaly is the key inference task (Chapter 4). The former case is explored in the context of vertex nomination information retrieval, where we consider both analytic methods for countering the adversarial noise (Chapter 2) and also the addition of a user-in-the-loop in the retrieval algorithm to counter potential adversarial noise (Chapter 3). In the latter case we use graph embedding methods to discover sequential anomalies in network time series. Before delving into the above-mentioned topics, we present a few definitions to set the foundation.

Notation: Here we will establish some of the commonly used notation in the work. For a nonnegative integer n, we define

$$[n] = \{1, 2, 3, \cdots, n\}.$$

For a matrix $A \in \mathbb{R}^{m \times n}$, we use A_{ij} to denote the i, j-th element of A and $A_{i,\cdot}$ (resp., $A_{\cdot,i}$) the i-th row (resp, column) of A. We define the usual Frobenius norm via

$$\|A\| = \sqrt{\sum_i \sum_j A_{ij}^2},$$

and the 2-to-infinty norm [10] via

$$\|A\|_{2 \mapsto \infty} = \|A\|_{2,\infty} = \text{sup}_{\|x\|_2=1} \|Ax\|_\infty = \max_{i \in [m]} \|A_{i,\cdot}\|_2.$$

For $f, g : \mathbb{Z} > 0 \mapsto \mathbb{R}$, we adopt the usual asymptotic notation

$$f = o(g) \text{ if } \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

$$f = O(g) \text{ if } \text{limsup}_{n \to \infty} \frac{f(n)}{g(n)} < \infty$$

$$f = \omega(g) \text{ if } \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

$$f = \Omega(g) \text{ if } \text{liminf}_{n \to \infty} \frac{f(n)}{g(n)} > 0$$

$$f = \Theta(g) \text{ if } f = O(g) \text{ and } f = \Omega(g).$$

1.0.1 Network representation matrices

A common and concise way of representing a graph is by its adjacency matrix, where the elements of the matrix indicate whether or not pairs of vertices in the corresponding graph are adjacent.

Definition 1 (Adjacency matrix)**.** Let $G = (V, E)$ be a graph, with vertices $V = [n] = \{1, ..., n\}$

and edges $E \subset \binom{[n]}{2}$. Its adjacency matrix A is an $n \times n$ binary, symmetric and hollow matrix (*hollow* here meaning that the graph does not contain any self-loops and hence the diagonal entries of A are zero). It can be described as follows:

$$A_{ij} = \begin{cases} 1 & \text{if there is an edge between vertices } i \text{ and } j \\ 0 & \text{if there is no edge between vertices } i \text{ and } j \end{cases}$$

This matrix can be considered one of the main building blocks of graph analysis. It is often used as a tool to embed the graph into lower dimensions and extract information including latent positions, for example. Moreover, the linear algebraic properties of A encode significant network structure in G [11]. For example, the exponential of the adjacency matrix provides valuable information about connectivity, as well as about the relative importance or centrality of nodes. Another useful application is computing the Perron vector of the adjacency matrix in order to rank the nodes of a network (see [12]).

The Laplacian matrix is another matrix that encodes important information with regard to a graph.

Definition 2 (The Laplacian matrix)**.** The (normalized) Laplacian matrix is defined as:

$$\mathcal{L}(A) = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$$

where I is the identity matrix and $D = diag(d_i)$ is the diagonal matrix with degrees $d_i = \sum_{j=1}^{n} A_{ij}$ on the diagonal. A few basic facts about the Laplacian matrix are mentioned below. For a more thorough exploration, see [13].

1. The spectrum of $\mathcal{L}(A)$ is a subset of $[0, 2]$.

2. The smallest eigenvalue is always zero (as $D - A$ has a 0 eigenvalue with eigenvector $\vec{1}$.)

The Laplacian matrix has been given a lot of attention in past times, given its application in many fields, such as randomized algorithms, combinatorial optimization problems and machine learning. More specific examples of the use of the Laplacian matrix include calculating the number of spanning trees for a given graph, approximating the sparsest cut of a graph, and it is also used in graph-based signal processing [14]. The spectral decomposition of the Laplacian matrix can be used to analyze the lower dimensional embeddings which can give us insights into graph properties, via clustering, for instance.

1.1 Random Graph Models

A host of random graph models have been proposed in the literature (see [8, 15]), and two of the more popular models in the statistical network inference community are the stochastic blockmodel and the random dot product graph model. The above work on random graph models allows us to situate our analysis of network data in the context of traditional statistical inference. They provide a setting in which one can better analyze graph properties, including visualizing graphs in latent spaces and/or clustering the nodes, node and edge property prediction or even subgraph property prediction, etc [16, 17] (see [18] for an extensive exploration of many common random graph models).

1.1.1 Erdős-Rényi model (ER)

Erdős was the first one to use random graphs, with the purpose of giving a probabilistic construction of a graph with large girth and chromatic number [19]. Only later did Erdős and Rényi begin studying random graphs as the main object of interest in their own right, publishing the well known paper [20]. This paper laid the foundation of random graph models and its ideas are still used all over the literature today. They propose one of the most simple and fundamental graph models, namely the Erdős-Rényi (ER) graph model [21–24].

Definition 3 (Erdős-Rényi model (ER))**.** Let $G = (V, E)$ be an n-vertex undirected graph-valued random variable. We say that G is distributed according to the Erdős-Rényi(n,p) (abbreviated ER(n,p)) random graph model if the following holds:

1. For any $i, j \in \binom{V}{2}$, we have $P(\{i, j\} \in E) = p$.
2. The events $\left\{\{i, j \in E\}\right\}_{i,j \in \binom{V}{2}}$ are mutually independent.

This model is not only interesting because of its mathematical properties (see below), but is also widely used in graph theory research, for example it is used as a benchmark when testing graph matching algorithmic performance and matching feasibility [25, 26]. The mathematical properties of the ER graph model explored in [20] are quite fascinating, and some highlights are summarized below.

Consider the set $E_{n,N}$ of all graphs $G = (V, E)$, where $V = v_1, ..., v_n$ are the vertices of the graph and N is the number of edges. Choosing a graph G randomly from the set $E_{n,N}$ then simply consists of choosing N out of the total number of edges $\binom{n}{2}$ that a graph with n vertices can have. This means that the number of elements in the set $E_{n,N}$ is $\binom{\binom{n}{2}}{N}$. One can think of the

number of edges N of a random graph $G_{n,N} \in E_{n,N}$ to be a time variable, where one can then analyze the *evolution of a random graph*, as it is called in [20]. In more mathematical terms, we can think of a random graph formation as a stochastic process, so that at time $t = 1$, we randomly choose one edge, e_1 (out of $\binom{n}{2}$ possible edges connecting the vertices $v_1, ..., v_n$), at time $t = 2$, we randomly choose another edge, $e_2 \neq e_1$ out of the now $\binom{n}{2} - 1$ possible edges, and so on. At time $t = k + 1$ we then randomly choose one of the $\binom{n}{2} - k$ possible edges (different from the edges $e_1, ..., e_k$ that were already chosen), where each of the remaining edges can again be chosen with equal probability, in this case $\frac{1}{\binom{n}{2}-k}$. For ease of notation below, we denote the graph described above by $\Gamma_{n,N}$, with vertices $v_1, ..., v_n$ and edges $e_1, ..., e_N$.

In [20], five phases are considered, when looking at the evolution of the random graph. In phase 1 (where N(n) = o(n)), $\Gamma_{n,N(n)}$ almost surely only contains components that are trees. In phase 2 (where $N(n) \sim cn$, where $0 < c < \frac{1}{2}$), $\Gamma_{n,N(n)}$ almost surely consists of components that are either trees or components that contain exactly one cycle (components in which the number of edges and vertices is equal). Phase 3 (where $N(n) \sim cn$) is the most interesting phase of them all, and maybe one of the most fascinating results discovered in [20], namely that the structure of $\Gamma_{n,N(n)}$ suddenly changes! When $N(n) \sim cn$, where $c < \frac{1}{2}$, then the greatest component of $\Gamma_{n,N(n)}$ is a tree with approximately $\frac{1}{\alpha}\Big(\log_n - \frac{5}{2}\log\log n\Big)$ vertices (with probability tending to 1 for $n \to +\infty$ and where $\alpha = 2c - 1 - \log(2c)$). When $N(n) \sim \frac{n}{2}$, then surprisingly, the greatest component has approximately $n^{2/3}$ vertices (with probability tending to 1 for $n \to +\infty$ and its structure is a lot more complicated and complex. Finally, in this phase, for $N(n) \sim cn$ with $c > \frac{1}{2}$, the greatest component of $\Gamma_{n,N(n)}$ has approximately $G(c)n$ vertices (with probability tending to 1 for $n \to +\infty$), where

$$G(c) = 1 - \frac{1}{2c} \sum_{k=1}^{+\infty} \frac{k^{k-1}}{k!} \left(2ce^{-2e}\right)^k \tag{1.1}$$

and where $G(1/2) = 0$ and $\lim_{c \to +\infty} G(c) = 1$. Here, there is the above "giant" component, and the rest of the components are comparatively small trees, the total number of vertices of which (the latter components) are almost surely $n(1 - G(c)) + o(n)$ for $c \geq \frac{1}{2}$. In phase 4 (where $N(n) \sim cn \log_n$, $c \leq \frac{1}{2}$), the graph becomes connected (almost surely). Finally, in phase 5 (where $N(n) \sim (n \log_n) w(n)$, when $w(n) \to +\infty$), not only the same as in phase 4 holds, but the orders of the vertices are almost surely asymptotically equal. In [20] they call this an *asymptotically regular* graph.

For a simple example, consider the case when $G \sim ER(n, p)$, where $n = 20$ and $p = 0.3$. The corresponding adjacency matrix will then contain entries (above the diagonal) that are i.i.d Bernoulli(0.3) random variables (see Figure 1.3). Note that the structure of the connections in the graph are reflected in the density of the adjacency matrix, i.e. for a graph where the number of connections (edges between vertices) is high, the corresponding adjacency matrix contains a lot of 1s and only a few 0s, whereas in a setting with fewer edges in the graph, the adjacency matrix contains fewer 1s and more 0 values. In Figure 1.3 the graph is less dense, since the probability p of an edge existing between any two vertices is only 0.3. On the other hand, in Figure 1.4, the probability of edges is 0.7, which gives rise to a more dense graph and therefore a more dense adjacency matrix. As the block probability values are closer to 0.5, the graphs become more entropic and hence more complex.

In the ER model, all vertices are stochastically identical. Much of modern modeling of random graphs focuses on the consideration of networks with intrinsic latent structure among the

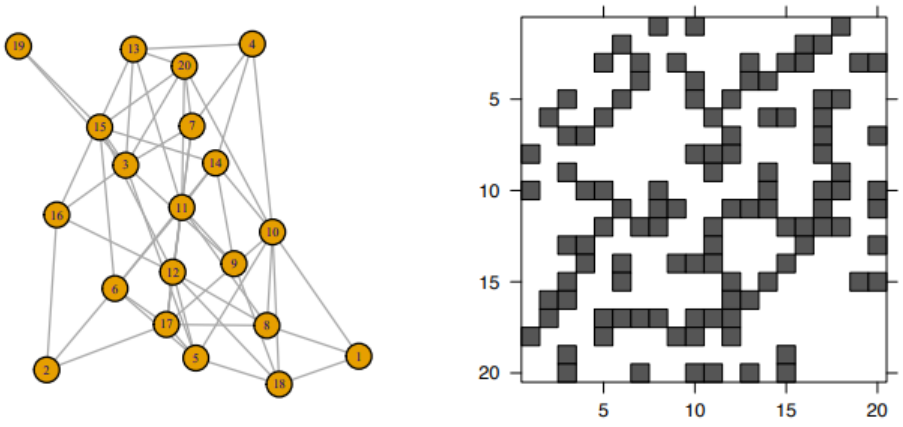

Figure 1.3: An Erdős-Rényi graph $G \sim ER(n, p)$, where $n = 20$ and $p = 0.3$. This graph is less dense than the one in Figure 1.4 and corresponds to a less dense adjacency matrix.

vertices. For example, community structures arise and are one of the main areas of research in network theory. Because of the "flatness" of the ER model (all edges have the same probability p), a model that lends itself well to studying communities of a graph and is easier to work with was developed, namely the Stochastic Block model [27–30].

1.1.2 Stochastic Block Model (SBM)

The Stochastic Block Model (SBM), introduced in [31], provides a simple model for networks with latent community structure, and the SBM and its variants (degree corrected SBM [32], mixed membership SBM [33], hierarchical SBM [34, 35], etc.) have been popular models for exploring inference tasks such as community detection/clustering [36–40] and community testing [41, 42]. Furthermore, while rather simple, SBMs can also be viewed as an analogue of

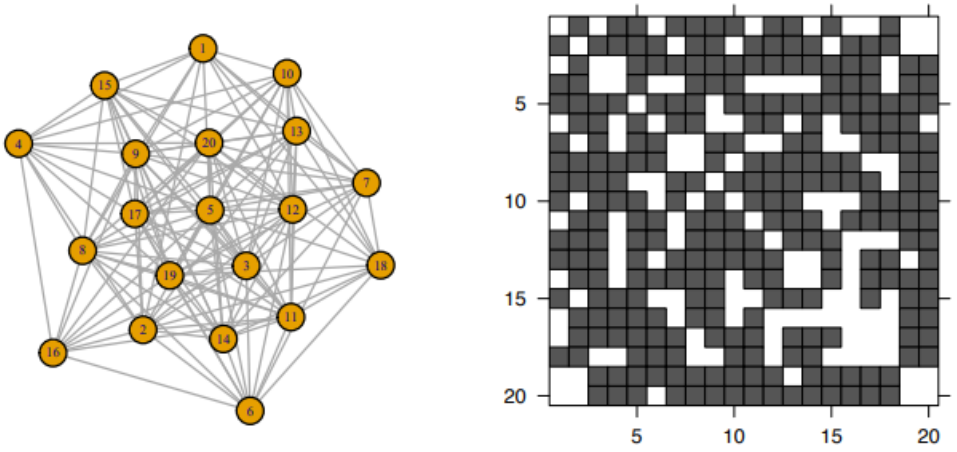

Figure 1.4: An Erdős-Rényi graph $G \sim ER(n, p)$, where $n = 20$ and $p = 0.7$. This graph is more dense than the one in Figure 1.3 and corresponds to a more dense adjacency matrix.

network histograms and thus provide a universal representation for unlabeled graphs [43]. The most simple definition of an SBM can be stated as follows:

Definition 4 (Stochastic Block Model (SBM) with sparsity parameter ν)**.** A random graph $G = (V, E)$ on n vertices is distributed according to a Stochastic Blockmodel with parameters K, **B**, π, and sparsity parameter ν (written $G \sim$SBM$(n, K, \mathbf{B}, \pi, \nu)$) if the following hold:

i. Each vertex $i \in \{1, ..., n\}$ is independently assigned to a community/block $\{1, 2, ..., K\}$ according to the probability vector $\pi \in \mathbb{R}^K$; we will denote the block membership of vertex $v \in V$ via b_v.

ii. $\mathbf{B} = [B_{ij}] \in [0, 1]^{K \times K}$, the block probability matrix, is a $K \times K$ symmetric matrix, whose entries provide (up to the sparsity factor ν) the probability of a vertex in one block communicating with a vertex in another block; $\nu \in [0, 1]$ is a sparsity factor controlling the graph density.

iii. Conditional on the block-membership for each vertex, the (undirected) edges of the graph

are independently sampled according to:

$$\text{If } u, v \in V, \text{ then } \mathbb{1}_{u \sim_G v} \sim \text{Bernoulli}(\nu B_{b_v, b_u}).$$

When working across pairs of SBMs, it is often convenient to work within the context of an SBM model that allows for structured dependence across the edges of multiple networks; towards this end, we next introduce the ρ-correlated Stochastic Blockmodel from [25].

Definition 5 (Sparse ρ-Correlated Stochastic Block Model (SBM))**.** A pair of graphs (G_1, G_2), is an instantiation of a correlated Stochastic Blockmodel with parameters K, **B**, π and ρ and with sparsity parameter ν (written $(G_1, G_2) \sim \text{SBM}(n, K, \mathbf{B}, \pi, \nu, \rho)$) if the following hold:

i. Marginally, $G_1 \sim SBM(n, K, \mathbf{B}, \pi, \nu)$ and $G_2 \sim SBM(n, K, \mathbf{B}, \pi, \nu)$; moreover, the block memberships are chosen so that the block membership function is identical across graphs.

ii. Conditional on the block-membership for each vertex in each graph, the collection of the following indicator random variables is mutually independent,

$$\left\{ \{\mathbb{1}_{u \sim_{G_1} v}\}_{\{u,v\} \in \binom{V}{2}} \cup \{\mathbb{1}_{u \sim_{G_2} v}\}_{\{u,v\} \in \binom{V}{2}} \right\}$$

except that for each $\{u, v\} \in \binom{V}{2}$, the correlation between $(\mathbb{1}_{u \sim_{G_1} v})$ and $(\mathbb{1}_{u \sim_{G_2} v})$ is ρ.

Remark 1.1.1. There is an alternate parameterization of the SBM that we will find convenient in experiments, namely the case when the block sizes and memberships are fixed. In this case (written $G \sim \text{SBM}(n, K, \mathbf{B}, \vec{n}, \nu)$), the block probability assignment vector π is replaced by

$\vec{n} \in \mathbb{Z}^K$ satisfying $n_i \geq 0$ for all $i \in [K]$ and $\sum_{i=1}^{K} n_i = n$. Here, vertices are preassigned into the K blocks (so that $|\{v : b_v = i\}| = n_i$) and edges are conditionally independent given these assignments. The remainder of the definition is essentially unchanged.

In Figure 1.5, we visualize a simple example of a graph $G_1 \sim SBM$ with 3 blocks (each of size 100) and $n = 300$ vertices, with the following probability matrix:

$$\mathbf{B}_1 = \begin{pmatrix} 0.6 & 0.02 & 0.02 \\ 0.02 & 0.6 & 0.02 \\ 0.02 & 0.02 & 0.6 \end{pmatrix}$$

The graph is visualized on the left panel of the figure, and its corresponding adjacency matrix in the right panel. Note the clear community structure in A here, as compared to the flat ER examples considered earlier. Note that the diagonal entries of the probability matrix are higher probability values than the rest of the entries, and the corresponding adjacency matrix clearly represents this fact.

Now consider $G_2 \sim SBM$ with 3 blocks and $n = 300$ vertices, with the following probability matrix:

$$\mathbf{B}_2 = \begin{pmatrix} 0.02 & 0.6 & 0.6 \\ 0.6 & 0.02 & 0.6 \\ 0.6 & 0.6 & 0.02 \end{pmatrix}$$

In this case, the diagonal entries of the probability matrix are much smaller values than the rest of the matrix, and again, it is easy to see this pattern in the corresponding adjacency matrix.

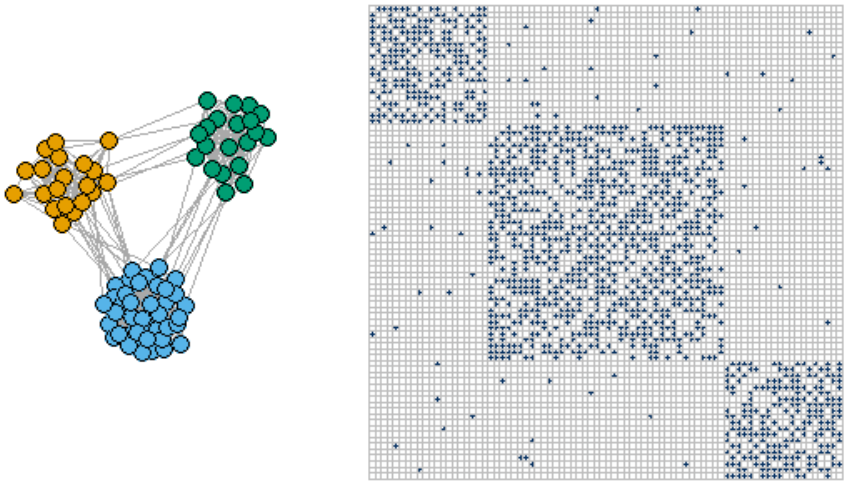

Figure 1.5: Example SBM graph and adjacency matrix based on the probability matrix $\mathbf{B}_1$, where the probability of two vertices being connected through an edge is much higher for vertices in the same block (community). This can be clearly seen in the adjacency matrix, where the diagonal entries are more dense, and the off-diagonal entries very sparse.

Remark 1.1.2. To allow for degree variability inside the classical SBM, the degree corrected SBM of [32] endows each vertex with an additional parameter c_v. The definition is then identical to the classical SBM except that conditional on the block-membership for each vertex, the (undirected) edges of the graph are independently sampled according to: If $u, v \in V$, then

$$\mathbb{1}_{u \sim_G v} \overset{ind.}{\sim} \text{Bernoulli}(\nu c_v c_u B_{b_v, b_u}).$$

Oftentimes, the SBM may not capture certain details of connections between the communities in the graph or the complex nature of community overlap. In these cases, a more appropriate

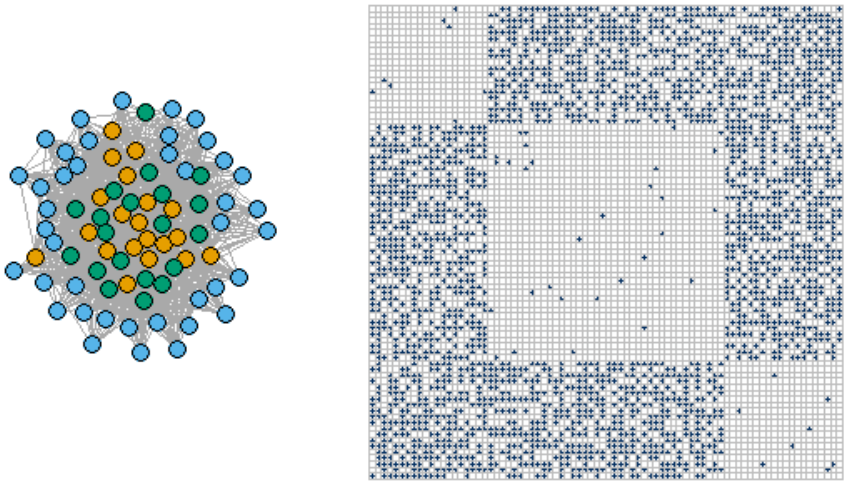

Figure 1.6: Example SBM graph and adjacency matrix based on the probability matrix $\mathbf{B}_2$, where the probability of two vertices being connected through an edge is much higher for vertices in different blocks (communities). This can be clearly seen in the adjacency matrix, where the diagonal entries are a lot more sparse and the off-diagonal entries very dense.

model to use may be the Mixed Membership Stochastic Block Model (MMSBM) [33]. This model is an extension of the simple SBM, where each vertex has a distribution of group memberships (rather than being assigned to a particular community, like in the SBM model). Examples of MMSBMs being used range from summarizing and de-noising complex connectivity patterns in a network of interactions amongst proteins in yeast [44] to analyzing the brain patterns of patients with Alzheimer's disease [45]. A formal definition of the MMSBM is given below.

Definition 6 (Mixed Membership Stochastic Block Model (MMSBM) with sparsity parameter ν)**.** A random graph $G = (V, E)$ on n vertices is distributed according to a Mixed Membership Stochastic Blockmodel (MMSBM) with parameters K, $\mathbf{B}$, $\{\pi_v\}_{v \in V}$, and sparsity parameter ν

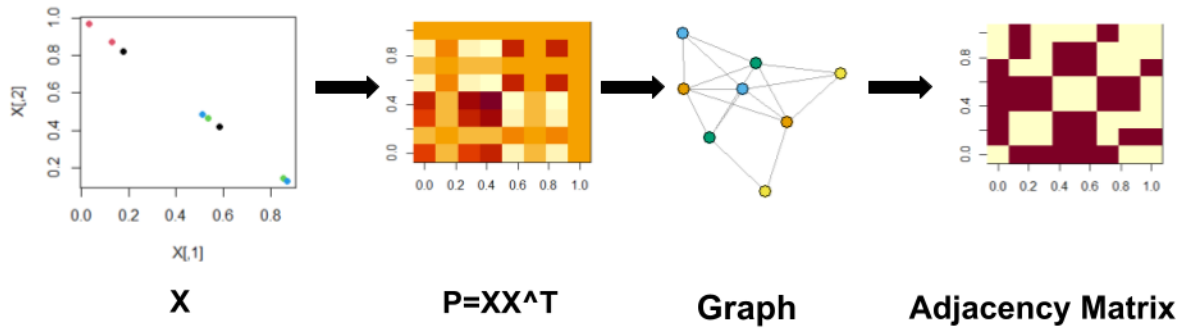

Figure 1.7: Example of a graph based on an RDPG model, where the latent structure is represented by the latent positions X, which can be approximately retrieved by embedding the adjacency matrix of the graph.

(written $G \sim$SBM$(n, K, \mathbf{B}, \{\pi_v\}_{v \in V}, \nu)$) if the following hold:

i. Each vertex v is (either randomly or deterministically) assigned a membership vector π_v in the $K-1$ simplex.

ii. $\mathbf{B} = [B_{ij}] \in [0,1]^{K \times K}$, the block probability matrix, is a $K \times K$ symmetric matrix, whose entries provide (up to the sparsity factor ν) the probability of a vertex in one block communicating with a vertex in another block; $\nu \in [0,1]$ is a sparsity factor controlling the graph density.

iii. Conditional on the block-membership vectors for each vertex, the (undirected) edges of the

graph are independently sampled according to:

$$\text{If } u, v \in V, \text{ then } \mathbb{1}_{u \sim_G v} \sim \text{Bernoulli}(\nu \pi_v^T \mathbf{B} \pi_u).$$

Another extension of the SBM model is the so-called Hierarchical Stochastic Block Model (HSBM). In this setting, not only does the whole graph have an SBM structure, but the graph contains R subgraphs, which all also have SBM structure within themselves. The connections between each two subgraphs are hence comparatively sparse, whereas the connections within the subgraphs are comparatively dense. Note that a HSBM can also be thought of as one SBM graph with $K \times R$ blocks. For different formal definitions of the HSBM, see for example [30, 35, 46].

The above SBM models and their extensions are very specific graph models used in cases where the graph contains a certain 'block structure'. This is not always the case, and therefore a more general model—which incidentally encompasses all the models above—may be more useful for cases where the graph contains latent structure that is not block-like. This is provided by the Generalized Random Dot Product Graph (GRDPG).

1.1.3 Generalized Random Dot Product Graph (GRDPG)

Another popular network model in the statistical network inference literature is the Generalized Random Dot Product Graph (GRDPG), which posits that the edge connectivity is a function of latent vertex attributes that are (potentially) more general than simple community membership [9, 47–49]. In the GRDPG setting, inference often begins with estimation of the latent positions [9], as these estimates often provide low-dimensional Euclidean representations for the graph at the vertex level. Given sufficient control over the estimation error of the latent positions [10],

various inference tasks can be profitably pursued in the embedding space, including clustering [36, 37, 50], classification [51, 52], and testing [53–56], among others.

Definition 7 (Generalized Random Dot Product Graph (GRDPG) with sparsity parameter ν)**.** Let $d \geq 1$ be given and let $\mathcal{X}$ be a subset of $\mathbb{R}^d$ such that $x^\top I_{p,q} y \in [0,1]$. Here $I_{p,q}$ is a $d \times d$ diagonal matrix with diagonal entries containing p "+1's" and q "-1's" for integers $p \geq 1, q \geq 0$, $p+q=d$. Let F be a distribution supported on $\mathcal{X} \subset \mathbb{R}^d$. A random n-vertex graph $G=(V,E)$ is distributed according to a Generalized Random Dot Product Graph with parameters $\mathbf{X}$ and F and sparsity parameter ν (written $G \sim \mathrm{GRDPG}(\mathbf{X}, F, \nu)$) if the following hold:

i. $\mathbf{X}$ is a $n \times d$ matrix whose rows $X_1, \cdots, X_n$ are i.i.d. random vectors sampled from $\mathcal{X}$ according to F.

ii. Conditional on $\mathbf{X}$, the (undirected) edges of the graph are independently Bernoulli random variables with $\mathbb{P}(u \sim_G v) = \nu X_u^\top I_{p,q} X_v$. Written compactly (where $\mathbf{A}$ is the adjacency matrix of G),

$$\mathbb{P}(A|\mathbf{X}) = \prod_{\{i,j\} \in \binom{V}{2}} (\nu X_i^\top I_{p,q} X_j)^{A_{i,j}} (1 - \nu X_i^\top I_{p,q} X_j)^{1-A_{i,j}}. \tag{1.2}$$

Similar to the correlated SBM setting, the following model from [57, 58] allows us to work with pairs of correlated GRDPGs.

Definition 8 (Sparse ρ-Correlated GRDPG)**.** A pair of graphs (G_1, G_2), is an instantiation of a correlated GRDPG with parameters F, $\mathbf{X}$, and ρ and with sparsity parameter ν (written $(G_1, G_2) \sim \mathrm{GRDPG}(\mathbf{X}, F, \nu, \rho)$) if the following hold:

i. $\mathbf{X}$ is a $n \times d$ matrix whose rows $X_1, \cdots, X_n$ are i.i.d. random vectors sampled from $\mathcal{X}$ according to F.

ii. Marginally, $G_1 \sim \text{GRDPG}(\mathbf{X}, F, \nu)$ and $G_2 \sim \text{GRDPG}(\mathbf{X}, F, \nu)$.

iii. Conditional on $\mathbf{X}$, the collection of the following indicator random variables is mutually independent,

$$\left\{ \{\mathbb{1}_{u \sim_{G_1} v}\}_{\{u,v\} \in \binom{V}{2}} \cup \{\mathbb{1}_{u \sim_{G_2} v}\}_{\{u,v\} \in \binom{V}{2}} \right\}$$

except that for each $\{u, v\} \in \binom{V}{2}$, the correlation between $(\mathbb{1}_{u \sim_{G_1} v})$ and $(\mathbb{1}_{u \sim_{G_2} v})$ is ρ.

We note here that the GRDPG model encompasses the SBM model and its popular variants (degree-corrected, hierarchical, etc.) as well as any (conditionally) edge independent random graph for which the edge probabilities matrix is low rank. Furthermore, any latent position graph [59] on n vertices can be approximated by a GRDPG with latent positions $\mathbf{X} \in \mathbb{R}^{n \times d}$ for some sufficiently large d.

Remark 1.1.3. In the context of the GRDPG latent position random graph models, two different sources of nonidentifiability arise naturally: subspace nonidentifiability and model-based nonidentifiability [60]. Recall that the edge probability matrix for a GRDPG is given by $\mathbf{P} = \nu \mathbf{X} I_{p,q} \mathbf{X}^T$ for some $n \times d$ matrix $\mathbf{X}$. *Subspace nonidentifiability* arises in the context of the non-uniqueness of the eigenbasis of the subspaces corresponding to repeated eigenvalues; i.e., we cannot hope to exactly recover the columns of $\mathbf{X}$ (i.e., the scaled eigenvectors of $\mathbf{P}$) corresponding to repeated eigenvalues of $\mathbf{P}$. More pressing here is the issue of *model nonidentifiability*; specifically, transformations to the inputs $\mathbf{X}$ under which $\mathbf{P}$ is invariant. More specifically, for any indefinite orthogonal matrix $\mathbf{W}_{p,q}$ (so that $\mathbf{W}_{p,q} I_{p,q} \mathbf{W}_{p,q}^T = I_{p,q}$), we have $\mathbf{P} = \mathbf{X} I_{p,q} \mathbf{X}^T = \mathbf{X} \mathbf{W}_{p,q} I_{p,q} \mathbf{W}_{p,q}^T \mathbf{X}^T$;

hence $G_1 \sim GRDPG(\mathbf{X})$, and $G_2 \sim GRDPG(\mathbf{XW})$ yield $G_1 \stackrel{\mathcal{L}}{=} G_2$.

1.2 Adjacency Spectral Embedding

In the setting of GRDPGs and more general latent position random graphs, spectral embedding-based methods have proven effective at estimating the latent vertex features $\{X_i\}$. Two popular spectral embedding techniques are the Laplacian Spectral Embedding (LSE) [36, 61] and the Adjacency Spectral Embedding (ASE) [37]. In our text, we will focus our attention on the ASE.

Definition 9 (Adjacency Spectral Embedding (ASE))**.** Given the adjacency matrix **A** of an undirected graph, the d-dimensional adjacency spectral embedding of A is defined as follows:

$$\text{ASE}(\mathbf{A}, d) := \hat{\mathbf{X}}_d = \mathbf{U}\Sigma^{1/2} \in \mathbb{R}^{n \times d}, \tag{1.3}$$

where in the above expression,

i. The singular value decomposition (SVD) of $|\mathbf{A}|$ is given via

$$|\mathbf{A}| = (\mathbf{A}^T\mathbf{A})^{1/2} = [\mathbf{U}|\mathbf{U}^\perp][\Sigma \oplus \tilde{\Sigma}][\mathbf{U}|\mathbf{U}^\perp]^T;$$

ii. $\Sigma \in \mathbb{R}^{d \times d}$ is the diagonal matrix whose diagonal entries correspond to the d largest singular values of $|\mathbf{A}|$.

iii. $\mathbf{U} \in \mathbb{R}^{n \times d}$ is the $n \times d$ matrix whose orthonormal columns correspond to the singular vectors of $|\mathbf{A}|$ associated with the singular values in Σ.

ASE provides a theoretically tractable embedding of G, with consistency [37] and central limit

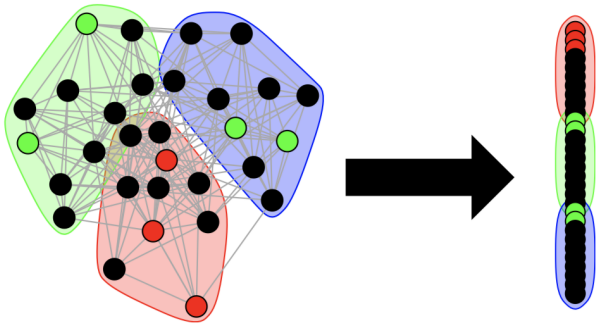

Figure 1.8: Figure originally from [2]: A visual representation of the VN framework with **multiple** vertices of interest in a single graph. Here, given vertices of interest S (here, the vertices denoted in red) in a graph G and vertices that are not of interest (in green), the goal is to produce rank lists of the vertices of G graphs, where the initially unknown vertices of interest (those in the red set colored black) rank at the top of that rank list.

theorems [9, 48] both having been proven for the ASE estimating the underlying latent positions $\mathbf{X}$ in the GRDPG setting. Note that one key model parameter that must be estimated when practically computing the ASE is the embedding dimension d. Herein, we will estimate d by choosing an elbow in the scree plot of the singular values of $\mathbf{A}$ [62], a (principled) heuristic outlined in [9], and then take p and q to be the number of positive and negative eigenvalues of $\mathbf{A}$ corresponding to these leading singular values.

Remark 1.2.1. As the ASE can only recover latent positions up to an indefinite orthogonal transform, methods that are invariant to such transforms (e.g., spectral clustering) are unaffected by the nonidentifiability here. Importantly, inter-point distances are *essentially* preserved; see [60] for detail.

1.3 Vertex Nomination (VN)

A setting that provides a useful set of tools to analyze complex networks and extract relevant information is Vertex Nomination (VN). It can be thought of as a semi-supervised information retrieval framework for network data, in which a vertex (or vertices) of interest are used to discover additional vertices of interest (either in the same graph, or in a different graph under consideration) [57, 58, 63, 64]. The output of a VN scheme, like in other information retrieval tasks, is a ranked list of the unknown vertices, where the identified vertices of interest ideally rank on top of the list. In our text, we look at two different settings in which VN schemes are useful to extract graph information. The first one is a setting in which there are two graphs under consideration, one of which is subject to an adversarial attack. Here, Definition 11 is fitting. The other setting is one in which we are given a query (or a vertex of interest) and are asked to produce a ranked list of further interesting vertices (in the same graph). Definition 10 is more appropriate in this case. To gain some intuition, below we present two informal definitions of Vertex Nomination: for the single graph and multiple graph settings (see Figures 1.8 and 1.9 for visual representations of this). In Section 1.3.1, we then provide the details of the VN problem formulation.

Definition 10 ((Informal) Vertex Nomination (Single graph))**.** Consider a graph $G = (V, E)$ and a vertex of interest v^* (or a set of vertices of interest V^*) . The goal in Vertex Nomination is to return a ranked list of the rest of the vertices, where ideally, the remaining vertices of interest rank on top of the list (see Figure 1.8; note this figure originally appeared in [2]).

Definition 11 ((Informal) Vertex Nomination (Two graphs))**.** Given vertices of interest S_1 in a

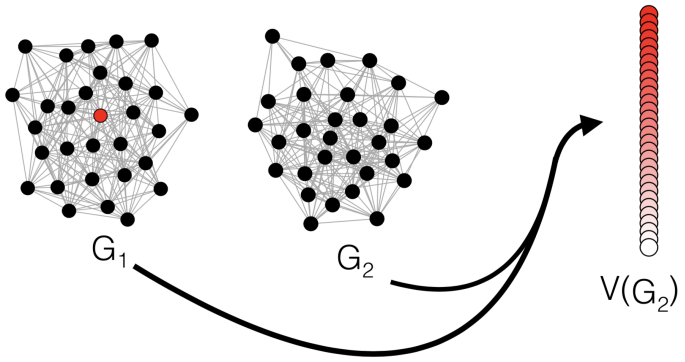

Figure 1.9: Figure originally from [2]: A visual representation of the generalized Vertex Nomination framework with **one** vertex of interest. Here, given a vertex of interest v^* in G_1 (denoted in red), and a second graph G_2, the goal is to produce a rank list of the vertices of G_2 with the unknown vertex (or vertices) of interest in G_2 ideally ranking at the top of the rank list.

graph G_1 and a second graph G_2 with a portion of its vertices being also of interest, $S_2 \subset V(G_2)$, produce a rank list of the vertices of G_2 with the unknown vertices in S_2 ideally concentrating at the top of the rank list (see Figure 1.9; note this figure originally appeared in [2]).

What defines vertices as "interesting" is vague above, and this is intentional. Indeed, in different settings, what makes a vertex "interesting" may vary; examples in the literature range from membership in a community of interest [65], vertices involved in illicit activity [65], or vertices corresponding to a particular user in a social network [57].

Early work in VN considered community membership as the trait defining interesting vertices as interesting [66–68], with notable applications including nominating fraudulent activity at Enron [66], recommending items in an entity transition graph using data from Bing [69] and helping identify websites involved in human trafficking [65]. Rich theory establishing the notions of consistency and Bayes optimality were derived in this community-focused setting [65, 70, 71]. More recent work defines the problem of nominating across networks, with interestingness defined more broadly—for example a particular user or collection of users across social networks

[57] or vertices corresponding to a particular neuron/region in a connectome [63] might be the vertices of interest. Theory was developed in [2, 3], where the notion of consistent vertex nomination schemes is defined and developed, with the role of features being further explored in [63]. In addition to problem formulation and theory, a significant number of methods have been developed to practically tackle the vertex nomination problem including those based on spectral decomposition [58,65,71], likelihood maximization [65], localized graph matching, [57] Bayesian MCMC [72], and specialized ILP formulations [69], to name a few.

1.3.1 Vertex Nomination and consistency (theoretical details)

Below, we provide a rigorous definition of the Vertex Nomination problem. Before doing so, we need to define the framework on which the VN problem is built. First, we define what a *nominatable distribution* is.

Definition 12 (Nominatable distribution [3])**.** Let $n, m \in \mathbb{Z} > 0$. The set $\mathcal{N}_{n,m}$ of *Nominatable Distributions of order (n,m)* is the set of all distribution families $\mathbf{F}_{\Theta}^{(n,m)}$ of the form

$$\left\{ F_{c,\theta}^{(n,m)} \text{ s.t. } 0 \leq c \leq \min(n,m) \in \mathbb{Z}, \theta \in \Theta \subset \mathbb{R}^{d(n,m)} \right\}$$

where $F_{c,\theta}^{(n,m)}$ is supported on $\mathcal{G} \times \mathcal{G}$, parameterized by $\theta \in \Theta$, and satisfies the following:

1. The vertex sets $V_1 = \{v_i\}_{i=1}^n$ and $V_2 = \{u_j\}_{j=1}^m$ satisfy $v_i = u_i$ for all $i \in [c]$; these vertices (naturally paired across graphs) will be called core vertices and denoted via C;

2. The "junk" vertices (those without correspondence across graphs), namely $J_1 = V_1 \backslash C$ and $J_2 = V_2 \backslash C$, satisfy $J_1 \cap J_2 = \emptyset$;

3. The induced subgraphs satisfy $G_1[J_1] \perp G_2[J_2] \,|\, \theta$

In order to model the situation in which the vertex labels across the graphs G_1 and G_2 are hidden, we consider passing the vertex labels of G_2 through what is known as an *obfuscation function* [2, 3].

Definition 13 (Obfuscating Function [3])**.** Let $(G_1, G_2) \sim F_{c,\theta}^{(n,m)} \in \mathcal{N}_{n,m}$. Consider a set W such that $W \cap V_i = \emptyset$ for $i = 1, 2$ with $|W| = |V_2|$. An obfuscating function is a bijection $\mathfrak{o} : V_2 \to W$. The set W is an obfuscating set, and $\mathfrak{D}_W$ the set of all obfuscating functions from V_2 to W.

We now have the tools to define the vertex nomination scheme. Below is the definition for a single vertex of interest, adapted here from [2, 3].

Definition 14 (VN Scheme for single VOI [2, 3])**.** We define the *vertex nomination scheme* as a function $\Phi : \mathcal{G}_n \times \mathfrak{o}(\mathcal{G}_m) \times V_1 \to \mathcal{T}_W$ (where for a set A, $\mathcal{T}_A$ denotes the set of all total orderings of the elements of A and where $n, m \in \mathbb{Z} > 0$) such that for any $g_1 \in \mathcal{G}_n, g_2 \in \mathcal{G}_m, v^* \subset V_1$ (where $v^* \in V_1$ is the vertex of interest in G_1), obfuscating functions $\mathfrak{o}_1, \mathfrak{o}_2 \in \mathfrak{D}_W$ (where W be an obfuscating set) and any $u \in V(g_2)$, the following holds for all $k \in [m]$:

$$\mathfrak{r}_\Phi(g_1, g_2, \mathfrak{o}_1, v^*, \mathcal{I}(u; g_2)) = \mathfrak{r}_\Phi(g_1, g_2, \mathfrak{o}_2, v^*, \mathcal{I}(u; g_2)) \iff$$

$$\mathfrak{o}_2 \circ \mathfrak{o}_1^{-1}(\mathcal{I}(\Phi(g_1, \mathfrak{o}_1(g_2)), V^*))[k]); \mathfrak{o}_1(g_2)) = \mathcal{I}(\Phi(g_1, \mathfrak{o}_2(g_2), v^*)[k]; \mathfrak{o}_2(g_2))$$

where $\Phi(g_1, \mathfrak{o}(g_2), v^*)[k]$ denotes the k-th element in the ordering $\Phi(g_1, \mathfrak{o}(g_2), v^*)$ and $\mathcal{I}(u; g)$ is defined as $\mathcal{I}(u; g) = \left\{ w \in V(g) \text{ s.t. } \exists \text{ an automorphism } \sigma \text{ of g, s.t. } \sigma(u) = w \right\}$ where $g \in \mathcal{G}_m$, $u \in V(g)$. The set of all such VN schemes will be noted as $\mathcal{V}_{nm}$.

Note that this extends to multiple vertices in the query set (i.e., multiple v^*) in a straightforward manner; see [3] for detail.

For a VN scheme Φ and obfuscation function $\mathfrak{o}$, we define $\text{rank}_{\Phi(g_1,\mathfrak{o}(g_2),v^*)}(\mathfrak{o}(u))$, for each $u \in V_2$, to be the position of $\mathfrak{o}(u)$ in the total ordering of $\Phi(g_1, \mathfrak{o}(g_2), v^*)$ and we let $\mathfrak{r}_\Phi : \mathcal{G}_n \times \mathcal{G}_m \times \mathfrak{D}_W \times V_1 \times 2^{V_2} \to 2^{[m]}$ be defined via

$$\mathfrak{r}_\Phi(g_1, g_2, \mathfrak{o}, v^*, S) = \{\text{rank}_{\Phi(g_1,\mathfrak{o}(g_2),v^*)}(\mathfrak{o}(u)) \text{ s.t. } u \in S\}. \tag{1.4}$$

With notation as above, for (g_1, g_2) sampled from $(G_1, G_2) \sim F_{c,n,m,\theta}$ with vertex of interest $v^* \in C$, and $k \in [m-1]$, we define the level-k nomination loss via

$$\ell_k(\Phi, g_1, g_2, v^*) = \mathbb{1}\left\{\text{rank}_{\Phi(g1,\mathfrak{o}(g_2),v^*)}\left(\mathfrak{o}(v^*)\right) \geq k+1\right\} = 1 - \mathbb{1}\left\{\text{rank}_{\Phi(g1,\mathfrak{o}(g_2),v^*)}\left(\mathfrak{o}(v^*)\right) \leq k\right\}$$

The level k error of Φ at v^* is then defined to be

$$L_k(\Phi, v^*) = \mathbb{E}_{(G_1,G_2)\sim F_{c,n,m,\theta}}\left[\ell_k(\Phi, G_1, G_2, v^*)\right]$$

To define the notion of VN consistency, we consider sequences $\mathbf{F} = \left(F^{(n,m_n)}_{c_n,\theta_n}\right)_{n=n_0}^{\infty}$ be a sequence of nominatable distributions with nested cores (see [2]). In this setting, the Bayes optimal VN scheme sequence $(\Phi^*_{F^{(n,m_n)}_{c_n,\theta_n}})$ is defined in [2] which allows for the following definition of VN consistency.

Definition 15 (Level k_n Consistent VN Rules in the single v.o.i. setting [2, 3])**.** With notation as above, for a given non-decreasing sequence (k_n) of integers, the VN rule $\mathbf{\Phi} = \left(\Phi_{n,m_n}\right)_{n=n_0}^{n=\infty}$ is

level-(k_n) consistent for vertex of interest v^* with respect to $\mathbf{F}$ if

$$\lim_{n\to\infty} L_{k_n}(\Phi_{n,m_n}, v^*) - L_{k_n}(\Phi^*_{F^{(n,m_n)}_{c_n,\theta_n}}, v^*) = 0$$

.

Notable in the VN setting, *universally level-*(k_n) *consistent* vertex nomination schemes do not exist (i.e., schemes that are level-(k_n) consistent for all nested-core nominatable sequences $\mathbf{F}$) [2]. This is a sharp and intriguing contrast to the related setting of classification, where universally consistent classifiers are well known to exist [73]. This is formalized in the following theorem from [2].

Theorem 1.3.1 (Corollary 28 of [2])**.** Let $\epsilon \in (0, 1)$ be arbitrary, and consider a VN rule $\mathbf{\Phi} = (\Phi_{n,m})$. For any nondecreasing sequence $(k_n)_{n=n0}^{\infty}$ satisfying $k_n = o(m)$, there exists a sequence of distributions $F_{c,n,m,\theta}$ in $\mathcal{N}$ with nested cores such that

$$\text{limsup}_{n\to\infty} L^*_{k_n}(v^*) = \epsilon < 1 = \lim_{n\to\infty} L_{k_n}(\Phi_{n,m}, v^*)$$

1.4 Thesis Outline

The unifying theme of this thesis is the task of detecting the latent structure of a graph in the presence of network anomalies. This is an area of increasing importance as latent structure models gain popularity [9, 48] in theory and practice. In Chapters 2 and 3, we consider the setting in which the anomaly inhibits graph inference (Chapter 2-3). Specifically, in Chapter 2 we develop two novel regularization methods to increase graph concentration in both structured

and unstructured / white noise contamination settings (see [74]: submitted for publication). In Chapter 3 we consider another setting in which the anomaly can be thought of as inhibiting graph inference. Specifically, we consider the task of ranking objects across multiple data views/modalities relative to a given query. The anomaly here is in the form of errorful training and/or data modalities where the query's signal is particularly weak. Our work builds on [75]'s work in which an integer linear programming (ILP) learning-to-rank framework is introduced. Originally, a set of objects known to be similar to the query is assumed to be provided. Our work here involves introducing a set of objects known to be dissimilar to the query and adding a user-in-the-loop to the algorithm, both of which are shown to increase algorithmic performance and protect against certain algorithmic anomalies (e.g., errorful training data). Finally, in Chapter 4, we consider a setting where the anomaly in the latent structure space *is* the signal that we wish to uncover. Here, we use graph embedding methods to discover sequential anomalies in network time series. Specifically, we use the DCRDPG (see 17) to naturally encode a network time-series and show the capacity of this model to capture biologically relevant signal in a connectomic time-series setting.

Chapter 2: Adversarial Contamination

2.1 Novel contribution

In this Chapter, we present two novel regularization methods to increase graph concentration in both structured and unstructured / white noise contamination settings (see our paper [74], which is submitted for publication). The structured noise setting that we build upon from [3]'s is one in which the adversarial contamination model is formulated as a probabilistic mechanism acting on the network via edge/vertex deletion or addition. Here, we develop a novel trimming method (situating our exploration in the context of random dot product graphs (RDPGs) [9, 47, 59] and stochastic blockmodel graphs) which is theoretically more tractable, computationally less expensive *and* shows empirically better performance. Since in many real data settings the distribution of the noise (contamination) is more nuanced or altogether unknown, we develop yet another regularization method for a more general diffuse noise setting, based on the k-means algorithm. Both above-mentioned trimming methods are shown to combine seamlessly and succeed in retrieving adequate information of the contaminated graph in both simulated and real-life data.

2.2 Robust graph vertex nomination

As already mentioned, graph data has become more prevalent in recent times, and therefore there has been a need for robust graph algorithms to operate in these complex data domains. In many cases of interest, inference is further complicated by the presence of adversarial data contamination designed to reduce algorithmic effectiveness. Some examples include attacks in Graph Neural Networks (GNN) via gradient-based attack procedures [76] and via reinforcement learning [77] to name a few (for a review of adversarial attacks and defenses on graphs in Deep Neural networks (DNNs) see [78]). Often, the effect of the adversary is to change the data distribution in ways that negatively affect statistical inference and algorithmic performance. In this chapter we study this phenomenon in the context of the *vertex nomination* (VN) inference task [2, 65–68], a semi-supervised information retrieval task akin to personalized recommender systems [79] on graphs. Succinctly, the vertex nomination problem can be stated as "given a pair of graphs G_1 and G_2 and a set of vertices of interest in G_1, find a corresponding set of vertices of interest in G_2 and create a rank list of the vertices in G_2" [57] (the original, single graph analogue tasked the user with finding additional vertices of interest in G_1 given a training set of vertices of interest in G_1). The corresponding vertices of interest in G_2 should ideally (if they exist) appear at the top of the nomination list. In the recent literature, algorithms for approximately solving the vertex nomination task have been implemented in myriad applied problem spaces such as finding fraud in the Enron email corpus [66, 68], identifying web advertisements associated with human trafficking [65], identifying search queries across Bing transition rate networks [3, 58], and identifying neurons across hemispheres in a Drosophila larva brain connectome [69], among others.

While the theoretical and practical impacts of adversarial noise and subsequent regularization have been widely studied in graph-valued data applications (see, for example, [77, 80–83] among myriad others), the development of these concepts in vertex nomination is relatively nascent and is an area of current research. In [3], the effect of adversarial data contamination was introduced and studied in the context of vertex nomination. In addition to developing a theoretical basis for understanding the action of an adversary in vertex nomination, the authors in [3] empirically demonstrate the expected cycle of performance degradation due to adversarial noise followed by data regularization (via the trimming method inspired by [84]) recovering much of the lost performance. The adversarial contamination model in [3] is formulated as a probabilistic mechanism acting on the network via edge/vertex deletion or addition. The goal of the adversary is to move the distribution out of the consistency class of a given vertex nomination rule (see [2]), and thus diminish algorithmic performance. Within the context of stochastic blockmodel graphs [31], a noise model (initially proposed in [81]) is introduced in [3] in which the addition/deletion of edges and vertices in the network effectively introduces "noise" blocks into the original blockmodel distribution. A network regularization method is then introduced which seeks to trim the noise blocks via a network analogue of the classical trimmed-mean estimator [85,86] for outlier contaminated data (see Section 2.3.2 for detail). While the trimming regularization of [3] is empirically demonstrated to be effective in both real and synthetic applications, it is both difficult to analyze theoretically and practically limited, as it is designed to combat a specific noise-type.

Building upon this work, we consider the effect on vertex nomination performance of a combination of both structured block adversarial noise (as defined in [3, 81]) and diffuse white noise. Situating our exploration in the context of random dot product graphs (RDPGs) [9, 47, 59] and stochastic blockmodel graphs, we propose multiple (theoretically more tractable)

regularization methods that can be combined to combat very general noise settings (see Section 2.3.4), and show empirically superior performance.

2.2.1 Vertex Nomination

Informally, the vertex nomination (VN) problem we pursue herein can be stated as follows: Given vertices of interest S_1 in a graph G_1 and a second graph G_2 with a portion of its vertices being also of interest, $S_2 \subset V(G_2)$, produce a rank list of the vertices of G_2 with the unknown vertices in S_2 ideally concentrating at the top of the rank list. What defines vertices as "interesting" is vague above, and this is intentional. Indeed, we allow the user broad leeway in defining what makes a vertex interesting, whether it be membership in a community of interest [65], vertices involved in illicit activity, or vertices corresponding to a particular user in a social network [57]. While a formal definition of the above is presented in [2, 3] and reiterated in Section 1.3.1, we do not present the full formal definition here as it would introduce a needless notational complexity, and the informal definition suffices for our present purposes.

In [3], our motivating work for adversarial VN, the authors considered the VN problem situated across a pair of networks with latent community structure. Their contamination model (see Section 2.3.1) corrupted the community communication probabilities and memberships by introducing into each community anomalous vertices with anomalous connection probabilities. In light of this, a natural model to situate our initial VN analysis is the stochastic blockmodel of [31] (see Definition 4) which posits a simple network model with latent community structure. This model allows us to handle both the network with community structure and community-structured noise (as considered in [3]). In some settings, more nuanced noise structures (that

depend on features beyond community structure) may be more appropriate, and so we will also consider in our analysis the generalized random dot product graph of [47,48] in our contamination modeling. The generalized random dot product graph (see Definition 7) will allow us to consider more "diffuse" contamination frameworks (see Section 2.3.3), including white-noise settings and manifold-structured noise settings as well. Below, we delve deeper into the VN problem setting, these two contamination models, and our novel methodologies for regularizing this noise.

2.3 Contamination and regularization in VN

In order to study the effect of contamination (and regularization) on VN performance, we will adopt the following graph model for (G_1, G_2) moving forward. We posit that $(G_1, G_2) \sim \mathrm{SBM}(n, K, \mathbf{B}, \pi, \rho, \nu)$ is the uncontaminated base graph pair. This functions to endow a natural notion of vertex correspondence between the vertices of G_1 and G_2 (as induced by the correlation ρ) and hence a canonical definition of vertices of interest in G_2 for any subset of vertices S_1 in G_1. We further assume that we are given G_1, S_1 and G_2^c, where G_2^c is the network G_2 contaminated by one of the stochastic noise models outlined below.

2.3.1 Block Contamination

Our first noise model, inspired by [81] and presented as in [3], corrupts the block structure of our underlying blockmodel network G_2. In brief, this model operates as follows. Given parameters π_+, π_-, s_+, s_- (which can vary with n, to account for sparsity):

i. Initialize $E(G_2^c) = E(G_2)$.

ii. Create the random set W_+ by independently selecting each vertex to be in W_+ with some

probability π_+. Given W_+, independently create the random set W_- by independently selecting each vertex in $V \setminus W_+$ to be in W_- with some probability π_-.

iii. For each vertex pair $\{v, u\} \in W_+ \times (V \setminus W_-)$:

1. If $\{v, u\} \in E(G_2^c)$, nothing happens.
2. If $\{v, u\} \notin E(G_2^c)$, an edge is independently *added* connecting $\{v, u\}$ in G_2^c with probability s_+.

iv. For each vertex pair $\{v, u\} \in W_- \times (V \setminus W_+)$,

1. If $\{v, u\} \notin E(G_2^c)$, nothing happens.
2. If $\{v, u\} \in E(G_2^c)$, the edge is independently *deleted* from G_2^c with probability s_-.

Note that this noise model acts on G_2 by adding and removing edges amongst subsets of the vertices. In this sense, this is an edge contamination model. However, by considering the portion of the graph that is uncorrupted (edges amongst the vertices not in $W_+ \cup W_-$) as a core network correlated to the corresponding core part of G_1, we can envision this noise model as adding vertices and edges to the core in order to corrupt the signal.

Remark 2.3.1. *In a dense SBM setting (where $\nu = 1$) with 2 blocks, consider an n-vertex stochastic blockmodel with block probability matrix given by*

$$\boldsymbol{B} = \begin{pmatrix} \mathbf{p} & \mathbf{r} \\ \mathbf{r} & \mathbf{q} \end{pmatrix}$$

where, wlog, $p \geq q \geq r \geq 0$. In this setting, the above adversarial model gives rise to a new

6-*block stochastic blockmodel, whose block probability matrix* $\boldsymbol{B}^c$ *is given by*

$$\boldsymbol{B}^c = \begin{array}{c} \begin{matrix} \tilde{B}_1 & \tilde{B}_1^+ & \tilde{B}_1^- & \tilde{B}_2 & \tilde{B}_2^+ & \tilde{B}_2^- \end{matrix} \\ \begin{pmatrix} \boldsymbol{p} & x_1 & x_2 & \boldsymbol{r} & x_3 & x_4 \\ x_1 & x_1 & p & x_3 & x_3 & r \\ x_2 & p & x_2 & x_4 & r & x_4 \\ \boldsymbol{r} & x_3 & x_4 & \boldsymbol{q} & x_5 & x_6 \\ x_3 & x_3 & r & x_5 & x_5 & q \\ x_4 & r & x_4 & x_6 & q & x_6 \end{pmatrix} \end{array} \begin{matrix} \tilde{B}_1 \\ \tilde{B}_1^+ \\ \tilde{B}_1^- \\ \tilde{B}_2 \\ \tilde{B}_2^+ \\ \tilde{B}_2^- \end{matrix}$$

where

$$x_1 := p + s_+(1-p)$$

$$x_2 := p(1-s_-)$$

$$x_3 := r + s_+(1-r)$$

$$x_4 := (1-s_-)r$$

$$x_5 := q + s_+(1-q)$$

$$x_6 := q(1-s_-)$$

Letting B_1 *and* B_2 *denote the blocks in the original SBM, in the above,* $\tilde{B}_1^+$ *are the vertices in* $W_+ \cap B_1$*;* $\tilde{B}_1^-$ *are the vertices in* $B_1 \cap W_-$*; and* $\tilde{B}_1$ *are the vertices in* $B_1 \setminus (\tilde{B}_1^- \cup \tilde{B}_1^+)$*.* $\tilde{B}_2$ *is defined analogously. Note first that the induced subgraph amongst* $\tilde{B}_1 \cup \tilde{B}_2$ *is an SBM with block probability matrix* $\boldsymbol{B}$*, and we will often consider this "core" of the contaminated* G_2 *to be*

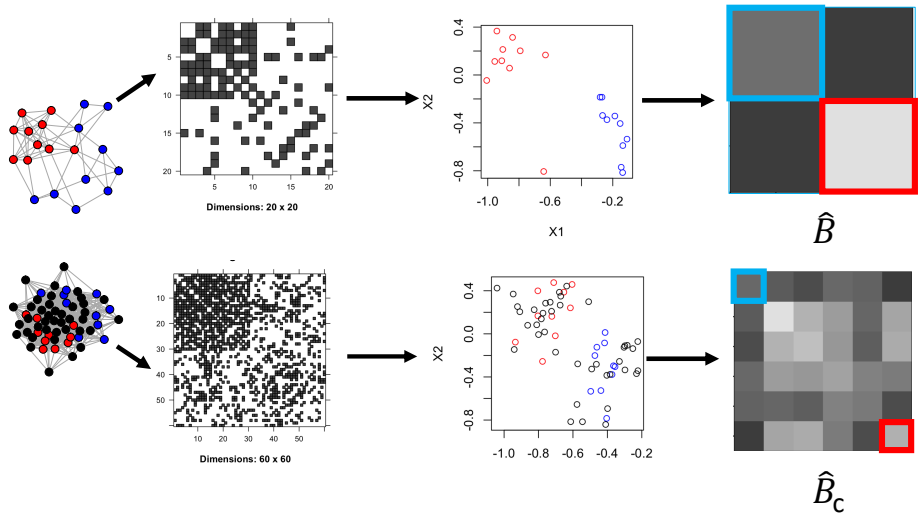

Figure 2.1: Block contamination and regularization pipeline in a simple 2-block SBM model. The top row represents the clean G_1, the bottom row the contaminated G_2. The aligned blocks in the matching step ideally recover the true correspondence across the red communities and across the blue communities (i.e., across the uncontaminated communities).

correlated to the correspondingly structured graph of G_1*. Also note that, given a* K*-block SBM, this contamination model yields a* $3K$*-block contaminated SBM.*

2.3.2 Block Regularization

To mitigate the effect of the adversary in the above-described setting, [3] proposes a graph-trimming-based regularization method that is empirically demonstrated to retrieve much of the original inferential performance. Once trimmed, the procedure they use to nominate first separately computes the ASE of the two graph, then uses seeded vertices to align the networks in the embedding space via orthogonal Procrustes analysis [87]. The vertices are then jointly clustered using model-based Gaussian mixture modeling (here BIC-penalized GMM as in [88]). Finally, candidate matches/vertices of interest are ranked based on increasing Mahalanobis distance to the vertex (or vertices) of interest [3]. One of the crucial steps in the above-described procedure

is precisely the regularization method employed, which is the network analogue of the classical trimmed mean estimator from robust statistics. This approach seeks to trim the top $h\%$ and bottom $\ell\%$ of vertices ordered by degree, where (h, ℓ) is chosen adaptively via a modularity maximization procedure.

The trimming in the above method happens *before* creating the adjacency spectral embeddings of the graphs, and the impact of the trimming on the distribution of the ASE is difficult to theoretically parse due to the complicated dependency structures that appear as a side product of the regularization. While [89] have shown that regularization enforces concentration in sparse random graphs in the spectral norm, similar concentration results for the type of row-wise norms (i.e., the $2 \to \infty$ norm) that would be needed for a finer-grained analysis are still open. In light of the myriad works proving consistency of the ASE for estimating the latent position parameters in random dot product graphs [9, 48], trimming after embedding the graphs seems a plausible alternative method to avoid inference complications.

Our new method is based precisely on this point. We first embed the graphs and estimate block/community structure (i.e., estimating the $\mathbf{B}$ matrix in an SBM setting) for each graph in spectral space. The networks are then aligned in model space (via graph matching the estimated $\mathbf{B}$ matrices), yielding a subgraph of G_2 corresponding to G_1, with the remainder of G_2 being trimmed. This induced subgraph is re-embedded and aligned to the ASE of G_1. The same ranking procedure as in the first method is used to rank the candidate matches in G_2. Below we give a more detailed description of our regularization method (see Algorithm 1 for pseudocode).

1. Separately embed G_1 (into $\mathbb{R}^{d_1}$) and G_2 (into $\mathbb{R}^{d_2}$) via ASE, where d_1 and d_2 are estimated as in Section 1.2. Estimate p_1, q_1 by counting the number of positive (respectively negative)

eigenvalues of the adjacency matrix of G_1. Estimate p_2, q_2 by following a similar procedure.

2. Separately cluster the vertices of G_1 and G_2 in the embedded space using GMM as employed by `MClust` [88]. Denote the cluster centers obtained from clustering G_1 (resp., G_2) via ξ_1 (resp., ξ_2). Modeling G_1 and G_2 via SBMs, estimate block probability matrices via $\hat{\mathbf{B}} = \xi_1 I_{p_1,q_1} \xi_1^T \in \mathbb{R}^{K_1 \times K_1}$ and $\hat{\mathbf{B}}^{(c)} = \xi_2 I_{p_2,q_2} \xi_2^T \in \mathbb{R}^{K_2 \times K_2}$. For a proof of the Frobenius norm consistency of these estimates in the SBM setting, see Theorem 2.6.1 in Appendix 2.6.1.

3. For $K_1 \leq K_2$, then we will proceed to trim the graph in model space as follows.

   i. First, align $\hat{\mathbf{B}}$ to $\hat{\mathbf{B}}^{(c)}$ by finding

   $$P \in \text{argmin}_{Q \in \Pi_{K_1,K_2}} \|\hat{\mathbf{B}} - Q\hat{\mathbf{B}}^{(c)}Q^T\|_F^2.$$

   where

   $$\Pi_{K_1,K_2} = \{P \in \{0,1\}^{K_1 \times K_2} \text{ s.t. } \vec{1}_{K_2} P \leq \vec{1}_{K_1}, P\vec{1}_{K_2} = \vec{1}_{K_1}\}.$$

   Note that while solving the above problem is NP-hard in general, there are computationally feasible options for relatively small K_1.

   ii. For P in the above argmin (which need not be unique in general), denote $\xi_{2,t} := P\xi_2$. Letting the collection of vertices whose blocks were selected by P be denoted $\mathcal{I}$ (i.e., whose blocks were *not* trimmed), we have that the trimmed network is $G_2[\mathcal{I}]$, with estimated block probability matrix (abusing notation) denoted $\hat{\mathbf{B}}^{(c,t)} = \xi_{2,t} I_{p_2,q_2} \xi_{2,t}^T$.

4. Embed $G_2[\mathcal{I}]$ into $\mathbb{R}^{d_1}$ using ASE, denoting the embeddings of G_1 and $G_2[\mathcal{I}]$ via $\hat{\mathbf{X}}_1$ and

$\hat{\mathbf{X}}_2$ respectively.

5. As we know which cluster centers in G_1 have been aligned via graph matching to those in $G_2[\mathcal{I}]$, we can use this information to align the graphs in the embedded space without the need for seeds. To wit, solve the indefinite orthogonal Procrustes problem

$$\mathbf{W} \in \text{argmin}_{\mathbf{O} \in \mathcal{O}_{p_1,q_1}} \|\xi_1 \mathbf{O} - \xi_{2,t}\|_F,$$

and set $\hat{\mathbf{X}}_{1,a} = \hat{\mathbf{X}}_1 \mathbf{W}$ (where $\mathcal{O}_{p_1,q_1} = \{\mathbf{M} \in \mathbb{R}^{d_1 \times d_1} \text{s.t. } \mathbf{M}^T I_{p_1,q_1} \mathbf{M} = I_{p_1,q_1}\}$ is known as the indefinite orthogonal group). See [90] for an approach to approximately solving the indefinite Procrustes problem. In many cases, it is appropriate to solve instead the orthogonal Procrustes problem in which we seek

$$\mathbf{W} \in \text{argmin}_{\mathbf{O} \in \mathcal{O}_d} \|\xi_1 \mathbf{O} - \xi_{2,t}\|_F,$$

rather than the indefinite version. Indeed, in our experiments below, we found it sufficient to use the orthogonal Procrustes solution here (which also obviates the need to estimate the extra parameters p_1 and q_1). We present the algorithm here in its fullest generality for use in settings where the indefinite Procrustes problem is needed.

6. Cluster the rows of $Z = \begin{pmatrix} \widehat{X}_{1,a} \\ \widehat{X}_2 \end{pmatrix}$ using GMM (here, we employ `MClust` again), and finally, rank the candidate matches in $G_2[\mathcal{I}]$ according to the following Mahalanobis-distance-based scheme.

   i. Let $u \in V(G_1)$ and $v \in V(G_2[\mathcal{I}])$ be clustered points in G_1 and $G_2[\mathcal{I}]$, and let Σ_u and

Σ_v be their respective covariance matrices obtained by the GMM-based clustering.

ii. Compute (where for a matrix $\mathbf{M}$, $\mathbf{M}^\dagger$ represents the Moore-Penrose pseudoinverse of $\mathbf{M}$)

$$\triangle(u, v) = \max(D_u(u, v), D_v(u, v)) \tag{2.1}$$

where

$$D_u(u, v) = \sqrt{(u - v)\Sigma_u^\dagger(u - v)^T}$$

$$D_v(u, v) = \sqrt{(u - v)\Sigma_v^\dagger(u - v)^T}$$

iii. Rank the vertices in $G_2[\mathcal{I}]$ by increasing value of $\min_{u \in S_1} \triangle(u, v)$, where S_1 is the set of vertices of interest in G_1.

Seeded vertices can be computationally (and financially) expensive to obtain. In addition to being more amenable to theoretical analysis, one of the principle benefits of the current regularization scheme is that it obviates the need for seeded vertices. Step 5.in the above algorithm replaces the seeded Procrustes alignment needed in [3] with an unseeded alignment of cluster centers. See Figure 2.5 for a comparison of our regularization procedures with seeds and without.

In Algorithm 1, we estimate the block probability matrices via $\hat{\mathbf{B}} = \xi_1 I_{p_1,q_1} \xi_1^T \in \mathbb{R}^{K_1 \times K_1}$ and $\hat{\mathbf{B}}^{(c)} = \xi_2 I_{p_2,q_2} \xi_2^T \in \mathbb{R}^{K_2 \times K_2}$. The following Theorem shows the Frobenius norm consistency of the analogous estimates obtained by the optimal Mean Squared Error-based clustering (in Step 2 of Algorithm 1) in the Stochastic Blockmodel (SBM) setting.

Theorem 2.3.1. Let $\hat{\boldsymbol{\mu}}$ be the matrix of cluster centroids provided by the optimal Mean Squared

Error-based clustering of $\hat{\mathbf{X}}$ in the uncontaminated network (with $\hat{\boldsymbol{\mu}}^{(c)}$ defined analogously for the contaminated SBM network as defined in [74]). Let $\mathbf{B}$ and $\mathbf{B}^{(c)}$ be the block probability matrices for the uncontaminated and contaminated networks, under suitable assumptions on the SBM model parameters (see Theorem 1 of [74] for specific details), we have that

$$\| \underbrace{\hat{\boldsymbol{\mu}} I_{p_1,q_1} \hat{\boldsymbol{\mu}}^T}_{:=\hat{\mathbf{B}}} - \nu_n \mathbf{B} \|_F = O_{\mathbb{P}}\left(\frac{K_1 \log^{\alpha} n}{n}\right) \quad \| \underbrace{(\hat{\boldsymbol{\mu}}^{(c)}) I_{p_2,q_2} (\hat{\boldsymbol{\mu}}^{(c)})^T}_{:=\hat{\mathbf{B}}_c} - \nu_n \mathbf{B}^{(c)} \|_F = O_{\mathbb{P}}\left(\frac{K_2 \log^{\alpha} n}{n}\right)$$

We then use the estimates $\hat{\mathbf{B}}$ and $\hat{\mathbf{B}}^{(c)}$ to trim the contaminated vertices in G_2. We can accomplish this by solving the following graph matching type problem. For each $a, b \in \mathbb{Z} > 0$, define $\Pi_{a,b} = \{P \in \{0,1\}^{a \times b} : P\vec{1}_b = \vec{1}_a,\ \vec{1}_a^T P \leq \vec{1}_b\}$. Note that there exists a $P^* \in \Pi_{K,3K}$ such that $\|\mathbf{B} - P^* \mathbf{B}^{(c)} (P^*)^T\|_F = 0$. We then seek to show that with high probability, $P^* = \text{argmin}_{P \in \Pi_{K,3K}} \|\hat{\mathbf{B}} - P \hat{\mathbf{B}}^{(c)} P^T\|_F^2$. This would imply that P^* also recovers the correspondence between the original, non-contaminated blocks in $\hat{\mathbf{B}}$ and the uncontaminated portion of $\hat{\mathbf{B}}^{(c)}$, with the remaining blocks (which ideally capture the contamination) then trimmed by our procedure. This is formalized in the following result:

Theorem 2.3.2. Let $\mathbf{B}$ and $\mathbf{B}^{(c)}$ be such that

$$\min_{P \in \Pi_{K,3K} \setminus \{P^*\}} \|\mathbf{B} - P \mathbf{B}^{(c)} P^T\|_F = \Omega(1)$$

For K fixed, we then have

$$\mathbb{P}\left(P^* = \text{argmin}_{P \in \Pi_{K,3K}} \|\hat{\mathbf{B}} - P \hat{\mathbf{B}}^{(c)} P^T\|_F\right) \to 1$$

We prove Theorems 2.6.1 and 2.6.2 in Section 2.6.

2.3.2.1 Trimming in model versus graph space

Since the trimming [3] happens in the graph space, we will refer to it as the "graph trimming" method. Similarly, we will refer to our newly presented method as "model trimming". We next explore the comparative performance of these two methods using the same simulation setup as that described in [3] (albeit, with different noise levels). In our simulations, we consider a graph G_1 with 500 core vertices and a contaminated graph G_2 with $500 + m$ vertices (here m represents the level of contamination, where we consider $m = 200$ and $m = 400$). To wit, the graphs are sampled from the following SBM model:

i. $G_1 \sim \text{SBM}(500, 2, \mathbf{B}, \vec{n} = (250, 250), \nu = 1)$ where

$$\mathbf{B} = \begin{bmatrix} 0.7 & 0.2 \\ 0.2 & 0.3 \end{bmatrix}$$

Here we use the true $d_1 = 2$ in the ASE embedding.

ii. $G_2 \sim \text{SBM}(500 + m, 6, \mathbf{B}^{(c)}, \vec{n} = (250, m/4, m/4, 250, m/4, m/4), \nu = 1)$ where $\mathbf{B}^{(c)}$ is

generated from $\mathbf{B}$ as described in Section 2.3.1 with $s+ = s_- = 0.2$, i.e.,

$$\mathbf{B}^{(c)} = \begin{bmatrix} 0.70 & 0.76 & 0.56 & 0.20 & 0.36 & 0.16 \\ 0.76 & 0.76 & 0.70 & 0.36 & 0.36 & 0.20 \\ 0.56 & 0.70 & 0.56 & 0.16 & 0.20 & 0.16 \\ 0.20 & 0.36 & 0.16 & 0.30 & 0.44 & 0.24 \\ 0.36 & 0.36 & 0.20 & 0.44 & 0.44 & 0.30 \\ 0.16 & 0.20 & 0.16 & 0.24 & 0.30 & 0.24 \end{bmatrix}$$

Here we use the true $d_2 = 6$ in the ASE embedding.

iii. Letting $\mathcal{C} = [1 : 250, (250 + m/2 + 1) : (500 + m/2)]$ $(G_1, G_2[\mathcal{C}]) \sim \text{SBM}(500, 2, \mathbf{B}, \vec{n} = (250, 250), \rho)$, we obtain two graphs which have correlated core vertex sets.

We analyzed the performance of the VN task after trimming under both methods: graph trimming of [3] and model trimming (using, for ease of comparison, simple orthogonal Procrustes in step 3.i of our procedure as opposed to the generalized Procrustes solver); the results are summarized in Figure 2.2. Note that the graph trimming method requires seeded vertices to run to completion and thus, for this experiment we had used 10 randomly chosen seeds from $\mathcal{C}$ to align the embeddings when doing graph trimming. In each panel of the figure, we plot on the y-axis the number of vertices in G_1 (when considered as the vertex of interest) with their corresponding vertex of interest ranked in the top x. In other words, for $x = k$, $y = f(x)$ gives us the number of vertices in G_1 which have their corresponding vertex of interest (in G_2) ranked $1^{st}, 2^{nd}, ...,$ or, k^{th}. In the middle (resp., bottom) row, we plot the performance of the trimming method proposed herein (resp., the trimming method of [3]). In the top row, we plot the difference in performance across

the two methods (model trimming - graph trimming). In the first column (resp., second and third), we show performance for $m = 200$ and $\rho = 0.7$ (resp., $m = 400$, $\rho = 0.7$ and $m = 200$, $\rho = 0.9$). Each figure represents 30 Monte Carlo simulations (paired within each column), with performance in each simulation plotted in gray and the average over all MC plotted in red (top) or black (bottom two rows). In the bottom two rows, the blue line represents chance performance.

In an ideal setting, where all the noise is trimmed perfectly and the nomination task performs perfectly, we would expect a horizontal line at $y = k^*$, where k^* is the number of vertices in G_1. From the figure, we observe that a higher correlation provides better VN performance for both methods, as we would intuitively expect. Furthermore, in graphs with greater number of noise vertices, the model trimming method performs significantly better than the graph trimming method. In the graphs with less noise, the model trimming method still performs better on average but the performance difference is less pronounced. We emphasize that the model trimming method uses no seed vertices while the graph trimming method uses ten seed vertices. The effectiveness of the graph trimming method in [3] was demonstrated in graphs with a relatively smaller amount of noise, and we suspect that our current method is indeed preferable in high-noise settings. In all cases, both methods are significantly better than chance here.

2.3.3 Diffuse noise contamination and regularization

Although the above-described block regularization method is empirically (and theoretically) effective in the contaminated SBM setting above, in many real data settings the distribution of the noise is more nuanced or altogether unknown. Developing further regularization strategies to deal with broader noise models is a natural next step. In this section we describe another

Algorithm 1 Block regularization pseudocode

Input: G_1 and G_2 and S_1

1. $\hat{\mathbf{X}}_1 = \text{ASE}(G_1, d_1)$ and $\hat{\mathbf{X}}_2 = \text{ASE}(G_2, d_2)$ (See Definition 1.3) and estimate p_1, q_1, p_2, q_2, where $p_1 + q_1 = d_1$ and $p_2 + q_2 = d_2$.
2. Separately cluster the rows of $\hat{\mathbf{X}}_1$ and $\hat{\mathbf{X}}_2$ via `MClust` (see [88]); Denote the cluster centers obtained from clustering $\hat{\mathbf{X}}_1$ (resp., $\hat{\mathbf{X}}_2$) via ξ_1 (resp., ξ_2);
3. Set $\hat{\mathbf{B}} = \xi_1 I_{p_1,q_1} \xi_1^T \in \mathbb{R}^{K_1 \times K_1}$ and $\hat{\mathbf{B}}^{(c)} = \xi_2 I_{p_2,q_2} \xi_2^T \in \mathbb{R}^{K_2 \times K_2}$;
4. Find $P \in \text{argmin}_{Q \in \Pi_{K_1,K_2}} \|\hat{\mathbf{B}} - Q\hat{\mathbf{B}}^{(c)}Q^T\|_F$;
5. For P in the above argmin, denote $\xi_{2,t} := P\xi_2$. Let

   $$\mathcal{I} = \{v \in V_2 \text{ s.t. } v's \text{ corresponding cluster center is selected by } P\};$$

   redefine $\hat{\mathbf{X}}_2 = \text{ASE}(G_2[\mathcal{I}], d_1)$;
6. Solve $\mathbf{W} \in \text{argmin}_{O \in \mathcal{O}_{p_1,q_1}} \|\xi_1 O - P\xi_2\|_F$ and set $\hat{\mathbf{X}}_{1,a} = \hat{\mathbf{X}}_1 \mathbf{W}$;
7. Cluster the rows of $[\widehat{X}_{1,a}^T | \widehat{X}_2^T]^T$ via `MClust`, and rank the vertices by increasing value of $\min_{u \in S_1} \triangle(u, v)$ where $\triangle$ is the Mahalanobis-like distance computed in Eq. 2.1.

technique, based on a robust K-means clustering method, that can be used in the setting where the contamination is unstructured (or less structured) diffuse noise. We first describe the contaminated latent positions in the diffuse noise model, before giving a detailed description of our K-means based cleaning method.

Our starting point is that G_2 is a k-block SBM, that we further contaminate with unstructured noise. To wit, let $\mathcal{X}_d$ be a subset of $\mathbb{R}^d$ such that $x^\top I_{p,q} y \in [0, 1]$ for all $x, y \in \mathcal{X}_d$ and Ω be a convex subset of $\mathcal{X}_d$. Next let $\{\mathfrak{z}^i\}_{i=1}^k$ be a collection of k distinct points in $\mathcal{X}_d$. We then sample n latent positions $\mathbf{Y}$ for a SBM graph from $F = \sum_{i=1}^k \pi_i \delta_{\mathfrak{z}^i}$ with $\pi_i > 0$ for $i \in [k]$; note that $\mathbf{Y} \in \mathbb{R}^{n \times d}$ has at most k distinct rows. We then contaminate $\mathbf{Y}$ with i.i.d. "white noise" latent positions, $\mathbf{Z} \in \mathbb{R}^{m \times d}$, whose rows are i.i.d. uniformly distributed over Ω. Our contaminated model's latent positions can therefore be written as $\mathbf{X} = [\mathbf{Y}^T | \mathbf{Z}^T]^T \in \mathbb{R}^{(n+m) \times d}$.

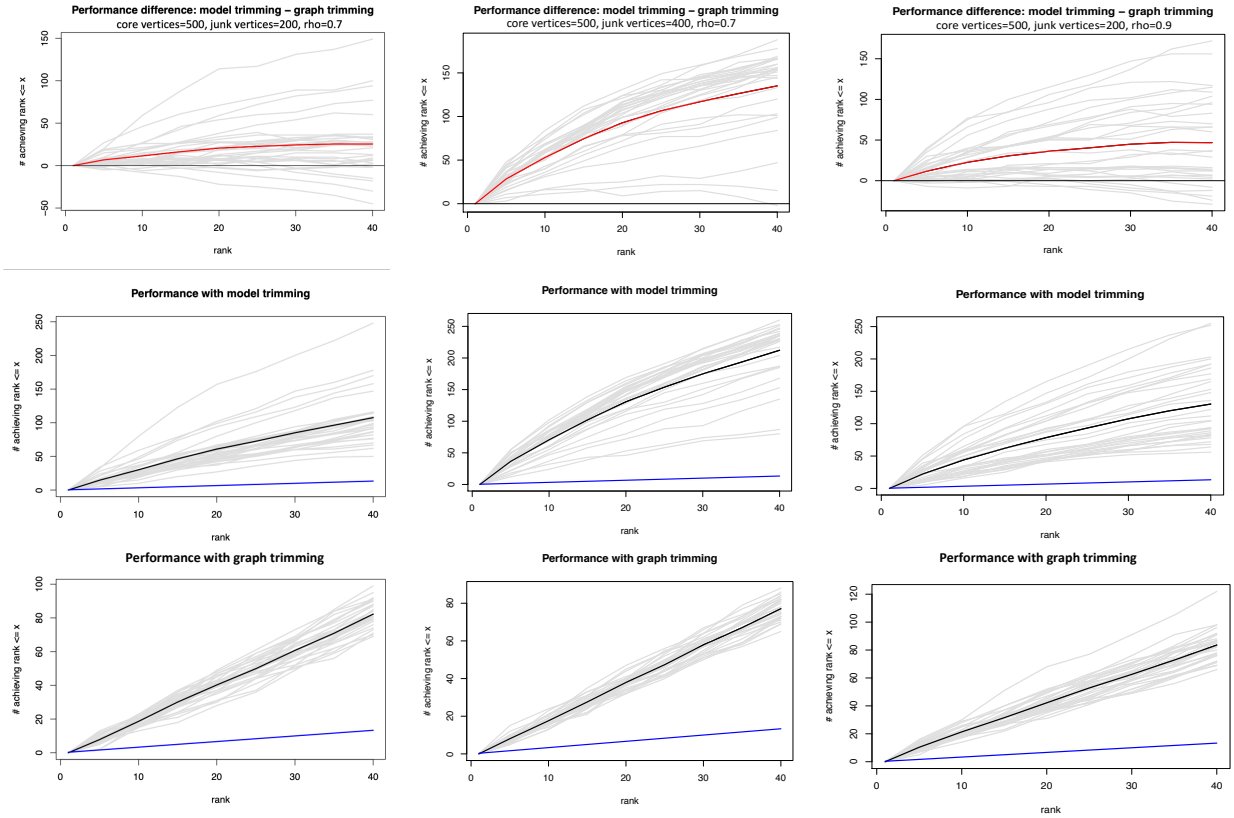

Figure 2.2: In each panel of the figure, we plot on the y-axis the number of vertices in G_1 (when considered as the vertex of interest) with their corresponding vertex of interest ranked in the top x. In the middle (resp., bottom) row, we plot the performance of the trimming method proposed herein (resp., the trimming method of [3]). In the top row, we plot the difference in performance across the two methods. In the first column (resp., second and third), we show performance for $m = 200$ and $\rho = 0.7$ (resp., $m = 400$, $\rho = 0.7$ and $m = 200$, $\rho = 0.9$). Each figure represents 30 Monte Carlo simulations (paired within each column), with performance in each simulation plotted in gray and the average over all MC plotted in red (top) or black (bottom two rows). In the bottom two rows, the blue line represents chance performance. Note that the range of the y-axis changes from figure to figure due to the difference in performance, number of noise vertices added, and vertices trimmed.

The contaminated graph $G_2 \sim$GRDPG($\mathbf{X}, \nu$). Note that the general nature of Ω makes $\mathbf{Z}$ a natural model for more diffuse manifold noise contamination.

In order to regularize the diffuse noise out of G_2, we will employ a robust K-means clustering algorithm on $\hat{\mathbf{X}}$ =ASE(G_2, d) as described below. Let $\mathcal{P}_2$ be the set of partitions of $[n + m]$ into two groups, and $\mathcal{C}_K$ the collection of sets of K distinct points in $\mathcal{X}_d$. We seek to

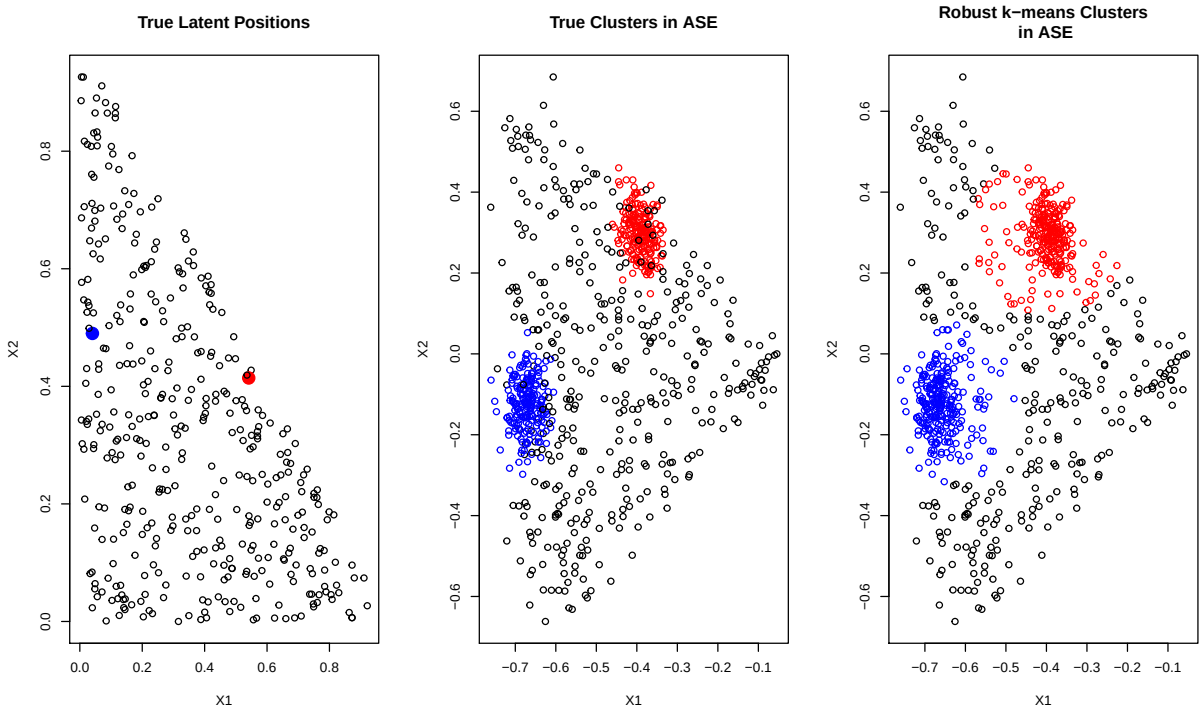

Figure 2.3: Diffuse noise contamination example in a 2-block SBM. In the left panel, we plot the true latent positions of the signal (red/blue) and the noise (black); in the center panel, we plot the estimated latent positions provided via ASE of the signal vertices (red/blue) and the noise (black)—note the rotation inherent to the GRDPG model; in the right panel, we plot the clusters (in color) recovered by the robust K-means with $K = 2$ and $\lambda = 0.2$.

solve the following optimization problem with tuning parameter $\lambda > 0$,

$$\min_{\Phi \in \mathcal{C}_K, \mathfrak{p} \in \mathcal{P}_2} \underbrace{\left(\sum_{i \in \mathfrak{p}_1} \min_{\phi \in \Phi} \|\phi - \hat{X}_i\|_2 + \lambda |\{j : j \in \mathfrak{p}_2\}| \right)}_{\Gamma(\Phi, \mathfrak{p})}. \tag{2.2}$$

The partition of the vertices provided by $\mathfrak{p} \in \mathcal{P}_2$ divides the vertex set of G_2 into estimated signal vertices (i.e., those in $\mathfrak{p}_1$) and estimated noise vertices (those in $\mathfrak{p}_2$). The vertices in $\mathfrak{p}_1$ contribute the the error in Eq. (2.2) via the usual K-means term and are further clustered into K disjoint groups. Those vertices in $\mathfrak{p}_2$ are far from the cluster centers, and are not included in any one of the K clusters. These unclustered points incur a constant cost (here λ) in Eq. (2.2); λ here is designed

to penalize partitions that would cluster too few vertices in the noisy graph G_2 while also allowing for noise vertices to be excluded from the final clustering. In practice, we can choose λ based on the size of the clusters obtained by clustering the clean graph G_1; if r is the largest cluster radius in G_1 then one simple heuristic is to choose λ to be approximately $r + \log^2(n+m)/\sqrt{n+m}$ (see Eq. 4.3) (we found $\lambda = 0.2$ sufficient in most of our experiments). The graph G_2 is then regularized by considering the "cleaned" graph $G_2[\mathfrak{p}_1]$ in which the unclustered (ideally noise) vertices have been trimmed. In the context of the above model, in Appendix 2.6.2, we establish the consistency of the optimal K-means clusters solving Eq.(2.2).

Remark 2.3.2. *In order to approximately solve Eq.(2.2), we adopt the following simple heuristic. We estimate a maximum cluster radius r^* from a clustering of G_1, and use this optimal clustering in variant of the classical K-means procedure as follows:*

i. Set $\mathfrak{p}_1 = V, \mathfrak{p}_2 = \emptyset$, and initialize the algorithm via an unconstrained K-means clustering of the rows of $\widehat{X}$; Denote the cluster centers via $\{\mu_i\}_{i=1}^K$ and the cluster assignment vector via

$$b_v^{(K)} = \begin{cases} \textit{argmin}_{i\in[K]} \|\widehat{X}_v - \mu_i\| & \textit{if } v \in \mathfrak{p}_1; \\ 0 & \textit{if } v \in \mathfrak{p}_2; \end{cases}$$

ii. Iterate the following two K-means adjacent steps until stopping conditions are met

a. Update the K cluster centers

$$\mu_i = \frac{1}{|\{v \in \mathfrak{p}_1 \textit{ s.t. } b_v^{(K)} = i\}|} \sum_{\{v\in\mathfrak{p}_1 \textit{ s.t. } b_v^{(K)}=i\}} \widehat{X}_v$$

b. For each $v \in V(G_2)$*, if*

$$\min_i \|X_v - \mu_i| < r^*,$$

set $v \in \mathfrak{p}_1$ *and* $b_v^{(K)} = i$*. Else, set* $v \in \mathfrak{p}_2$ *and* $b_v^{(K)} = 0$*.*

Note that, in practice we iterate Step ii.until the cluster assignments do not change or a maximum number of iterates has been met. We often run the above clustering multiple times (keeping the clustering minimizing Eq. (2.2)) to account for randomness in the initialization of the K*-means algorithm. For an example of this algorithm in practice, see Figure 2.3.*

2.3.4 Regularizing multiple noise sources

Heterogeneous and multimodal noise settings are very common when working with real data. The two above-described cleaning methods can be strung together seamlessly to potentially ameliorate multiple noise sources simultaneously. Consider the setting where G_2 is contaminated by both the block-noise of Section 2.3.1 and the diffuse noise of Section 2.3.3. This represents an idealized version of contamination that is both structured (here, designed to obfuscate the true graph model in model space) and unstructured. Combining the two regularization strategies outlined above yields a two-step approach in which we first clean out the unstructured noise using the robust K-means method and then use our block-noise regularization algorithm to trim the structured noise. Below we present simulation experiments showing the performance of the VN task using our two-step cleaning procedure (again using regular orthogonal Procrustes rather than indefinite orthogonal Procrustes). Note that we will often compare performance post-regularization to a non-regularized VN procedure that operates via: using seeded vertices to align the embeddings of the two networks (without trimming) and nominating based on Step 6 of the

procedure in Section 2.3.2.

We first consider the contaminated SBM model from Section 2.3.2.1 with $m = 1000$ noise vertices and $\rho = 0.7$. To this, we add 500 additional white noise vertices sampled from (suitably rotated to yield feasible latent positions) the positive orthant in $\mathbb{R}^6$. In our cleaning procedures, we consider $d_1 = 2$, $d_2 = 6$, and $K = 6$. We use the true number of clusters in the initial `MClust` clustering steps (Step 2 of Algorithm 1). Figure 2.4 shows the performance on the VN task after the implementation of the two-stage regularization procedure. In the figure, we again plot on the y-axis the number of vertices in G_1 (when considered as the vertex of interest) with their corresponding vertex of interest ranked in the top x. In the (L) panel, we plot the absolute performance of the 2-stage cleaning regularization procedure; in the (R) panel, we plot the difference in performance of the core spectral VN procedure post-regularization versus without regularization (post-pre). Note that the post-cleaning method does not require seeds, while the pre-cleaning method does (embed the graphs, seeded Procrustes to align, cluster and rank based on the computed Mahalanobis distance). Each grey line represents one of 30 Monte Carlo simulations, with the average performance over all MC plotted in black (L) or red (R). The blue line represents chance performance. We see that our 2-stage cleaning procedure is effective at mitigating the effect of the noise, even without seeds, and allows for significantly improved nomination performance versus running the core VN procedure sans cleaning. This confirms our presumption that our regularization method proves to be effective in retrieving most of the original data signal, at least in simulation.

Seeds are often an algorithmic *luxury* and not always available in real-life settings. Therefore, presenting a version of our algorithm that does not require seeds is a highly advantageous task. A natural question though is what is lost (performance-wise) in the un-seeding? While it is clear

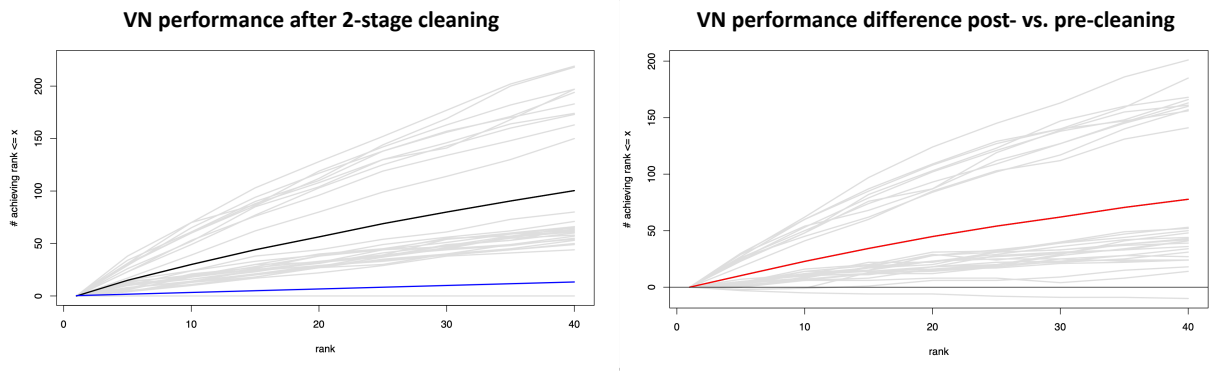

Figure 2.4: Vertex Nomination performance after the implementation of the two-stage regularization with $\lambda = 0.2$. We plot on the y-axis the number of vertices in G_1 (when considered as the vertex of interest) with their corresponding vertex of interest ranked in the top x. In the (L) panel, we plot the absolute performance of the 2-stage cleaning regularization procedure; in the (R) panel, we plot the difference in performance post-regularization versus without regularization (post-pre). Each grey line represents one of 30 Monte Carlo simulations, with the average performance over all MC plotted in black (L) or red (R). The blue line represents chance performance.

that optimally using seeds will always yield enhanced (or, at least, no worse) performance, this is not necessarily the case in the present algorithmic setting. In the VN procedure outlined in Algorithm 1, seeds would be incorporated in Step 4, where seeded-Procrustes would replace the unseeded alignment of the cluster centers. Surprisingly, incorporating seeds in this (natural) way yields significantly poorer algorithmic performance in simulations. With the above setup, we consider the effect of incorporating seeds in Figure 2.5. In each panel, we plot on the y-axis the number of vertices in G_1 (when considered as the vertex of interest) with their corresponding vertex of interest ranked in the top x. In the (L) panels, we plot the absolute performance of the 2-stage cleaning regularization procedure with (top) and without (bottom) seeds; in the (R) panel, we plot the difference in performance with and without seeds (with-without). Each grey line represents one of 30 (paired) Monte Carlo simulations, with the average performance over all MC plotted in black (L) or red (R). The blue lines on the left represent chance performance. We see significant performance improvement when applying the block-regularization procedure

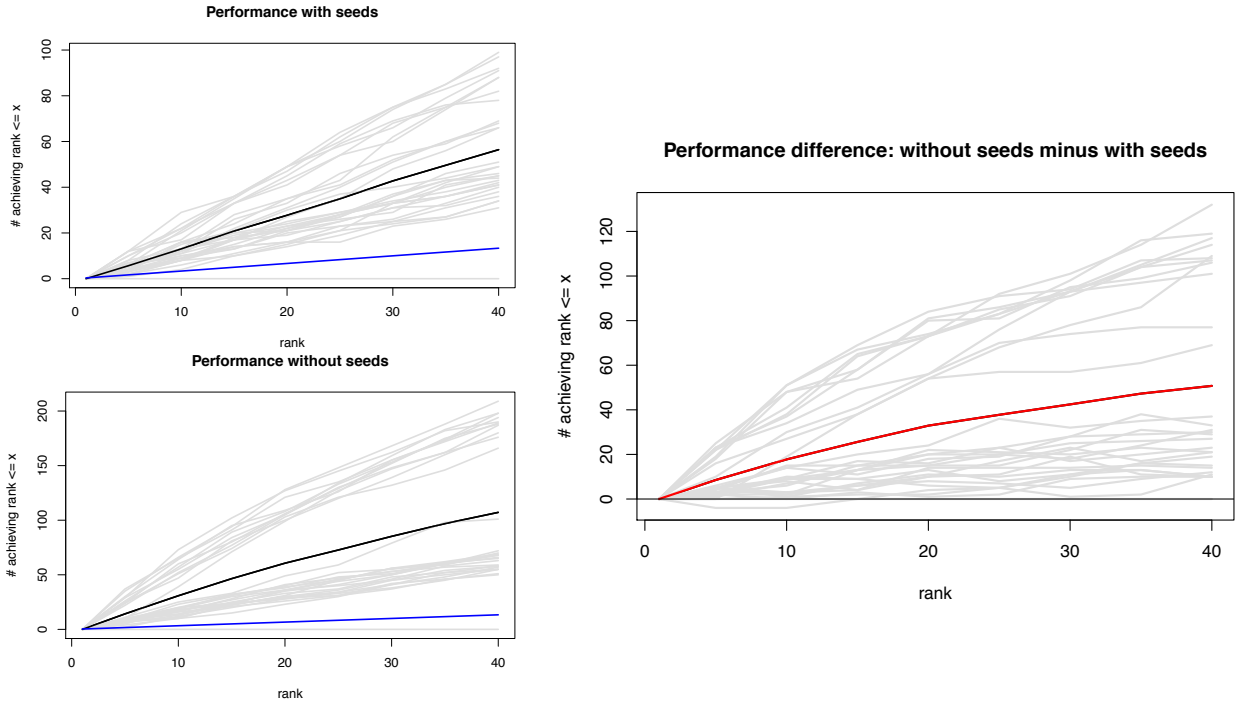

Figure 2.5: Vertex Nomination performance after with and without seeds. In each panel, we plot on the y-axis the number of vertices in G_1 (when considered as the vertex of interest) with their corresponding vertex of interest ranked in the top x. In the (L) panels, we plot the absolute performance of the 2-stage cleaning regularization procedure with (top) and without (bottom) seeds; in the (R) panel, we plot the difference in performance with and without seeds (with-without). Each grey line represents one of 30 (paired) Monte Carlo simulations, with the average performance over all MC plotted in black (L) or red (R), with λ chosen to be 0.2 in the robust k-means cleaning. The blue lines on the left represent chance performance.

(as part of the 2-stage pipeline) without seeds; we postulate that this is because of the de-noising due to averaging in the estimated cluster centers versus the relatively noisy latent positions of the individual seeded vertices.

2.4 Real data experiments

In this section we present experiments based on three real data sets: the High School Friendship social network dataset from [91], connectomic Brain Data registered via the common DS data template [92], and the political blogs data of [93]. In both these settings, we describe how

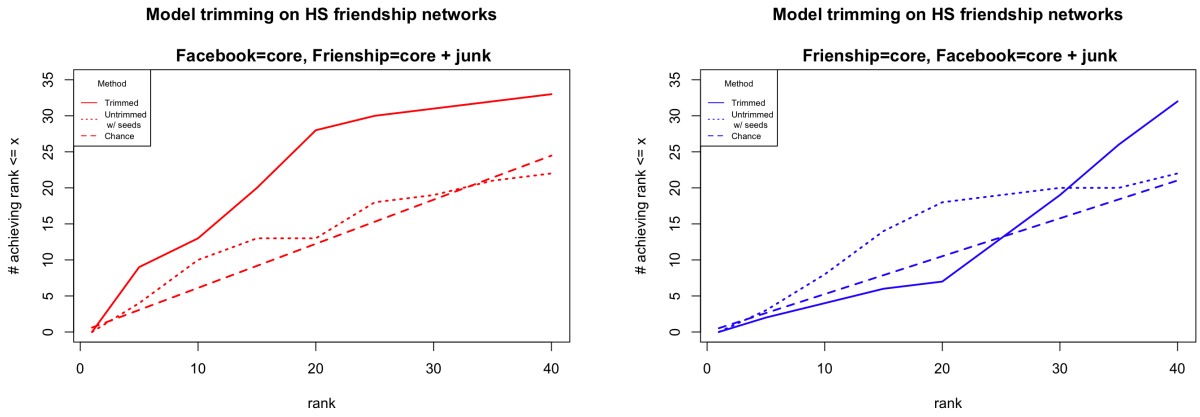

Figure 2.6: Block regularization for Vertex Nomination across the HS Friendship Social Networks. In each panel, we plot on the y-axis the number of vertices in the uncontaminated network (when considered as the vertex of interest) with their corresponding vertex of interest ranked in the top x. In the left (resp., right) panel, the uncontaminated network is the core Facebook (resp., core Friendship) network and the contaminated graph is the full Friendship (resp., full Facebook) network. The solid lines represent performance post-cleaning, the dashed lines chance and the dotted lines performance without cleaning (with seeds).

the algorithms we propose are used to help mitigate the effect of an adversary in the respective settings.

2.4.1 VN on High School Friendship Social Networks

The data was collected from high school students at Lycée Thiers in Marseilles, France [91], and is composed of two different friendship social networks, each encapsulating a different interaction dynamic.

i. **Facebook Friendship**

Students were asked to use the Netvizz application which then created the network of Facebook friendship relations between the Facebook friends of each student who uses the application. Since only 17 students gave access to their local network, it was not possible to build the entire network of Facebook relationships between the students through this

information. Instead, a list of pairs of students, ("known-pairs" for which the existence or not of a friendship relation on Facebook was known), was used. The number of students considered here was 156.

ii. **Self-reported Friendship**

Students were also asked to complete a survey in which they were asked to name their friends in the high school. We consider 134 friendship surveys in our analysis.

Specifically, our data set consists of two graphs, G_1 (the self-reported friendship graph; we will refer to this as the Friendship network) and G_2 (the Facebook friendship graph), containing 134 and 156 vertices respectively. In both graphs, the vertices represent students and the edges represent their friendship, and there is a core set of 82 vertices appearing in both graphs. In G_1, two vertices are adjacent if at least one of the students reported to be friends with the other one, whereas in G_2 two vertices are adjacent if the corresponding students are friends on Facebook.

In this setting, we consider two experimental setups. In the first one, we only consider the core vertices in G_1 and treat the non-core vertices of G_2 as a contamination. We then use our new block-contamination trimming method to regularize G_2 and analyse the performance of our VN procedure on the regularized graph. In the second setting, we "switch roles". We let G_1 be the contaminated graph and consider G_2 to be the clean graph, i.e. we only consider the core vertices of G_2. Results are summarized in Figure 2.6, and we note here that the method is implemented without seeds in both cases (contrasting with the similar VN analysis in [57] that was seed dependent). From the figure, we see that performance here is highly dependent on the nature of the model "noise." Indeed, when the contaminated graph is the Friendship graph, it is clear that the trimmed setting performs much better than both the untrimmed setting and

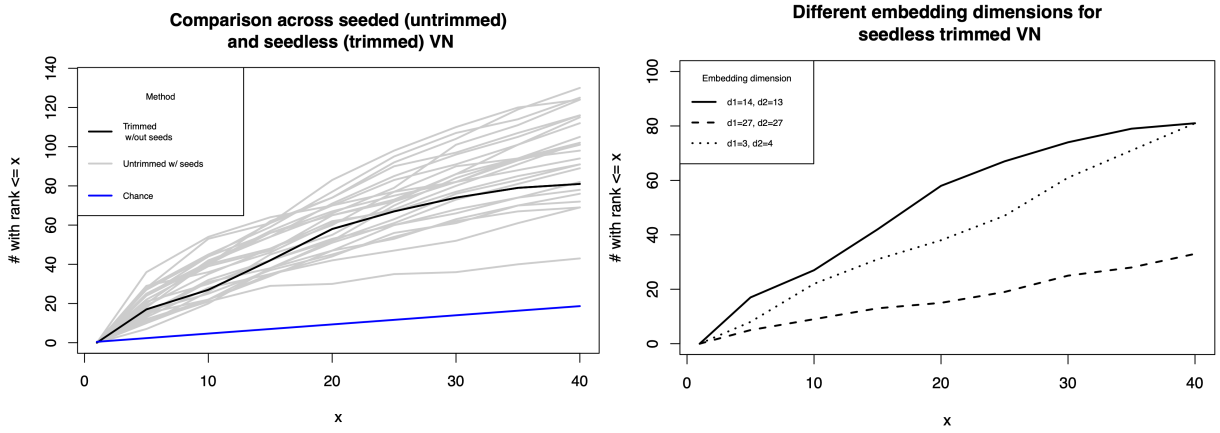

Figure 2.7: The left Figure shows the performance of the Vertex Nomination task in the brain data setting when trimming the noise vertices without seeds (grey lines) and when no trimming is performed and seeds are used. These performances are compared to chance. One can conclude that the trimmed setting seems to perform at least as good as the untrimmed setting, but without the need of seeds. The right Figure shows performance of the Vertex Nomination task for different embedding dimensions when computing the ASE of the graphs.

chance. This seems to imply that our algorithm helps mitigate the affect of the adversary and retrieves most of the original graph structure in this setting. In contrast, when the contaminated graph is the Facebook graph, the VN task performance after trimming is not that much better than before trimming (or even chance). Two plausible explanations for these differences are as follows. Firstly the optimum number of clusters chosen by `MClust` for the Facebook core (resp., junk) is 4 (resp., 8) and for the Friendship core (resp., junk) is 8 (resp., 7). Model trimming is thus possibly ineffective when G_1 is taken to be the Friendship core network while the contaminated G_2 is taken to be the Facebook graph as they both have 8 estimated blocks. The second explanation is that there is significant difference in the nature of online friendships versus reported friendships such as sampling bias inherent to friendship survey data [91].

2.4.2 Brain Data

The data used in this experiment is a subset of the Connectivity-based Brain Imaging Research Database (C-BIRD) at Beijing Normal University (BNU). It contains data from 57 healthy young volunteers, 30 males and 27 males of ages 19-30, who completed two MRI scan sessions within an interval of approximately 6-weeks. A graph was generated from each of the two sessions. In our experiments, vertices of the brain graph represent voxel regions in the brain (after they have been registered to a common template), with edges measuring neural connection amongst regions. In our experiment setup, we consider two of the above-described brain scans of the same individual, which are inherently correlated, and define our graphs G_1 and G_2 as follows. G_1 is a subgraph of the first brain scan graph, in which we only consider the vertices in the regions that lie in the left hemisphere of the individual. For G_2 we consider the whole second brain scan graph. Based on the regularization theory presented above, one can consider the second graph, G_2 to be a contaminated version of the graph G_2', where G_2' consists of the vertices in the second brain scan that occur in the regions that lie in the left hemisphere of the brain. The rest of the vertices can be considered "contaminated" vertices, having no analogue in G_1. Note that one can consider a homology between the two hemispheres of the brain and thus the contamination described above can be thought of as structured noise rather than simply white noise. Our goal is to find the vertices in the second scan of the individual, which correspond to the vertices in the regions in the left hemisphere of the brain from the first scan.

There are 70 regions in each brain (derived via the Desikan atlas). Regions $1 - 35$ occur in the left hemisphere of the brain. Hence, we can easily create the induced subgraph G_1 by accessing the vertices that lie in these regions. Using `MClust` to estimate the block structure

in G_1 and G_2, we use Algorithm 1 to trim the "block-structured" noise from G_2. We then compare the performance of our algorithm using *no seeds* with the case when no trimming is performed and 5 seeds are being used to align the embedded networks. Figure 2.7 (left) shows the performance of the above described experiments with (gray) and without (black) seeds, compared to chance performance (blue). The experiments with seeds are repeated 25 times, each time using a randomly selected seeds set. As one can easily deduce from the figure, both methods (trimmed with no seeds and untrimmed with seeds) perform better than chance. Moreover, the trimmed setting often outperforms the untrimmed setting with seeds. This is encouraging, as seeds are expensive and hence being able to retrieve information without being provided seeds is a very much desired outcome.

A common source of error in experiments including embeddings arises when choosing the dimension of the embedding. We compare algorithmic performance of our experiment by choosing different embedding dimensions when computing the ASE of the graphs. In Figure 2.7 (RIGHT) one can see that $d_1 = 14$ and $d_2 = 13$ (chosen by finding the second elbow of the SCREE plot of G_1 and G_2 [9]), outperforms embedding methods that underestimate (using the first SCREE elbow) or overestimate (using the third SCREE elbow) the embedding dimension.

2.4.3 Political blogs

In this section, we consider the robust K-means procedure followed by nomination in the network of hyperlinks between weblogs on US politics [94]. Specifically, the posts of 40 blogs were analyzed over the period of two months preceding the U.S. Presidential Election of 2004. The blogs are natural segmented into two parts—liberal-leaning and conservative-leaning—and

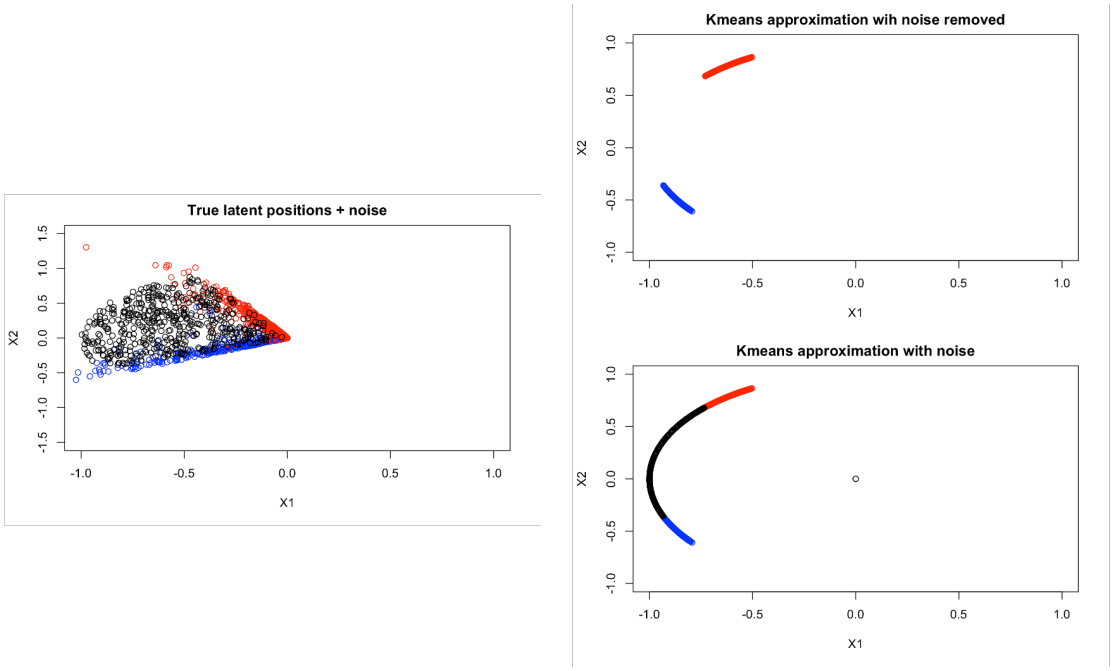

Figure 2.8: LEFT: The true latent positions (liberal/conservative) plus added noise. RIGHT: The clustering results (projected on the unit sphere) after the implementation of our kmeans method, with noise (above) and without the noise (below).

edges in the network capture how these blogs referred to each other both within and across communities. In order to test the performance of our robust K-means algorithm, we adopt the following synthetic data approach. We first consider the blog graph G_1 (the original data) as a 2-dimensional RDPG and estimate the latent positions of the network using ASE. To these points, we add an additional sample of $m = 500$ points drawn uniformly at random from the 2-sphere (first orthant) to artificially create noise data points. We then sample G_2 as an RDPG from the signal plus noise latent positions. Here, G_2 represents a version of G_1 corrupted by two noise sources, the m diffuse noise vertices and the noise from the RDPG resampling procedure. We then proceed as follows:

i. Re-embed G_2 (call this $\hat{\mathbf{X}}_2$), and project the embedded data to the sphere (this is a common tactic in clustering sparse graphs [48, 50]); use the robust 2-means clustering to clean the

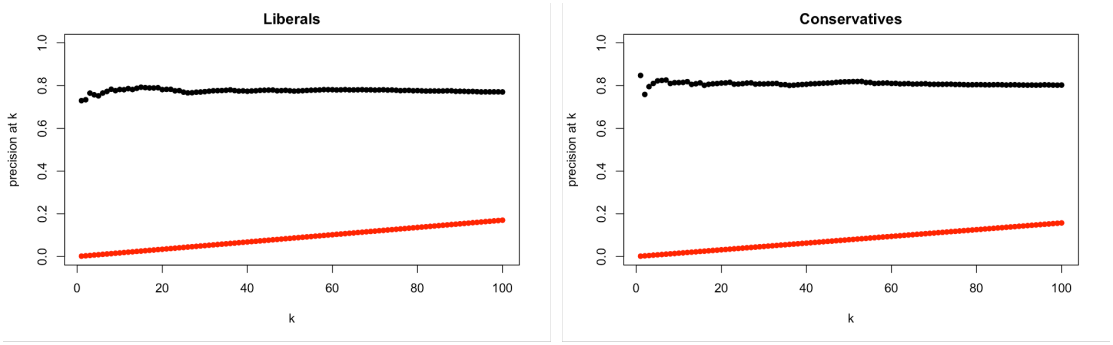

Figure 2.9: The mean precision at k of each vertex representing a liberal blog post (when considered as the vertex of interest) for different values of k, namely $k = 1, ..., 50$. Specifically, for each vertex of interest (when the vertex represents a liberal blog post), we calculate the number of other liberal blog posts that rank in the top k and divide this number by k. We proceed similarly for each vertex representing a conservative blog post.

noise vertices from $\hat{\mathbf{X}}_2$. Call the cleaned (unprojected) data $\hat{\mathbf{X}}_{2,t}$. The input and output of the 2-means clustering is plotted in figure 2.8.

ii. Cluster the embedding of G_1 (call it $\hat{\mathbf{X}}_1$) into 2 clusters and use orthogonal Procrustes to align these cluster centers to those obtained in the robust 2-means clustering of $\hat{\mathbf{X}}_2$.

iii. Use `MClust` to cluster the combined data (the aligned $\hat{\mathbf{X}}_1$ and $\hat{\mathbf{X}}_{2,t}$) and rank matches based on our Mahalanobis distance VN procure.

In Figure 2.9 we plot (in black) the precision at k of our ranking scheme where we consider a positive match for precision purposes as follows: when nominating matches for a liberal (resp., conservative) blog, we consider a nominated blog of the same political persuasion as a positive match. Chance precision is plotted in red. In the figure, the precision at k is averaged over all liberal (resp., conservative) blogs in the left (resp., right) panel, and we consider $k \in \{1, ..., 100\}$. As can be seen on Figure 2.9, we observe that our algorithm is performing well, and highly exceeds that of chance when considering both liberal and conservative vertices.

2.5 Discussion and Future Work

Adversarial attacks on networks are phenomena that unfortunately happen frequently and thus, trimming methods that help mitigate the affect of the contamination are much sought after. After presenting the adversarial model of [3], we introduced a novel trimming method, where the trimming happens in model space, rather than graph space. We extended this procedure to a setting where we might have two types of noise: structured and unstructured (white noise). Here, we further introduce a robust kmeans algorithm, where a fixed maximum cluster radius is used to gather points into clusters and "clean out" (ideally) the noise points, whose distance from any cluster center is greater than the maximum radius. A tuning parameter λ helps with this process, by requiring the payment of a higher penalty in the objective function for the points that do not lie in any of the clusters. Experiments show that the two cleaning methods combine seamlessly and succeed in retrieving adequate information of the contaminated graph in both simulated and real-life data.

The above exploration is only the tip of the iceberg in an area where a lot of research can and should be done. The Stochastic Block Model is a simple, yet rich model to work with, but theory can be developed for more general structured noise settings. A natural next step would be exploring regularization methods in other, more complex contaminated models. In the latter case, an option might be to provide a few different robust algorithms, created precisely to be implemented in different contexts, and see which one gives the best results. Note that the regularization methods mentioned above act globally on the graph, but one can also consider local regularizers, which is another open area of research. For contamination procedures that act locally, regularization methods performed at a local level might prove to be effective in many

settings. Another path to explore is lifting the above problem to higher dimensions, including multilayered graphs and time series. In this case, tensors are a natural framework to work in and provide the relevant theory that will support a thorough exploration of the subject.

2.6 Supporting results and pseudocode

Herein we collect the supporting results and proofs.

2.6.1 Consistency of block probability estimates

In the theory below, we will make extensive use of the result, Theorem 3, from [48], which states that if both second moment matrices ($\mathbb{E}(\mathbf{X}I_{p,q}\mathbf{X}^T)$ for $\mathbf{X} \sim F$ or $\mathbf{X} \sim F^{(c)}$) are full rank, then there exists a sequence of indefinite orthogonal matrices $\mathbf{Q}_n \in \mathcal{O}(p,q)$ (so that $\mathbf{Q}_n$ satisfies $\mathbf{Q}_n^T I_{p,q}\mathbf{Q}_n = I_{p,q}$) and a universal constant $\alpha > 0$ such that if the sparsity factor $\nu_n = \omega\left(\frac{\log^{4\alpha} n}{n}\right)$, then

$$\max_{i=1,2,\ldots,n} \|\mathbf{Q}_n \hat{X}_i - \nu_n^{1/2} X_i\| = O_{\mathbb{P}}\left(\frac{\log^{\alpha} n}{n^{1/2}}\right) \tag{2.3}$$

The constant α appearing in Eq. 4.3 will be used throughout the appendix and proofs contained therein.

Let the true latent position matrix for $\mathbf{B}$ be denoted $\boldsymbol{\mu} \in \mathbb{R}^{K_1 \times d_1}$ and of $\mathbf{B}^{(c)}$ be denoted $\boldsymbol{\mu}^{(c)} \in \mathbb{R}^{K_2 \times d_2}$. Let the distribution over latent positions for the uncontaminated (resp., contaminated) network be denoted

$$F = \sum_{i=1}^{K_1} \pi_i \delta_{\mu_i} \text{ (resp., } F^{(c)} = \sum_{i=1}^{K_2} \pi_i^{(c)} \delta_{\mu_i^{(c)}}).$$

Suppose further that $\min_i \pi_i = \Theta(1)$ and similarly for $\min_i \pi_i^{(c)}$. We further assume that the

latent positions in the uncontaminated model (and in the contaminated model) satisfy (for $\alpha > 0$ defined in Eq. 4.3)

$$|\mu_i - \mu_j| = \omega\left(\frac{\log^\alpha n}{n^{1/2}}\right).$$

Our proof will proceed with a MSE-based (Mean Square Error-based) clustering heuristic rather than the (more difficult to analyze) GMM-based clustering in our algorithm. Note that the GMM-based clustering is more appropriate in the current setting [48], and often achieves superior performance in application [61]. The MSE clustering of the rows of $\hat{\mathbf{X}}$ into K clusters provides

$$\hat{\mathbf{C}} = \min_{\mathbf{C}\in\mathcal{C}_K} \|\mathbf{C} - \hat{\mathbf{X}}\|_F, \text{ where}$$

$$\mathcal{C}_K := \{\mathbf{C} \in \mathbb{R}^{n\times d} : \mathbf{C} \text{ has K distinct rows}\},$$

as the optimal cluster centroids for the K clusters.

As mentioned in the main text, we claim that $\hat{\mathbf{B}}$ and $\hat{\mathbf{B}}^{(c)}$ are suitable estimates of our original block membership matrices $\mathbf{B}$ and $\mathbf{B}^{(c)}$. The following Theorem supports this claim.

Theorem 2.6.1. Let $\hat{\boldsymbol{\mu}}$ be the matrix of cluster centroids provided by the optimal MSE-based clustering of $\hat{\mathbf{X}}$ in the uncontaminated network (with $\hat{\boldsymbol{\mu}}^{(c)}$ defined analogously for the contaminated network). Given the block probability matrices $\mathbf{B}$ and $\mathbf{B}^{(c)}$ for the uncontaminated and contaminated networks, with assumptions on ν_n, π_i, $\pi_i^{(c)}$, μ_i and $\mu_i^{(c)}$ as above we have that

$$\|\underbrace{\hat{\boldsymbol{\mu}} I_{p_1,q_1} \hat{\boldsymbol{\mu}}^T}_{:=\hat{\mathbf{B}}} - \nu_n \mathbf{B}\|_F = O_{\mathbb{P}}\left(\frac{K_1 \log^\alpha n}{n}\right) \quad \|\underbrace{(\hat{\boldsymbol{\mu}}^{(c)}) I_{p_2,q_2} (\hat{\boldsymbol{\mu}}^{(c)})^T}_{:=\hat{\mathbf{B}}_c} - \nu_n \mathbf{B}^{(c)}\|_F = O_{\mathbb{P}}\left(\frac{K_2 \log^\alpha n}{n}\right)$$

Proof. The assumption on $\min_i \pi_i$ is sufficient to ensure that (via a simple Hoeffding's inequality)

$\min_i n_i = \min_i |\{v : b_v = i\}| = \Theta_{\mathbb{P}}(n)$. The assumption that

$$|\mu_i - \mu_j| = \omega\left(\frac{\log^{\alpha} n}{n^{1/2}}\right)$$

then ensures that the optimal MSE clustering yields consistent estimates of block memberships. To wit let the estimated cluster assignment vector for the the optimal MSE clustering routine be denoted $\hat{b}_v$, then $\min_{\tau \in S_K} |\{v \in [n] : \tau(b_v) \neq \hat{b}_v\}| = o_{\mathbb{P}}(1)$ (note that the proof is essentially identical to the proof of Theorem 2.6 in [50] and is thus omitted).

Adopting the notation of [95], we let (with $d = \text{rank}(\mathbf{B})$ assumed known)

$$\mathbf{A} = \hat{\mathbf{U}}\hat{\mathbf{\Lambda}}\hat{\mathbf{U}}^T + \hat{\mathbf{U}}_{\perp}\hat{\mathbf{\Lambda}}_{\perp}\hat{\mathbf{U}}_{\perp}^T$$

be the eigendecompostion of $\mathbf{A}$, where the columns of $\hat{\mathbf{U}} \in \mathbb{R}^{n \times d}$ are the eigenvectors associated with the d largest eigenvalues in modulus of $\mathbf{A}$. Let the associated eigendecompostion of $\mathbf{P} := \nu_n \mathbf{X} I_{p_1,q_1} \mathbf{X}^T$ (where $\nu_n^{1/2}\mathbf{X} \in \mathbb{R}^{n \times d}$ matrix of latent positions of $\mathbf{A}$) be given by

$$\nu_n \mathbf{X} I_{p_1,q_1} \mathbf{X}^T = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$$

Define also,

$$\Pi_{\mathbf{U}} = \mathbf{U}\mathbf{U}^T; \quad \Pi_{\mathbf{U}}^{\perp} = \mathbf{I}_n - \Pi_{\mathbf{U};};$$

$$\mathbf{P}^{\dagger} = \mathbf{U}\mathbf{\Lambda}^{-1}\mathbf{U}^T; \quad \mathbf{E} = \mathbf{A} - \mathbf{P}.$$

For each $k \in K$, let the vector of cluster membership for class k in $\mathbf{Y}$ be denoted via $\mathbf{s}_k$, so that

$$\mathbf{s}_k(i) = \mathbb{1}\{\mathbf{X}_i \text{ is in cluster } k\}.$$

Similarly, for each $k \in K$, let the vector of the estimated cluster membership for class k obtained by clustering $\hat{\mathbf{X}}$ be denoted via $\hat{\mathbf{s}}_k$; with this notation $\xi_{1,k} = \frac{1}{\hat{n}_k}\hat{\mathbf{s}}_k^T\hat{\mathbf{X}}$.

The proof next proceeds with the following decomposition, adapted here from Equation (A.5) in [95],

$$\begin{aligned}
&\frac{n}{\nu_n^{1/2}}(\hat{\mathbf{B}}_{kl} - \nu_n \mathbf{B}_{kl}) \\
&\quad = \frac{n}{n_k n_l \nu_n^{1/2}}(\mathbf{s}_k^T \mathbf{E}\mathbf{\Pi}_{\mathbf{U}}\mathbf{s}_l + \mathbf{s}_l^T \mathbf{\Pi}_{\mathbf{U}}^{\perp}\mathbf{E}\mathbf{\Pi}_{\mathbf{U}}\mathbf{s}_k) \qquad (2.4) \\
&\quad + \frac{n}{n_k n_l \nu_n^{1/2}}(\mathbf{s}_k^T \mathbf{\Pi}_{\mathbf{U}}^{\perp}\mathbf{E}^2\mathbf{P}^{\dagger}\mathbf{s}_l + \mathbf{s}_l^T \mathbf{\Pi}_{\mathbf{U}}^{\perp}\mathbf{E}^2\mathbf{P}^{\dagger}\mathbf{s}_k) \qquad (2.5) \\
&\quad + O_{\mathbb{P}}(n^{-1/2}\nu_n^{-1})
\end{aligned}$$

We have from [95] that conditional on $\mathbf{P}$, $\nu_n^{1/2}$ times Eq. 2.5 converges a.s. to a constant. Therefore $\frac{\nu_n^{1/2}}{\log^{\alpha} n}$ times Eq. 2.5 is $o_{\mathbb{P}}(1)$. From Equation A.24 in [95], we have that conditional on $\mathbf{P}$, $\mathbf{s}_k^T \mathbf{E}\mathbf{\Pi}_{\mathbf{U}}\mathbf{s}_l + \mathbf{s}_l^T \mathbf{\Pi}_{\mathbf{U}}^{\perp}\mathbf{E}\mathbf{\Pi}_{\mathbf{U}}\mathbf{s}_k$ is the sum of $n(n+1)/2$ independent mean 0 random variables with bounded variance (all variances bounded by d for example). A simple application of Markov's inequality implies that for any $\epsilon > 0$,

$$\begin{aligned}
\mathbb{P}\left(\frac{\nu_n^{1/2}}{\log^{\alpha} n}\frac{n}{n_k n_l \nu_n^{1/2}}|\mathbf{s}_k^T \mathbf{E}\mathbf{\Pi}_{\mathbf{U}}\mathbf{s}_l + \mathbf{s}_l^T \mathbf{\Pi}_{\mathbf{U}}^{\perp}\mathbf{E}\mathbf{\Pi}_{\mathbf{U}}\mathbf{s}_k| > \epsilon\right) &\leq \frac{O(n(n+1)/2)}{\frac{\log^{2\alpha} n}{\nu_n}\frac{n_k^2 n_l^2 \nu_n}{n^2}\epsilon^2} \\
&= O_{\mathbb{P}}\left(\frac{1}{\log^{2\alpha} n}\right) = o_{\mathbb{P}}(1)
\end{aligned}$$

We then have that

$$\frac{\nu_n^{1/2}}{\log^\alpha n}\left(\frac{n}{\nu_n^{1/2}}(\hat{\mathbf{B}}_{kl}-\nu_n\mathbf{B}_{kl})\right)=\frac{n}{\log^\alpha n}(\hat{\mathbf{B}}_{kl}-\nu_n\mathbf{B}_{kl})=o_{\mathbb{P}}(1)+\underbrace{O_{\mathbb{P}}\left(\frac{n^{-1/2}}{\nu_n^{1/2}\log^\alpha n}\right)}_{o_{\mathbb{P}}(1)}.$$

This implies then that

$$|\hat{\mathbf{B}}_{kl}-\nu_n\mathbf{B}_{kl}|=o_{\mathbb{P}}\left(\frac{\log^\alpha n}{n}\right)$$

A union bound over all the entries of $\mathbf{B}$ completes the proof. The proof for $\hat{\mathbf{B}}^{(c)}$ is analogous. ■

Now that we have proven the above claim, we want to use the estimates $\hat{\mathbf{B}}$ and $\hat{\mathbf{B}}^{(c)}$ to trim the contaminated vertices in G_2. We can accomplish this by solving the following graph matching type problem. For each $a, b \in \mathbb{Z} > 0$, define

$$\Pi_{a,b}=\{P\in\{0,1\}^{a\times b}: P\vec{1}_b=\vec{1}_a,\ \vec{1}_a^T P\leq\vec{1}_b\}.$$

Note that there exists a $P^*\in\Pi_{K,3K}$ such that

$$\|\mathbf{B}-P^*\mathbf{B}^{(c)}(P^*)^T\|_F=0.$$

We then seek to show that with high probability,

$$P^*=\text{argmin}_{P\in\Pi_{K,3K}}\|\hat{\mathbf{B}}-P\hat{\mathbf{B}}^{(c)}P^T\|_F^2. \tag{2.6}$$

This would imply that P^* recovers the correspondence between the original, non-contaminated blocks in $\hat{\mathbf{B}}$ and the uncontaminated portion of $\hat{\mathbf{B}}^{(c)}$, with the remaining blocks (which ideally

capture the contamination) trimmed by our procedure. This is formalized in the following result:

Theorem 2.6.2. Let $\mathbf{B}$ and $\mathbf{B}^{(c)}$ be such that

$$\min_{P \in \Pi_{K,3K} \setminus \{P^*\}} \|\mathbf{B} - P\mathbf{B}^{(c)}P^T\|_F = \Omega(1)$$

For K fixed, we then have

$$\mathbb{P}\left(P^* = \text{argmin}_{P \in \Pi_{K,3K}} \|\hat{\mathbf{B}} - P\hat{\mathbf{B}}^{(c)}P^T\|_F\right) \to 1$$

Proof. Recall that in our model, the dimensions of $\mathbf{B}$ and $\mathbf{B}^{(c)}$ are respectively $K \times K$ and $3K \times 3K$. Then, from Theorem 2.6.1, we have that:

$$\|\hat{\mathbf{B}} - \nu_n \mathbf{B}\|_F = O_{\mathbb{P}}\left(\frac{K \log^{\alpha} n}{n}\right) \tag{2.7}$$

$$\|\hat{\mathbf{B}}^{(c)} - \nu_n \mathbf{B}^{(c)}\|_F = O_{\mathbb{P}}\left(\frac{K \log^{\alpha} n}{n}\right) \tag{2.8}$$

With P^* such that $\|\mathbf{B} - P^*\mathbf{B}^{(c)}(P^*)^T\|_F = 0$, we then have the following:

$$\begin{aligned}
\|\hat{\mathbf{B}} - P^*\hat{\mathbf{B}}^{(c)}(P^*)^T\|_F &= \|\hat{\mathbf{B}} - \nu_n P^*\mathbf{B}^{(c)}(P^*)^T + \nu_n P^*\mathbf{B}^{(c)}(P^*)^T - P^*\hat{\mathbf{B}}^{(c)}(P^*)^T - \nu_n\mathbf{B} + \nu_n\mathbf{B}\| \\
&\leq \|\hat{\mathbf{B}} - \nu_n\mathbf{B}\|_F + \|P^*\hat{\mathbf{B}}^{(c)}(P^*)^T - \nu_n P^*\mathbf{B}^{(c)}(P^*)^T\|_F \\
&\quad + \|\nu_n\mathbf{B} - \nu_n P^*\mathbf{B}^{(c)}(P^*)^T\| \\
&= O_{\mathbb{P}}\left(\frac{K \log^{\alpha} n}{n}\right)
\end{aligned}$$

Consider $P \in \Pi_{K,3K}$ satisfying $P \neq P^*$. Then the Assumptions A1 and A2 are sufficient to guarantee

$$\|\hat{\mathbf{B}} - P\hat{\mathbf{B}}^{(c)}P^T\|_F = \Omega(\nu_n) = \omega\left(\frac{K\log^{4\alpha} n}{n}\right).$$

For fixed K, the probability that $\|\hat{\mathbf{B}} - P\hat{\mathbf{B}}^{(c)}P^T\|_F > \|\hat{\mathbf{B}} - P^*\hat{\mathbf{B}}^{(c)}(P^*)^T\|_F$ converges to 1. A union over the (finitely many) $P \in \Pi_{K,3K} \setminus P^*$ then yields the desired result. ■

Remark 2.6.1. The condition that $\min_{P\in\Pi_{K,3K}\setminus\{P^*\}} \|\mathbf{B} - P\mathbf{B}^{(c)}P^T\|_F = \Omega(1)$ is implied by either one of the two following two sets of conditions holding (where we are considering the parametrization for the edge addition/deletion probabilities as $\nu_n s_+$ and $\nu_n s_-$ respectively):

$$\min_{i,j\,:\,i\neq j} |\mathbf{B}_{i,i} - \mathbf{B}_{j,j}| = \Omega(1) \tag{A1.1}$$

$$\min_{i,j} |\mathbf{B}_{i,i} - \mathbf{B}_{j,j} - s_+(1 - \mathbf{B}_{j,j})| = \Omega(1) \tag{A1.2}$$

$$\min_{i,j} |\mathbf{B}_{i,i} - \mathbf{B}_{j,j}(1 - s_-)| = \Omega(1) \tag{A1.3}$$

or

$$\min_{i,j,k,\ell\,:\,i\neq j,k\neq\ell,\{i,j\}\neq\{k,\ell\}} |\mathbf{B}_{ij} - \mathbf{B}_{k\ell}| = \Omega(1) \tag{A2.1}$$

$$\min_{i,j,k,\ell\,:\,i\neq j,k\neq\ell} |\mathbf{B}_{ij} - \mathbf{B}_{k\ell} - s_+(1 - \mathbf{B}_{k\ell})| = \Omega(1) \tag{A2.2}$$

$$\min_{i,j,k,\ell\,:\,i\neq j,k\neq\ell} |\mathbf{B}_{ij} - \mathbf{B}_{k\ell}(1 - s_-)| = \Omega(1) \tag{A2.3}$$

The first set of conditions would ensure sufficient disagreement on the matched diagonal of $\|\mathbf{B} - P\mathbf{B}^{(c)}P^T\|_F$ to guarantee the growth rate on $\min_{P\in\Pi_{K,3K}\setminus\{P^*\}} \|\mathbf{B} - P\mathbf{B}^{(c)}P^T\|_F$, while the second

set of conditions guarantees sufficient error on the matched off-diagonal of $\|\mathbf{B} - P\mathbf{B}^{(c)}P^T\|_F$.

2.6.2 Consistency of robust K-means clustering

We first prove a consistency result in the dense setting (i.e., in the setting where $\nu_n = \theta(1)$), and then will outline how to adapt the results to sparser settings. In the dense setting, we will assume that the penalty parameter $\lambda = \lambda_{n,m}$ is bounded away from 0.

Assumption 1: *With* $\boldsymbol{X}$ *defined as in Section 2.3.3, let the* K *distinct rows of* $\boldsymbol{Y}$ *represent the latent position vectors in* $\mathbb{R}^d$ *for the signal blockmodel component of* G_2*. We assume that there exists a constant* $\eta > 0$ *such that if* Y_i *and* Y_j *are two distinct rows of* $\boldsymbol{Y}$*, then* $\|Y_i - Y_j\| > \eta$*. Also assume that* m *(the number of rows of* $\mathbf{Z}$*) satisfies* $m = o(n)$*.*

Again we use Theorem 3 from [48], which states that there exists a sequence of indefinite orthogonal matrices $\mathbf{Q}_n \in \mathcal{O}(p,q)$ (so that $\mathbf{Q}_n$ satisfies $\mathbf{Q}_n^T I_{p,q} \mathbf{Q}_n = I_{p,q}$) and constants $c, C > 0$ such that in this dense framework, the following holds with high probability for sufficiently large n

$$\max_{i=1,2,\ldots,n} \|\mathbf{Q}_n \hat{X}_i - \nu_n^{1/2} X_i\| := \epsilon_{n,m} \leq C \frac{\log^c(n+m)}{(n+m)^{1/2}} \tag{2.9}$$

Below, we will drop the subscript on $\epsilon_{n,m}$ (as well as on $\lambda_{n,m}$) to ease notation. Conditioning on the event in Eq. (2.9), consider the partition π such that $\pi_1 = \{1, 2, \cdots, n\}$ and $\pi_2 = \{n+1, n+2, \cdots, n+m\}$, and consider Φ composed of the K distinct rows of $\mathbf{Y}$. For this choice of π and

Φ, we have that

$$\Gamma(\Phi, \pi) \leq Cn\frac{\log^c(n+m)}{(n+m)^{1/2}} + \lambda m. \tag{2.10}$$

Next, we let $(\hat{\Phi}, \hat{\pi})$ be an element of the argmin of Eq.(2.2), and consider balls $\{\mathcal{B}_i\}_{i=1}^K$ of radius

$$r := \min\left(\frac{\lambda}{3}, \frac{\eta}{6}\right)$$

about the K distinct rows of $\mathbf{Y}$. By Assumption 1, these balls are disjoint. Under our assumption that the penalty λ is bounded away from 0, we have that $r + \epsilon < \lambda$ for n sufficiently large. For n sufficiently large, we then observe the following:

i. If a ball $\mathcal{B}_i$ contains no centers in $\hat{\Phi}$, then

$$\begin{aligned}\Gamma(\hat{\Phi}, \hat{\pi}) \geq & \min_{y \in \{\mathfrak{z}^i\}_{i=1}^k} \left(\left|\{j : Y_j = y \text{ and } j \in \hat{\pi}_1\}\right| \cdot (r - \epsilon) + \left|\{j : Y_j = y \text{ and } j \in \hat{\pi}_2\}\right| \cdot \lambda\right) \\ \geq & \min_i n_i \cdot \min(r - \epsilon, \lambda) \\ = & \Omega_P\left(n \cdot (\lambda - \epsilon)\right)\end{aligned}$$

where the final equality holds with high probability as each n_i has a Binomial distribution with parameters n and $\pi_i > 0$. This yields the desired contradiction as this asymptotically dominates Eq.(2.10) for n sufficiently large.

ii. Note that if a ball $\mathcal{B}_i$ contains two or more centers of $\hat{\Phi}$, then there is at least one ball with no cluster centers and the desired contradiction follows from i. above.

Observations i. and ii. above yield that each $\mathcal{B}_i$ contains exactly one of the centers in $\hat{\Phi}$. This

yields then that all of the signal vertices in the ASE are properly clustered according to $(\hat{\Phi}, \hat{\pi})$, as signal points will not be unclustered (as this induces a penalty of λ in $\Gamma(\hat{\Phi}, \hat{\pi})$ while assigning the signal point to the cluster center closest to its true latent position induces a penalty of at most $r + \epsilon < \lambda$) or misclustered (as this induces a penalty of at least $\eta - 2r > 2\eta/3$ in $\Gamma(\hat{\Phi}, \hat{\pi})$ while assigning the signal point to the cluster center closest to its true latent position induces a penalty of at most $r + \epsilon < 2\eta/3$).

Next, we ask how many noise points are assigned (incorrectly) a cluster label. Noise points will only be assigned a label if they are within λ of a cluster center (and hence, their true latent position is within $\lambda + r + 2\epsilon$ of a true signal latent position). This probability is bounded above by (where $\Gamma(\cdot)$ is the usual Γ function and K is a universal constant)

$$\mathfrak{p} = \frac{K \cdot (\lambda + r + 2\epsilon)^d \pi^{d/2} / \Gamma(d/2 + 1)}{\frac{1}{2^d} \pi^{d/2} / \Gamma(d/2 + 1)} = K \cdot (2\lambda + 2r + 4\epsilon)^d.$$

A simple application of Hoeffding's inequality yields that, with high probability, at most

$$O(m \cdot K \cdot (8\lambda/3)^d)$$

(and hence $o(n)$) noise points fall within $\lambda + r + 2\epsilon$ of a true signal latent position, and hence are assigned a cluster label.

Remark 2.6.2. *In the sparse setting all the results above still hold, the only difference being that all parameters are scaled by $\sqrt{\nu}$. Indeed, the above argument follows mutatis mutandis (where*

the 's' subscript denotes the sparse parameter and a '$\mathfrak{d}$' subscript the dense) with:

$$\lambda_s = \sqrt{\nu}\lambda_{\mathfrak{d}}$$

$$r_s = \sqrt{\nu}r_{\mathfrak{d}}$$

$$\eta_s = \sqrt{\nu}\eta_{\mathfrak{d}}$$

Note that ϵ is defined the same way in the sparse and dense settings, and that here $\lambda_s \gg \epsilon$ (assuming $\lambda_{\mathfrak{d}}$ is bounded away from 0) as

$$\epsilon_{n,m} = O_P\left(\frac{\log^c(n+m)}{(n+m)^{1/2}}\right)$$

and

$$\nu_n = \omega\left(\frac{\log^{4c} n}{n}\right).$$

Chapter 3: Learning to rank with human-in-the-loop

3.1 Novel contribution

In this Chapter, we consider another setting in which the anomaly can be thought of as inhibiting graph inference. Specifically, we consider the task of ranking objects relative to a given query. Our work builds on [75]'s work in which an integer linear programming (ILP) learning-to-rank framework is introduced. The approach uses knowledge of some given dissimilarity measures, relative to a query. These are leveraged to learn an approximately optimal linear combination of these dissimilarities for discovering new data points near to the query. Originally, a set S of objects known to be similar to the query is assumed to be provided. Our work here involves introducing a set T of objects known to be dissimilar to the query, modifying the ILP to account for the T set, and adding a user-in-the-loop to the algorithm. These are shown to increase algorithmic performance, especially in settings where there are adversarial/anomalous data (e.g., errors in training data and diffuse signal in any one data modality). We also uncover a novel relationship between the ILP learning to rank setting and more classical maximum margin classifiers (SVMs); see Section 5.2.1.

3.2 Learning to rank with human-in-the-loop

The above vertex nomination task can be seen as a graph-based example of the more general task of ranking objects relative to a given query. This more general task has numerous important applications, such as identifying potential human traffickers [64], identifying users across social networks [57] or improving search engines [3, 96, 97]. Often, in order to rank the given objects relative to the query, a domain specific notion of similarity needs to be defined or computed, and this is not always an easy task [75, 98]. In some settings a notion of similarity is provided a priori and the ranking straightforward, while in other settings the definition of similarity is poorly defined and needs to be learned using supervisory information or data [99, 100]. The latter usually requires a significant amount of data. One possible solution for this problem is to leverage pre-existing models or representations in the similarity learning task, that have proven to be useful for similar downstream inference tasks. In settings where resources are scarce, this has proven to be successful, including transfer learning [101], domain adaptation [102], learning-to-rank [103] and continual learning [104].

In this text, we extend the integer linear programming (ILP) learning-to-rank framework proposed in [75], in which knowledge of some given dissimilarity measures, relative to a query, are leveraged to learn an approximately optimal linear combination of these dissimilarities for discovering new data points near to the query. This new dissimilarity measure is customized to the query under consideration and does not necessarily imply a generalization to other settings. Informally, the ILP proceeds as follows. Letting v^* be our query vertex (the setting where the query set is composed of multiple vertices will be addressed in Section 3.3.1), we posit an a priori unknown oracle dissimilarity Δ^*—oracle in that if $\Delta^*(v^*, v)$ is small (resp., large) then v is

(resp., is not) a vertex of interest in the graph; see Definition 16 for the definition of dissimilarity used in the sequel. A set S of objects known to be similar (under Δ^*) to the query is assumed to be provided. Given a collection of perhaps disparate dissimilarities $\Delta_1, \cdots, \Delta_k$, the goal is to discover an "optimal" linear combination of the Δ_i's that well approximates Δ^*. This is achieved in the ILP framework by ranking the elements of S as highly as possible in the combined measure—here a linear combination of the $\Delta_1, \cdots, \Delta_k$.

We extend the above setting to one in which in addition to the S set, we are given a set T of objects known to be dissimilar under Δ^* to the query under consideration. Our task now is to learn an approximately optimal linear combination of the dissimilarities provided which will simultaneously rank elements of S as high as possible in the ranked list and elements of T relatively lower in the same list. Along the way, we discovered that the formulation of this newly-introduced extension is structurally similar to the classical Support Vector Machine (SVM) [105–107]. Therefore, in addition to the ILP, we consider a hybridized ILP-SVM approach to solve the problem at hand, obtaining better results, while being more computationally efficient. Finally, we introduce a human-in-the-loop (HitL) to both ranking settings (ILP and ILP-SVM). At each iteration in the HitL framework, the user is provided with the top k elements of the ranked list and is asked to return information on whether each object is similar or dissimilar to the query at hand. Given this information, each of the top k elements in the list is then added to the S or T set in the next iteration (depending on whether the object is judged similar / dissimilar to the query). This process is continued until certain stopping criteria are met. Both the addition of the T set and the introduction of the human-in-the-loop to the setting empirically (in both simulations and real data experiments) increase algorithmic performance and provide an improved ranking. We support our claims with simulations and real data experiments.

Remark 3.2.1. Instead of introducing the T set to the above algorithm, one could consider using Schroedinger Eigenmaps, a semi-supervised manifold learning and recovery technique [108], as a dissimilarity measure to learn our optimal linear combination of dissimiliarites [109–111] that can incorporate both the information in S and in T using well constructed barrier potentials. Instead of a Laplacian, we could use a Schroedinger map with some potential parameter μ. We can then proceed the same way to introduce a human-in-the-loop (HitL). We are currently exploring this as an alternative to the T and S set formalization.

3.2.1 Vertex nomination and the ILP recommender system

In our motivating work in [75], the authors consider the problem of creating a recommender system in the presence of light supervision provided by S; see Section 3.3.1 for full detail. The final output of the ILP problem is a ranking of the items under consideration, with the items most similar (according to Δ^*) to the query ideally being placed near the top of the rank list. When applied to ranking the vertices of graphs, this ILP is an example of a *Vertex Nomination* (VN) learning system, and herein, we consider the ILP recommender system problem applied in the setting of VN [2, 65–68]. Informally, the VN problem in this context can be stated as follows: Given vertices of interest in a graph (provided here by v^* and S in the ILP), rank the remaining vertices in the graph so that previously unknown vertices of interest concentrate near the top of the rank list. In recent years, various different approaches to VN have been presented. Examples include model-based approaches where the vertices of interest are posited to be in a common community in a stochastic blockmodel [65, 70, 112] or vertices with similar latent positions in a random dot product graph [68, 74]. Another line of approaches considers solving the VN problem

via learned dissimilarities in spectral and other embeddings of the networks [3, 58, 113].

Before discussing the ILP in the context of VN, we will first formally define the notion of dissimilarity used in the sequel.

Definition 16. A dissimilarity on a set V is a function $\Delta : V \times V \mapsto \mathbb{R}^{\geq 0}$ such that

i. For all $v \in V$, $\Delta(v, v) = 0$;

ii. For all $u, v \in V$, $\Delta(u, v) = \Delta(v, u)$;

ii. For $u, v, w \in V$, if $\Delta(u, v) < \Delta(w, v)$ then u is more similar (according to Δ) to v than w.

Note that Δ does not need to satisfy the triangle inequality or non-negativity requirements of a metric. Let the set of all dissimilarities on $V \times V$ be denoted $\mathcal{D}_V$.

In the ILP problem cast in the setting of VN, the collection of items under consideration can be thought of as vertices of a graph $G = (V, E)$, the query as the vertex of interest v^* and the set S as a set of vertices known to be interesting (similar to v^*). The goal in [75], is to produce a VN scheme by solving an integer linear program that weights the provided dissimilarity measures $\Delta_1, \cdots, \Delta_k$ in such a way that the vertices in S are ranked highly in the list. This learned dissimilarity is then used to rank all other vertices, and the information in S can be thought of as a form of additional light supervision. For the learned dissimilarity measure $\Delta : V \times V \to \mathbb{R} \geq 0$, a natural ranking function to consider is one that returns the rank of the real number $\Delta(v^*, v)$ amongst the collection of $\{\Delta(v^*, v')\}_{v' \neq v^*}$, given a vertex $v \neq v^*$. More specifically, the vertex $v \in (V \backslash \{v^*\})$ that minimizes $\Delta(v^*, v)$ is mapped to 1—the top of the nomination list—and the vertex most dissimilar the vertex of interest v^* is mapped to $n - 1$. We denote this mapping from the vertex set to the set $[n - 1]$ for the dissimilarity Δ by h^{Δ}.

The ILP recommender system nomination scheme of [75] takes as input $(G, v^*, S, \{\Delta_i\}_{i=1}^k)$ and outputs a function that maps each vertex in $V\backslash\{v^*\}$ to an element of the set $\{1, ..., n-1\}$. In the set of unlabeled vertices, let $S^* \subset V\backslash\{v^*\}$ be the set of vertices that are truly similar to v^* according to the oracle dissimilarity Δ^* (i.e., those ranked above a certain threshold ranking by Δ^*). Here $S \subset S^*$ is the supervisory set and $S^*\backslash S$ the set of similar vertices that are unknown, and that we hope to rank high in our final nomination list. In addition, we define

$$\mathcal{H} = \{h : V\backslash\{v^*\} \to \{1, ..., n-1\}\}$$

to be the set of ranking functions that map a vertex $v \in V \setminus \{v^*\}$ to an element of the set $\{1, ..., n-1\}$ and we let $\mathcal{G}_V$ be the set of graphs with vertex set V. We can then more formally define a *S-supervised ILP nomination scheme* as a mapping

$$f : \mathcal{G}_V \times V \times 2^V \times \mathcal{D}_V^k \to \mathcal{H},$$

where our goal is to find an f such that $f(G, v^*, S, \{\Delta_i\}_{i=1}^k)$ outputs small values (i.e., ranks highly) for elements of S^*. Letting $\mathcal{N}_S$ be the set of S-supervised ILP nomination schemes, we lastly define the ranking associated with each $f \in \mathcal{N}_S$. Letting $C = (V\backslash(\{v^*\} \cup S))$ be the candidate set (where $(S^*\backslash S) \subset C$), we define the rank function of f as follows. For a list of positive integers U of length m, $\text{rk}(U)$ returns the relative rank (smallest-to-largest) of each element in the list; for example

$$\text{rk}(8, 12, 4, 2, 11) = (3, 5, 2, 1, 4).$$

We then define

$$r_f(G, v^*, S, \{\Delta_i\}_{i=1}^k) = \text{rk}\left(f(G, v^*, S, \{\Delta_i\}_{i=1}^k) \restriction_C\right);$$

i.e., r_f returns the ranks of the candidate nodes in C.

3.3 Extension to T set

In [75] and other proposed VN schemes in the literature [64,70,112,114], the only information being used to determine the optimal dissimilarity measure are the vertices in V^* or S, which are known to be similar to the vertex of interest v^*. We extend this setting to one in which, in addition to the vertices S, we are provided with vertices T that are known to be highly dissimilar to v^* according to Δ^*. Our motivation in doing so is many-fold: Ideally, this T set will allow us a better understanding of the similarity space, which could be useful for learning a more complex dissimilarity structures defined by Δ^* (e.g., disjoint, non-convex, etc.). We additionally hope that this extra information will help 'pull down' the vertices that are similar to elements in T (and therefore dissimilar to v^*) towards the end of the ranked list. This, in turn, we expect, will produce a more optimal learned dissimilarity combination and hence a better ranked list. In the course of introducing the T set into the ILP framework, we note that the new ILP problem formulation is strikingly similar to a Support Vector Machine (SVM), and therefore we also explore using a variant of a SVM to solve our problem, in addition to the ILP. The SVM-inspired variant of the ILP achieves better computational efficiency, and no worse (and sometimes better) performance. Below, we present the extended setting (with the T set) and also provide a brief description of the Integer Linear Programming (ILP) method used. We conclude the section by describing how to use the SVM method to solve the problem at hand.

3.3.1 The T set in the ILP setting

Let $G = (V, E)$ be a graph, where $V = [n] = \{1, ..., n\}$ is the set of vertices and $E \subset \binom{V}{2}$ is the set of edges. Suppose we are given a vertex of interest, v^*, and two sets $S \subset (V \backslash \{v^*\})$ and $T \subset (V \backslash \{v^* \cup S\})$, which consist of vertices that are known to be similar and dissimilar to v^* respectively. The goal in the ILP framework is to *learn* a ranking scheme from the sets S and T, ideally ranking previously unknown vertices in V similar to s^* (according to Δ^*) high in the rank list. We next develop a scheme f (similar to the scheme in [75]) that takes as input $(G, v^*, S, T, \{\Delta_i\}_{i=1}^k)$—a graph, a vertex of interest, a set of vertices known to be similar to the vertex of interest, a set of vertices known to be dissimilar to the vertex of interest and a collection of dissimilarities—and outputs a function that maps each vertex in $\{V \backslash v^*\}$ to an element of $[n-1]$.

Recall that in the set of unlabeled vertices, $S^* \subset V \backslash \{v^*\}$ is the set of vertices that are truly similar to v^* according to the oracle dissimilarity Δ^* (i.e., those ranked above a certain threshold ranking by Δ^*). Similarly, let $T^* \subset V \backslash \{v^*, S^*\}$ be the set of vertices that are highly dissimilar to v^* according to the oracle dissimilarity Δ^* (i.e., those ranked below a certain threshold ranking by Δ^*). Suppose that $S \subset S^*$ and $T \subset T^*$ are subsets of the set of all vertices that are truly similar and truly dissimilar. We then define the *S,T-supervised ILP nomination scheme* as a mapping

$$f : \mathcal{G}_V \times V \times 2^V \times 2^V \times \mathcal{D}_V^k \to \mathcal{H},$$

where our goal is to find an f such that $f(G, v^*, S, T, \{\Delta_i\}_{i=1}^k)$ outputs small values (i.e., ranks highly) for elements of S^* and large values for elements of T^*. As the solution of the optimization

problem in [75] proves to be useful for learning to rank in general, and for settings of supervised vertex nomination in particular, we first present a modification of the ILP VN scheme of [75] to encompass the T set.

Let $\{v_1, ..., v_n\}$ be the vertices in our graph and without loss of generality, let $v_1 = v^*$. Let $\{\Delta_1, ..., \Delta_k\}$ be a collection of k distinct dissimilarity measures and assume we are given the dissimilarity between v_1 and v_i, $i = 2, ..., n$ for each of these measures. Let $d_{ji} = \Delta_j(v_1, v_i)$ denote the dissimilarity between v_1 and v_i in the j-th dissimilarity, where $j = 1, ..., k$. Informally, we want to choose an appropriate weighted combination of the k dissimilarities that ranks the given set $S \subset \{v_2, ..., v_n\}$ highly and the given set $T \subset \{v_2, ..., v_n\}$ lower in our ranked list. To wit, we want to choose a set of weights $\alpha_1, ..., \alpha_k \geq 0$ such that $\sum_j \alpha_j d_{ji}$ will be as relatively small as possible for $v_i \in S$ (i.e. $v_i \in S$ will be as close to the top of the ranked list) and as relatively large as possible for $v_i \in T$ (i.e. $v_i \in T$ will be as close to the bottom of the ranked list). This is achieved by here optimizing the following expression (where $f_\alpha := f_\alpha(G, v^*, S, T, \{\Delta_i\}_{i=1}^k)$ is the ranking achieved by ordering vertices by increasing value of $\sum_j \alpha_j d_{ji}$)

$$\min_{\alpha \geq 0} \left(C_1 \max \{\text{rk}\left(f_\alpha \restriction_S\right)\} + C_2 \max \{n - \text{rk}\left(f_\alpha \restriction_T\right)\} \right) \tag{3.1}$$

where $\alpha = (\alpha_1, ..., \alpha_k)$ is a given tuple of weights. In order to solve the above optimization problem, we can use the *integer linear programming* (ILP) framework of [75], which we present below.

Consider real valued decision variables/weights $\alpha_1, ..., \alpha_k$ that are constrained to be nonnegative (these are the weights we are looking for in Eq. 3.1). As the solution to Eq. 3.1 does not change when scaling the weights by the same positive factor, we impose the following normalization

constraint: $\alpha_1 + ... + \alpha_k = 1$. We define two integer variables x_v and y_v for each $v \in C$ and impose the linear constraints $0 \leq x_v \leq 1$ and $0 \leq y_v \leq 1$ (as x_v and y_v are constrained to be integral, this effectively forces $x_v, y_v \in \{0, 1\}$ to maintain feasibility). These added slack variables capture the following behaviors:

- If $x_v = 1$ and $y_v = 0$, then v is ranked better than at least one element in S (under α) and better than every element of T (under α).

- If $x_v = 0$ and $y_v = 1$, then v is ranked worse than at least one element in T, and worse than every element of S (under α).

- If $x_v = 0$ and $y_v = 0$, then v is ranked better than every element in T (under α) and worse than every element in S (under α).

We then wish to minimize $\sum_{v \notin S} x_v$ and $\sum_{y \notin C \setminus \{S \cup T\}} y_v$; i.e., we seek

$$\min \Big(\sum_{v \notin \{S \cup T\}} x_v + \sum_{v \notin \{S \cup T\}} y_v \Big) \tag{3.2}$$

subject to the following: $x_v \in \{0, 1\}$ and $y_v \in \{0, 1\}$, and

$$\text{for all } v_i \in S, v_\ell \in V \setminus (S \cup T), \quad \sum_{j=1}^{k} \alpha_j d_{ji} \leq \sum_{j=1}^{k} \alpha_j d_{j\ell} + M x_{v_\ell} \tag{3.3}$$

$$\text{for all } v_i \in T, v_\ell \in V \setminus (S \cup T), \quad \sum_{j=1}^{k} \alpha_j dj\ell - M y_{v_\ell} \leq \sum_{j=1}^{k} \alpha_j d_{ji} \tag{3.4}$$

where $M := \max_{i,j} d_{ji}$.

Note that the above inequalities establish the following conditions. For any $v \in C$:

- If $x_v = 0$, then v should be ranked worse than every element in S. In other words, its dissimilarity under α from v^* should be greater than or equal to the dissimilarity of every element of S from v^*.

- If $x_v = 1$, then constraint Eq. 3.3 becomes trivial. This is due to the fact that as all Δ's are nonnegative, and we are considering values of α (≥ 0 entry-wise) such that $\sum_j \alpha_j = 1$. The inequality therefore holds trivially, given that M is defined as the maximum of all possible dissimilarities.

- If $y_v = 0$, then v should be ranked better than every element in T. In other words, its dissimilarity under α from v^* should be smaller than or equal to the dissimilarity of every element of T from v^*.

- Finally, if $y_v = 1$, then constraint Eq. 5.3 becomes trivial given a similar argument to the case when $x_v = 1$.

3.3.1.1 Multiple queries

Although the above model works well in various settings, oftentimes the data provided is extremely sparse and the ranking function produced might not be very helpful. In these cases, [98] suggests using information from multiple semantically similar queries (rather than from a single query), when available. In this setting, the model changes slightly, as now, instead a single (v^*, S, T) triplet, we have access to K triplets of queries, sets of elements known to be similar to the queries and sets of elements known to be dissimilar to the query, namely $\{(v^*, S, T) \cup \{q_k, S_k, T_k\}_{k=2}^K\}$. As before, we have access to J dissimilarity measures between the queries and the remaining vertices. Finally, the goal is the same as in the one query setting:

to produce an optimal ranking function relative to the query v^*. More specifically, consider a graph $G = (V, E)$ and let $\Delta_j : V \times V \rightarrow \mathcal{R}$ be dissimilarity measures corresponding to the j^{th} representation. As in the case of a single query, we define the parameters of the convex combinations by $\overrightarrow{\alpha} \in \mathbb{S}^{J-1}$ (the $J-1$ simplex), where we let the α_k correspond to the combinations related to q_k respectively. Also, for simplicity of notation below, let $q_1 := v^*$ and $S_1 := S$. Since we are now dealing with multiple triplets of queries and vertices that are known to be similar and dissimilar, we introduce the notion of probability distributions on triplets (q_k, S_k, T_k). We assume that $(q_k, S_k, T_k) \overset{i.i.d}{\sim} P_{(q,S,T)}$ for $k = 1, ..., K$, where $P_{(q,S,T)}$ is a probability distribution over (q, S, T). Note that in the case of multiple queries, we now have k functions $\{f_\alpha^1, ..., f_\alpha^k\}$, where $f_\alpha^1 := f_\alpha(G, v^*, S, T, \{\Delta_i\}_{i=1}^k)$ and for $\ell \in \{2, ..., k\}$, $f_\alpha^\ell := f_\alpha(G, q_\ell, S_\ell, T_\ell, \{\Delta_i\}_{i=1}^k)$ are the rankings achieved by ordering vertices by increasing value of $\sum_j \alpha_j d_{ji}^\ell$. We let the average optimal, unknown linear combination of dissimilarities for each k be:

$$\bar{\alpha} = \mathbb{E}_{(q,S,T)\sim P_{(q,S,T)}}\Big[\arg\min_{\overrightarrow{\alpha}}\Big(C_1 \max\{\text{rk}\,(f_\alpha \restriction_S)\} + C_2 \max\{n - \text{rk}\,(f_\alpha \restriction_T)\}\Big)\Big] \tag{3.5}$$

where C_1 and C_2 are appropriately chosen constants.

Given the fact that the similarity measures are defined to be equivalent on average, they have the same expectation, and therefore we expect the ideal ranking function to map elements of S_k onto small values and elements of T_k onto large values on average. Therefore, we can easily define the objective function below:

$$\min_{\overrightarrow{\alpha}\geq 0} \sum_{k=1}^{K}\Big(C_1 \max_{s\in S_k}\big\{\text{rk}\,\big(f_\alpha^k \restriction_S\big)\big\} + C_2 \max_{t\in T_k}\big\{n - \text{rk}\,\big(f_\alpha^k \restriction_T\big)\big\}\Big) \tag{3.6}$$

where C_1 and C_2 are appropriately chosen constants.

As in the case of a single query, the above objective function can be solved using an ILP. In this case, we define the candidate set for q_k to be $C_k = V \backslash \{\{q_k\} \cup S_k \cup T_k\}$. The Multiple Query ILP (MQ-ILP) objective function can then be defined as

$$\min_{\overrightarrow{\alpha}} \sum_{k=1}^{K} \Big(\sum_{v \notin \{S \cup T\}} x_{v,k}^{\overrightarrow{\alpha}} + \sum_{v \notin \{S \cup T\}} y_{v,k}^{\overrightarrow{\alpha}} \Big) \tag{3.7}$$

where $x_{x,v}^{\overrightarrow{\alpha}}$ are indicator variables for all $v \in C_k$ and all $q_k \in \{q_1, ..., q_K\}$, defined as follows:

- If $x_{v,k}^{\overrightarrow{\alpha}} = 1$ and $y_{v,k}^{\overrightarrow{\alpha}} = 0$, then v is ranked better than at least one element in S_k (under $\overrightarrow{\alpha}$) and better than every element of T_k (under $\overrightarrow{\alpha}$).

- If $x_{v,k}^{\overrightarrow{\alpha}} = 0$ and $y_{v,k}^{\overrightarrow{\alpha}} = 1$, then v is ranked worse than at least one element in T_k, and worse than every element of S_k (under $\overrightarrow{\alpha}$).

- If $x_{v,k}^{\overrightarrow{\alpha}} = 0$ and $y_{v,k}^{\overrightarrow{\alpha}} = 0$, then v is ranked better than every element in T_k (under $\overrightarrow{\alpha}$) and worse than every element in S_k (under $\overrightarrow{\alpha}$).

The linear constraints are now

$$\text{for all } v_i \in S_k, v_\ell \in V \setminus (S_k \cup T_k), \quad \sum_{j=1}^{k} \alpha_j d_{ji}^k \leq \sum_{j=1}^{k} \alpha_j d_{j\ell}^k + M x_{v_\ell}^{\overrightarrow{\alpha}} \tag{3.8}$$

$$\text{for all } v_i \in T_k, v_\ell \in V \setminus (S_k \cup T_k), \quad \sum_{j=1}^{k} \alpha_j d_{j\ell}^k - M y_{v_\ell}^{\overrightarrow{\alpha}} \leq \sum_{j=1}^{k} \alpha_j d_{ji}^k \tag{3.9}$$

where $M := \max_{k,i,j} d_{ji}^k$, for all $(s, v) \in (S_k, C_k)$, $(t, v) \in (T_k, C_k)$ and for all $k \in 1, ..., K$. This

guarantees that $x_{v,k}^{\vec{\alpha}}$ is only $= 0$ when $v \in C_k$ is ranked lower than all elements of S_k. Finally, note that the MQ-ILP and ILP are equivalent when either $K = 1$ or $\|S_k\| = 0$ for all $k \in 2, ..., K$.

3.4 Human-in-the-loop

Oftentimes the performance of an algorithm can be significantly improved in the presence of a human-in-the-loop, an iterative feedback process through which a human can interact with algorithmically-generated models [115]. Examples of usage of such models in machine learning include Security Systems, Code Production Tools, Simulation Systems and Search Engines [116]. The information provided by the human aids the algorithm's learning process, which in turn impacts its performance. In settings of recommender systems, a human-in-the-loop is a great source of additional information that helps produce a more accurate ranked list. The majority of recommender systems use Machine Learning to recommend post, products, and other items, usually produced by the users [117]. In our setting, a user can help improve the recommendation list presented to them, by providing the computer with binary dissimilarity information for each element presented to it (i.e. whether each element is similar to the vertex of interest). The human-in-the-loop can be easily integrated into the above-described algorithm (for all three settings: ILP, ALP and SVM) as follows:

1. A user provides the algorithm with a query and a set of dissimilarity measures (we assume the dissimilarity between the query and the set of items to be ranked is known for each of these dissimilarity measures).

2. In addition, let S_0 be the initial set of items known to be similar to the query and T_0 be the initial set of items known to be dissimilar to it.

3. Using the above-described methods, we produce a ranked list and provide the user with the top k elements of that list.

4. The user then returns information of whether each of the top k elements in the ranked list is similar or dissimilar to the vertex of interest v^*.

5. Given this information, each of the k top elements in the list is then added to the initial S or T set (depending on whether the object is similar / dissimilar to the query) in the next iteration.

6. This process is continued until certain stopping criteria are met. Examples of stopping criteria include going through a certain number of iterations or the difference between the current ranked list and previous one being smaller than a certain threshold. For a sketch of this method, see Diagram 3.1.

Remark 3.4.1. Note that a source of error might be the fact that the user is not 'perfect'. In other words, the sets S_i and T_i might not be completely accurate, at each iteration. This obviously has an effect on the accuracy of the ranking list that we produce at each step, which again has an effect on the input that the user provides us with in the following step which results in a cascading error.

3.5 Simulations and Real Data Experiments

To support our above-described claims, we present results of both simulated and real data experiments in the section below. The simulations validate the effect of the T set and the HitL in better algorithmic performance and analyze the impact of the number of dissimilarity measures

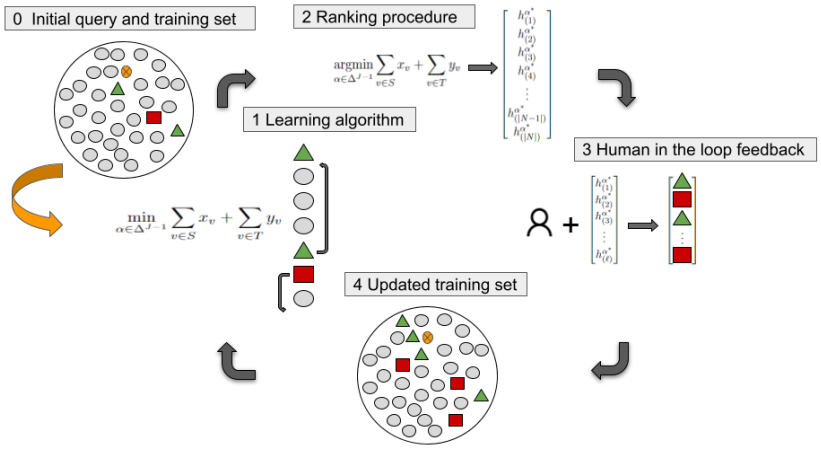

Figure 3.1: A sketch of the human-in-the-loop algorithm. A user provides a query and a set of dissimilarity measures. Given S_0 (the initial set of items known to be similar to the query) and T_0 (the initial set of items known to be dissimilar to it), we produce a ranked list and provide the user with the top k elements of that list. The user then returns information of whether each of the top k elements in the ranked list is similar or dissimilar to the vertex of interest v^*. Given this information, each of the k top elements in the list is then added to the current S or T set (depending on whether the object is similar / dissimilar to the query) in the next iteration. This process is continued until certain stopping criteria are met.

used on performance. Finally, we demonstrate how the quality of the initial seeds (whether the S and T sets initially contain vertices that are similar or dissimilar to v^*) affects algorithmic performance. Additionally, we explore the performance of our algorithms on a few different real data sets.

Remark 3.5.1 (Dissimilarity measures)**.** In the setting of [75], the dissimilarity measures used for the simulation experiments were based on the ASE (Adjacency Spectral Embedding) and LSE (Laplacian Spectral Embedding) of the graph under consideration. The introduction of the T set above necessarily gave rise to a new definition of the dissimilarity measures in our experiments below, based here on the S and T sets themselves. This was a result of the observation that the distance values of the ASE and LSE were too close to each other in our simulation settings, which contributed to artificially high performance values. This issue was solved by defining

another set of dissimilarity measures, based on the graph and the introduction of node covariates of the graph. More specifically, the dissimilarity measures were defined as $\Delta_1, \Delta_2, .., \Delta_J$, where Δ_1 is based on the ASE of the graph and $\Delta_2, ..., \Delta_J$ are based on node covariates/node features. This effectively decorrelated the distances, enabling a setting that allows for the performance of the proposed algorithms to be studied in a more nuanced manner.

3.5.1 Simulations

We conducted multiple experiments on simulated data. Among other things, these simulations highlight and explore the usefulness of the presence of the T set, the role of the human-in-the-loop and the number of dissimilarity measures used in algorithmic performance. In addition, we analyze the effect of the quality of the initial seeds (both S and T sets) in algorithmic performance. The Figures and descriptions below give insight into how the above factors affect the performance of our algorithm.

The experimental setup for all simulations below is as follows. We model $A \sim RDPG(XX^T)$ where X is sampled from a Uniform distribution on the positive unit disk (i.e., first quadrant) and $Z_i \overset{i.i.d.}{\sim} Normal(\vec{0}, I_m)$ independent of X. We then define the 'oracle' dissimilarity via

$$\Delta^*_{i,j} = \alpha^* \|X_i - X_j\| + (1 - \alpha^*) \sum_{k=1}^{m} \|Z_{i,k} - Z_{j,k}\|,$$

where $\alpha^* = 1/(m+1)$. To gain some more intuition on the problem at hand, we start our analysis by providing a simple visual representation/example of the HitL in a simple network setup. Modeling G via a Random Dot Product graph with X and Z_i defined like above, the 51 closest vertices to v^* (note, v^* is chosen randomly in the graph) are initially defined as S^*, and T^*

Figure 3.2: A visual representation of the problem at hand. Each row represents an iteration of the human-in-the-loop algorithm. The green points represent those vertices that are labeled as elements of S at the given iteration and similarly, the black points represent the vertices that are labeled as elements of the T set in the same human-in-the-loop iteration. The red point (representing the vertex of interest v^*) and the orange points (elements of S^*) remain constant throughout the figures, as they represent ground truths. Finally, the gray points simply represent all other vertices in the graph that have not been assigned a label yet.

is chosen to be the set of all 948 other vertices. The initial S_0 set is then chosen to be a random subset of S^* and the initial T_0 set is a random subset of $V \backslash (S_0 \cup v^*)$, where V is the set of total vertices of the graph. Before showing the experiments, we consider an idealized version of the problem to aid in intuition. Each row in Figure 3.2 represents an iteration of the human-in-the-loop algorithm (idealized here in that we use the true latent positions and true features for this

example only). The green points represent those vertices that are labeled as elements of S at the given iteration and similarly, the black points represent the vertices that are labeled as elements of the T set in the same iteration. The red point (representing the vertex of interest v^*) and the orange points (elements of S^*) remain constant throughout the figures, as they represent ground truths. Finally, the grey points simply represent all other vertices in the graph that have not been assigned a label yet. Note that in all three rounds of iterations, the shape of the point cloud is different for each of the three representations. In the combined representation figures one can clearly see the clustering of the vertices in S close to the vertex of interest v^*. Furthermore, vertices in S^* are also clustered around v^*, which is what we expect. Finally, the vertices in T seem to be 'far' from v^* as desired.

As mentioned in the section above, the dissimilarity measures chosen for our experiments are based on the graph features and node features. Specifically, we use the ASE of the adjacency matrix and the covariates to define our dissimilarity measures (where the goal is to learn the optimal $\vec{\alpha}$)

$$\Delta_{i,j}^{\vec{\alpha}} = \alpha_1 \|\hat{X}_i - \hat{X}_j\| + \sum_{k=1}^{m} \alpha_{1+k} \|Z_{i,k} - Z_{j,k}\|.$$

In the experiments below, we rank all vertices in $V \backslash v^*$ based on this dissimilarity measure, and we define S^* to be the top 16 elements in the ranked list. We define T^* to be anything that is not in $S^* \cup v^*$ and run 100 Monte Carlo iterations for each experiment. Given an initial S and T set (note that these sets change from experiment to experiment), our task is to find the optimal linear combination of our dissimilarity measures Δ_1 (ASE-based; i.e., based on estimates $\hat{X}$ =ASE(X) and not on the true X's) and Δ_2 (based on the observed true features/covariates) that would rank the vertices in S towards the top of the list and the vertices of T towards the bottom. We measure

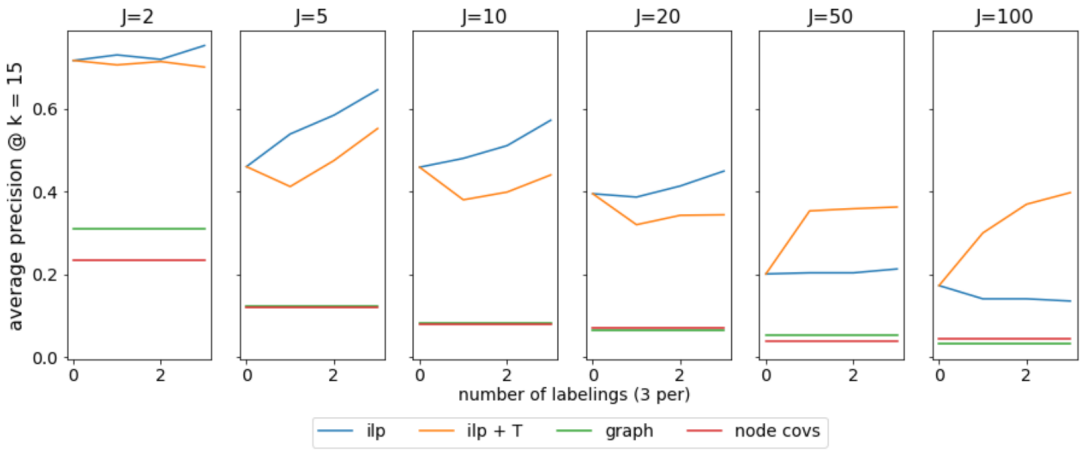

Figure 3.3: Algorithmic performance for different numbers of covariate distances used. It is interesting to note that as this number increases (≥ 50), the extension-to-T-set algorithm performs much better than the original algorithm.

the performance of our algorithm by computing the precision-at-k of the ranking procedure (here $k = 15$).

Figure 3.3 shows the performance of our algorithms for different numbers of covariate distances used (i.e., for m varying). Here, the initial S_0 set is a random subset of S^* and the initial T_0 set is a random subset of $V \backslash S_0$, where V is the set of total vertices of the graph. It is interesting to note that as this number increases ($m \geq 50$), the extension-to-T-set algorithm performs much better than the original algorithm. Both algorithms with and without the T set (no matter what the number of covariate distances used) have a much higher performance rate than when *only* the graph features or *only* the node covariates are used.

In Figure 3.4, we explore how the human-in-the-loop can increase algorithmic performance as we increase the number of initial elements in the T set and vary the quality of the initial seeds (elements of the initial S and T sets chosen here randomly from the S^* and T^* sets). In the first three experiments (row 1) in Figure 3.4, the initial T set is empty, and we analyze how the quality of the initial S seeds affects algorithmic performance. When both initial S seeds are elements of

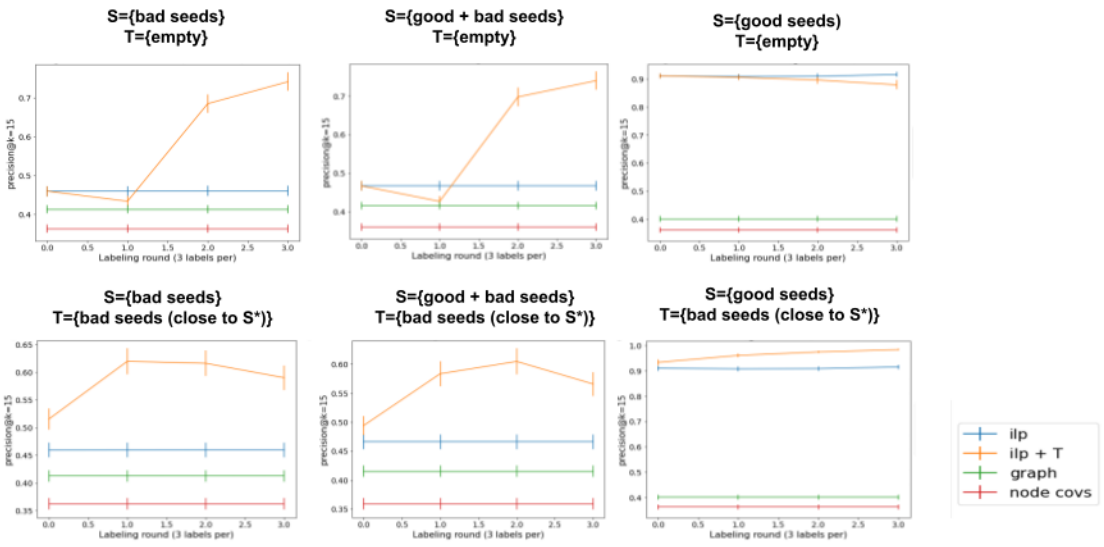

Figure 3.4: The experiments in each column have the same three initial S sets (elements of T^* [left], elements of T^* and S^* [middle] and elements of S^* only [right]), whereas the T sets are changing in each row. The initial T set in the first row is empty and in the second row the initial T set contains elements of T^* that are close to S^*. In all experiments we run 100 Monte Carlo iterations and the error bars represent 2 standard errors across precisions when running the 100 MCs.

S^* (top right), the algorithm seems to be performing well for all iterations of the human-in-the-loop. The vertices in S are hence initially correctly labeled as vertices that are close to v^*. This in turn, helps rank the other vertices that are close to v^* by placing them towards the top of the ranked list. If the initial S set contains only elements of T^* (top left), then we expect the initial performance to not be as good, since we are telling the algorithm that vertices that are far from v^* should be considered as being close to v^*, thus providing 'wrong' information. Note that the user feedback makes a big difference in this case, as within a few iterations of the human-in-the-loop, we have a drastic increase in our algorithm's performance. Here, the user's feedback adds new elements (which are correctly placed up to user errors) to the S and T sets, which in turn increases algorithmic performance. Finally, if the initial S set contains both elements of S^* and

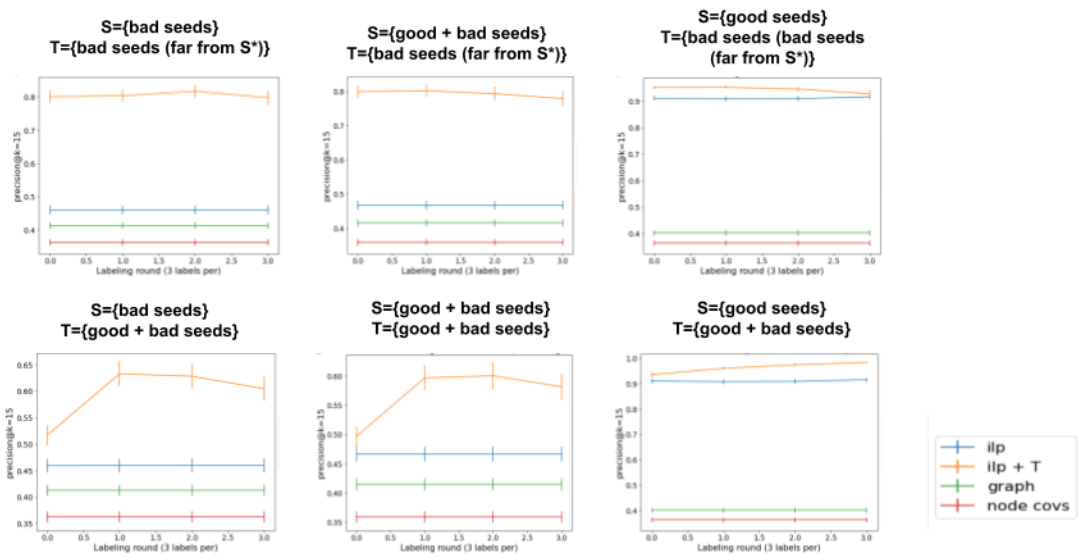

Figure 3.5: The experiments in each row have the same three initial S sets (elements of T^* [left], elements of T^* and S^* [middle] and elements of S^* only [right]), whereas the T sets are changing in each row. The initial T set in the third row contains elements of T^* that are far from S^* and in the fourth row it contains both elements of S^* and T^*. In all experiments we run 100 Monte Carlo iterations and the error bars represent 2 standard errors across precisions when running the 100 MC.

T^*, then we do expect an improvement of the algorithm as we go through iterations of the human-in-the-loop. The fact that we initially provide the algorithm with some 'wrong' information (an element of T^* being in the initial S set), decreases the total performance slightly, but we still see overall improvement, as we initially also provided the algorithm with 'correct' information (an element of T^* in the initial T set). We see a similar trend in the second three experiments (row 2) in Figure 3.4, where the initial T set is chosen to be bad seeds close to S^*; allowing us to analyze how the quality of the initial T seeds affects algorithmic performance.

The plots in Figures 3.5 tell a similar story. The experiments in each column have the same three initial S sets (elements of T^*, elements of T^* and S^* and elements of S^* only), whereas the T sets are changing in each row; here the elements of the initial S and T sets are randomly

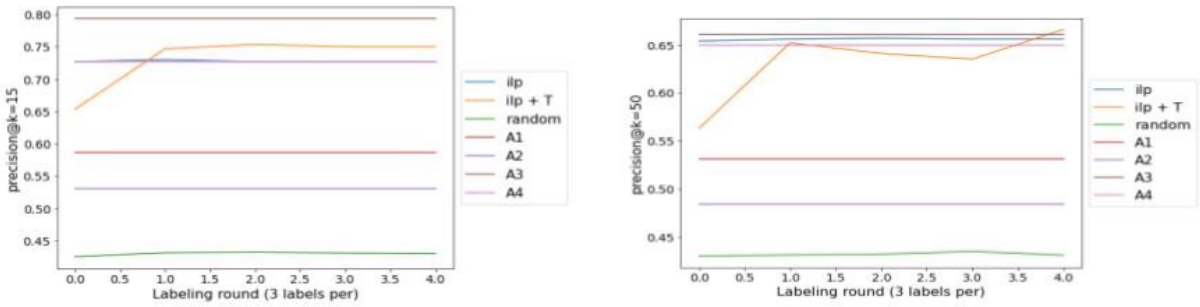

Figure 3.6: We compare the performance of our learned optimal $\overrightarrow{\alpha}$ to the performance when using each of the embeddings (adjacency and Laplacian) separately on each of the two graphs. We call these latter 4 plots $A1, A2, A3$ and $A4$. When running 32 Monte Carlo iterates, the performance of the ILP setting with the T set setting seems to be increasing over the number of human-in-the-loop iterations, when compared to the ILP (without the T set) setting and when compared to most of the A_i settings. The left figure in Figure 3.6 shows the performance of the algorithms at precision at $k = 15$, whereas the right figure shows this performance for precision at $k = 50$.

chosen from S^* and T^*. In the first row the initial T set contains elements of T^* that are far from S^* (and hence helpful) and in the second row it contains both elements of S^* and T^*. To summarize, the presence of T^* vertices in the initial T set and/or S^* vertices in the initial S seem to increase algorithmic performance, as intuitively expected. Furthermore, the presence of T^* vertices in the initial S set and/or S^* vertices in the initial T set decreases performance, since the initial information provided in these cases is 'incorrect'. Finally, the human-in-the-loop seems to have the biggest effect on algorithmic performance in those settings where the initial seeds contain 'incorrect' information.

3.5.2 Real Data Experiment: C. elegans

Caenorhabditis elegans is a transparent roundworm (about 1 mm in length) that lives in temperate soil environments. The name is derived from the Greek words *caeno* (recent) and *rhabditis* (rod-like) together with the Latin word *elegans* (elegant). In 1963, Sydney Brenner

suggested researching *C. elegans*, focusing on the area of neuronal development. Later his research involved the molecular and developmental biology of *C. elegans*. Since then, it has been greatly used as a model organism and was the first multicellular organism to have its whole genome sequenced [118].

Most *C. elegans* are female hermaphrodites and in our experiments we consider data from the brain of such elegans. Specifically, we look at the brain of these *C. elegans* and the two different types of connections between the brain neurons: chemical synapses (A_c) and electrical synapses (A_e). These provide us with two different graphs, where vertices represent neurons and edges represent the electrical and chemical synapses respectively. We binarize and symmetrize the graphs and then perform an OMNI embedding [119, 120] on both graphs simultaneously.

Each vertex belongs to one of three categories of neurons: inter/motor/sensory. In our experiments, we consider a random interneuron vertex as our query and let all other interneuron vertices be S^*. The motor and sensory neurons are considered T^*. We compute the adjacency and Laplacian spectral embeddings of the OMNI matrix [119, 120]

$$M = \begin{bmatrix} A_c & \frac{A_c+A_e}{2} \\ \frac{A_c+A_e}{2} & A_e \end{bmatrix}$$

and use these two embeddings to define our distance measures Δ_i used to learn the optimal $\overrightarrow{\alpha}$. We compare the performance of our learned optimal $\overrightarrow{\alpha}$ to the performance when using each of the embeddings (adjacency and Laplacian) separately on each of the two graphs (yielding 4 dissimilarities). We call these latter 4 plots $A1, A2, A3$ and $A4$. As one can see from Figure 3.6 when running 32 Monte Carlo iterates, the performance of the ILP setting with the T set

seems to be increasing over the number of human-in-the-loop iterations, when compared to the ILP (without the T set) setting and when compared to most of the A_i settings. The fact that some of the A_i settings seem to be performing better than the $ILP + T$ setting may be due to the fact that we do not know what the true optimal linear combination is, and whether or not a combination of the embeddings does indeed increase algorithmic performance as compared to using the simple embeddings in the first place. The left figure in Figure 3.6 shows the performance of the algorithms at precision at $k = 15$, whereas the right figure shows this performance for precision at $k = 50$. Note that all algorithms seem to be performing better at precision at $k = 15$, suggesting that correct nominations here concentrate higher in the nomination list as desired.

Chapter 4: Discovering sequential anomalies in network time series

4.1 Novel contribution

Finally, we consider a setting where the anomaly *is* the signal that we wish to uncover. This project is motivated by data of neuronal activity of a sea slug brain collected by colleagues Prof. William Frost at Rosalind Franklin University and Prof. Jeffrey Brown at Penn State University. The above-mentioned Professors used consensus clustering to analyze the behavior of the neurons and group together neurons that seem to be behaving similarly. Below, we introduce a novel method to analyze this behaviour and discover neuronal clusters, namely one in which we use graph embedding methods to discover sequential anomalies in network time series. Specifically, we introduce and use the DCRDPG (see 17) to naturally encode a network time-series, and apply functional data clustering to discover groups of neurons with similar behavior in the networks.

4.2 Sequential anomalies in network time series

In the previous two projects, the anomaly came in the form of adversarial noise which was regularized via trimming or the incorporation of a HitL. In this chapter, the anomaly is the signal that we wish to uncover, and this last project explores using graph embedding methods to discover sequential signal anomalies in network time series. Our motivation in this chapter is

the following real-data biological setting. We are given temporal recordings of the dorsal pedal ganglia of Apylsia Californica slugs. In the experiment setting, the slug is given nerve stimuli at several different times, and the goal is to understand and categorize the behaviour of the neurons in the brain through their reaction to the sequential stimuli. Specifically, we seek to uncover potential *learning* behaviour (here in the form of serial sensitization), where the firing rates of certain brain neurons do not return as quickly to resting state after the second or third stimuli, and full activation of the motor program does not require as significant a change in firing rate. This may be implying that they have learned to expect/are primed for sequential stimuli, given previous stimuli.

One way to analyze the behaviour of the neurons is by looking at graph embeddings of the neuronal time-series. In general terms, we can use these embeddings to cluster neurons into groups of similar activity patterns which in turn provides information about the way each neuron behaves when sequential stimuli are applied. In this chapter, we consider a tractable model to understand these time-series, namely the Degree Corrected Random Dot Product Model, abbreviated as the DCRDPG.

Definition 17 (d-dimensional Degree Corrected Random Dot Product Graph Model (DCRDPG))**.** Let F be a distribution with support in a set $\mathcal{X} \subset \mathbb{R}^d$ where if $x, y \in \mathcal{X}$ then $\langle x, y\rangle \in (0, 1)$. Let the matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ have rows that are distributed i.i.d. according to F. A random graph $(G, \mathbf{X}) \sim DCRDPG(F, \nu, \vec{c})$ is an instantiation of a degree corrected random dot product graph with parameters F, $\vec{c}$ and sparsity factor ν, if, conditional on $\mathbf{X}$ and $\vec{c}$, edges in the graph are independently distributed with

$$\mathbb{1}\{\,\{i, j\} \in E(G)\} \sim \text{Bernoulli}(\nu c_i c_j X_i^T X_j).$$

The upper triangular portion of the adjacency matrix A of G is then conditionally distributed via

$$P(A|\mathbf{X}, \vec{c}) = \prod_{i<j} (\nu c_i c_j X_i^T X_j)^{A_{ij}} (1 - \nu c_i c_j X_i^T X_j)^{1-A_{ij}}.$$

We use the DCRDPG to naturally encode a network time-series $(A_t)_{t=t_1}^{t_m}$ as follows. We sample a common latent position matrix $\mathbf{X}$ with rows i.i.d. according to F. For each $t \in [t_1, ..., t_m]$, we then have

$$P(A_t|\mathbf{X}, \vec{c}_t) = \prod_{i<j} (\nu c_{t,i} c_{t,j} X_i^T X_j)^{A_{ij}} (1 - \nu c_{t,i} c_{t,j} X_i^T X_j)^{1-A_{ij}}.$$

so that we assume $\mathbf{X}$ and ν are invariant in t, but that the degree correction parameters $\vec{c}_t = (c_{t,1}, c_{t,2}, \ldots, c_{t,n})$ vary in t. In this regard, $\vec{c}^{(i)} := (c_{1,i}, c_{2,i}, ..., c_{m,i})$ is a one-dimensional time-series for each vertex v_i, where we uniformly set $c_{1,i} = 1$ for identifiability reasons. Our goal is to estimate and cluster the $\vec{c}^{(i)}$'s with the applied goal of this one-dimensional data projection providing (ideally) a better understanding of the differences in temporal behavior across the different neurons in the motor program.

4.3 Motivating Example

As mentioned briefly in the introduction, the data under consideration is neuronal activity of a sea slug brain collected by colleagues Prof. William Frost at Rosalind Franklin University and Prof. Jeffrey Brown at Penn State University. Specifically, we are provided with 33 min recordings of $n = 127$ neurons in the dorsal pedal ganglia of Apylsia Californica slugs. These recordings involve the subsequent application of four different nerve stimuli over the course of

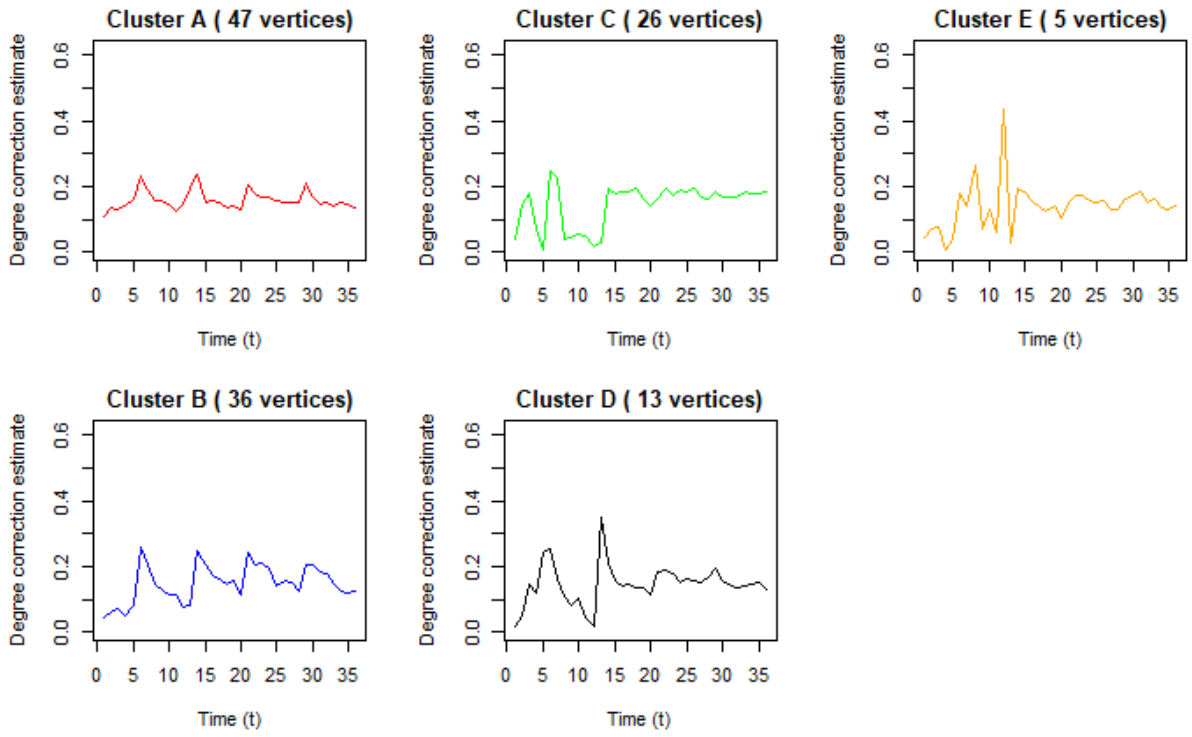

Figure 4.1: The mean trajectory for each cluster of neuronal degree corrections, when we consider 5 clusters in our FDC algorithm. These time-series clusters help us extract important trends of neuron activity over time, specifically the way each neuron group reacts to the sequential stimuli given, and the desensitization that occurs as the second, third and fourth stimuli are applied.

33-mins at times 5, 12, 19, and 26 minutes. We cut the time (in seconds) into 36 intervals and for each of these intervals we create a matrix of the computed **sttcs** (Spike Time Tiling coefficients) for the neuronal interactions [121]. This produces 36 activity correlation matrices, which we model as our network time-series. Note that 36 was chosen to allow for the stimuli to occur sufficiently far from the start/end of their respective intervals. Finally, we add a small correlation value ($\epsilon = 0.001$) to each edge to offset the effect of extreme sparsity in the subsequent embedding.

We use the Adjacency Spectral Embedding to separately embed each of these matrices $(A_t)_{t=1}^{36}$ into a lower dimensional Euclidean space to obtain the estimated latent positions $\hat{Y}_i^{(t)}$ for each vertex i in each time indexed graph A_t in the time-series (where ideally (up to a rotation)

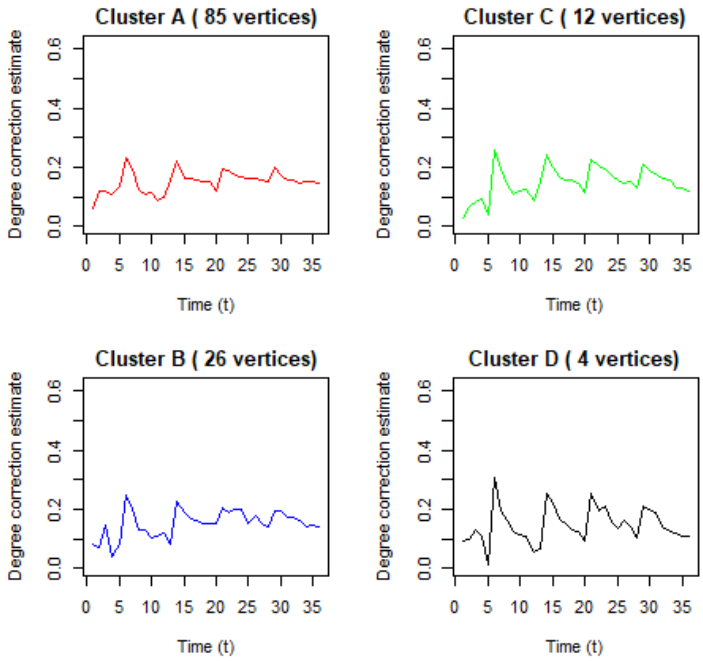

Figure 4.2: The mean trajectory for each cluster of neuronal degree corrections, when we consider 4 clusters in our FDC algorithm. These time-series clusters help us extract important trends of neuron activity over time, specifically the way each neuron group reacts to the sequential stimuli given, and the desensitization that occurs as the second, third and fourth stimuli are applied.

$\hat{Y}_i^{(t)} \approx \sqrt{\nu} c_{t,i} X_i$ in the notation of the DCRDPG). Note that the common embedding dimension is chosen via an automated elbow detection in the singular value scree plot [62, 122]). Typically, such an estimation/embedding procedure would either need to proceed jointly or align the vertices across embeddings via a Procrustes (or similar) alignment. In the joint embedding case, one such possibility would be to use the Omnibus spectral embedding method [119] with flat weights [123] to jointly embed these matrices into a lower dimension. Specifically, given the 36 correlation networks $\{A_t\}_{t=1}^{36}$, we create the block omnibus matrix $M \in \mathbb{R}^{4572 \times 4572}$ (here $4572 = 127 \times 36$ accounts for M being 36 blocks-by-36 blocks each of size 127) where the t, s-th block in M is

$$\frac{A_t + A_s}{2}.$$

We would then embed M using adjacency spectral embedding, which would produce simultaneous embeddings of all vertices across all graphs; note that this comes at the expense of inducing correlation in the embedding space across embeddings [123], and this correlation can obfuscate subsequent inference.

In the present separate embedding setting, even with no Procrustean alignment, we can easily obtain the estimated parameters $\hat{c}_{t,i}$ for all i and t via

$$\hat{c}_{t,i} = \|\hat{Y}_i^{(t)}\| / \|\hat{Y}_i^{(1)}\|,$$

noting that these estimates are invariant to rotations of the ASE embeddings. We can hence deduce information about the neuron's activity by clustering the estimated time-series $\hat{\vec{c}}^{(i)} = (\hat{c}_{1,i}, \hat{c}_{2,i}, ..., \hat{c}_{36,i})$. To this end, we can use Functional Data Clustering (FDC) to group neurons based on the 'similarity' of their trajectories over time. Here, each function, $\hat{\vec{c}}^{(i)}$, is considered a data point, and we compute a distance between pairs of functions and then cluster them using the R-package **kml** [124] an implementation of the classical K-means clustering algorithm, but for longitudinal data. Figure 4.1 (resp., 4.2) shows the mean trajectory for each cluster of neurons, when we consider 5 clusters in our algorithm (resp., 4 clusters). Ideally, these time-series clusters will help us extract important trends of neuron activity over time, specifically the way each neuron group reacts to the sequential stimuli given, and the desensitization that occurs as the second, third and fourth stimuli are applied.

In each of the clusters in Figure 4.1, we note that there are notable spikes in neuronal activity (here captured by a spike in the neuronal degree-correction parameters) at each of the four stimuli applied to the sea slug. In Clusters A, B and C , the neurons seem to have reacted

strongly to the first stimuli (corresponding to a large spike in the graphs at index ≈ 5). In Clusters A and B, the subsequent stimuli did not cause as large of a reaction due to the $\hat{c}$'s not returning to their baseline level after the first stimuli, suggesting a possible desensitization of the neurons to the stimuli. This similarly occurs in Cluster C after the second stimulus, and to a lesser extent in Cluster D. In the four cluster setting, we see similar potential desensitization in Cluster A, and to a lesser extent in B and C; see Figure 4.2. In Figure 4.1 (resp., Figure 4.2), the neurons in cluster E (resp., cluster D) seem to behave more chaotically to the stimuli, and these do not exhibit the same level of desensitization.

Below we present the theoretical details of the graph embedding and clustering methods used in our analysis. We show that $\hat{c}_{t,i}$ is indeed a consistent estimate of the parameter $c_{t,i}$ and provide some background on Functional Data Clustering (FDC) [125].

4.4 Background Theory

Our estimation procedure begins by computing $\hat{Y}^{(t)} = ASE(A_t, d)$ (here ASE is the Adjacency Spectral Embedding of [9,126]) and then estimating each $c_{t,i}$ via $\hat{c}_{t,i} = \|\hat{Y}_i^{(t)}\|/\|\hat{Y}_i^{(1)}\|$. Below we show that each $\hat{c}_{t,i}$ is indeed a consistent estimate of the parameters $c_{t,i}$ in the DCRDPG model.

Theorem 4.4.1. With notation as above, let m be fixed, and assume that $\vec{c}_t = \Theta(1)$ entry-wise for all t. Further assume that there exists a constant C such that F is supported on $\mathcal{X}$ such that

i. $x \in \mathcal{X} \Rightarrow \|x\| > C$

ii. $x, y \in \mathcal{X} \Rightarrow \langle x, y \rangle \in (0, 1)$

Then there exists a constant $\alpha > 0$ such that if (where the implicit dependence on n is dropped to ease notation)

$$\nu = \omega\left(\frac{\log^{4\alpha} n}{n}\right),$$

then

$$\sup_{i,t}|c_{t,i} - \widehat{c}_{t,i}| = O\left(\frac{\log^{\alpha} n}{n^{1/2}\nu^{1/2}}\right) \tag{4.1}$$

holds with probability at least $1 - 1/n^2$ for all n sufficiently large.

Proof. Let the random graph $(G, \mathbf{X}) \sim DCRDPG(F, \nu, \vec{c})$ be an instantiation of a degree corrected random dot product graph with parameters F, ν and $\vec{c}$ as defined in Definition 17. The latent position matrix $Y^{(t)}$ at time t can then be defined via $Y_i^{(t)} = \nu^{1/2} c_{t,i} X_i$. For vertex i, we then have the following

$$\begin{aligned}
|\hat{c}_{i,t_j} - c_{i,t_j}| &= \left|\frac{\|\hat{Y}_{i,t_j}\|}{\|\hat{Y}_{i,t_1}\|} - \frac{\|Y_{i,t_j}\|}{\|Y_{i,t_1}\|}\right| \\
&\leq \left|\frac{\|\hat{Y}_{i,t_j}\|}{\|\hat{Y}_{i,t_1}\|} - \frac{\|\hat{Y}_{i,t_j}\|}{\|Y_{i,t_1}\|}\right| + \left|\frac{\|\hat{Y}_{i,t_j}\|}{\|Y_{i,t_1}\|} - \frac{\|Y_{i,t_j}\|}{\|Y_{i,t_1}\|}\right| \\
&= \left|\frac{\|\hat{Y}_{i,t_j}\|\|Y_{i,t_1}\| - \|\hat{Y}_{i,t_j}\|\|\hat{Y}_{i,t_1}\|}{\|\hat{Y}_{i,t_1}\|\|Y_{i,t_1}\|}\right| + \left|\frac{\|\hat{Y}_{i,t_j}\| - \|Y_{i,t_j}\|}{\|Y_{i,t_1}\|}\right| \\
&= \underbrace{\left|\frac{\|\hat{Y}_{i,t_j}\|}{\|\hat{Y}_{i,t_1}\|\|Y_{i,t_1}\|}\left(\|Y_{i,t_1}\| - \|\hat{Y}_{i,t_1}\|\right)\right|}_{I} + \underbrace{\left|\frac{\|\hat{Y}_{i,t_j}\| - \|Y_{i,t_j}\|}{\|Y_{i,t_1}\|}\right|}_{II}
\end{aligned} \tag{4.2}$$

To bound the above expression, note that the following holds. From Theorem 3 in [127], we

know that under the assumptions of the theorem

$$\max_{i\in[n],j\in[m]} \|\mathbf{Q}_{n,j}\hat{Y}_{i,t_j} - Y_{i,t_j}\| = O_{\mathbb{P}}\left(\frac{\log^{\alpha} n}{n^{1/2}}\right), \tag{4.3}$$

where $\mathbf{Q}_{n,j} \in \mathcal{O}(d)$ is a sequence of orthogonal matrices. Next note that under the assumptions of the theorem, we have

$$\sqrt{nu} = \omega\left(\frac{\log^{2\alpha} n}{n^{1/2}}\right)$$

and that $\|Y_{i,t_j}| = \Theta(\sqrt{\nu})$ for each i, j. This provides that

$$\|\hat{Y}_{i,t_j}\| \geq \|Y_{i,t_j}\| - \|\mathbf{Q}_{n,j}Y_{i,t_j} - \hat{Y}_{i,t_j}\| = \Omega_{\mathbb{P}}(\sqrt{\nu});$$

$$\|\hat{Y}_{i,t_j}\| \leq \|Y_{i,t_j}\| + \|\mathbf{Q}_{n,j}Y_{i,t_j} - \hat{Y}_{i,t_j}\| = O_{\mathbb{P}}(\sqrt{\nu}).$$

Therefore, we can bound Term I in Eq. 4.2 via

$$\begin{aligned}\sup_{i,j}\left|\frac{\|\hat{Y}_{i,t_j}\|}{\|\hat{Y}_{i,t_1}\|\|Y_{i,t_1}\|}\left(\|Y_{i,t_1}\| - \|\hat{Y}_{i,t_1}\|\right)\right| &= O_{\mathbb{P}}\left(\frac{\nu^{1/2}}{\nu}\right) O_{\mathbb{P}}\left(\frac{\log^{\alpha} n}{n^{1/2}}\right) \\ &= O_{\mathbb{P}}\left(\frac{\log^{\alpha} n}{n^{1/2}\nu^{1/2}}\right)\end{aligned}$$

We can bound Term II in Eq. 4.2 similarly via

$$\sup_{i,j}\left|\frac{\|\hat{Y}_{i,t_j}\| - \|Y_{i,t_j}\|}{\|Y_{i,t_1}\|}\right| = O_{\mathbb{P}}\left(\frac{\log^{\alpha} n}{n^{1/2}\nu^{1/2}}\right)$$

We hence conclude that

$$\sup_{i,j}|c_{t_j,i} - \widehat{c}_{t_j,i}| = O_{\mathbb{P}}\left(\frac{\log^{\alpha} n}{n^{1/2}\nu^{1/2}}\right)$$

as desired.

■

4.4.1 Functional Data Clustering

In our motivating example, once we obtain the time series based on the DCRDPG model, we use Functional Data Clustering to group the neurons based on their behavioral similarities. Functional Data Analysis (FDA) is a branch of statistics where each data point under consideration is a random function or surface (usually considered to be sufficiently smooth and independent for ease of analysis). The FDA methodology we apply herein is nonparametric, utilizing smoothing methods and functional distance metrics (rather than explicit parametric assumptions on the generating time-series process), and allows for flexible modeling [128]. Functional data clustering is a method of FDA whose task is the clustering of data that is continuous in nature (i.e., the data points are continuous functions that underlie the discrete measurements or observations) [125]. Cluster analysis, in itself, is a common method used to group a set of objects based on grouping similar data points into coherent sets (note that similarity here can be defined in many different ways: a common way is by considering Euclidean Distance). In clustering, often the goal is for objects in the same group/cluster to be more similar to each other than to those in other groups/clusters.

In our setting, the data is represented by curves, rather than single points in Euclidean space.

There are many different methods used for functional data clustering, including methods that are borrowed from discrete data clustering. A survey of functional data clustering can be found here [129] and more literature on different functional data clustering methods can be found in, for example, [130,131] including another review of clustering methods for functional data. In our setting, we consider each neuron's degree-correction trajectory to be a single data point and use Functional Data Clustering to group neurons based on the 'similarity' of their behavior/trajectory over time. More specifically, we compute the Euclidean distance between pairs of functions and run a K-means algorithm on this distance data using the R-package *kml* [124, 132], an implementation of K-means, but for longitudinal data. The `kml` algorithm is a "hill-climbing" algorithm, which allows it to always converge towards a maximum. Given that it is not possible to know whether one is obtaining a local or global maximum, the hill-climbing algorithm is run multiple times to ensure the best quality partition. By default, the algorithm is run 20 times and the partition that maximizes the determinant of the matrix between is then chosen (see [133] for more detail).

Remark 4.4.1. Our degree-correction functional data, and the underlying data stochastic process $X = \{X(t), t \in T\}$, can be modeled via a random time-series. As such, we can consider each time series to be a stochastic process, where $\mathbf{c}$ is a collection of random variables $\mathbf{c}(t)$ indexed by the bounded interval $t \in [0, 1]$, with mean $\mu(t) = \mathcal{E}(\mathbf{c}(t))$ and covariance function $\Sigma(s, t) = Cov(\mathbf{c}(s), \mathbf{c}(t))$. For the t_j-th time, the realization is then a sample of n independent observations, $\mathbf{c}_{t_j,\bullet}$. This formalism allows us to apply the powerful tools of change-point and anomaly detection from classical time-series analysis in the present spike train setting. However, there is work to be done here to classify each stimulus as either an anomaly or change-point, and

also in formally considering a return to resting-state after a given stimulus. This formalism is the subject of present research and will not be considered further herein.

Remark 4.4.2. To get a sense of the importance of each vertex in the connectivity of the network, we can use the notion of *betweenness centrality*. Formally, for a vertex $v \in V(G)$, the betweenness centrality $\mathfrak{b}(v)$ is defined via

$$\mathfrak{b}(v) = \sum_{u,w \neq v} \frac{p_{u \to v \to w}}{p_{u \to w}},$$

where $p_{u \to w}$ is the number of shortest paths in G from u to w, and $p_{u \to v \to w}$ the number of shortest paths in G from u to w passing through v. This can be thought of as the fraction of geodesics (shortest paths) going through a vertex v. In a way, one can think of a vertex with a high betweenness value to be one that has an important role in the flow of the network, as many shortest paths in the network pass through this vertex. These vertices are key to the network's structure and hence to the development of the motor program, and our formalism allows us to see how their importance is visualized/instantiated via the degree-correction time-series. To this end, we calculate the betweenness measure of each vertex at each time point and plot a time series of these values for those vertices with the highest betweenness scores (in this particular setting, we plot time series for those vertices v that have at least one betweenness measure in the time series satisfying $\mathfrak{b}_t(v) > 50$ ($\mathfrak{b}_t(v)$ being the betweenness centrality measure at time t). As can be seen in Figure 4.5, the high betweenness values for all the top vertices seem to appear at those time points when the stimuli are applied. This may suggest that in those moments when the stimuli happen, the particular vertices with high betweenness values play an important role in the flow of the neuronal information in the network at that time.

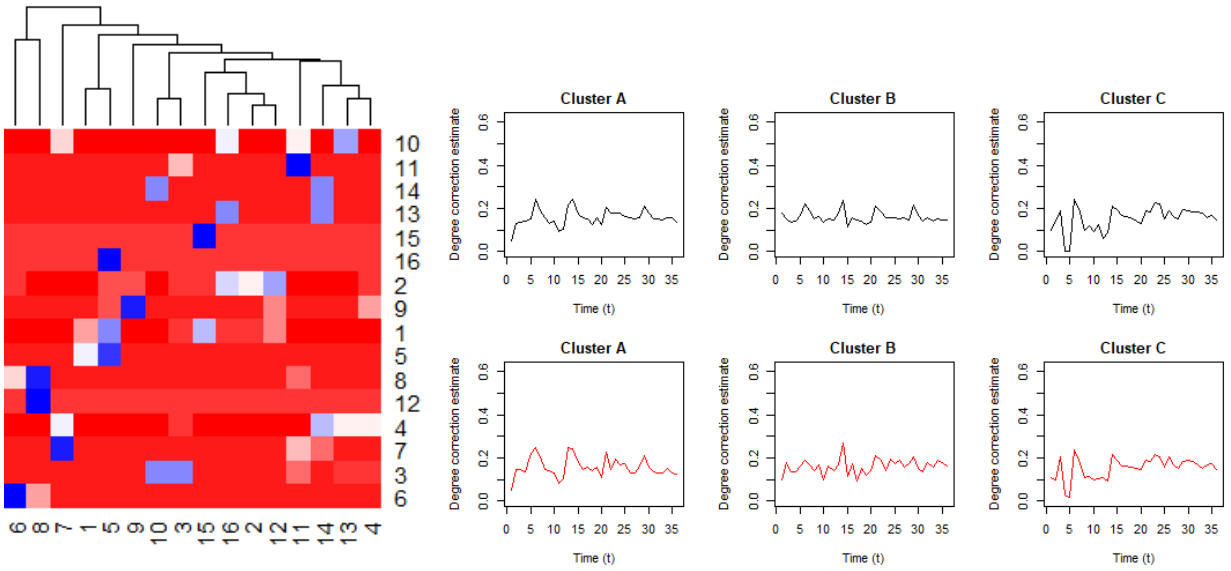

Figure 4.3: We consider FDC with 16 clusters and the consensus clustering performed by our neuroscience colleagues (similar to that done in [4]). On the left, we plot a heat map of the Jaccard index for each pair of clusterings, with one element of the pair being a FDC clustering and the other element being a consensus clustering. Here, the range of values is from 0 to 0.54 and warmer colors (like red) in the heat map correspond to lower values, cooler colors (like blue) correspond to higher values, and white corresponds to some middle value. On the right, we plot three FDC clusters and their assigned similar consensus clusters (computing an l2-norm between the means of the FDC (black) and consensus clusters (red), and assigning pairs via solving a linear assignment problem with this cost matrix).

4.5 Comparing Methods

In order to better understand our findings, we next compare our result to consensus clustering [134–136], a clustering tool used by our neuroscientist colleagues to cluster and analyze the neurons in the spike-train data under consideration. Below, we will see that although our FDC method produces somewhat similar clustering results to the consensus clustering method (which is what we want and expect), the FDC and consensus clustering methods produce clusters that differ in significant and interesting ways.

Consensus clustering is usually used to create stable results out of a set of partitions

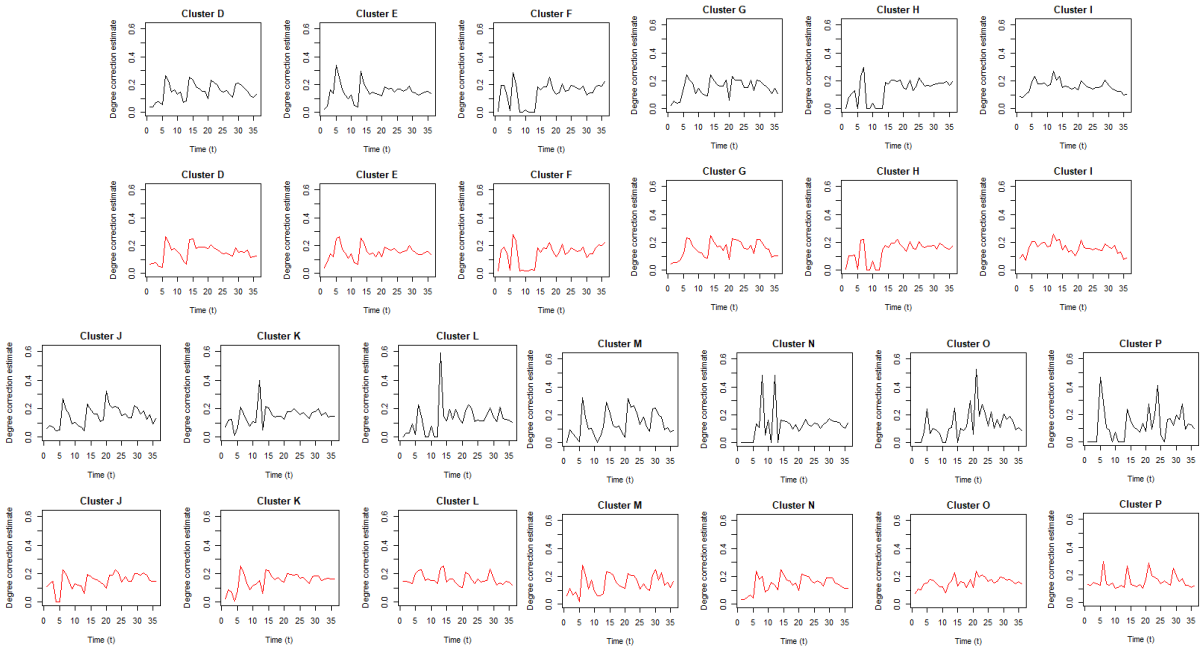

Figure 4.4: We consider FDC with 16 clusters and the consensus clustering performed by our neuroscience colleagues (similar to that done in [4]). We plot the remaining 13 FDC clusters and their assigned similar consensus clusters (computing an l2-norm between the means of the FDC (black) and consensus clusters (red), and assigning pairs via solving a linear assignment problem with this cost matrix); see Figure 4.3 for the first three pairings.

generated by different clustering algorithms [134]. A problem with a lot of clustering techniques is that they do not produce a unique answer, as often times the partition corresponds to extrema of a cost function. In these cases, the results need to be approximated, usually depending on either random seeds or initial conditions [137]. Finding a partition that is representative of the community structure of the given system when there are several outputs for a given method is crucial. In order to choose this partition, one can combine the information of different outputs into a new partition; specifically, when working with time-stamped network datasets, we can combine partitions corresponding to different time windows [137]. Consensus clustering is a well known data analysis technique that is used to solve this problem. In simple terms, the goal in consensus clustering is to look for the *median* (or *consensus*) partition. This partition is

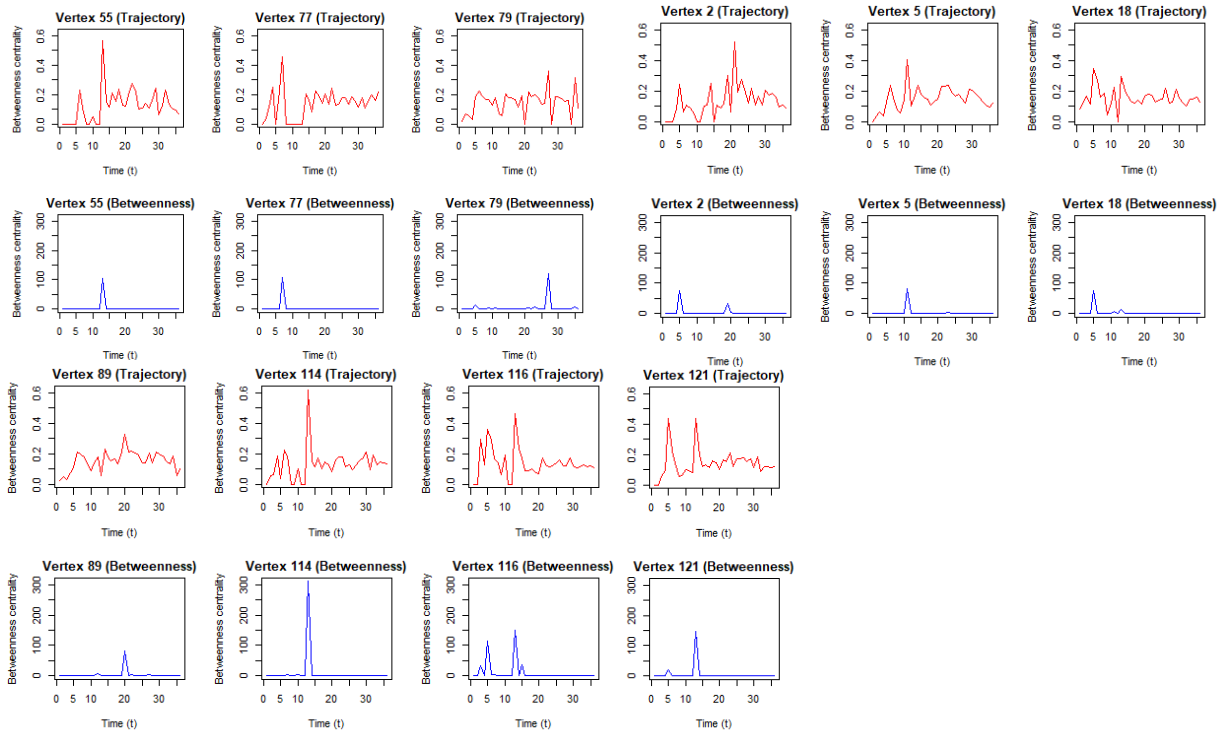

Figure 4.5: We calculate the betweenness centrality measure of each vertex at each time point and plot a time series of these values (blue) for those vertices with the highest betweenness scores. In this particular setting, we plot time series for those vertices that have at least one betweenness centrality measure whose value is > 50 in blue. For each of these 10 vertices, we plot the corresponding degree-correction time series in red.

meant to be the most similar, on average, to all the input partitions. Different similarity measures have been used for computing the distance between clusters (to compute a median), including the Normalized Mutual Information (NMI) [138]. This formulation, however, results in a hard combinatorial optimization problem, and an alternative method uses the consensus matrix (a matrix based on vertices that co-occur in clusters of the input partition) to create the median cluster [134]. This matrix is used as an input for the clustering technique and leads to a new set of partitions which then in turn generate a new consensus matrix. This process iterates until a unique partition is reached which does not change with further iterations, and this process has been shown to quickly create consistent and stable partitions in real networks [139].

In Figure 4.3, we compare the mean trajectories of the two clustering methods, with number of clusters set to 16 in the FDC (16 here being the number of clusters in the consensus clustering provided by our neuroscience colleagues; their consensus clusters were obtained similarly to those in [4]). On the left panel of Figure 4.3, we plot a heat map of the Jaccard index for each pair of clusterings, with one element of the pair being an FDC clustering and the other element being a consensus clustering. This index, which measures similarity between finite sample sets, is defined as the size of the intersection divided by the size of the union of the sets. Through this computation, we can determine how many neurons each of the clustering pairs have in common. As the heat map suggests, there is enough overlap of neurons in cluster pairs for us to deduce that the clustering methods produce similar results. At the same time, there seems to be enough of a difference between clustering methods to make us believe that the FDC clustering can provide novel insights with regard to the neuron activity. On the right panel in Figure 4.3, we plot three FDC (black) and their similar consensus clusters (red). In order to get a pairing of similar clusterings, we solve a linear assignment problem (LAP) to pair clusters that are 'closest' to each other (each pair containing one cluster from the FDC clustering and one from the consensus clustering). 'Similarity' here is measured by the l2-norm (Euclidean distance) between the corresponding mean degree-correction time series we are working with above. We store these values in a cost matrix, where entry a_{ij} of the matrix corresponds to the l2 norm between cluster i (FDC method) and cluster j (consensus clustering method). We then use the `solve_LSAP` function in R to solve the linear assignment problem and assign cluster pairs. The remaining 13 pairings are plotted in Figure 4.4. We note first that the pairings achieved via the LAP and the most similar pairs according to the Jaccard index (obtained via an LAP of the Jaccard matrix) are different (here they have an overlap of $9/16$ assignments). While most of the trajectory pairs seem

to have similar trends across clusterings, they interestingly capture different neuronal groupings that exhibit similar underlying behavior. This is evidence that our FDC method is recovering distinct (and similarly biologically relevant) clusterings to the previous consensus approach.

Chapter 5: Conclusion and Future work

The main focus of this text was to find ways of extracting the latent structure in the presence of network anomalies, in the case where the anomaly is signal and in the case where it is noise. We looked at three different scenarios: one in which the graph is subject to an adversarial attack and the anomaly is inhibiting inference (Chapter 2-3), one in which detecting the anomaly is the key inference task (Chapter 4), and lastly we explored graph embedding methods to discover sequential anomalies as signal in a network time series. Below, we provide some concluding remarks and potential future avenues of research for each of these sections.

5.1 Future work on adversarially contaminated settings

Adversarial attacks on networks are phenomena that unfortunately happen frequently and thus, trimming methods and HitL regularization methods that help mitigate the affect of the contamination are much sought after. In our work, we introduced a novel trimming method, where the trimming happens in model space, rather than graph space. We extended this procedure to a setting where we might have two types of noise: structured and unstructured (white noise). We further introduce a robust K-means algorithm, where a fixed maximum cluster radius is used to gather points into clusters and"clean out" (ideally) the noise points, whose distance from any cluster center is greater than the maximum radius. Experiments show that the two cleaning

methods combine seamlessly and succeed in retrieving adequate information of the contaminated graph in both simulated and real-life data. While Stochastic Block Models and RDPG models are simple, yet rich models to work with, a natural next step would be to develop the analogous theory for more general structured noise settings, and exploring regularization methods in other, more complex contaminated models. One option might be to provide a suite of different robust regularization algorithms, created precisely to be implemented in different contexts, and see which one gives the best results. One can also consider local regularization methods (for more localized noise sources), which is another open area of research. For contamination procedures that act locally, regularization methods performed at a local level might prove to be effective in many settings.

Another path to explore is lifting the above ranking problems to higher dimensions, including multilayered graphs and time series. In this case, tensors are a natural framework to work in and provide the relevant theory that will support a thorough exploration of the subject.

5.2 Future work on ILP extensions

One natural scenario is to consider our new ILP algorithm in the case where we are given multiple sets of queries/vertices of interest and sets S and T for each query. In the setting of recommendation systems in VN, we also further explore the connection between the ILP and a variant (defined below) we coin the 'ALP' and also consider the relationship between the ILP and the classical SVM in our data exploration.

5.2.1 Relation of the ILP to Support Vector Classification

In the course of our formulation of the ILP with T sets, we discovered that the formulation of this newly-introduced extension is structurally similar to the classical Support Vector Machine (SVM) [105–107]. Therefore, in addition to the ILP, we consider a hybridized ILP-SVM approach to solve the problem at hand, which often obtains better results while being more computationally efficient.

The Support Vector Machine (also known as Support Vector Classifier) is a popular supervised learning method used for classification tasks [105, 106, 140, 141]. In the two-class SVM setting, the goal is to find a maximum margin hyperplane that separates the data classes. Once the hyperplane has been determined, new data can be classified, based on the side of the hyperplane on which it falls. There are many settings in which the SVM is a useful tool, including settings in which the data contains multiple features, or where the margin of separation in the data is very clear. Examples of areas in which SVM has been a useful tool include text and hypertext categorization (where the number of labeled training examples can be significantly reduced by using SVM) [142], classification of images [143], classification of satellite data (for example, using supervised SVM) [144] and many more [145–147].

Recall that the ILP that we are trying to solve is trying to minimize

$$\sum_{v \notin S} x_v, \quad \text{and} \quad \sum_{y \notin C \setminus \{S \cup T\}} y_v.$$

In other words, we seek

$$\min \Big(\sum_{v \notin \{S \cup T\}} x_v + \sum_{v \notin \{S \cup T\}} y_v \Big) \tag{5.1}$$

subject to the following: $x_v \in \{0,1\}$ and $y_v \in \{0,1\}$, and

$$\text{for all } v_i \in S, v_\ell \in V \setminus (S \cup T), \quad \sum_{j=1}^{k} \alpha_j d_{ji} \leq \sum_{j=1}^{k} \alpha_j d_{j\ell} + M x_{v_\ell} \tag{5.2}$$

$$\text{for all } v_i \in T, v_\ell \in V \setminus (S \cup T), \quad \sum_{j=1}^{k} \alpha_j d_{j\ell} - M y_{v_\ell} \leq \sum_{j=1}^{k} \alpha_j d_{ji} \tag{5.3}$$

where $M := \max_{i,j} d_{ji}$.

5.2.1.1 SVM

Given training data $\{(x_i, y_i)\}_{i=1}^n$, a soft-margin SVM is defined by the following objective function:

$$\begin{aligned} &\min_{\vec{w},b,\zeta} \frac{\|\vec{w}\|_2^2}{2} + C\sum_{i=1}^{n} \zeta_i \\ &\text{s.t. } y_i(\vec{w}^T x_i - b) \geq 1 - \zeta_i \text{ for all } i \in [n] \\ &\quad \zeta_i \geq 0 \text{ for all } i \in [n] \end{aligned} \tag{5.4}$$

The hyperparameter $C \geq 0$ is often tuned via cross-validation. We next cast the ILP above in a similar form to the classical SVM, to tease out a deeper connection between the two methods. Considering the "training data" here as

$$x_\ell^i = d_{\bullet,\ell} - d_{\bullet,i},\ y_\ell^i = 1 \text{ for } v_i \in S, v_\ell \in V \setminus S \cup T$$

$$x_\ell^i = d_{\bullet,\ell} - d_{\bullet,i},\ y_\ell^i = -1 \text{ for } v_i \in T, v_\ell \in V \setminus S \cup T$$

The penalty variables are then defined via

$$\zeta_\ell^i = x_\ell \text{ for } v_i \in S, v_\ell \in V \setminus \{S \cup T\}$$

$$\zeta_\ell^i = y_\ell \text{ for } v_i \in T, v_\ell \in V \setminus \{S \cup T\}$$

We can then write the ILP as

$$\begin{aligned}
&\min_{\xi} \Big(\sum_{i,\ell} \xi_\ell^i \Big) \\
&\text{s.t. } y_\ell^i \left(\vec{\alpha} x_\ell^i \right) \geq -M \xi_\ell^i \text{ for all } i \in S \cup T, \ell \in V \setminus \{S \cup T\} \\
&\quad \xi_\ell^i \in \{0, 1\} \text{ for all } i \in S \cup T, \ell \in V \setminus \{S \cup T\} \\
&\quad \vec{\alpha} \geq 0, \quad \sum_i \alpha_i = 1
\end{aligned}$$

We can relax the above ILP formulation to obtain an "approximate ILP" (ALP) setting with less affine (here, essentially normalizing) constraints on α. We do this by adding an $\|\alpha\|_2^2$ term to the above minimization (acting as a normalizing/regularizing term on the entries of α_i) and removing the nonnegativity condition on the α_i's. We can then write the ALP as:

$$\begin{aligned}
&\min_{\xi} \Big(\sum_{i,\ell} \xi_\ell^i + \|\alpha\|_2^2 \Big) \\
&\text{s.t. } y_\ell^i \left(\vec{\alpha} x_\ell^i \right) \geq -M \xi_\ell^i \text{ for all } i \in S \cup T, \ell \in V \setminus \{S \cup T\} \\
&\quad \xi_\ell^i \in \{0, 1\} \text{ for all } i \in S \cup T, \ell \in V \setminus \{S \cup T\}
\end{aligned}$$

5.2.2 Simulations comparing ILP and ALP

We further investigate the relationship between the ILP and the ALP by running experiments similar to those of Chapter 3. Specifically, we compare the algorithmic performance of our novel ILP method (denoted as *ilp + T* in Figures 5.1 and 5.2) to the original ILP method and the graph and node covariates separately, as was done in the previous experiments of Chapter 3 (see Figures 3.4 and 3.5). We further compare these results to the ALP method described in Section 5.2.1.1. To computationally solve the ALP problem, we here use the Nelder-Mead algorithm: one of the most commonly used direct search methods to solve optimization functions. In general terms, this algorithm approximates the solution of the objective function by iteratively producing a sequence of simplices (see [148] for a more detailed description). Here, we call the *Nelder-Mead* method when using the *minimize* function in Python's SciPy library.

In Figure 5.1 we consider the initial S set containing 3 randomly chosen elements of S^* and 3 randomly chosen elements of T^*. T initially contains only one randomly chosen element of T^*. We run 100 Monte Carlo iterations and compute the precision at 15. As can be seen in the figure, our novel ILP algorithm (ilp + T) seems to be performing best after a single round of human-in-the-loop retraining. We consider a further example where the initial S set contains 4 randomly chosen elements of S^* and T initially contains only 3 randomly chosen element of T^* (see Figure 5.2). We again run 100 Monte Carlo iterations and compute the precision at 15. In this experiment, the original ILP algorithm seems to be performing best at the start, but our new method (ilp + T) 'catches up' and performs as well as the old method by the third round of the human-in-the-loop. Both methods seem to outperform the ALP here, though in ongoing work we see the potential of the ALP to outperform the ILP, especially in high dimensional data settings.

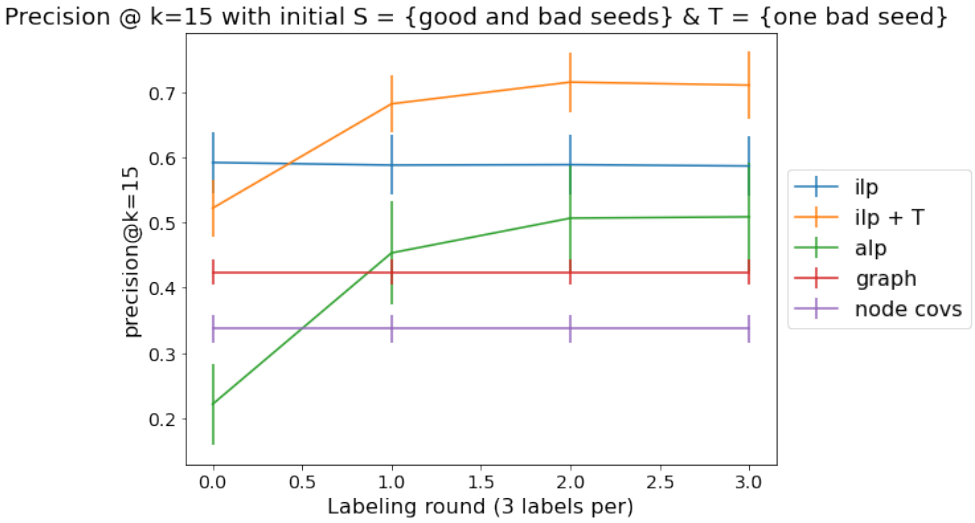

Figure 5.1: With the same network and experimental setup as in Chapter 3, in this experiment, the initial S set contains 3 randomly chosen elements of S^* and 3 randomly chosen elements of T^*. T initially contains only one randomly chosen element of T^*. We run 100 Monte Carlo iterations and compute the precision at 15; we plot the mean precision with error bars $\pm$2se. As can be seen, our novel ILP algorithm (ilp + T) seems to be performing best.

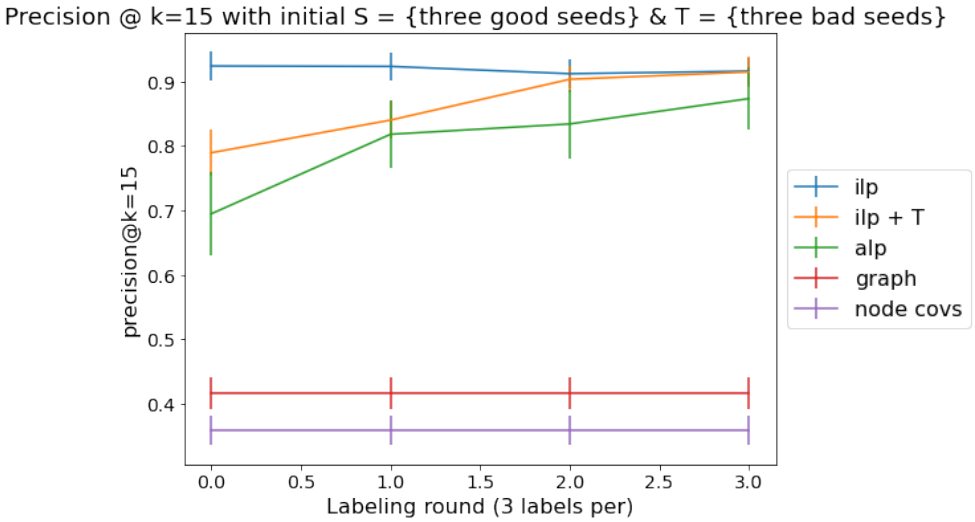

Figure 5.2: With the same network and experimental setup as in Chapter 3, in this experiment, the initial S set contains 4 randomly chosen elements of S^* and T initially contains only 3 randomly chosen element of T^*. We run 100 Monte Carlo iterations and compute the precision at 15; we plot the mean precision with error bars $\pm$2se. In this experiment, the original ILP algorithm seems to be performing best at the start, but our new method (ilp + T) 'catches up' and performs as well as the old method by the third round of the human-in-the-loop. Both methods seem to outperform the ALP.

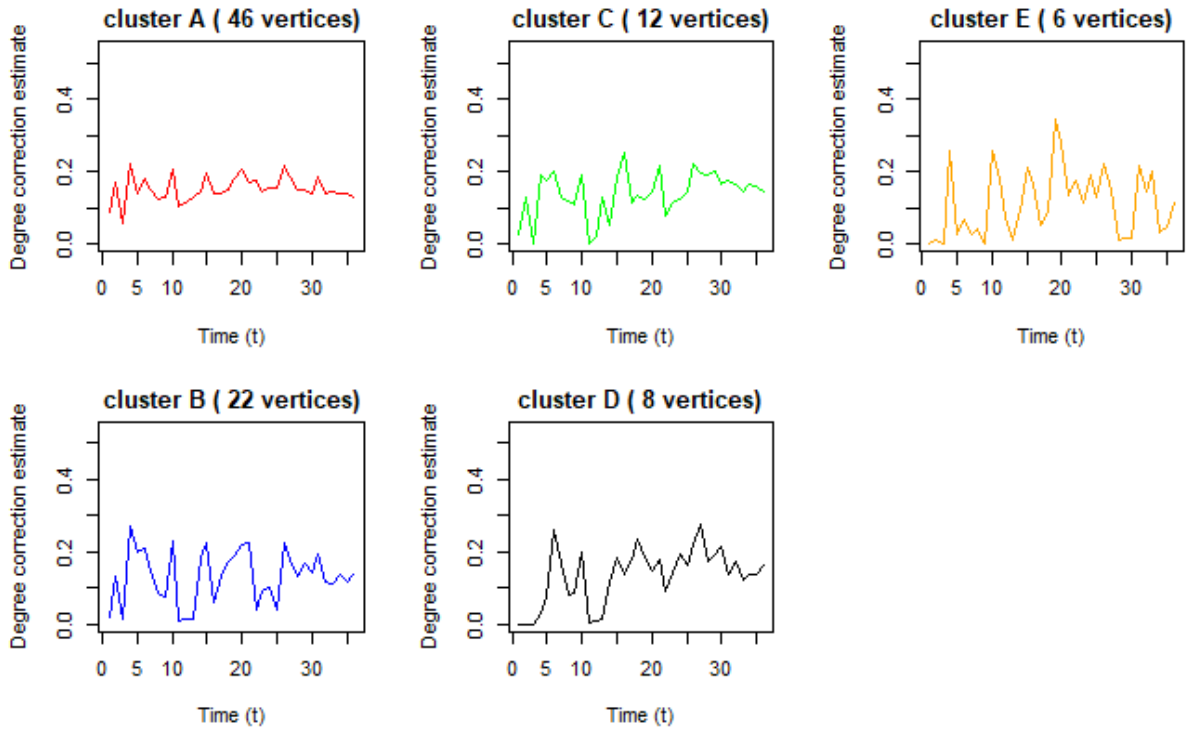

Figure 5.3: The mean trajectory for each cluster of neuronal degree corrections, when we consider 5 clusters in our FDC algorithm. In this new setting, stimuli are applied more frequently (specifically at 5, 12, 19, 26, 33 and 40 minutes, which correspond to the points 3.8, 9.2, 14.6 , 17.6 , 25.3 , 30.7 on the x-axis of the plot). In addition, the stimuli at 33-minutes (25.3 on the plot) was more intense than the other ones.

5.3 Future work on time-series signal detection

In our work using the DCRDPG model to analyze embeddings of neuron spike trains, our goal is to further understand the biological relevance of our methodology. To this end, we have compared our results to consensus clustering [134–136], a tool commonly used by neuroscientists to cluster and analyse the neurons in the spike-train data under consideration. While our obtained clusters capture signal common to the consensus clusterings, our method also sheds light on neuron behaviour that the consensus clustering does not necessarily pick up. Work is currently underway with neuroscience colleagues to better understand the biological relevance

of our obtained clusters especially as compared to the better understood consensus clusters.

Further work in this area includes analyzing a setting where the strength and number of stimuli applied changes in order to better understand how this changes our clustering algorithm's effectiveness. More specifically, we consider data from experiments where in contrast to our earlier work, the recording time is longer (47 minutes) and stimuli are applied more frequently and with varying intensity (specifically at 5, 12, 19, 26, 33 and 40 minutes, which correspond to the points 3.8, 9.2, 14.6, 17.6, 25.3, 30.7 on the x-axis of the plot). In addition, one of the stimuli was of higher intensity than the rest, namely the one at 33-minutes (25.3 on the plot). Initial work in this new setting includes plotting the mean trajectory for each cluster of neuronal degree corrections, when we again consider 5 clusters in the FDC algorithm (see Figure 5.3) in order to better understand the sensitivity of our algorithm to detecting stimuli level rather than just the presence of stimuli. We plan to further explore this setting by comparing the clustering to the corresponding consensus clustering results, as was done in the previous setting. Furthermore, given the difference in intensity of stimuli, we consider different methods in order to perform signal correction. Some of the most common methods to explore would be filtering techniques, including low and high pass filters, smoothing techniques such as Savitzky–Golay filter, moving average or exponential smoothing and normalization techniques to name a few [149].

Furthermore, we would like to apply more formal time-series modeling (e.g., ARMA and ARIMA models [150]) to our degree-correction time-series data. This would allow us to formalize, within the context of the time-series models, methods for both for anomaly detection and change-point detection, as well as developing methodology for testing whether a detected stimuli is an anomaly, a change-point or neither. Furthermore, we would like to develop formal methods to test whether after a stimuli the time-series has returned to its steady-state/resting-state

behavior or if sensitization has occurred.

Bibliography

[1] Josh A Firth, Joel Hellewell, Petra Klepac, Stephen Kissler, Adam J Kucharski, and Lewis G Spurgin. Using a real-world network to model localized covid-19 control strategies. *Nature medicine*, 26(10):1616–1622, 2020.

[2] V. Lyzinski, K. Levin, and C. E. Priebe. On consistent vertex nomination schemes. *Journal of Machine Learning Research*, 20(69):1–39, 2019.

[3] J. Agterberg, Y. Park, J. Larson, C. White, C. E. Priebe, and V. Lyzinski. Vertex nomination, consistent estimation, and adversarial modification. *Electronic Journal of Statistics*, 14(2):3230–3267, 2020.

[4] Evan S Hill, Jeffrey W Brown, and William N Frost. Photodiode-based optical imaging for recording network dynamics with single-neuron resolution in non-transgenic invertebrates. *JoVE (Journal of Visualized Experiments)*, (161):e61623, 2020.

[5] A. L. Traud, P. J Mucha1, and M. A. Porter. Social structure of facebook networks. *arXiv preprint arXiv:1102.2166v1*, 2011.

[6] A. Mele. A structural model of segregation in social networks. *doi:10.1920/wp.cem.2010.3210*, 2013.

[7] K Eichler, F Li, AL Kumar, Y Park, I Andrade, C Schneider-Mizell, T Saumweber, A Huser, D Bonnery, B Gerber, et al. The complete wiring diagram of a high-order learning and memory center, the insect mushroom body. *Nature*, 548(175-182):23, 2017.

[8] E. D. Kolaczyk. *Statistical Analysis of Network Data: Methods and Models*. Springer, 2009.

[9] A. Athreya, D. E. Fishkind, M. Tang, C. E. Priebe, Y. Park, J. T. Vogelstein, K. Levin, V. Lyzinski, and Y. Qin. Statistical inference on random dot product graphs: a survey. *Journal of Machine Learning Research*, 18(1):8393–8484, 2017.

[10] J. Cape, M. Tang, and C. E. Priebe. The two-to-infinity norm and singular subspace geometry with applications to high-dimensional statistics. *Annals of Statistics*, 47(5):2405–2439, 2019.

[11] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[12] M. A. Mugahwi, O. De la Cruz Cabrera, S. Noschese, and L. Reichel. Functions and eigenvectors of partially known matrices with applications to network analysis. *Applied Numerical Mathematics*, 159:93–105, 2021.

[13] F. R. K Chung. *Spectral Graph Theory*. American Mathematical Society, 2017.

[14] R. Singh, A. Chakraborty, and B. S. Manoj. Graph fourier transform based on directed laplacian. *arXiv preprint https://arxiv.org/pdf/1601.03204.pdf*, 2016.

[15] E. D. Kolaczyk and G. Csárdi. *Statistical analysis of network data with R*, volume 65. Springer, 2014.

[16] P. Louis, S. A. Jacob, and A. Salehi-Abari. Simplifying subgraph representation learning for scalable link prediction. *arXiv preprint https://arxiv.org/abs/2301.12562*, 2023.

[17] X. Wang, H. Zhao, L. Wei, and Q. Yao. Graph property prediction on open graph benchmark: A winning solution by graph neural architecture search. *arXiv preprint https://arxiv.org/pdf/2207.06027.pdf*, 2022.

[18] B. Bollobás. Random graphs. 2001.

[19] P. Erdős. Graph theory and probability. *Canadian Journal of Mathematics*, 1959.

[20] P. Erdős and A. Rényi. On the evolution of random graphs. *Transactions of the American Mathematical Society*, 1984.

[21] C. McDiarmid and F. Skerman. Modularity of erdős–rényi random graphs. *arXiv preprint https://arxiv.org/pdf/1808.02243.pdf*, 2019.

[22] L. Erdős, A. Knowles, H. Yau, and J. Yin. Spectral statistics of erdős–rényi grpahs i: Local semicircle law. *The Annals of Probability*, 41, 2019.

[23] E. Hiesmayr and T. McKenzie. The spectral edge of constant degree erdős–rényi graphs. *https://arxiv.org/pdf/2309.11007.pdf*, 2013.

[24] A. Sealfon and J. Ullman. Efficiently estimating erdős–rényi graphs with node differential privacy. *https://arxiv.org/pdf/1905.10477.pdf*, 2019.

[25] D. E. Fishkind, S. Adali, H. G. Patsolic, L. Meng, D. Singh, V. Lyzinski, and C. E. Priebe. Seeded graph matching. *Pattern Recognition*, 87:203–215, 2019.

[26] V. Lyzinski, D. E. Fishkind, and C. E. Priebe. Seeded graph matching for correlated erdos-renyi graphs. *arXiv preprint https://arxiv.org/pdf/1304.7844v1.pdf*, 2013.

[27] C. Lee and D. J. Wilkinson. A review of stochastic block models and extensions for graph clustering. *arXiv preprint https://arxiv.org/abs/1903.00114*, 2019.

[28] P. W. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5:109–137, 1983.

[29] B. Karrer and M. E. J. Newman. Stochastic blockmodels and community structure in networks. *Physical Review*, 83, 2011.

[30] V. Lyzinski, M. Tang, A. Athreya, Y. Park, and C. E. Priebe. Community detection and classification in hierarchical stochastic blockmodels. *arXiv preprint https://arxiv.org/pdf/1503.02115.pdf*, 2016.

[31] P. W. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[32] B. Karrer and M. E. J. Newman. Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1):016107, 2011.

[33] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 2008.

[34] V. Lyzinski, M. Tang, A. Athreya, Y. Park, and C. E. Priebe. Community detection and classification in hierarchical stochastic blockmodels. *IEEE Transactions on Network Science and Engineering*, 4(1):13–26, 2016.

[35] Tiago P Peixoto. Hierarchical block structures and high-resolution model selection in large networks. *Physical Review X*, 4(1):011047, 2014.

[36] K. Rohe, S. Chatterjee, and B. Yu. Spectral clustering and the high-dimensional stochastic blockmodel. *Annals of Statistics*, 39(4):1878–1915, 2011.

[37] D. L. Sussman, M. Tang, D. E. Fishkind, and C. E Priebe. A consistent adjacency spectral embedding for stochastic blockmodel graphs. *Journal of the American Statistical Association*, 107(499):1119–1128, 2012.

[38] Y. Zhao, E. Levina, and J. Zhu. Consistency of community detection in networks under degree-corrected stochastic block models. *Annals of Statistics*, 40(4):2266–2292, 2012.

[39] A. A. Amini, A. Chen, P. J. Bickel, and E. Levina. Pseudo-likelihood methods for community detection in large sparse networks. *Annals of Statistics*, 41(4):2097–2122, 2013.

[40] M. E. J. Newman. Equivalence between modularity optimization and maximum likelihood methods for community detection. *Physical Review E*, 94(5):052315, 2016.

[41] P. J. Bickel and P. Sarkar. Hypothesis testing for automated community detection in networks. *Journal of the Royal Statistical Society, Series B*, pages 253–273, 2016.

[42] J. Lei. A goodness-of-fit test for stochastic block models. *Annals of Statistics*, 44(1):401–424, 2016.

[43] S. C. Ohlede and P. J. Wolfe. Network histograms and universality of blockmodel approximation. *Proceedings of the National Academy of Sciences*, 111(41):14722–14727, 2014.

[44] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9:1981–2014, 2008.

[45] D. Moyer, B. Gutman, G. Prasad, G. andVer Steeg, and P. Thompson. Mixed membership stochastic blockmodels for the human connectome. *https://dcmoyer.github.io/pubs/mixed-membership-stochastic.pdf*, Accessed: 2/14/24.

[46] V. Lyzinski, M. Tang, A. Athreya, Y. Park, and C. E. Priebe. Community detection and classification in hierarchical stochastic blockmodels. *arXiv preprint: https://arxiv.org/pdf/1503.02115v3.pdf*, 2015.

[47] S. J. Young and E. R. Scheinerman. Random dot product graph models for social networks. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 138–149. Springer, 2007.

[48] P. Rubin-Delanchy, J. Cape, M. Tang, and C. E. Priebe. A statistical interpretation of spectral embedding: the generalised random dot product graph. *Journal of the Royal Statistical Society, Series B*, 2022+.

[49] M. Fiori, B. Marenco, F. Larroca, P. Bermolen, and G. Mateos. Gradient-based spectral embeddings of random dot product graphs. *arXiv preprint https://arxiv.org/abs/2307.13818*, 2023.

[50] V. Lyzinski, D. Sussman, M. Tang, A. Athreya, and C. Priebe. Perfect clustering for stochastic blockmodel graphs via adjacency spectral embedding. *Electronic Journal of Statistics*, 8(2):2905–2922, 2014.

[51] D. L. Sussman, M. Tang, and C. E. Priebe. Consistent latent position estimation and vertex classification for random dot product graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36:48–57, 2014.

[52] M. Tang, D. L. Sussman, and C. E. Priebe. Universally consistent vertex classification for latent positions graphs. *Annals of Statistics*, 41(3):1406–1430, 2013.

[53] M. Tang, A. Athreya, D. L. Sussman, V. Lyzinski, Y. Park, and C. E. Priebe. A semiparametric two-sample hypothesis testing problem for random graphs. *Journal of Computational and Graphical Statistics*, 26(2):344–354, 2017.

[54] M. Tang, A. Athreya, D. L. Sussman, V. Lyzinski, and C. E. Priebe. A nonparametric two-sample hypothesis testing problem for random graphs. *Bernoulli*, 23(3):1599–1630, 2017.

[55] A. A. Alyakin, J. Agterberg, H. S. Helm, and C. E. Priebe. Correcting a nonparametric two-sample graph hypothesis test for graphs with different numbers of vertices. *arXiv preprint arXiv:2008.09434*, 2020.

[56] J. Agterberg, M. Tang, and C. Priebe. Nonparametric two-sample hypothesis testing for random graphs with negative and repeated eigenvalues. *arXiv preprint arXiv:2012.09828*, 2020.

[57] H. G. Patsolic, Y. Park, V. Lyzinski, and C. E. Priebe. Vertex nomination via seeded graph matching. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 13(3):229–244, 2020.

[58] R. Zheng, V. Lyzinski, C. E. Priebe, and M. Tang. Vertex nomination between graphs via spectral embedding and quadratic programming. *Journal of Computational and Graphical Statistics*, pages 1–15, 2022.

[59] P. D. Hoff, A. E. Raftery, and M. S. Handcock. Latent space approaches to social network analysis. *Journal of the American Statistical Association*, 97(460):1090–1098, 2002.

[60] J. Agterberg, M. Tang, and C. E. Priebe. On two distinct sources of nonidentifiability in latent position random graph models. *arXiv preprint arXiv:2003.14250*, 2020.

[61] M. Tang and C. E. Priebe. Limit theorems for eigenvectors of the normalized Laplacian for random graphs. *Annals of Statistics*, 46(5):2360–2415, 2018.

[62] M. Zhu and A. Ghodsi. Automatic dimensionality selection from the scree plot via the use of profile likelihood. *Computational Statistics & Data Analysis*, 51(2):918–930, 2006.

[63] K. Levin, C. E. Priebe, and V. Lyzinski. Vertex nomination in richly attributed networks. *arXiv preprint https://arxiv.org/abs/2005.02151v3*, 2023.

[64] D. E. Fishkind, V. Lyzinski, H. Pao, L. Chen, and C. E. et al. Priebe. Vertex nomination schemes for membership prediction. *The Annals of Applied Statistics*, 9(3):1510–1532, 2015.

[65] D. E. Fishkind, V. Lyzinski, H. Pao, L. Chen, and C. E. Priebe. Vertex nomination schemes for membership prediction. *Annals of Applied Statistics*, 9(3):1510–1532, 09 2015.

[66] G. A. Coppersmith and C. E. Priebe. Vertex nomination via content and context. *arXiv preprint arXiv:1201.4118*, 2012.

[67] G. Coppersmith. Vertex nomination. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(2):144–153, 2014.

[68] D. Marchette, C. E. Priebe, and G. Coppersmith. Vertex nomination via attributed random dot product graphs. In *Proceedings of the 57th ISI World Statistics Congress*, volume 6, 2011.

[69] H. S. Helm, A. Basu, A. Athreya, Y. Park, J. T. Vogelstein, M. Winding, M. Zlatic, A. Cardona, P. Bourke, J. Larson, C. White, and C. E. Priebe. Learning to rank via combining representations. *arXiv preprint arXiv:2005.10700v2*, 2020.

[70] V. Lyzinski, K. Levin, D. E. Fishkind, and C. E. Priebe. On the consistency of the likelihood maximization vertex nomination scheme: Bridging the gap between maximum likelihood estimation and graph matching. *Journal of Machine Learning Research*, 17(1):6206–6239, 2016.

[71] J. Yoder, Li C., H. Pao, E. Bridgeford, K. Levin, D. Fishkind, C. Priebe, and V. Lyzinski. Vertex nomination: The canonical sampling and the extended spectral nomination schemes. *Computational Statistics & Data Analysis*, 145:106916, 2020.

[72] D. S. Lee and C. E. Priebe. Bayesian vertex nomination. *arXiv:1205.5082v1*, 2012.

[73] Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media, 2013.

[74] Sheyda Peyman, Minh Tang, and Vince Lyzinski. Adversarial contamination of networks in the setting of vertex nomination: a new trimming method. *arXiv preprint arXiv:2208.09710*, 2022.

[75] Hayden S Helm, Amitabh Basu, Avanti Athreya, Youngser Park, Joshua T Vogelstein, Carey E Priebe, Michael Winding, Marta Zlatic, Albert Cardona, Patrick Bourke, et al. Distance-based positive and unlabeled learning for ranking. *Pattern Recognition*, 134:109085, 2023.

[76] J. Li, T. Xie, L. Chen, F. Xie, X. He, and Z. Zheng. Adversarial attack on large scale graph. *arXiv:2009.03488v2*, 2021.

[77] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song. Adversarial attack on graph structured data. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1115–1124, 2018.

[78] W. Jin, Y. Li, H. Xu, Y. Wang, Sh. Ji, Ch. Aggarwal, and J. Tang. Adversarial attacks and defenses on graphs: A review, a tool and empirical studies. *ACM SIGKDD Explorations Newsletter Volume 22 Issue 2 December, pp 19*, 2020.

[79] P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[80] J. Jia, B. Wang, X. Cao, and N. Z. Gong. Certified robustness of community detection against adversarial structural perturbation via randomized smoothing. In *Proceedings of The Web Conference 2020*, pages 2718–2724, 2020.

[81] T. T. Cai and X. Li. Robust and computationally feasible community detection in the presence of arbitrary outlier nodes. *Annals of Statistics*, 43(3):1027–1059, 2015.

[82] D. Zügner, A. Akbarnejad, and S. Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2847–2856, 2018.

[83] N. Entezari, S. A. Al-Sayouri, A. Darvishzadeh, and E. E. Papalexakis. All you need is low (rank) defending against adversarial attacks on graphs. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 169–177, 2020.

[84] D. Edge, J. Larson, M. Mobius, and C. White. Trimming the hairball: Edge cutting strategies for making dense graphs usable. In *IEEE International Conference on Big Data*, pages 3951–3958, 2018.

[85] S. M. Stigler. The asymptotic distribution of the trimmed mean. *Annals of Statistics*, 1(3):472–477, 1973.

[86] P. J. Huber. *Robust statistics*. John Wiley & Sons, 2004.

[87] P. H. Schonemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1—-10, 1966.

[88] L. Scrucca, M. Fop, T. B. Murphy, and A. E. Raftery. mclust 5: Clustering, classification and density estimation using gaussian finite mixture models. *The R Journal*, 8/1:289–317, 2016.

[89] C. M. Le, E. Levina, and R. Vershynin. Concentration and regularization of random graphs. *Random Structures & Algorithms*, 51(3):538–561, 2017.

[90] J. Chung, B. Varjavand, J. Arroyo, A. Alyakin, J. Agterberg, M. Tang, C. E. Priebe, and J. T. Vogelstein. Valid two-sample graph testing via optimal transport procrustes and multiscale graph correlation with applications in connectomics. *Stat*, 11, 2022.

[91] R. Mastrandrea, J. Fournet, and A. Barrat. Contact patterns in a high school: a comparison between data collected using wearable sensors, contact diaries and friendship surveys. *PLOS One*, 10 e0136497, 2015.

[92] R. M. Lawrence, E. W. Bridgeford, P. E. Myers, G. C. Arvapalli, S. C. Ramachandran, D. A. Pisner, P. F. Frank, A. D. Lemmer, A. Nikolaidis, and J. T. Vogelstein. Standardizing human brain parcellations. *Scientific Data, Volume 8, Article number: 78*, 2021.

[93] L. A Adamic and N. Glance. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43, 2005.

[94] L. A. Adamic and Glance N. The political blogosphere and the 2004 u.s. election: Divided they blog. *Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem*, 2005.

[95] M. Tang, J. Cape, and C. E. Priebe. Asymptotically efficient estimators for stochastic blockmodels: The naive MLE, the rank-constrained MLE, and the spectral. *Bernoulli*, 28:1049–1073, 2022.

[96] D. E. Fishkind, V. Lyzinski, H. Pao, L. Chen, and C. E. et al. Priebe. Letor: A benchmark collection for research on learning to rank for information retrieval. *Inf. Retr.*, 13(4):346–374, 2010.

[97] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. *ser. Proceedings of Ma- chine Learning Research*, 14, 2011.

[98] H. S. Helm, M. Abdin, B. D. Pedigo, S. Mahajan, V. Lyzinski, Y. Park, A. Basu, P. Choudhury, C. M. White, W. Yang, and C. E. Priebe. Leveraging semantically similar queries for ranking via combining representations. *arXiv preprint https://arxiv.org/pdf/2106.12621.pdf*, 2021.

[99] R. Manrique, F. Cueto-Ramirez, and O. Mariño. Comparing graph similarity measures for semantic representations of documents. *CCIS*, 885, 2018.

[100] M. Roy, S. Schmid, and G. Tredan. Modeling and measuring graph similarity: The case for centrality distance. *arXiv preprint https://arxiv.org/abs/1406.5481*, 2014.

[101] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[102] M. Wang and W. Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312(10):135–153, 2018.

[103] Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[104] J. T. Vogelstein, H. S. Helm, R. D. Mehta, J. Dey, W. Levine, W. Yang, B. Tower, J. Larson, C. White, and C. E. Priebe. A general approach to progressive learning. 2020.

[105] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez. A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, 408:189–215, 2020.

[106] R. Hu, X. Zhu, Y. Zhu, and J. Gan. Robust svm with adaptive graph learning. *https://doi.org/10.1007/s11280-019-00766-x*, 2020.

[107] M. Awad and R. Khanna. Support vector machines for classification. *Efficient Learning Machines*, pages 39–66, 2015.

[108] W. Czaja and M. Ehler. Schroedinger eigenmaps for the analysis of biomedical data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35, 2013.

[109] J. Benedetto, W. Czaja, J. Dobrosotskaya, T. Doster, K. Duke, and D. Gllis. Semi-supervised learning of heterogeneous data in remote sensing imagery. *Independent Component Analyses, Compressive Sampling, Wavelets, Neural Net, Biosystems, and Nanoengineering X*, 8401:34–45, 2012.

[110] Czaja W. Cahill, N. D. and D. W. Messinger. Schroedinger eigenmaps with nondiagonal potentials for spatial-spectral clustering of hyperspectral imagery. *Algorithms and technologies for multispectral, hyperspectral, and ultraspectral imagery*, 9088:27–39, 2014.

[111] J. J. Benedetto, W. Czaja, J. Dobrosotskaya, T. Doster, and K. Duke. Spatial-spectral operator theoretic methods for hyperspectral image classification. *Algorithms and technologies for multispectral, hyperspectral, and ultraspectral imagery*, 9088:275–297, 2016.

[112] Jordan Yoder, Li Chen, Henry Pao, Eric Bridgeford, Keith Levin, Donniell E Fishkind, Carey Priebe, and Vince Lyzinski. Vertex nomination: The canonical sampling and the extended spectral nomination schemes. *Computational Statistics & Data Analysis*, 145:106916, 2020.

[113] M. Sun, M. Tang, and C. E. Priebe. A comparison of graph embedding methods for vertex nomination. *International Conference on Machine Learning and Applications*, 1:398–403, 2012.

[114] Vince Lyzinski, Keith Levin, and Carey E Priebe. On consistent vertex nomination schemes. *Journal of Machine Learning Research*, 20(69), 2019.

[115] Eduardo Mosqueira-Rey, Elena Hernández-Pereira, David Alonso-Ríos, José Bobes-Bascarán, and Ángel Fernández-Leal. Human-in-the-loop machine learning: a state of the art. *Artificial Intelligence Review*, 56(4):3005–3054, 2023.

[116] X. Wu, L. Xiao, Y. Sun, J. Zhang, T. Ma, and L. He. A survey of human-in-the-loop for machine learning. *arXiv preprint: https://arxiv.org/pdf/2108.00941.pdf*.

[117] D. Ustalov, N. Fedorova, and N. Pavlichenko. Improving recommender systems with human-in-the-loop. *Proceedings of the 16th ACM Conference on Recommender Systems*, page 708–709, 2022.

[118] S. Brenner. The genetics of caenhorhabditis elegans. *Genetics*, 77:71–94, 1974.

[119] Keith Levin, Avanti Athreya, Minh Tang, Vince Lyzinski, Youngser Park, and Carey E Priebe. A central limit theorem for an omnibus embedding of multiple random graphs and implications for multiscale network inference. *arXiv preprint arXiv:1705.09355*, 2017.

[120] K. Levin, A. Athreya, M. Tang, V. Lyzinski, Y. Park, and C. E. Priebe. A central limit theorem for an omnibus embedding of multiple random graphs and implications for multiscale network inference. *arXiv preprint: https://arxiv.org/pdf/1705.09355.pdf*, 2019.

[121] Catherine S Cutts and Stephen J Eglen. Detecting pairwise correlations in spike trains: an objective comparison of methods and application to the study of retinal waves. *Journal of Neuroscience*, 34(43):14288–14303, 2014.

[122] Sourav Chatterjee. Matrix estimation by Universal Singular Value Thresholding. *The Annals of Statistics*, 43(1):177 – 214, 2015.

[123] Konstantinos Pantazis, Avanti Athreya, Jesús Arroyo, William N Frost, Evan S Hill, and Vince Lyzinski. The importance of being correlated: Implications of dependence in joint spectral inference across multiple networks. *The Journal of Machine Learning Research*, 23(1):6297–6373, 2022.

[124] Ch. Genolini and B. Falissard. Kml: k-means for longitudinal data. *computational Statistics*, 25:317–328, 2010.

[125] M. Zhang and A. Parnell. Review of clustering methods for functional data. *arXiv preprint https://arxiv.org/pdf/2210.00847.pdf*, 2022.

[126] Daniel L Sussman, Minh Tang, Donniell E Fishkind, and Carey E Priebe. A consistent adjacency spectral embedding for stochastic blockmodel graphs. *Journal of the American Statistical Association*, 107(499):1119–1128, 2012.

[127] P. Rubin-Delanchy, J. Cape, M. Tang, and C. E. Priebe. A statistical interpretation of spectral embedding: the generalised random dot product graph. *arXiv preprint arXiv:1709.05506v5*, 2021.

[128] H. Müller. Functional data analysis. *International Encyclopedia of Statistical Science*, pages 554–555, 2014.

[129] J. Jacques and C. Preda. Functional data clustering: a survey. *Advances in Data Analysis and Classification*, 8:231–255, 2014.

[130] F. Centofanti, A. Lepore, and B. Palumbo. Sparse and smooth functional data clustering. *arXiv preprint https://arxiv.org/pdf/2103.15224.pdf*, 2022.

[131] R. Wu, B. Wang, and A. Xu. Functional data clustering using principal curve methods. *Communications in Statistics - Theory and Methods*, 51:7264–7283, 2021.

[132] C. Genolini and B. Falissard. Kml: a package to cluster longitudinal data. *Computer methods and programs in biomedicine*, 104:112–121, 2011.

[133] Overview: K-means for longitudinal data. *https://search.r-project.org/CRAN/refmans/kml/html/kml-package.html*.

[134] A. Strehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partition. *Journal of Machine Learning Research*, 2002.

[135] A. Topchy, A. K. Jain, and W. Punch. Clustering ensembles: models of consensus and weak partitions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27, 2005.

[136] A. Goder and V. Filkov. Consensus clustering algorithms: Comparison and refinement. *2008 Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2008.

[137] A Lancichinetti and S. Fortunato. Consensus clustering in complex networks. *arXiv preprint https://arxiv.org/pdf/1203.6093.pdf*, 2012.

[138] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas. Comparing community structure identification. *arXiv preprint https://arxiv.org/pdf/cond-mat/0505245.pdf*, 2005.

[139] H. Kwak, Y. Eom, Y. Choi, H. Jeong, and S. Moon. Consistent community identification in complex networks. *arXiv preprint https://arxiv.org/pdf/0910.1508.pdf*, 2009.

[140] M. Ardeshir, C. Sanford, and D. Hsu. Support vector machines and linear regression coincide with very high-dimensional features. *arXiv preprint https://arxiv.org/pdf/2105.14084.pdf*, 2021.

[141] P. Corcoran. An end-to-end graph convolutional kernel support vector machine. *Applied Network Science*, 5, 2020.

[142] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. *Machine Learning: ECML*, 98:137–142, 1998.

[143] L. Barghout. Spatial-taxon information granules as used in iterative fuzzy-decision-making for image segmentation. *Studies in Big Data*, 10:285–318, 2015.

[144] A. Maity. Supervised classification of radarsat-2 polarimetric data for different land features. *arXiv preprint https://arxiv.org/abs/1608.00501*, 2016.

[145] R. Cuingnet, C. Rosso, M. Chupin, S. Lehéricy, D. Dormont, H. Benali, Y. Samson, and O. Colliot. patial regularization of svm for the detection of diffusion alterations associated with stroke outcome. *Medical Image Analysis*, 1, 2011.

[146] B. Gaonkar and C. Davatzikos. Analytic estimation of statistical significance maps for support vector machine based multi-variate image analysis and classification. *NeuroImage*, 78:270–283, 2013.

[147] A. Statnikov, D. Hardin, and C. Aliferis. Using svm weight-based methods to identify causally relevant and non-causally relevant variables. 2006.

[148] F. Gao and L. Han. Implementing the nelder-mead simplex algorithm with adaptive parameters. *Computational Optimization and Applications*, 51:1:259–277, 2012.

[149] E. Isufi, F. Gama, D. I. Shuman, and S. Segarra. Graph filters for signal processing and machine learning on graphs. *arXiv preprint: https://arxiv.org/pdf/2211.08854.pdf*, 2024.

[150] James D Hamilton. *Time series analysis*. Princeton university press, 2020.