LOW RANK METHODS FOR NETWORK ALIGNMENT

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Huda Nassar

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL

STATEMENT OF DISSERTATION APPROVAL

To my nieces and nephews:

Tia Nassar, Rena Nassar, Yara Nassar, Mohammad Nassar, Helen Nassar, Mohammad Nassar, Lea El-Agha, Angela Nassar, Judy El-Agha.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Nassar, Huda PhD, Purdue University, August 2019. Low Rank Methods for Network Alignment. Major Professor: David F. Gleich.

Network alignment is the problem of finding a common subgraph between two graphs, and more generally k graphs. The results of network alignment are often used for information transfer, which makes it a powerful tool for deducing information or insight about networks. Network alignment is tightly related to the subgraph isomorphism problem which is known to be NP-hard, this makes the network alignment problem supremely hard in practice. Some algorithms have been devised to approach it via solving a form of a relaxed version of the NP-hard problem or by defining certain heuristic measures. These algorithms normally work well for problems when there is some form of prior known similarity between the nodes of the graphs to be aligned. The absence of such information makes the problem more challenging. In this scenario, these algorithms would often require much more time to finish executing, and even fail sometimes. The version of network alignment that this thesis tackles is the one when such prior similarity measures are absent. In this thesis, we address three versions of network alignment: (i) multimoal network alignment, (ii) standard pairwise network alignment, and (iii) multiple network alignment. A key common component of the algorithms presented in this thesis is exploiting a low rank structure in the network alignment problem and thus producing algorithms that run much faster than classic network alignment algorithms.

1 INTRODUCTION

As networks have become a more common way to represent data, processing and making predictions on such types of data has become increasingly important. There are many domains that utilize networks for their data representation and a common example is social networks. For example, in social networks, users are encoded as *nodes* and friendships are encoded as *edges*. It is standard practice to use the connections of a certain user to make predictions about this user in the future. Network science, thus, utilizes the topology of a graph (the way a graph is connected) to form new predictions or make new conclusions about the data. For instance, link prediction (Liben-Nowell and Kleinberg, 2007; Lü and Zhou, 2011), which is a common problem in network science aims at using a node's current connections in a network to predict new links in the graph. Another problem, network alignment (Bayati et al., 2013; Kollias et al., 2014; Nassar and Gleich, 2017; Malmi et al., 2017; Klau, 2009), uses more than one network's toplogy to predict similarities between two or more nodes. This prediction of similarity across more than one graph can allow for the deduction of a more complete version of a graph; this happens when the networks aligned represent the same entity in reality but each network is missing some information often attributed to collection errors.

Simply put, network alignment aims to identify pairs of nodes from two different graphs as *similar*. Similarity here varies from one field to another. For instance, biologists are often interested in pairing up nodes in two protein-protein interaction (PPI) networks from two different species in the hope of learning new functions of the proteins in the network belonging to the less studied species. Aligning two social networks, in contrast, aims at finding nodes that represent the same individual, and the goal here would be to find a more accurate global representation of an underlying network. In general, network alignment is important for two main reasons.

- Network alignment allows us to model transfer of information from one network to another. If we are able to identify nodes that are, in fact, related in two distinct graphs, we can infer deduce information about nodes between the graphs.
- Network alignment can be used to suggest a more global structure of the graph (and this can be thought of as predicting links to form a more complete version of the graph). This is analogous to image registration assuming one has two images of the same scene with some distorted pixels in each figure—in this case, the two graphs are snapshots of the same underlying structure but each of them has missing edges.

The network alignment problem is an extremely hard problem in practice due to its tight relationship with the subgraph isomorphism problem (Klau, 2009; Nassar et al., 2018). Nevertheless, multiple algorithms have been assigned for this task either by introducing certain heuristics or relaxing some of the constraints. One common component that a lot of network alignment algorithms rely on for their success is a *prior guess* (e.g. (Klau, 2009; Bayati et al., 2013)). Often, network alignment algorithms take as input the graphs to be aligned, as well as a set of possible node matchings (or the prior guesses), and these guesses play a key role in guiding the network alignment algorithm. It is indeed problematic for some algorithms when such information is not available – as solving the problem may simply become infeasible in terms of the required running time and memory requirement.

The key reason why the absence of the prior guesses becomes problematic is due to forcing the network alignment algorithm to evaluate every possible pair of matchings. These computations are often expensive, and the absence of a prior guess implies computing $|V_A||V_B|$ such evaluations (where $|V_A|$ and $|V_B|$ are the number of nodes in the two graphs to be aligned). **In the algorithms introduced in this thesis, we focus on the case when this prior knowledge is not available. Yet, we avoid computing pairwise similarity scores of every pair of nodes by exploiting**

a low rank structure in multiple formulations of the network alignment problem.

Prior to our work on network alignment, there has been a number of network alignment algorithms that formulated network alignment using spectral methods which identified a matrix that often encoded the pairwise scores of every pair of nodes from two graphs. In our work, we theoretically prove that these matrices (or tensors in the case of multiple network alignment) which store the similarity scores are in fact low rank. We then use this low rank representation and introduce matching algorithms that solve the matching problem on the low rank factors of this matrix or tensor without explicitly forming any of these quantities.

The rest of this thesis is organized as follows

- Chapter 2 (Preliminaries) introduces the necessary concepts and notation used throughout this thesis.
- Chapter 3 (Background) covers related work that has been done on network alignment and discusses the IsoRank network alignment algorithm.
- Chapter 4 (Multimodal network alignment) studies the problem of network alignment on a special kind of networks–multimodal networks.
- Chapter 5 (Low rank spectral network alignment) discusses a new fast and memory efficient low rank method that solves the pairwise network alignment problem.
- Chapter 6 (Multiple network alignment) addresses the problem of multiple network alignment, and presents a new scalable multiple network alignment method.
- Chapter 7 (Summary and further directions) serves as a summary of this thesis and incorporates further findings related to network alignment and bipartite matching and recommends future directions for research in this area.

All work in this thesis has been conducted with multiple collaborators. The work in Chapter 4 was in collaboration with my advisor David F. Gleich (Nassar and Gleich, 2017). The work in Chapter 5 was a collaboration with Ananth Grama, Nate Veldt, Shahin Mohammadi, and David F. Gleich (Nassar et al., 2018). The work in Chapter 6 was a collaboration with Ananth Grama, Georgios Kollias, and David F. Gleich (Nassar et al., 2018). A summary of the network alignment problems that are addressed in this thesis is shown in Table 1.1.

Table 1.1.
Summary of the network alignment problems that we will address in this thesis.

Pairwise network alignment (Chapter 5)	$\underset{\boldsymbol{P}}{\text{maximize}}$ subject to	$\sum_{ij}\sum_{rs}\boldsymbol{A}_{ij}\boldsymbol{B}_{rs}\boldsymbol{P}_{ir}\boldsymbol{P}_{js}$ $\sum_i \boldsymbol{P}_{ij} \le 1$ for all j $\sum_j \boldsymbol{P}_{ij} \le 1$ for all i $\boldsymbol{P}_{ij} \in \{0,1\}$.
Multimodal network alignment (Chapter 4)	$\underset{\boldsymbol{P}}{\text{maximize}}$ subject to	$\sum_{k=1}^{m}\sum_{ij}\sum_{rs}\boldsymbol{A}_{ij}^{(k)}\boldsymbol{B}_{rs}^{(k)}\boldsymbol{P}_{ir}\boldsymbol{P}_{js}$ $\sum_i \boldsymbol{P}_{ij} \le 1$ for all j $\sum_j \boldsymbol{P}_{ij} \le 1$ for all i $\boldsymbol{P}_{ij} \in \{0,1\}$.
Multiple network alignment (Chapter 6)	$\underset{\underline{\boldsymbol{X}}}{\text{maximize}}$ subject to	$\sum_{i_1,i_2,\dots,i_k}\sum_{j_1,j_2,\dots,j_k}\boldsymbol{A}_{i_1j_1}^{(1)}\boldsymbol{A}_{i_2j_2}^{(2)}\dots\boldsymbol{A}_{i_kj_k}^{(k)}\underline{X}_{i_1,i_2,\dots,i_k}\underline{X}_{j_1,j_2,\dots,j_k}$ $\sum_{i_2,\dots,i_k}\underline{\boldsymbol{X}}_{i_1,i_2,\dots,i_k} \le 1$ for all i_1 $\sum_{i_1,i_3,\dots,i_k}\underline{\boldsymbol{X}}_{i_1,i_2,\dots,i_k} \le 1$ for all i_2 $\dots$ $\sum_{i_1,i_2,\dots,i_{k-1}}\underline{\boldsymbol{X}}_{i_1,i_2,\dots,i_k} \le 1$ for all i_k $\underline{\boldsymbol{X}}_{i_1,i_2,\dots,i_k} \in \{0,1\}$.

2 PRELIMINARIES

Throughout this thesis, we will use the matrix representation of a graph $G_{\boldsymbol{A}} = (V_{\boldsymbol{A}}, E_{\boldsymbol{A}})$ which contains $n = |V_{\boldsymbol{A}}|$ nodes and $m = |E_{\boldsymbol{A}}|$ edges to be the matrix $\boldsymbol{A}$, and we call $\boldsymbol{A}$ the adjacency matrix correspondng to $G_{\boldsymbol{A}}$. The dimension of $\boldsymbol{A}$ is $n \times n$ and it has $2m$ nonzero entries when the graph $G_{\boldsymbol{A}}$ is undirected. For an unweighted graph, $\boldsymbol{A}_{ij} = \boldsymbol{A}[i, j] = 1$ if nodes i and j are adjacent, and unless explicitly stated later in this thesis, we assume all graphs we deal with are undirected – eventhough most of the concepts developed still apply to directed graphs. We denote $\boldsymbol{A}[i, :]$ and $\boldsymbol{A}[:, j]$ to be the i^{th} row and j^{th} column of $\boldsymbol{A}$ respectively, and we use $\mathbf{a}_j$ to denote the j^{th} column of $\boldsymbol{A}$ as well. A vector is denoted by a small bold letter (e.g. $\mathbf{x}$), the vector $\mathbf{e}$ is the vector of all ones and the vector $\mathbf{e}_i$ is the i^{th} column of the identity matrix (i.e. $\mathbf{e}_i[i] = 1$ and $\mathbf{e}_i[j] = 0$ when $j \neq i$). Later in this thesis, we use tensors which can be k-dimensional. We use the definition of a slice of a tensor from (Kolda and Bader, 2009) to be a two dimensional section of a tensor defined by fixing all but two indices. As an example, the i^{th} slice in a three dimensional tensor $\underline{\boldsymbol{T}}$, is $\underline{\boldsymbol{T}}[:, :, i]$.

Another notion that we will repeatedly use throughout the thesis is the *vec* and *unvec* operations. The operation *vec* applied on a matrix stacks the columns of the matrix into a new vector as follows.

$$\text{vec}(\boldsymbol{A}) = \text{vec}(\begin{bmatrix} | & | & & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_k \\ | & | & & | \end{bmatrix}) = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_k \end{bmatrix}$$

And the *vec* operation applied on a tensor stacks the columns of each slice of the tensor in order starting with the first slice. As an illustration, the *vec* operation applied on a three dimensional tensor $\underline{\boldsymbol{T}}$ can be expressed as follows.

$$\mathrm{vec}(\underline{\boldsymbol{T}}) = \begin{bmatrix} \mathrm{vec}(\underline{\boldsymbol{T}}[:,:,1]) \\ \mathrm{vec}(\underline{\boldsymbol{T}}[:,:,2]) \\ \vdots \\ \mathrm{vec}(\underline{\boldsymbol{T}}[:,:,k]) \end{bmatrix}$$

The *unvec* operation in contrast is the inverse function of *vec* and it produces a tensor or a matrix with the appropriate dimensions in the chapters to follow.

2.1 Kronecker products

Kronecker products are a recurent concept in this thesis, and we revisit them briefly here. A Kronecker product of a matrix $\boldsymbol{A}$ of dimension $n_1 \times n_2$ and another matrix $\boldsymbol{B}$ of dimension $m_1 \times m_2$ produces a matrix $\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B}$ of dimension $n_1 m_1 \times n_2 m_2$. For example, the Kronecker product of the following two matrices is:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11}\boldsymbol{B} & a_{12}\boldsymbol{B} \\ a_{21}\boldsymbol{B} & a_{22}\boldsymbol{B} \\ a_{31}\boldsymbol{B} & a_{32}\boldsymbol{B} \end{bmatrix}$$

The adjacency matrix of the product graph of two graphs whose adjacency matrices are $\boldsymbol{A}$ and $\boldsymbol{B}$ is $\boldsymbol{A} \otimes \boldsymbol{B}$. The product graph contains $|V_A||V_B|$ nodes and every node corresponds to a pair of nodes from the two previous graphs.

2.1.1 A Kronecker product property

A Kronecker product property that is specifically popular in several domains and is repeatedly used in this thesis is the following.

$$(\boldsymbol{B}^T \otimes \boldsymbol{A})\,\text{vec}(\boldsymbol{X}) = \text{vec}(\boldsymbol{AXB})$$

where vec($\boldsymbol{X}$) is the vectorization operation described above. The proof follows from the following simple fact.

$$\begin{aligned}(\boldsymbol{AXB})[:,k] &= \boldsymbol{AXB}[:,k] = \boldsymbol{A}\textstyle\sum_i \boldsymbol{X}[:,i]\boldsymbol{B}[i,k] = (\boldsymbol{B}[:,k]^T \otimes \boldsymbol{A})\begin{bmatrix}\boldsymbol{X}[:,1]\\ \boldsymbol{X}[:,2]\\ \vdots\\ \boldsymbol{X}[:,m]\end{bmatrix}\\ &= (\boldsymbol{B}[:,k]^T \otimes \boldsymbol{A})\,\text{vec}(\boldsymbol{X})\end{aligned}$$

Using the above derivation, we can conclude that

$$\text{vec}(\boldsymbol{AXB}) = \begin{bmatrix}(\boldsymbol{AXB})[:,1]\\ \vdots\\ (\boldsymbol{AXB})[:,n]\end{bmatrix} = \begin{bmatrix}(\boldsymbol{B}[:,1]^T \otimes \boldsymbol{A})\\ \vdots\\ (\boldsymbol{B}[:,n]^T \otimes \boldsymbol{A})\end{bmatrix}\text{vec}(\boldsymbol{X}) = (\boldsymbol{B}^T \otimes \boldsymbol{A})\,\text{vec}(\boldsymbol{X})$$

2.2 Low rank representations

In this thesis, we will repeatedly exploit the low rank structure of one of the matrices used in the network alignment optimization problem and we use various representations of this low rank matrix. In this section we revisit a few ways to express a low rank matrix. A low rank $n \times n$ matrix is a summation of k rank-1 matrices where k is considerably smaller than n. And this leads us to the first representation.

$$\boldsymbol{X} = \boldsymbol{X}_1 + \boldsymbol{X}_2 + \ldots + \boldsymbol{X}_k = \sum_{i=1}^{k} \boldsymbol{X}_i$$

Where each $\boldsymbol{X}_i$ is a rank-1 matrix. Each $\boldsymbol{X}_i$ can be represented as an outer product, so if $\boldsymbol{X}_i = \mathbf{u}_i\mathbf{v}_i^T$, then the above equation becomes

$$\boldsymbol{X} = \sum_{i=1}^{k} \mathbf{u}_i\mathbf{v}_i^T$$

Stacking the left vectors together and the right vectors leads to our third representation. Let $\boldsymbol{U} = \Big[\mathbf{u}_1|\mathbf{u}_2|\dots|\mathbf{u}_k\Big]$ and $\boldsymbol{V} = \Big[\mathbf{v}_1|\mathbf{v}_2|\dots|\mathbf{v}_k\Big]$, then

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{V}^T$$

The outer product of two vectors and the Kronecker product has tight connections when the vec and unvec operations are employed. Using the fact that $\text{vec}(\mathbf{u}\mathbf{v}^T) = \mathbf{v} \otimes \mathbf{u}$, we can deduce a fourth representation of a low rank matrix.

$$\text{vec}(\boldsymbol{X}) = \sum_{i=1}^{k} \mathbf{v}_i \otimes \mathbf{u}_i$$

2.3 Bipartite matching

Another concept that is critical to this thesis is bipartite matching. A matching between two sets is a one-to-one mapping between elements in these sets such that an element in the first set can be matched with zero or one element in the second set. In the context of graphs, maximum weight bipartite matching is the problem of pairing up nodes between two sets such that the total edge weights of all matched pairs is maximized (refer to equation (2.1)). A naïve algorithm to find the best matching computes the overall weight from overy possible matching, but requires a runtime of $\mathcal{O}(n!)$ when each set has $\mathcal{O}(n)$ nodes. A greedy algorithm, which runs by picking the maximum possible weight at each step of the algorithm achieves a theoretical 2-approximation bound from the optimal bipartite matching. The greedy algorithm runs in $\mathcal{O}(n^2 \log n)$ assuming that the bipartite graph is dense. The hungarian algorithm (Kuhn, 1955) is one of the most well known algorithms for this problem as it solves the problem exactly and runs in $\mathcal{O}(n^3)$. As it will be clear later, a standard step in many existing network alignment algorithms is to solve a maximum weight bipartite matching problem as a final step. In this thesis, we introduce the first class of algorithms that can solve the maximum weight bipartite matching problem when the matrix that stores edge scores is stored in its low rank factors.

$$\begin{array}{ll} \underset{\boldsymbol{P}}{\text{maximize}} & \sum_{ij} \boldsymbol{P}_{ij}\boldsymbol{X}_{ij} \\ \text{subject to} & \sum_{u} \boldsymbol{P}_{u,v} \leq 1 \text{ for all } v \\ & \sum_{v} \boldsymbol{P}_{u,v} \leq 1 \text{ for all } u \\ & \boldsymbol{P}_{u,v} \in \{0,1\} \end{array} \tag{2.1}$$

3 NETWORK ALIGNMENT

The goal of pairwise network alignment is to identify related nodes between two distinct graphs that are expected to be similar or related. The relevance varies from one discipline to another—for instance, aligning one social network to another aims at identifying pairs of nodes from the two graphs that represent the same person, whereas aligning protein protein interaction networks of one species to another entails identifying proteins that are similar in function. In the standard formulation of network alignment, the result is a formal one-to-one matching between nodes from the first graph to nodes in the second graph, while the result of a local network alignment algorithm can produce many-to-many pairings of nodes. Some applications of network alignment include (i) finding similar nodes in social networks, (ii) social network deanonymization (Korula and Lattanzi, 2014), (iii) pattern matching in graphs (Conte et al., 2004), (iv) protein matching in Biology (Kelley et al., 2004; Singh et al., 2008; Liao et al., 2009), (v) ontology alignment (Hu et al., 2008), and more.

3.1 Global versus local network alignment

There are two major approaches to network alignment problems (Atias and Sharan, 2012): local network alignment, where the goal is to find multiple local regions of both graphs that are similar to each other, and global network alignment, where the goal is to find one global region that is similar in both graphs. Many approaches to network alignment rely on solving an optimization problem to compute what amounts to a topological similarity score between pairs of nodes in the two networks. Figure 3.1 illustrates the difference between global and local network alignment. As can be inferred from the figure, local network alignment produces a many-to-many pairing of nodes whereas global network alignment produces a set of one-to-one matchings.

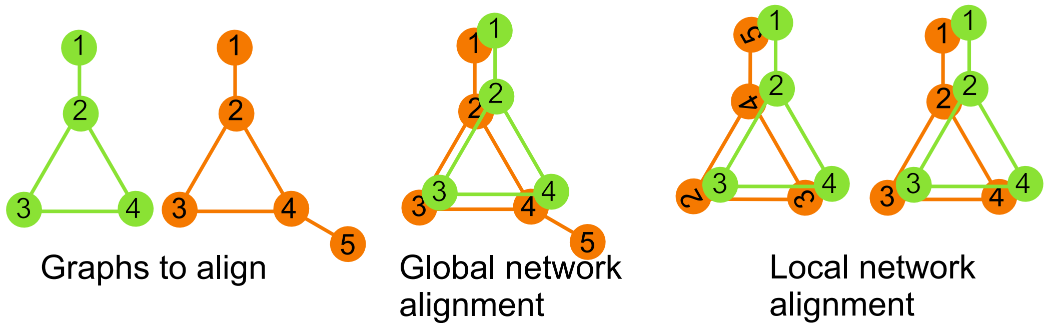


Figure 3.1. Global network alignment aims to find *one* set of one-to-one matching between the two graphs to maximize the global overlap. Local network alignment aims to find multiple matchings such that several repeated local patterns are identified.

In this thesis, we focus on global alignment with the aim of producing one-to-one matches between distinct graphs.

3.2 Network alignment formulation

We now turn our attention to define global pairwise network alignment more rigorously. For two graphs with adjacency matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, the goal is to find a permutation matrix $\boldsymbol{P}$ such that the objective below is maximized.

$$\begin{array}{ll} \underset{\boldsymbol{P}}{\text{maximize}} & \sum_{ij} [\boldsymbol{P}^T \boldsymbol{A} \boldsymbol{P}]_{ij} \boldsymbol{B}_{ij} \\ \text{subject to} & \sum_i \boldsymbol{P}[i,:] \leq 1 \text{ for all } i \\ & \sum_j \boldsymbol{P}[:,j] \leq 1 \text{ for all } j \\ & \boldsymbol{P}_{ij} \in \{0,1\} \end{array} \tag{3.1}$$

This quantity is often referred to as the *overlap*. In words, this formulation aims to find a re-labeling of the nodes in graph $G_{\boldsymbol{A}}$ such that this new labeling will generate the maximum possible overlap when the two graphs are overlayed on top of each other. This specific formulation is an instance of the quadratic assignment problem (QAP) (Burkard et al., 2012) and solves the maximum common subgraph problem.

The goal of the maximum common subgraph problem is to find the largest possible graph that will be isomorphic to a subgraph from $G_{\boldsymbol{A}}$ and a subgraph from $G_{\boldsymbol{B}}$. Network alignment stated as the above optimization problem generalizes the subgraph isomorphism problem, which is known to be NP-hard.

Recent variations of this objective include an extension to overlapping triangles (Mohammadi et al., 2016), an extension that combines edge overlapping with prior similarity scores (Singh et al., 2008; Bayati et al., 2013), as well as an extension specific to bipartite graphs (Koutra et al., 2013)

3.3 The network alignment pipeline

There exists a variety of algorithms for pairwise network alignment, and they often have a similar pipeline (Feizi et al., 2016; Singh et al., 2008; Bayati et al., 2013; Klau, 2009), and the pipeline consists of two steps. The first step is finding pairwise scores for every pair of nodes from the two networks, and this step is the distinguishing feature of network alignment algorithms. In IsoRank (Singh et al., 2008) for instance, nodes that participate in aligning more edges recieve higher scores, and in TAME (Mohammadi et al., 2016), nodes that participate in aligning more triangles receive higher scores and other algorithms define other heuristics. Later in this chapter, we will get into a more detailed discussion on IsoRank, and later in this thesis we will discuss EigenAlign (Feizi et al., 2016) and TAME (Mohammadi et al., 2016) in more detail.

Thus, the first step of many network alignment algorithms is to forming a scoring algorithm f which gives the similarity scores. After all the similarity scores have been computed and stored in a matrix $\boldsymbol{X}$, the goal is to maximize the global similiarity scores and thus, a maximum weight bipartite matching algorithm is run on $\boldsymbol{X}$. This produces the final one-to-one matchings.

$$f(\boldsymbol{A}, \boldsymbol{B}) = \boldsymbol{X} \rightarrow \text{bipartite matching } (\boldsymbol{X})$$

The problematic aspect of this pipeline is that it requires a memory storage that is quadratic in the size of the graphs.

3.4 Network alignment with a prior

Some applications also come with prior information about which nodes in one network may be good matches for nodes of another network, which implicitly imposes a restriction on the number of the similarity scores that must be computed and stored in practice (Bayati et al., 2013) (and this makes the matrix $\boldsymbol{X}$ introduced more sparse in practice). However, for problems that lack this prior, the data requirement for storing the similarity scores is quadratic, which severely limits the scalability of this class of approaches to solve the problem (for example, the Lagrangian relaxation method of Klau et al. (Klau, 2009) requires at least quadratic memory in this scenario and is infeasible for large graphs). There do exist memory-scalable heuristics for solving network alignment problems with no prior. One procedure designed for aligning networks without prior information is the the Graph Alignment tool (GRAAL) (Kuchaiev et al., 2010). GRAAL computes the so-called *graphlet degree signature* for each node, a vector that generalizes node degree and represents the topological structure of a node's local neighborhood. The method measures distances between graphlet degrees to obtain similarity scores, and then uses a greedy *seed and extend* procedure for matching nodes across two networks based on the scores. A number of algorithms related to this method have been introduced, which extend the original technique by considering other measures of topological similarity as well as different approaches to rounding similarity scores into an alignment (Milenkovic et al., 2010; Kuchaiev and Pržulj, 2011; Memisevic and Pržulj, 2012; Malod-Dognin and Pržulj, 2015). The seed-and-extend alignment procedure was also employed by the GHOST algorithm (Patro and Kingsford, 2012), which computes topological similarity scores based on a novel spectral signature for each node. The GHOST procedure or the GRAAL algorithm and its variants usually involve cubic or worse computation in terms of vertex neighborhoods in the graph (e.g. enumeration of all 5-node graphlets within a local region). In this thesis, our focus is on alignment problems where the prior information about matches is not present, yet, we avoid the quadratic memory requirement by employing low rank methods.

3.5 Network alignment in Biology

Network alignment is a popular problem in biology (Kelley et al., 2004; Singh et al., 2008; Liao et al., 2009; Memisevic and Pržulj, 2012; Milenkovic et al., 2010) due to its ability to help extract new information about less studied biological entities. Often in biology, one can extract valuable knowledge about proteins for which little information is known by aligning a protein network with another protein network that has been studied more. By doing so, one can draw conclusions about proteins in the first network by understanding their similarities to proteins in the second (Kuchaiev et al., 2010; Kuchaiev and Pržulj, 2011). For instance, if we understand how a particular experiment disrupts a subnetwork of protein interactions in a mouse and we know the related interactions in a human, we have evidence for the hypothesis that the experiment would disrupt the mapped interactions in a human. Furthermore, network alignment is used to compare the brain connectomes from fMRIs. The brain is one of the most challenging human body components to understand the function of the brain connectome, network alignment can be employed (Milano et al., 2016, 2017). With the biological importance of this problem in mind, there have been several biologically inspired algorithms for network alignment. These algorithms often build the biological component into the algorithm itself. For instance, GHOST (Patro and Kingsford, 2012) uses the protein sequence similarity scores (such as the ones extracted from BLAST (Altschul et al., 1990)), and Magna (Vijayan et al., 2015) uses a genetic algorithm and a *crossover* function aimed at producing an alignment that takes into account the evolutionary changes of the nodes in the networks. Our focus in this thesis is more generic to standard networks from any discipline.

3.6 Multiple network alignment

So far we have been referring to the network alignment problem in terms of two networks. The problem formulation extends neatly to multiple networks, but makes the problem even more challenging to solve in practice. Chapter 6 is entirely concerned with multiple network alignment and we defer the details to that chapter.

3.7 Literature review on network alignment algorithms

We now take a step back and review some of the existing network alignment methods that have been proposed for standard networks. The global network alignment problem is almost always stated as an attempt to maximize the number of overlapped edges combined with maximizing domain-specific notions of apriori known vertex similarity. Recently, (Mohammadi et al., 2016) introduced the notion of finding an alignment that maximizes the number of preserved higher order structures (such as triangles) across networks. This results in an integer programming problem that can be approximated by the Triangular Alignment algorithm (TAME), which obtains similarity scores by solving a tensor eigenvalue problem that relaxes the original objective. And alternative approaches to improve network alignment include active methods that allow users to select matches from a host of potential near equal matches (Malmi et al., 2017).

There are two prominent classes of methods: (i) embedding and (ii) integer optimization relaxations and heuristics. Embedding methods seek to compute a feature vector for each node of $G_{\boldsymbol{A}}$ and $G_{\boldsymbol{B}}$ independently and then use relationships between the feature vectors to generate the matching. Using eigenvectors is common for this task in pattern recognition (Knossow et al., 2009) and recent methods have proposed graphlet counts (Kuchaiev et al., 2010), eigenvector histograms (Patro and Kingsford, 2012), and one method generates the embeddings based on node and edge types (Fraikin and van Dooren, 2007). Obtaining these feature vectors can be extremely expensive in terms of computation, but results in memory efficient methods.

The second class of methods attempts to solve a relaxed version of the problem shown in (3.1), perhaps with an additional term reflecting the vertex similarity. IsoRank was one of the first methods in this class (Singh et al., 2008). Subsequent techniques include belief propagation methods (Bayati et al., 2013), Lagrangian relaxations (Klau, 2009), spectral methods (Feizi et al., 2016), and tensor eigenvectors for motif-alignment (Mohammadi et al., 2016). Essentially, all of these methods store a dense, real-valued heuristic matrix $\boldsymbol{X}$ of size $|V_A| \times |V_B|$. This needs memory that is *quadratic* in the size of the networks and greatly limits scalability. One of the most scalable methods results from a crucial analytical insight into the structure of the IsoRank heuristic. The resulting method–network similarity decomposition (Kollias et al., 2012)–can be considered a hybrid of embedding and integer optimization.

3.8 IsoRank

Now, we dive more into one of the methods mentioned earlier–IsoRank. IsoRank is one of the earliest network alignment algorithms that tackles the global network alginment problem (Singh et al., 2008), and it is used repeatedly in this thesis.

The intuition behind IsoRank is that two nodes i and j are good matches if the neighbors of i can match the neighbors of j. A *relevance* score based on this idea can be computed between every possible pair of nodes from two graphs $G_{\boldsymbol{A}}$ and $G_{\boldsymbol{B}}$.

$$\boldsymbol{R}(\boldsymbol{A}_i, \boldsymbol{B}_j) = \sum_{u \in N(\boldsymbol{A}_i)} \sum_{v \in N(\boldsymbol{B}_j)} \frac{1}{|N(u)||N(v)|} \boldsymbol{R}(u, v)$$

Where $N(w)$ is the set of all nodes that are adjacent to node w. Initially, the values in $\boldsymbol{R}$ are uniform, and as the relevance scores are iteratively recomputed, $\boldsymbol{R}$ converges to become the matrix that stores the similarities of every pair of nodes from the two graphs.

We can derive a similar relationship via matrix notation. First, we form a new matrix $\boldsymbol{C}$ which is the adjacency matrix of the product graph of $\boldsymbol{A}$ and $\boldsymbol{B}$. Here, $\boldsymbol{C}[\texttt{sub2ind}[A_i, B_k], \texttt{sub2ind}[A_j, B_l]] = 1$ if $(\boldsymbol{A}_i, \boldsymbol{A}_j)$ is an edge in $G_{\boldsymbol{A}}$ and $(\boldsymbol{B}_k, \boldsymbol{B}_l)$ is an edge in $G_{\boldsymbol{B}}$, where `sub2ind` is the function that converts a subscript to a linear index. One iteration of solving for $\boldsymbol{R}$ is equivalent to solving for $\mathbf{r} = \tilde{\boldsymbol{C}}\mathbf{r}$, where $\mathbf{r} = \operatorname{vec}(\boldsymbol{R})$ and $\tilde{\boldsymbol{C}}$ is the column normalized matrix of $\boldsymbol{C}$. Thus, finding $\boldsymbol{R}$ as definded by IsoRank is equivalent to finding the dominant eigenvector of the matrix $\tilde{\boldsymbol{C}}$.

Nevertheless, as mentioned earlier, some network alignment problems come with prior known similarity about the nodes in both graphs. This prior can be attributed via a weighting parameter α. The final equation of IsoRank becomes equivalent to solving for the PageRank vector on the weighted product graph. If we let $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{B}}$ be the column normalized versions of $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively, then the IsoRank iterate can be stated as follows.

$$\mathbf{x} = \alpha\tilde{\boldsymbol{B}} \otimes \tilde{\boldsymbol{A}}\mathbf{x} + (1-\alpha)\operatorname{vec}(\boldsymbol{L})$$

where the entries in $\boldsymbol{L}$, $\boldsymbol{L}[i,j]$, store the prior known similarity scores between node i from $G_{\boldsymbol{A}}$ and node j from $G_{\boldsymbol{B}}$. $\boldsymbol{L}$ is a normalized matrix where the sum of entries add up to 1.

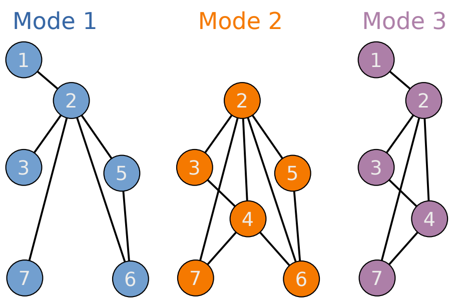
4 MULTIMODAL NETWORK ALIGNMENT

In this chapter we discuss the network alignment problem on a special kind of networks, multimodal networks. The results presented in this chapter are from our paper Multimodal Network Alignment (Nassar and Gleich, 2017).

A multimodal network encodes relationships between the same set of nodes in multiple settings and in this chapter, we propose a method for multimodal network alignment. This method computes a matrix which indicates the alignment, but produces the result as a low-rank factorization directly. We then propose new methods to compute approximate maximum weight matchings of low-rank matrices to produce an alignment. We evaluate our approach by applying it on synthetic networks and use it to de-anonymize a multimodal transportation network. Our experiments show that using a multimodal network alignment method is superior to employing a standard network alignment method agnostically on multimodal graphs.

4.1 Motivation

Recently, much of the network data emerging in scientific applications and engineering studies is multimodal and contains separate types of relational information between vertices (Szell et al., 2010; Cardillo et al., 2013; Battiston et al., 2014; Nicosia and Latora, 2015; Ni et al., 2014). For instance, transportation networks are often described as multimodal when they feature multiple interconnecting but distinct networks, such as when subway lines reach airports and national train systems. In biology, multimodal networks show different types of relationships that can occur between proteins and genes such as protein sequence similarity, co-occurrence in genes, co-occurrence in scientific papers, experimental interaction and more.



$$\mathbf{M} = \left(\begin{array}{ccccccc|ccccccc|ccccccc}
0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
1&0&1&0&1&1&1&0&1&0&0&0&0&0&0&1&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
\hline
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&1&1&1&1&1&0&1&0&0&0&0&0\\
0&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&0&1&0&0&0\\
0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&1&0&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&1\\
\hline
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&1&0&0&0&0&0&0&1&0&0&0&0&0&1&0&1&1&0&0&1\\
0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&1&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&1&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&1&0&1&0&0&0
\end{array}\right)$$

Figure 4.1. A multimodal network consists of a common set of vertices identifiers (in this case, 1-7) and a set of modes that define edges. Below, we illustrate the multimodal adjacency matrix, which we will use to align the multimodal networks.

This motivates our work in this chapter. We seek to produce effective methods to align a pair of multimodal networks. We will be more specific about our notion of a multimodal network in the following section. For now, consider multimodal networks in biology where proteins have many different types of relationships such as the ones described above. We may, for instance, be interested in aligning these networks generated from two species to infer related complexes and systems. In this case, the modes of each multimodal network are shared between the two networks to align. We propose a new method that takes in a pair of multimodal networks with the same set of modes and produces an alignment between the vertices encoded in the network with no apriori knowledge of the relationship between the vertices. As stated earlier, network alignment problems are notoriously difficult from a theoretical perspective and multimodal networks tend to be bigger in size, which makes the multimodal network alignment problem even more challenging in practice.

Key difference with multiple network alignment. In multimodal network alignment, the goal is to find *one* relabeling of nodes applied to all modes of one multimodal network, such that the total overlap from every modal alignment is maximized. In contrast, multiple network alignment for k networks entails finding $k-1$

such relabelings, each corresponding to a network, such that when the k networks are overlayed on top of each other according to the new labels, the total overlap is maximized.

Several principled heuristic solutions are often used (Mohammadi et al., 2016; Singh et al., 2008; Conte et al., 2004) to solve the network alignment problem, and in this chapter, we develop a principled and effective heuristic method that generalizes the ideas utilized in IsoRank (Singh et al., 2008) in concert with the insights from the network similarity decomposition method (Kollias et al., 2014). Our method is designed for multimodal networks with 10s-1000s of modes with 1,000-100,000 vertices.

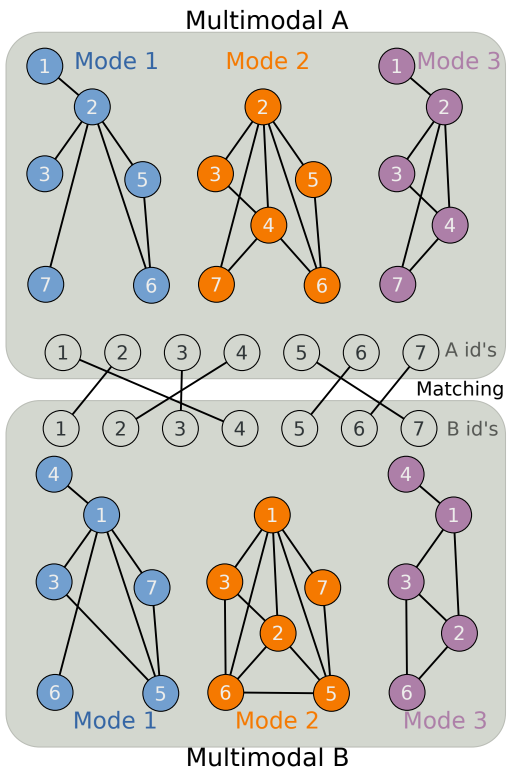


Figure 4.2. An illustration of the multimodal alignment problem. The goal is to identify the matching illustrated between the node id's given only the multimodal networks A and B. We assume the modes have a known alignment. The overlap of this matching is 20 (mode 1 contributes 6 overlapping edges, mode 2 contributes 9 overlapping edges, and mode 3 contributes 5 overlapping edges).

On top of this new method, we contribute a new theory about how to take the output from our heuristics and efficiently turn it into a matching between the graphs. This solves one of the key challenges in how to deal with multimodal networks and alignment. Many good ideas result in a polynomial explosion in problem size. This causes methods that have been developed for the pairwise case to be nearly impossible to use for the multimodal setup we propose (Bayati et al., 2013; Klau, 2009). More specifically, this is a memory bottleneck: they would need terabytes of memory to handle problems that start off as a megabyte of data. We state this new theory as an independently useful primitive of finding a matching in a low-rank matrix. For this problem, we investigate a highly efficient $1/k$ approximation for a rank-k matrix that finds the matching by using the low rank factors of the matrix *only*.

When we seek to evaluate our new methods, we will do so in (i) detailed synthetic experiments and (ii) a case-study and demonstration of our technique where we de-anonymize a publicly released transportation network that lacks meaningful vertex identifiers (Nicosia and Latora, 2015). The goal of the synthetic experiments is to address the hypothesis that using multimodal alignment is better than straightforward generalizations of using existing network alignment methods to pairwise align each mode or a combined network. In the case of the transportation data, only our multimodal alignment method is able to completely map the data between the public data and the original database with labels. Our goal with both of these experiments is to reveal properties of our method and ideas that would be useful in any domain-specific application of network alignment but without the engineering that tends to occur in these applications. The rest of this chapter is divided as follows:

- Section 4.2 presents a precise statement of a multimodal network alignment problem.
- Section 4.3 shows our method, multimodal similarity decomposition (MSD), to solve multimodal network alignment problem that uses all the information among the modes, and goes beyond baseline methods that align the modes individually.
- Section 4.4 discusses independently useful results about how to approximate a bipartite maximum weight matching where the matching matrix is low-rank.

- Section 4.6 is a case study on synthetic graphs and demonstrates that simple methods for multimodal alignment outperforms strong methods for pairwise network alignment of the individual modes when there are many vertices missing in each mode.

- Section 4.7 is a case-study with de-anonymizing a publicly released multimodal network of airports and airlines (Nicosia and Latora, 2015) that cannot be de-anonymized using state-of-the-art network alignment methods. We also study the set of most helpful modes to find this alignment in this section.

4.2 Multimodal networks *&* Multimodal network alignment

In the context of this thesis, a multimodal network is a common set of vertex labels (the set V) and multiple sets of undirected edges over these vertices. Each set of edges represents a mode, so we have $E^{(1)}, \dots, E^{(m)}$ for m modes and Figure 4.1 shows an example with three modes. Note that each mode can have a strict subset of the nodes V and not necessarily the entire set.

To manipulate multimodal network data computationally, we use a multimodal adjacency matrix. Our definition is in the spirit of how multislice, multiplex networks, and temporal networks are represented (Mucha et al., 2010), although the details of our specific construction differ. Let $\boldsymbol{A}_1, \dots, \boldsymbol{A}_m$ represent the adjacency matrices of each mode. Then the multimodal adjacency matrix is:

$$\boldsymbol{M}_{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{C}_{12} & \dots & \boldsymbol{C}_{1m} \\ \boldsymbol{C}_{12}^T & \boldsymbol{A}_2 & \dots & \boldsymbol{C}_{2m} \\ \vdots & \ddots & \ddots & \vdots \\ \boldsymbol{C}_{1m}^T & \dots & \dots & \boldsymbol{A}_m \end{bmatrix}. \tag{4.1}$$

Here, the matrix $\boldsymbol{C}_{ij}$ represents the cross-modal associations between mode i and mode j. This is always a binary diagonal matrix where $\boldsymbol{C}_{ij}(k,k) = 1$ if vertex k is present in both modes i and j. All other entries are 0. An example is illustrated in

Figure 4.1. Note that $\boldsymbol{M_A}$ is sparse, and the cross-modal edges do not introduce too many new entries beyond the data. The goal of the multimodal network alignment problem is to produce a matching between the vertices of two multimodal networks that *aligns* the networks by maximizing the number of edges of each mode that are preserved under the matching. Recall that a matching is a one-to-one relationship between two sets. Let $V_{\boldsymbol{A}}$ and $V_{\boldsymbol{B}}$ be the vertex sets of multimodal networks $G_{\boldsymbol{A}}$ and $G_{\boldsymbol{B}}$ respectively, and let $\boldsymbol{M_A}$ and $\boldsymbol{M_B}$ be the corresponding adjacency matrices of the multimodal networks. We use $v \leftrightarrow v'$ to denote a matching between $v \in V_A$ and $v' \in V_B$. Let $E_{\boldsymbol{A}}^{(k)}$ and $E_{\boldsymbol{B}}^{(k)}$ be the edge sets of the kth mode of $G_{\boldsymbol{A}}$ and $G_{\boldsymbol{B}}$ as well. Then we seek a matching between $V_{\boldsymbol{A}}$ and $V_{\boldsymbol{B}}$ that maximizes

$$\sum_{k=1}^{m} \sum_{\substack{(u_A,v_A) \\ \in E_A^{(k)}}} \sum_{\substack{(u_B,v_B) \\ \in E_B^{(k)}}} \begin{cases} 1 & \text{if } u_A \leftrightarrow u_B \text{ and } v_A \leftrightarrow v_B \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

Note that this is a single matching between the vertices that is then evaluated over all the modes, which is possible because we assume that the correspondence between each mode is known and mode k in network $G_{\boldsymbol{A}}$ corresponds to mode k in network $G_{\boldsymbol{B}}$. See Figure 4.2 for an example.

So far, the way we have defined our multimodal networks, the multimodal network alignment problem can be stated as as generalization of the classic pairwise network alignment problem presented in Chapter 3. If $\boldsymbol{P}$ is a matrix where the rows are indexed by vertices in V_A and columns by vertices in V_B, then $\boldsymbol{P}(v, v') = 1$ if $v \in V_A$ matches to $v' \in V_B$ and 0 otherwise. Also, let $\{\boldsymbol{A}_k\}$ and $\{\boldsymbol{B}_k\}$ be the adjacency matrices for each mode of the multimodal networks A and B. Then the multimodal network alignment problem is

$$\begin{array}{ll} \underset{\boldsymbol{P}}{\text{maximize}} & \frac{1}{2}\sum_{k=1}^{m}\sum_{ij}[\boldsymbol{P}^T\boldsymbol{A}_k\boldsymbol{P}]_{ij}[\boldsymbol{B}_k]_{ij} \\ \text{subject to} & \sum_i \boldsymbol{P}_{ij} \le 1 \text{ for all } j \\ & \sum_j \boldsymbol{P}_{ij} \le 1 \text{ for all } i \\ & \boldsymbol{P}_{ij} \in \{0,1\}. \end{array} \tag{4.3}$$

As it is the case for the pairwise network alignment problem, this integer optimization problem over the space of matrices is extremely challenging to solve exactly (note that it is equivalent to the pairwise network alignment problem when $m = 1$).

4.3 Multimodal Similarity Decomposition: A multimodal generalization of IsoRank and the Network Similarity Decomposition

We now present our approach to compute a multimodal network alignment. The high level idea is that we are going to run the IsoRank method to align the multimodal adjacency matrices $\boldsymbol{M_A}$ and $\boldsymbol{M_B}$. Doing so will involve a number of new insights about how the methods will behave on multimodal networks and exploiting the structure of the methods. The result of this section is a heuristic solution matrix $\boldsymbol{Y}_{MN}$ that we describe how to turn into a multimodal alignment in Section 4.4. We begin by reviewing the network similarity decomposition (Kollias et al., 2014) method.

4.3.1 IsoRank *&* the network similarity decomposition

Recall the IsoRank algorithm (Section 3.8) for pairwise network alignment. IsoRank is equivalent to solving the PageRank problem on the product graph of the two graphs to be aligned. Network Similarity Decomposition (NSD) (Kollias et al., 2014) uses a simple Kronecker product property (Chapter 2) to rewrite the key iterate of IsoRank.

$$\mathbf{x} = \alpha \tilde{\boldsymbol{B}} \otimes \tilde{\boldsymbol{A}} \mathbf{x} + (1-\alpha)\operatorname{vec}(\boldsymbol{L}) \leftrightarrow \boldsymbol{X} = \alpha \tilde{\boldsymbol{A}} \boldsymbol{X} \tilde{\boldsymbol{B}} \mathbf{x} + (1-\alpha)\operatorname{vec}(\boldsymbol{L}) \tag{4.4}$$

where $\operatorname{vec}(\boldsymbol{X}) = \mathbf{x}$ and $\boldsymbol{L}$ is the prior known similarity.

Note that when no prior similarity is known, this translates to making all the entries in $\boldsymbol{L}$ constant, reflecting a uniform similarity. The insight in the network similarity decomposition (NSD) is that if $\boldsymbol{L}$ is rank 1, as it would be in the case of uniform similarity, then running t iterations of the power method to compute the PageRank vector will result in a rank $t+1$ matrix $\boldsymbol{X}$. Moreover, the low-rank factorization of $\boldsymbol{X}$ can be easily computed by running the power method for PageRank on network $G_{\boldsymbol{A}}$ and network $G_{\boldsymbol{B}}$ *independently.* (In this sense, this is similar to an embedding method, because we compute the PageRank iterates of each network separately, then combine them.) This is easy to see when $\boldsymbol{L} = \mathbf{u}\mathbf{v}^T$ then t iterations of the power-method for (4.4) produce:

$$\boldsymbol{X}^{(t)} = (1-\alpha)\sum_{k=0}^{t-1} \alpha^k \tilde{\boldsymbol{A}}^k \mathbf{u}\mathbf{v}^T (\tilde{\boldsymbol{B}}^T)^k + \alpha^t \tilde{\boldsymbol{A}}^t \mathbf{u}\mathbf{v}^T (\tilde{\boldsymbol{B}}^T)^t$$

This matrix is a sum of $t+1$ rank-1 matrices each of the form $\texttt{scalar}\cdot(\tilde{\boldsymbol{A}}^t \mathbf{u})(\tilde{\boldsymbol{B}}^t \mathbf{v})^T$. In this way, a low-rank factorization of the IsoRank solution can be computed extremely efficiently. Run t steps of the power method for PageRank on networks $G_{\boldsymbol{A}}$ and $G_{\boldsymbol{B}}$ separately and store the iterations.

4.3.2 Our Multimodal Similarity Decomposition

Our goals with the multimodal similarity decomposition and multimodal alignment closely mirror those of NSD and IsoRank. In our case, we want to allow alignment information to *flow* between modes of the network to reinforce alignments between vertices that are present in various modes. This intuition suggests that the

IsoRank scores for the alignment of multimodal adjacency matrices will be an effective heuristic for our problem. This will flow alignment information between modes via the cross-modal edges $\boldsymbol{C}_{ij}$.

Since we assume that the alignment between modes of the network are already known, we can strengthen our heuristic by customizing the matrix $\boldsymbol{L}$. Recall that $\boldsymbol{L}_{ij}$ is the apriori similarity of node i in $G_{\boldsymbol{A}}$ and j in $G_{\boldsymbol{B}}$. The multimodal adjacency matrix has m copies of each vertex identifier – one for each vertex in each of the m modes. Thus, we use the matrix

$$\boldsymbol{L} = \begin{bmatrix} \gamma \cdot \texttt{ones} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \gamma \cdot \texttt{ones} \end{bmatrix}, \tag{4.5}$$

where $\texttt{ones}$ is a matrix of size $|V_A| \times |V_B|$ and γ is chosen such that all entries of $\boldsymbol{L}$ sum to 1 (that is, $\gamma = m^{-1}|V_A|^{-1}|V_B|^{-1}$). This choice of $\boldsymbol{L}$ corresponds to a rank-m matrix, where each mode constitutes a single rank-1 factor suggesting that nodes in mode 1 in multimodal network $G_{\boldsymbol{A}}$ should correspond to nodes in mode 1 of multimodal network $G_{\boldsymbol{B}}$.

Let $\boldsymbol{P}_{\boldsymbol{A}}$ and $\boldsymbol{P}_{\boldsymbol{B}}$ correspond to degree-normalization (by columns) of $\boldsymbol{M}_{\boldsymbol{A}}$ and $\boldsymbol{M}_{\boldsymbol{B}}$. Thus, the heuristic $\boldsymbol{X}$ we compute solves:

$$\boldsymbol{X} = \alpha \boldsymbol{P}_{\boldsymbol{A}} \boldsymbol{X} \boldsymbol{P}_{\boldsymbol{B}}^T + (1-\alpha)\boldsymbol{L} \tag{4.6}$$

where $\boldsymbol{L}$ is the multimodal similarity (4.5). The result $\boldsymbol{X}$ gives a score for aligning each node in each mode of A to each node in each mode B. This is not the type of output we want, so in subsequent sections, we discuss how to turn this output into the type of matching we expect ($1-1$ matching between the set $V_{\boldsymbol{A}}$ and $V_{\boldsymbol{B}}$).

At this point, we can utilize the same methodology underlying the network similarity decomposition to efficiently compute a low-rank decomposition of $\boldsymbol{X}$. Note that the matrix $\boldsymbol{L}$ is rank-m. Thus, we can compute t iterations of the power method to solve (4.6) and the result is a rank $(m)\cdot(t+1)$ matrix $\boldsymbol{X}$ following the exact same argument as expression for the network similarity decomposition. The pseudocode is shown in Algorithm 1. One small detail is that we assumed in the derivation that $\boldsymbol{P_A}$ and $\boldsymbol{P_B}$ have no empty rows or columns. If this is not the case, then each iterate needs to be normalized to be a probability distribution as in the pseudocode.

Algorithm 1 Pseudocode for the multimodal similarity decomposition: m is the number of modes of the multimodal networks $G_{\boldsymbol{A}}$ and $G_{\boldsymbol{B}}$, and the column-normalization computes: $\boldsymbol{P}[i,j] = \boldsymbol{M}[i,j]/\sum_\ell \boldsymbol{M_A}[\ell,j]$. The `normalize` function divides a vector by its sum. The reshaping occurs by column such that $\boldsymbol{U}[:,1] = \boldsymbol{Z}[:,0,1]$. We assume the reshaping happens the same way on $\boldsymbol{U}$ and $\boldsymbol{V}$ to ensure the low-rank factors are aligned. The result $\boldsymbol{U}$ and $\boldsymbol{V}$ must still be matched as in Section 4.4 to solve (4.3).

```
1: **Input**: Multimodal networks $\boldsymbol{M_A}$, $\boldsymbol{M_B}$, $\alpha$, iteration $t$
2: **Output**: Matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ such that $\boldsymbol{X}^{(t)} = \boldsymbol{U}\boldsymbol{V}^T$
3: $\boldsymbol{U}$ =PageRankPowers($\boldsymbol{M_A}$,$\alpha$,$t$)
4: $\boldsymbol{V}$ =PageRankPowers($\boldsymbol{M_B}$,$\alpha$,$t$)
5:
6: **PageRankPowers**($\boldsymbol{M}$,$\alpha$,$t$)
7: Set $\boldsymbol{P}$ to be the column normalized matrix $\boldsymbol{M}$.
8: Allocate $\boldsymbol{Z}$ as a $|V| \times (t+1) \times m$ array
9: for k=1 to m do                                  ▷ the number of modes
10:    for i=1 to $|V|m$ do
11:        $\boldsymbol{Z}[i,0,k] = \begin{cases} (\sqrt{m}|V|)^{-1} & i \text{ is an index in mode } k \\ 0 & \text{otherwise} \end{cases}$
12:    end for
13:    for j=1:t do
14:        $\boldsymbol{Z}[:,j,k] = \text{normalize}(\boldsymbol{P}\boldsymbol{Z}[:,j-1,k])$
15:        $\boldsymbol{Z}[:,j-1,k] = \sqrt{(1-\alpha)\alpha^{j-1}}\boldsymbol{Z}[:,j-1,k]$
16:    end for
17:    $\boldsymbol{Z}[:,t,k] = \sqrt{\alpha^t}\boldsymbol{Z}[:,t,k]$
18: end for
19: return $\boldsymbol{U}$ as $\boldsymbol{Z}$ reshaped to $|V| \times (t+1)(m)$
```

4.4 Matching algorithms & Resolving conflicts

After the multimodal network decomposition (MSD), the next step is to produce a matching between the vertices. A standard technique here is to run a max-weight bipartite matching on the matrix $\boldsymbol{X}$ for the network alignment case (Singh et al., 2008; Bayati et al., 2013). There are two issues with this step. The first is the size of the resulting matrix $\boldsymbol{X}$ can be prohibitively large if we realize it as a dense matrix. This is especially true if we have a large multimodal network with many modes. A problem with 100 modes and 5000 vertices would produce a $500,000 \times 500,000$ dense matrix whereas the low-rank factors for 8 iterations would require two $500,000 \times 900$ matrices. Thus, we investigate memory-efficient matching routines that deal with $\boldsymbol{X}$ via its low-rank decomposition. The second issue is that this output is a matching on the rows of the multimodal adjacency matrix and not a matching on the vertex set of the multimodal network. Aggregating them can result in conflicting matches and we describe a few ways to resolve these.

4.4.1 Approximate matching via low-rank factors

Bipartite matching is a common subroutine that has a well-known polynomial time algorithm (Kuhn, 1955). The problem with this algorithm is that it requires the matrix of weights associated with each edge of the bipartite graph. For our case, this matrix is large and dense, rendering the algorithm infeasible due to the memory required. However, note that the bottleneck is the density of the matrix. Bipartite matching is computationally tractable with *sparse* matrices of exactly the same size because the algorithms utilize the sparsity to lower the memory requirement. For the purposes of this section, let $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{V}^T$ be the matrix we wish to run a bipartite matching on and $\boldsymbol{U}$ be $m \times k$. We note that the low-rank factors from MSD are non-negative, so we assume $\boldsymbol{U}, \boldsymbol{V} \geq 0$.

We begin with an extremely idealized case. If $k = 1$, then $\boldsymbol{Y} = \mathbf{u}\mathbf{v}^T$ is rank-1, and it is extremely easy to compute a maximum weight matching. This only involves *sorting* the vectors $\mathbf{u}$ and $\mathbf{v}$ in decreasing order and taking the largest feasible set of components. (This fact is a simple corollary of the rearrangement inequality, so we omit a formal statement and proof.)

For the remainder of this derivation, we'll need one additional bit of notation: the matrix inner product $\boldsymbol{A} \bullet \boldsymbol{B} = \sum_{ij} A_{ij} B_{ij}$. Recall that a matching can be expressed as a matrix where $\boldsymbol{P}_{ij} = 1$ if i is matched to j and all other entries are 0. Given any matching $\boldsymbol{P}$ the weight of the matching is $\boldsymbol{P} \bullet \boldsymbol{X}$. We now show that using the *best matching* from each rank-1 factor of $\boldsymbol{X}$ gives a $1/k$-approximation when $\boldsymbol{X}$ has rank-k.

Theorem 4.4.1 *Let $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{V}^T$ be a rank-k non-negative decomposition for $\boldsymbol{X}$. Let $\boldsymbol{P}_i$ be the maximum weight matching corresponding to the rank-1 factor $\mathbf{u}_i\mathbf{v}_i^T$, and let $f_i = \boldsymbol{P}_i \bullet (\mathbf{u}_i\mathbf{v}_i^T)$ be its weight. If f^* is the largest value of f_i, and $\boldsymbol{P}_{i^*}$ is the corresponding maximum weight matching, then $\boldsymbol{P}_{i^*}$ achieves a $1/k$-approximation to the maximum weight matching on $\boldsymbol{X}$.*

Proof Let $\boldsymbol{P}^*$ be any optimal maximum weight matching. Then

$$\underbrace{\boldsymbol{P}^* \bullet \boldsymbol{X}}_{\text{optimal matching weight}} = \underbrace{\textstyle\sum_{i=1}^k (\boldsymbol{P}^* \bullet \mathbf{u}_i\mathbf{v}_i^T) \leq \sum_{i=1}^k f_i}_{f_i \text{ is optimal for } \mathbf{u}_i\mathbf{v}_i^T} \leq kf^* \leq k\boldsymbol{P}_{i^*}\mathbf{u}_{i^*}\mathbf{v}_{i^*}^T \leq k\boldsymbol{P}_{i^*} \bullet \boldsymbol{X}$$

■

While this provides a simple approximation bound, the value of k for our experiments is often around 1000, rendering it rather pointless in terms of theory. However, we find that a single factor often produces exceptionally good approximation factors–far beyond what the theory from this result would show.

In practice, there are a few additional improvements we can make. Two of these apply to general problems and also produce a $1/k$ approximation. The third improvement is specific to the multimodal network alignment objective. We also discuss a few other ideas suggested in the literature.

Maximum weight 1/k. In the first variant, we compute the weight of the matching $\boldsymbol{P}_i$ in the full matrix $\boldsymbol{X}$ via $\boldsymbol{M} \bullet \boldsymbol{P}_i$. The runtime of this computation is nk and it can be done entirely with the low-rank factors. Then we select the largest weight among matchings $\boldsymbol{P}_i$. This is still a $1/k$ approximation because $\boldsymbol{X} \bullet \boldsymbol{P}_i \geq (\mathbf{u}_i\mathbf{v}_i^T) \bullet \boldsymbol{P}_i$ due to the non-negativity.

Union of matchings. While we cannot always expect to solve the dense bipartite matching problem, we are able to solve sparse bipartite matching problems on the size of the vertex sets easily. In this case, we can take the union of edges produced by the matchings $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_k$ and find the best bipartite matching in this sparse subset of the entries of $\boldsymbol{X}$. Again, this can only make our approximation better and so we still get $1/k$.

Maximum overlap. The actual goal of our problem is to achieve the best alignment between two multimodal networks. Thus, rather than compute the *weight* of the matching in the heuristic $\boldsymbol{X}$, we directly evaluate the *overlap* produced by each of the matchings $\boldsymbol{P}_i$ between the multimodal adjacency matrices. (Note this is slightly different from the matching we will consider after we resolve conflicts below.) This picks the matching $\boldsymbol{P}_i$ that corresponds with the true objective functions.

Other possibilities. In ref. (Kollias et al., 2014), they suggest computing $\boldsymbol{Y}$ a row at a time and retaining the largest r entries as edges of a graph. This forms a sparse graph that can be used in a parallel maximum weight bipartite matching. Alternatively, it is feasible to run a greedy 1/2-approximate matching. We found both of these underperformed our ideas in terms of final quality and were no faster.

Runtime *&* Memory All of these schemes keep the matrix $\boldsymbol{X}$ stored as factors $\boldsymbol{U}$ and $\boldsymbol{V}$. Each matching takes $O(n)$ storage, so storing k matches requires $O(nk)$ memory, and this exactly mirrors the storage of the factors $\boldsymbol{U}$ and $\boldsymbol{V}$ themselves. We provide a comparison of runtime in Table 4.1.

Method	Runtime
Simple $1/k$	$nk \log n$
Max Weight $1/k$	$nk \log n + nk^2$
Union $1/k$	$nk \log n + nk^2 + \text{BM}(nk)$
Max Overlap	$nk \log n + \lvert E \rvert$
r-Sparsified	$n^2 k + \text{BM}(nr)$
Greedy $1/2$	$n^3 k$

Table 4.1.
We compare runtimes of low rank approximate bipartite matching where $\boldsymbol{U}$ and $\boldsymbol{V}$ are $n \times k$ for simplicity. Also, $\text{BM}(E)$ is the runtime for bipartite matching on n nodes with E total entries, and $|E|$ is the number of edges in the original networks.

Practical considerations. All of the approximations here are extremely fast to compute for values of n up to a few million. So in our codes, we will usually compute multiple matchings and choose the best based on the overall alignment scores after we resolve conflicts as discussed next.

4.4.2 Resolving conflicts

So far, we have presented methods to derive a matching on a low-rank approximation. In the context of multimodal networks, this matches at the level of the row and column identifiers of the multimodal adjacency matrices. This is a problem because each vertex id in the multimodal network occurs at m different rows of the multimodal adjacency matrix. For instance, if multimodal network A consists of 3 modes, vertex 1 can be matched to three different vertices, one for each mode!

To resolve these conflicts, we have two methods and choose the best among them based on the overlap of the alignment they produce. The first method is to greedily examine the multimodal matching, project each match to nodes in V_A and V_B, and accept it if it's still feasible in the projected alignment. The second method is to project the multimodal matchings to one bipartite network on V_A and V_B. Put another way, we take the union of all matches induced by the single multimodal matching. Each edge is weighted by the value in $\boldsymbol{X}$ (and duplicates are summed) that arose from the matching. Once we project all the matchings, we use a bipartite matching to obtain the final matching.

4.5 Baseline methods

Before we describe our experiments, we explain how the multimodal alignment problem can be addressed using existing methods for network alignment. These methods do not take advantage of the multimodality of the data to guide the alignment, but nevertheless provide strong baselines. The first baseline is to smash the networks together and look at an alignment of the unimodal networks with edge set $\cup_{k=1}^{m} E_{\boldsymbol{A}}^{(k)}$ and $\cup_{k=1}^{m} E_{\boldsymbol{B}}^{(k)}$. We can use any existing network alignment tool for this task. In the following experiments, we use Klau's relaxation (Klau, 2009) because it had the highest performance among a number of methods we evaluated. In addition to aligning the smashed unimodal networks, we can also compute a pairwise alignment between each mode. Thus, we align the edges $E_{\boldsymbol{A}}^{(k)}$ to $E_{\boldsymbol{B}}^{(k)}$, which provides another matching. Note that each alignment produces exactly the type of output we need and there is no need to resolve conflicts as we saw in the multimodal case. We can then pick the *best* among these $m + 1$ alignments (1 alignment for each mode and 1 alignment for the smashed network) based on the total overlap they induce in the full multimodal network. We call these methods *pairwise alignment* rather than *multimodal alignment*.

4.6 Synthetic experiments

In our first experiment, we want to understand when multimodal alignment results in better alignment compared with using a strong algorithm for the pairwise case (such as Klau's relaxation method (Klau, 2009)). We use our multimodal method with $\alpha = 0.9$, 10 iterations, and then compute the dense matrix $\boldsymbol{X}$ to use with bipartite matching because this test case is sufficiently small. We compare it against

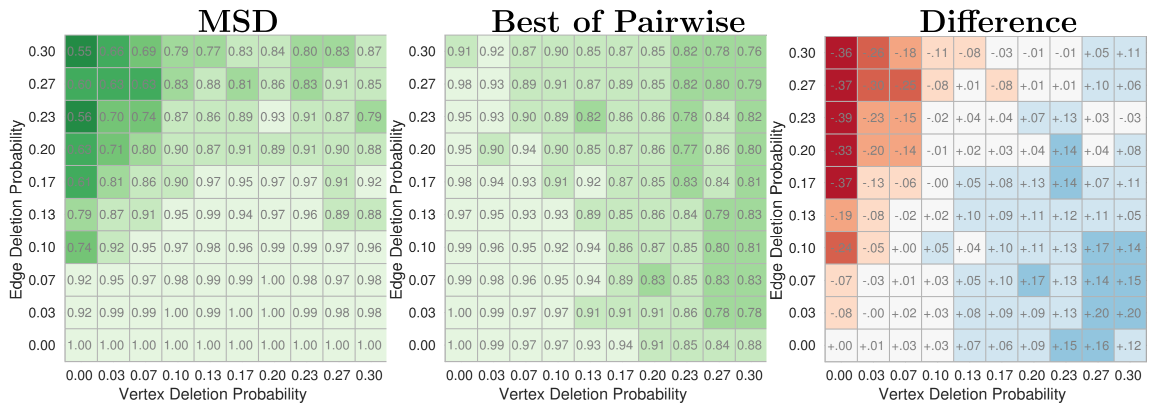


Figure 4.3. At left, the recovery results for our multimodal similarity decomposition (MSD) show excellent performance even in the high vertex deletion regime. In the middle, the best of any pairwise recovery result shows good results in the high edge deletion, but worse results with high vertex deletion. At right, the difference shows blue when MSD outperforms the best of the pairwise alignments by at least 5% recovery.

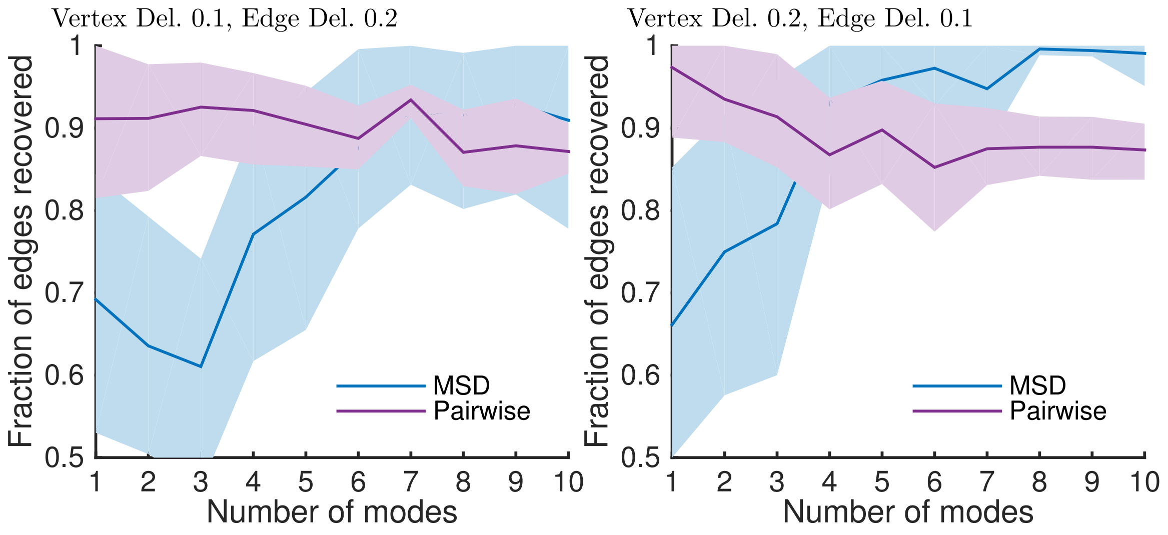


Figure 4.4. At left, we show how the recovery rate changes as we add modes when $p = 0.1$ and $q = 0.2$; at right we see $p = 0.2$ and $q = 0.1$. The performance of our multimodal alignment increases as we add modes. The line is the mean over 50 trials and the bands show the 10% and 90% percentiles.

Klau's method (Klau, 2009) as a pairwise aligner as in Section 4.5. Klau's algorithm takes significantly longer than MSD.

We generate a multimodal network alignment problem with 36 nodes and 6 modes as follows. We first create a 12-node Erdős-Rényi graph with average degree 3, and then join copies into a single 36-node network. The purpose of combining the networks is to put a degree of symmetry into the graph to make the alignment harder. This is our reference graph. Next, we generate a mode by randomly deleting vertices with probability p and randomly deleting edges with probability $q/2$. Then we generate two instances of this modal graph with deleted edges again with probability $q/2$. One instance goes to multimodal network $G_{\boldsymbol{A}}$ and the other goes to multimodal network $G_{\boldsymbol{B}}$. At the end, we have two multimodal networks $G_{\boldsymbol{A}}$ and $G_{\boldsymbol{B}}$ where each mode shares a number of relationships.

In figure 4.3, we show the fraction of edges aligned using a method over the total number of edges in the networks as we vary p and q for both MSD and the best of pairwise alignments (Section 4.5). We do not use node-based recovery due to many nodes of degree one, which cannot be resolved. The results are the mean over 50 trials, and they show a large regime where multimodal alignment is superior to any pairwise alignment. Specifically, when the vertex deletion probability p is large, the multimodal alignment is the only method to accurately align the networks. When edge deletion is high, smashing the networks effectively reconstitutes the original network and we see the superior performance of Klau's pairwise method.

Next, in Figure 4.4, we show how the behavior of the methods change as we vary the number of modes of the multimodal network. As expected, the performance of our method increases with additional modes, whereas the performance of the pairwise method is consistent (or even slightly decreasing).

4.7 De-anonymizing transportation datasets

In this section we present a case-study with de-anonymizing a publicly released dataset of airlines and airports, where we treat each airline as a mode. We show in this section that only our proposed multimodal alignment method can fully de-anonymize the network and align each edge. The anonymized network is the European air network from ref. (Nicosia and Latora, 2015), which was released with anonymized airport identifiers but with all airline identifiers. The data originally came from the OpenFlights repository (`http://openflights.org`). We wanted to take the original data and use it to restore identifiers to the anonymized data. We selected the May 2013 release based on the publication date of (Nicosia and Latora, 2015). These networks have 594 airports and 175 airlines and 6468 edges, respectively. Our goal is to align the multimodal network such that each edge is matched.

4.7.1 Performance at de-anonymization

We consider an alignment to de-anonymize the network if it is able to overlap all 6468 edges through the matched vertex identifiers. In Table 4.7.2, we show the overlap size for our multimodal methods and a number of different low-rank matching techniques from Section 4.4. This time we are not able to run the full bipartite matching unless we use a computer with $512GB$ of RAM and so we don't report those results. The best result from the baseline pairwise alignment methods miss 70 edges. All of our low-rank matching approximations achieve the full overlap score.

4.7.2 Which modes help multimodal alignment?

The final experiment seeks to understand *which* modes have the highest impact on the de-anonymization performance. In many applications, collecting additional data incurs a cost and this experiment is designed to provide insight as far as what type of multimodal data would be most helpful to collect. Our goal is to use only a subset of the 175 modes to produce an alignment, and then compute the overlap that results from using this alignment on all 175 modes. We sort the modes based on a number of graph theoretic measures to produce interesting subsets. The per-mode measures are: edge count, unique vertex count, average degree, triangle count, density. Figure 4.5 shows the results for these measures along with a few random orderings of the modes. The goal is to align the highest fraction of edges using the fewest modes. Most of the graph theoretic measures have similar performance, which suggests that any could be a proxy for which data to collect. Density is a notable outlier as some of the modes consist of a single edge, resulting in a high density, but little information about the global alignment.

Type	Method	Overlap
Multimodal	MSD Max Weight $1/k$	6468
Multimodal	MSD Union $1/k$	6468
Multimodal	MSD Max Overlap	6468
Pairwise	smashed	6398
Pairwise	best mode	3127

Table 4.2.
The multimodal airport networks have 6468 edges. All the multimodal methods are able to de-anonymize this network. The best pairwise results come from smashing the multimodal structure and aligning the resulting networks, which misses 70 edges compared to the multimodal alignments and takes more time.

4.8 Conclusions

In this chapter, we demonstrated that using multimodal features of data, when they exist makes alignment problems easier. We compared our multimodal network decomoposition method (MSD) against a carefully engineered method in pairwise alignment (Klau's). We also did all of the pairwise experiments reported here with the IsoRank method instead of Klau's and the results were uniformly worse as far as the alignment quality. However, when we use the multimodal extension to IsoRank

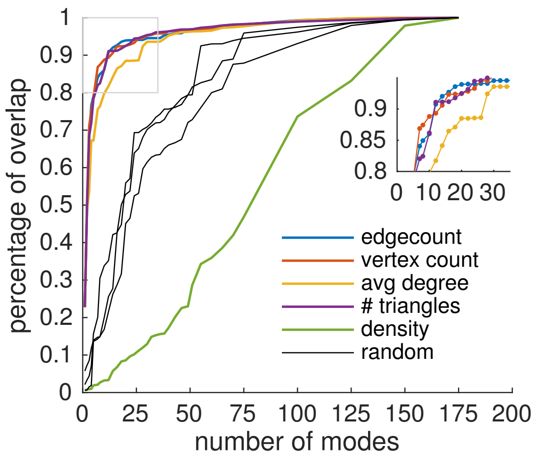


Figure 4.5. As we use more modes to produce an alignment for the airport data, the performance increases. This experiment shows that using modes that have many edges (edge count), touch many unique vertices (vertex count), have a high average degree (avg degree), or have high triangle count (# triangles) all outperform random selections. Alternatively, density does not work as discussed in the main text.

we have proposed, accurate alignments are easy to obtain. For instance, note that in Figure 4.4, MSD used the additional modes to improve the alignment, whereas the pairwise method did not.

MSD marks the first network alignment that can align data represented in a multimodal way. Taking advantage of modality proved to be useful and we believe that using this new methodology, combined with particular domain specific adaptations, will result in new insights about multimodal data specially in Bioninformatics research.

5 LOW RANK SPECTRAL NETWORK ALIGNMENT

We now bring our attention back to the standard network alignment problem presented in the beginning of this thesis. The aim of the standard pairwise network alignment problem is to produce one-to-one matches between nodes from two graphs such that the new alignment maximizes the number of overlapping edges. This chapter reports on our paper "Low rank spectral network alignment" (Nassar et al., 2018).

In this chapter, we use a recently proposed and theoretically grounded method (EigenAlign) as a building block. EigenAlign formulates the network alignment problem in a slightly different way than the standard maximization of overlap presented in the beginning of this thesis. Our adaptation of the EigenAlign algorithm admits a novel implementation which requires memory that is linear in the size of the graphs to be aligned. The key step to this insight is identifying low-rank structure in the node-similarity matrix used by EigenAlign for determining matches. With an exact, closed-form low-rank structure, we then solve a maximum weight bipartite matching problem on that low-rank matrix to produce the matching between the graphs. For this task, we show a new, a-posteriori, approximation bound for a simple algorithm to approximate a maximum weight bipartite matching problem on a low-rank matrix. The combination of our two new methods then enables us to tackle much larger network alignment problems than previously possible and to do so quickly. Problems that take hours with existing methods take only seconds with our new algorithm. We thoroughly validate our low-rank algorithm against the original EigenAlign approach. We also compare a variety of existing algorithms on problems in bioinformatics and social networks. Our approach can also be combined with existing algorithms to improve their performance and speed.

The IsoRank method is also based on eigenvectors, or more specifically, the PageRank vector of the product-graph of the two networks was used for the same purpose (Singh et al., 2008).

5.1 Motivation

EigenAlign (Feizi et al., 2016) approaches this problem in an innovative way. It uses the dominant eigenvector of a matrix related to the product-graph between the two networks in order to estimate the similarity. The eigenvector information is rounded into a matching between the vertices of the graphs by solving a maximum-weight bipartite matching problem on a dense bipartite graph (Feizi et al., 2016). In contrast to IsoRank, which is also based on eigenvectors, the key innovation of EigenAlign is that it explicitly models nodes that may not have a match in the network. In this way, it is able to provably align many simple graph models such as Erdős-Rényi when the graphs do not have too much noise. This gives it a firm theoretical basis *although it still suffers from the quadratic memory requirement.*

We highlight a number of innovations that enable the EigenAlign methodology to work without the quadratic memory requirement. We first show that the EigenAlign solution can be expressed via low-rank factors, and we can compute these low-rank factors exactly and explicitly using a simple procedure. A challenge in using low-rank information here is that the low rank factors used are no longer non-negative (in contrast to the previous chapter), and there are generally very few ideas on bipartite matching with low rank data (Nassar and Gleich, 2017). One study analyzes the locality of bipartite matching on low rank data (Liu and Teng, 2016), but there are generally a scarcity of algorithms that can compute a bipartite matching on a low rank matrix. We contribute a new analysis of a simple idea that improves our k-approximation bound empirically. Our new improved algorithm gives a computable a-posteriori approximation guarantee. In practice, this approximation guarantee is extremely good: around 1.1. The rest of this chapter is divided as follows.

- Section 5.2 shows an explicit expression for the solution of the EigenAlign eigenvector as a low-rank matrix (Theorem 5.3.1).
- Section 5.3 shows an $O(n \log n)$ method that will solve a maximum-weight bipartite matching problem on a low-rank matrix with an a-posteriori approximation guarantee (Algorithm 3, Theorem 5.4.1). In practice, these approximation guarantees are better than 1.1 for our experiments (Figure 5.5). This improves our k-approximation algorithm from Chapter 1 (where k is the rank).
- Section 5.5.1 is a thorough evaluation of our methodology to show that there appears to be little difference between the results of our low-rank methods and the original EigenAlign (Section 5.5.1), and our methods are more scalable.
- Section 5.5.2 is a demonstration that our low-rank methods can be combined with existing network alignment methods to yield better quality results.
- Section 5.5.3 is a demonstration that the methods are sufficiently scalable to be run for all pairs of networks induced by the vertex neighborhoods of every two connected nodes in a large graph. That is, we seek to align two vertex neighborhoods together whenever the vertices have an edge. To validate the alignments, we show that these track the Jaccard similarity between the set of neighbors. (Figure 6.7).

5.2 Key differences in EigenAlign formulation

We now revisit standard network alignment algorithms and contrast them with our specific setting and objective. A helpful illustration is shown in Figure 5.1. Recall that the standard network alignment problem seeks to find a binary matrix $\boldsymbol{P}$ that encodes a matching between the nodes of the networks and maximizes one of a few

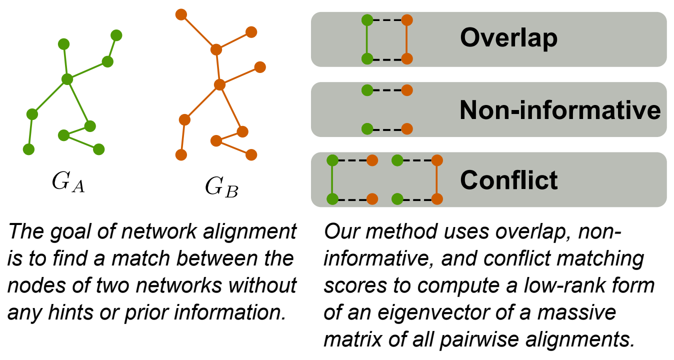

Overlap
Non-informative
Conflict
G_A
G_B
The goal of network alignment is to find a match between the nodes of two networks without any hints or prior information.
Our method uses overlap, non-informative, and conflict matching scores to compute a low-rank form of an eigenvector of a massive matrix of all pairwise alignments.

Figure 5.1. Our setup for network alignment follows Feizi et. al. (Feizi et al., 2016), where we seek to align two networks without any other metadata. Possible alignments between pairs of any nodes $(\boldsymbol{i}, \boldsymbol{j})$ in $G_{\boldsymbol{A}}$ and $(\boldsymbol{i}', \boldsymbol{j}')$ in $G_{\boldsymbol{B}}$ are scored based on one of three cases and assembled into a massive, but highly structured, alignment matrix $\boldsymbol{M}$.

possible objective functions discussed below. The matrix $\boldsymbol{P}$ encodes a matching when it satisfies the constraints

$$\boldsymbol{P}_{u,v} = \begin{cases} 1 & u \text{ is matched with } v \\ 0 & \text{otherwise.} \end{cases}, \quad \begin{array}{l} \sum_u \boldsymbol{P}_{u,v} \le 1 \text{ for all } v, \\ \sum_v \boldsymbol{P}_{u,v} \le 1 \text{ for all } u. \end{array}$$

The inequality constraints guarantee that a node in the first network is only matched with one or zero nodes in the other network. And this matrix $\boldsymbol{P}$ maximizes the number of overlapping edges between $G_{\boldsymbol{A}}$ and $G_{\boldsymbol{B}}$ in the standard pairwise network alignment problem, which we restate here for convenience.

$$\begin{array}{ll} \underset{\boldsymbol{P}}{\text{maximize}} & \sum_{ij} [\boldsymbol{P}^T \boldsymbol{A} \boldsymbol{P}]_{ij} [\boldsymbol{B}]_{ij} \\ \text{subject to} & \sum_u \boldsymbol{P}_{u,v} \le 1 \text{ for all } v \\ & \sum_v \boldsymbol{P}_{u,v} \le 1 \text{ for all } u \\ & \boldsymbol{P}_{u,v} \in \{0,1\}. \end{array}$$

5.2.1 The EigenAlign Algorithm

One of the drawbacks to the previous objective functions is there is no downside to matches that do not produce an overlap, i.e. edges in $G_{\boldsymbol{A}}$ that are mapped to non-edges in $G_{\boldsymbol{B}}$ or vice versa. Neither do these objective functions consider the case where non-edges in $G_{\boldsymbol{A}}$ are mapped to non-edges in $G_{\boldsymbol{B}}$. The first problem was recognized in Schellewald and Schnörr (2005) which proposed an SDP-based method to minimize the number of conflicting matches. More recently, the EigenAlign objective (Feizi et al., 2016) included explicit terms for these three cases: overlaps, non-informative matches, and conflicts, see Figure 5.1. The alignment score corresponding to $\boldsymbol{P}$ in this case is

$$\text{AlignmentScore}(\boldsymbol{P}) = s_O(\#\text{ overlaps}) + s_N(\#\text{ non-informatives}) + s_C(\#\text{ conflicts}) \tag{5.1}$$

where s_O, s_N, and s_C are weights for overlaps, non-informatives and conflicts. These constants should be chosen such that $s_O > s_N > s_C$. By setting s_N and s_C to zero we recover the ordinary notion of maximizing the number of overlaps. Although it may seem strange to maximize the number of conflicts, when the graphs have very different sizes or numbers of edges, this term acts as regularization. The important piece is that *non-informatives* are more valuable than conflicts.

This objective can be expressed formally by first introducing a massive *alignment matrix* $\boldsymbol{M}$ defined as follows: for all pairs of nodes i_A, j_A in $G_{\boldsymbol{A}}$ and all pairs i'_B, j'_B in $G_{\boldsymbol{B}}$, if $\boldsymbol{P}(i_A, i'_B) = 1$ and $\boldsymbol{P}(j_A, j'_B) = 1$, then

$$\boldsymbol{M}[(i_A, i'_B), (j_A, j'_B)] = \begin{cases} s_O, & \text{if } (i_A, j_A), (i'_B, j'_B) \text{ are overlaps} \\ s_N, & \text{if } (i_A, j_A), (i'_B, j'_B) \text{ are noninformatives} \\ s_C, & \text{if } (i_A, j_A), (i'_B, j'_B) \text{ are conflicts.} \end{cases}$$

We are abusing notations a bit in this definition and using pairs i_A and i'_B to index the rows and columns of this matrix. For a straightforward, canonical ordering of these pairs i_A, i'_B, the matrix $\boldsymbol{M}$ can be rewritten in terms of the adjacency matrices of $\boldsymbol{A}$ and $\boldsymbol{B}$:

$$\boldsymbol{M} = c_1(\boldsymbol{B} \otimes \boldsymbol{A}) + c_2(\boldsymbol{E}_B \otimes \boldsymbol{A}) + c_2(\boldsymbol{B} \otimes \boldsymbol{E}_A) + c_3(\boldsymbol{E}_B \otimes \boldsymbol{E}_A)$$

where $\otimes$ denotes the Kronecker product, $c_1 = s_O + s_N - 2s_C, c_2 = s_C - s_N, c_3 = s_N$ and $\boldsymbol{E}_A$, and $\boldsymbol{E}_B$ are the matrices of all ones and of the same size as $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively. The matrix $\boldsymbol{M}$ is symmetric as long as $G_{\boldsymbol{A}}$ and $G_{\boldsymbol{B}}$ are undirected.

Maximizing the alignment score (5.1) is then equivalent to the following quadratic assignment problem:

$$\begin{array}{ll} \underset{\mathbf{y}}{\text{maximize}} & \mathbf{y}^T \boldsymbol{M} \mathbf{y} \\ \text{subject to} & y_i \in \{0,1\} \\ & \sum_u y[u,v] \leq 1 \text{ for all } v \in V_B \\ & \sum_v y[u,v] \leq 1 \text{ for all } u \in V_A \end{array} \tag{5.2}$$

where V_A and $V_{\boldsymbol{B}}$ are the node sets of $G_{\boldsymbol{A}}$ and $G_{\boldsymbol{B}}$ respectively. The vector $\mathbf{y}$ is really just the *vector of data* representing the matching matrix $\boldsymbol{P}$, and the constraints are just the translation of the matching constraints from (5.2).

An empirically and theoretically successful method for optimizing this objective is to solve an eigenvector equation instead of the quadratic program. This is exactly the approach of EigenAlign, which computes network alignments using the following two steps:

1. Find the eigenvector $\mathbf{x}$ of $\boldsymbol{M}$ that corresponds to the eigenvalue of largest magnitude. Note, $\boldsymbol{M}$ is of dimension $n_A n_B \times n_A n_B$, where n_A and n_B are the number of nodes in $G_{\boldsymbol{A}}$ and $G_{\boldsymbol{B}}$ respectively; so, the eigenvector will be of dimension $n_A n_B$, and can be reshaped into an $n_A \times n_B$ matrix $\boldsymbol{X}$ where each entry represents a score for every pair of nodes between the two graphs. *We call this a similarity matrix because it reflects the topological similarity between vertices of $G_{\boldsymbol{A}}$ and $G_{\boldsymbol{B}}$.*

2. Run a bipartite matching algorithm on the similarity matrix $\boldsymbol{X}$ that maximizes the total weight of the final alignment.

In our work we extend the foundation laid by EigenAlign by considering improvements to both steps. We first show that the similarity matrix $\boldsymbol{X}$ can be accurately represented through an exact low-rank factorization. This allows us to avoid the quadratic memory requirement of EigenAlign. We then present several new fast techniques for bipartite matching problems on low-rank matrices. Together these improvements yield a low-rank EigenAlign algorithm that is far more scalable in practice.

Our work shares a number of similarities with the Network Similarity Decomposition (NSD) (Kollias et al., 2012) (the technique based on a low-rank factorization of the similarity matrix from IsoRank).

5.3 Low Rank Factors of EigenAlign

The first step of the EigenAlign algorithm is to compute the dominant eigenvector of the symmetric matrix $\boldsymbol{M}$. Feizi et al. suggest obtaining a similarity matrix $\boldsymbol{X}$ by first forming $\boldsymbol{M}$, performing a power iteration on this matrix, and reshaping the final output eigenvector $\mathbf{x}$ into $\boldsymbol{X}$ (Feizi et al., 2016). Because of the Kronecker structure in $\boldsymbol{M}$, this can equivalently be formulated directly as the matrix $\boldsymbol{X}$ that satisfies:

$$\begin{array}{ll} \underset{\boldsymbol{X}}{\text{maximize}} & \boldsymbol{X} \bullet (c_1 \boldsymbol{A}\boldsymbol{X}\boldsymbol{B}^T + c_2 \boldsymbol{A}\boldsymbol{X}\boldsymbol{E}^T + c_2 \boldsymbol{E}\boldsymbol{X}\boldsymbol{B}^T + c_3 \boldsymbol{E}\boldsymbol{X}\boldsymbol{E}^T) \\ \text{subject to} & \|\boldsymbol{X}\|_F = 1, \boldsymbol{X} \in \mathbb{R}^{n_A \times n_B}. \end{array} \tag{5.3}$$

In this expression, $\boldsymbol{X} \bullet \boldsymbol{Y} = \sum_{ij} X_{ij} Y_{ij}$ is the matrix inner-product and the translation from the eigenvector of $\boldsymbol{M}$ follows from the Kronecker product property $\text{vec}(\boldsymbol{A}\boldsymbol{X}\boldsymbol{B}^T) = (\boldsymbol{B} \otimes \boldsymbol{A})\text{vec}(\boldsymbol{X})$. We also dropped the dimensions from the matrices $\boldsymbol{E}$ of all ones. The eigenvector of $\boldsymbol{M}$ is the result of the *vec* operation on the matrix $\boldsymbol{X}$, which converts the matrix into a vector by concatenating columns.

Our first major contribution is to show that if the matrix $\boldsymbol{X}$ is estimated with the power-method starting from a rank 1 matrix, then the kth iteration of the power method results in a rank $k+1$ matrix that we can explicitly and exactly compute.

5.3.1 A Four Factor Low-Rank Decomposition

In the matrix form of problem (5.3), one step of the power method corresponds to the iteration:

$$\boldsymbol{X}_{k+1} = c_1 \boldsymbol{A}\boldsymbol{X}_k\boldsymbol{B}^T + c_2 \boldsymbol{A}\boldsymbol{X}_k\boldsymbol{E}^T + c_2 \boldsymbol{E}\boldsymbol{X}_k\boldsymbol{B}^T + c_3 \boldsymbol{E}\boldsymbol{X}_k\boldsymbol{E}^T. \tag{5.4}$$

If we begin with a rank-1 matrix $\boldsymbol{X}_0 = \mathbf{u}\mathbf{v}^T$ where $\mathbf{u} \in \mathbb{R}^{n_A}$ and $\mathbf{v} \in \mathbb{R}^{n_B}$ and let $\boldsymbol{U}_0 = \mathbf{u}$, and $\boldsymbol{V}_0 = \mathbf{v}$ so that $\boldsymbol{X}_0 = \boldsymbol{U}_0\boldsymbol{V}_0^T$. We will first prove by induction that $\boldsymbol{X}_k$ can be written as

$$\boldsymbol{X}_k = \boldsymbol{U}_k\boldsymbol{V}_k^T \tag{5.5}$$

where

$$\begin{aligned}\boldsymbol{U}_k &= [c_1\boldsymbol{A}\boldsymbol{U}_{k-1} \mid c_2\boldsymbol{E}\boldsymbol{U}_{k-1} \mid c_2\boldsymbol{A}\boldsymbol{U}_{k-1} \mid c_3\boldsymbol{E}\boldsymbol{U}_{k-1}] \\ \boldsymbol{V}_k &= [\boldsymbol{B}\boldsymbol{V}_{k-1} \mid \boldsymbol{B}\boldsymbol{V}_{k-1} \mid \boldsymbol{E}\boldsymbol{V}_{k-1} \mid \boldsymbol{E}\boldsymbol{V}_{k-1}].\end{aligned}$$

The base case of our induction follows directly from our definition of $\boldsymbol{X}_0$. Assume now that the equivalence between (5.4) and (5.5) holds up to k and we will prove the equivalence for $k+1$. We begin with equation (5.4) and plug in the decomposition of $\boldsymbol{X}_k$ from (5.5):

$$\begin{aligned}
&\boldsymbol{X}_{k+1} \\
&\quad = c_1 \boldsymbol{A}\boldsymbol{X}_k\boldsymbol{B}^T + c_2 \boldsymbol{A}\boldsymbol{X}_k\boldsymbol{E}^T + c_2 \boldsymbol{E}\boldsymbol{X}_k\boldsymbol{B}^T + c_3 \boldsymbol{E}\boldsymbol{X}_k\boldsymbol{E}^T \\
&\quad = c_1 \boldsymbol{A}\boldsymbol{U}_k\boldsymbol{V}_k^T\boldsymbol{B}^T + c_2 \boldsymbol{A}\boldsymbol{U}_k\boldsymbol{V}_k^T\boldsymbol{E}^T + c_2 \boldsymbol{E}\boldsymbol{U}_k\boldsymbol{V}_k^T\boldsymbol{B}^T + c_3 \boldsymbol{E}\boldsymbol{U}_k\boldsymbol{V}_k^T\boldsymbol{E}^T \\
&\quad = [c_1\boldsymbol{A}\boldsymbol{U}_k \mid c_2\boldsymbol{E}\boldsymbol{U}_k \mid c_2\boldsymbol{A}\boldsymbol{U}_k \mid c_3\boldsymbol{E}\boldsymbol{U}_k][\boldsymbol{B}\boldsymbol{V}_k \mid \boldsymbol{B}\boldsymbol{V}_k \mid \boldsymbol{E}\boldsymbol{V}_k \mid \boldsymbol{E}\boldsymbol{V}_k]^T \\
&\quad = \boldsymbol{U}_{k+1}\boldsymbol{V}_{k+1}^T.
\end{aligned}$$

This form of the factorization is not yet helpful, because the matrix $\boldsymbol{U}_k$ is of dimension $n_A \times 4^k$. To show that this is indeed a rank $k+1$ matrix, we show

$$\boldsymbol{X}_k = \boldsymbol{S}_k\boldsymbol{C}_k\boldsymbol{T}_k^T\boldsymbol{R}_k^T$$

where:

$$\boldsymbol{S}_k = [\boldsymbol{A}^k\mathbf{u} \mid \boldsymbol{A}^{k-1}\mathbf{e} \mid \ldots \mid \boldsymbol{A}\mathbf{e} \mid \mathbf{e}]$$

$$\boldsymbol{R}_k = [\boldsymbol{B}^k\mathbf{v} \mid \boldsymbol{B}^{k-1}\mathbf{e} \mid \ldots \mid \boldsymbol{B}\mathbf{e} \mid \mathbf{e}]$$

$$\boldsymbol{C}_k = \begin{bmatrix} c_1\boldsymbol{C}_{k-1} & \mathbf{0} & c_2\boldsymbol{C}_{k-1} & \mathbf{0} \\ \mathbf{0}^T & c_2\mathbf{r}_k^T\boldsymbol{C}_{k-1} & \mathbf{0}^T & c_3\mathbf{r}_k^T\boldsymbol{C}_{k-1} \end{bmatrix}$$

$$\boldsymbol{T}_k = \begin{bmatrix} \boldsymbol{T}_{k-1} & \boldsymbol{T}_{k-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0}^T & \mathbf{0}^T & \mathbf{h}_k^T\boldsymbol{T}_{k-1} & \mathbf{h}_k^T\boldsymbol{T}_{k-1} \end{bmatrix}.$$

In the above, $\mathbf{0}$ is the all zeros matrix or vector of appropriate size, and $\mathbf{e}$ is the all ones vector. Also, $\boldsymbol{C}_0 = \boldsymbol{T}_0 = 1$, and $\mathbf{r}_k$ and $\mathbf{h}_k$ are defined as follows:

$$\mathbf{r}_k = [\ \mathbf{e}^T\boldsymbol{A}^{k-1}\mathbf{u} \quad \mathbf{e}^T\boldsymbol{A}^{k-2}\mathbf{e} \quad \cdots \quad \mathbf{e}^T\boldsymbol{A}^1\mathbf{e} \quad \mathbf{e}^T\boldsymbol{A}^0\mathbf{e}\ ]^T$$

$$\mathbf{h}_k = [\ \mathbf{e}^T\boldsymbol{B}^{k-1}\mathbf{v} \quad \mathbf{e}^T\boldsymbol{B}^{k-2}\mathbf{e} \quad \cdots \quad \mathbf{e}^T\boldsymbol{B}^1\mathbf{e} \quad \mathbf{e}^T\boldsymbol{B}^0\mathbf{e}\ ]^T$$

with $\mathbf{r}_1 = [\mathbf{e}^T \boldsymbol{A}^0 \mathbf{u}]$ and $\mathbf{h}_1 = [\mathbf{e}^T \boldsymbol{B}^0 \mathbf{v}]$. Note that this form gives the rank $k+1$ decomposition we desire because $\boldsymbol{S}_k$ and $\boldsymbol{R}_k$ both have $k+1$ columns.

To complete our derivation, we show $\boldsymbol{U}_k = \boldsymbol{S}_k \boldsymbol{C}_k$ again using induction. The base case $k = 0$ is immediate from a simple expansion of the initial definitions, so assume that the result holds for up to integer k. Then,

$$\begin{aligned}
\boldsymbol{U}_{k+1} &= [c_1 \boldsymbol{A}\boldsymbol{U}_k \mid c_2 \boldsymbol{E}\boldsymbol{U}_k \mid c_2 \boldsymbol{A}\boldsymbol{U}_k \mid c_3 \boldsymbol{E}\boldsymbol{U}_k] \\
&= [\boldsymbol{A}\boldsymbol{U}_k \mid \boldsymbol{E}\boldsymbol{U}_k] \left[\begin{smallmatrix} c_1 \boldsymbol{I} & \mathbf{0} & c_2 \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & c_2 \boldsymbol{I} & \mathbf{0} & c_3 \boldsymbol{I} \end{smallmatrix}\right] \\
&= [\boldsymbol{A}\boldsymbol{S}_k \boldsymbol{C}_k \mid \boldsymbol{E}\boldsymbol{S}_k \boldsymbol{C}_k] \left[\begin{smallmatrix} c_1 \boldsymbol{I} & \mathbf{0} & c_2 \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & c_2 \boldsymbol{I} & \mathbf{0} & c_3 \boldsymbol{I} \end{smallmatrix}\right].
\end{aligned}$$

Now, note that $\boldsymbol{A}\boldsymbol{S}_k = \boldsymbol{S}_{k+1} \left[\begin{smallmatrix} \boldsymbol{I} \\ \mathbf{0}^T \end{smallmatrix}\right]$ and $\boldsymbol{E}\boldsymbol{S}_k = \boldsymbol{S}_{k+1} \left[\begin{smallmatrix} \mathbf{0} \\ \mathbf{r}_{k+1}^T \end{smallmatrix}\right]$. Thus

$$\boldsymbol{U}_{k+1} = \boldsymbol{S}_{k+1} \left[\begin{smallmatrix} \boldsymbol{I} & \mathbf{0} \\ \mathbf{0}^T & \mathbf{r}_{k+1}^T \end{smallmatrix}\right] \left[\begin{smallmatrix} \boldsymbol{C}_k & \mathbf{0} \\ \mathbf{0} & \boldsymbol{C}_k \end{smallmatrix}\right] \left[\begin{smallmatrix} c_1 \boldsymbol{I} & \mathbf{0} & c_2 \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & c_2 \boldsymbol{I} & \mathbf{0} & c_3 \boldsymbol{I} \end{smallmatrix}\right] = \boldsymbol{S}_{k+1} \boldsymbol{C}_{k+1}$$

Applying the same set of steps again will yield that $\boldsymbol{V}_k = \boldsymbol{R}_k \boldsymbol{T}_k$.

5.3.2 Three and Two Factor Decompositions

While this four factor decomposition is useful for revealing the rank of $\boldsymbol{X}_k$, we do not wish to work with matrices $\boldsymbol{C}_k$ and $\boldsymbol{T}_k$ in practice since each has 4^k columns. We now show that their product $\boldsymbol{C}_k \boldsymbol{T}_k^T$ yields a simple-to-compute matrix $\boldsymbol{W}_k$ of size $(k+1) \times (k+1)$, giving us a three-factor decomposition (3FD):

$$\boldsymbol{X}_k = \boldsymbol{S}_k \boldsymbol{W}_k \boldsymbol{R}_k.$$

The matrix $\boldsymbol{W}_k$ is defined iteratively by:

$$\boldsymbol{W}_k = \boldsymbol{C}_k \boldsymbol{T}_k^T = \begin{bmatrix} c_1 \boldsymbol{W}_{k-1} & c_2 \boldsymbol{W}_{k-1} \mathbf{h}_k \\ c_2 \mathbf{r}_k^T \boldsymbol{W}_{k-1} & c_3 \mathbf{r}_k^T \boldsymbol{W}_{k-1} \mathbf{h}_k \end{bmatrix}.$$

with $\boldsymbol{W}_0 = \boldsymbol{C}_0 \boldsymbol{T}_0^T = 1 \cdot 1 = 1$. This follows from multiplying $\boldsymbol{C}_k$ with $\boldsymbol{T}_k^T$ together.

This decomposition is a step closer to our final goal but suffers from poor scaling of numbers in the factors. Consequently, we can remedy this by using scaling diagonal matrices in order to present our final well-scaled three factor decomposition of $\boldsymbol{X}_k$, which we present as a summarizing theorem:

Theorem 5.3.1 *If* $\boldsymbol{X}_0 = \mathbf{u}\mathbf{v}^T$ *for vectors* $\mathbf{u} \in \mathbb{R}^{n_A \times 1}$ *and* $\mathbf{v} \in \mathbb{R}^{n_B \times 1}$*, then the kth iteration of update* (5.4) *permits the following low-rank factorization:*

$$\boldsymbol{X}_k = \tilde{\boldsymbol{U}}_k \tilde{\boldsymbol{W}}_k \tilde{\boldsymbol{V}}_k^T$$

where

$$\tilde{\boldsymbol{U}}_k = \begin{bmatrix} \frac{\boldsymbol{A}^k\mathbf{u}}{\|\boldsymbol{A}^{\mathbf{k}}\mathbf{u}\|_\infty} & \frac{\boldsymbol{A}^{k-1}\mathbf{e}}{\|\boldsymbol{A}^{\mathbf{k-1}}\mathbf{e}\|_\infty} & \cdots & \frac{\boldsymbol{A}\mathbf{e}}{\|\boldsymbol{A}\mathbf{e}\|_\infty} & \frac{\mathbf{e}}{\|\mathbf{e}\|_\infty} \end{bmatrix}$$

$$\tilde{\boldsymbol{V}}_k = \begin{bmatrix} \frac{\boldsymbol{B}^k\mathbf{u}}{\|\boldsymbol{B}^{\mathbf{k}}\mathbf{u}\|_\infty} & \frac{\boldsymbol{B}^{k-1}\mathbf{e}}{\|\boldsymbol{B}^{\mathbf{k-1}}\mathbf{e}\|_\infty} & \cdots & \frac{\boldsymbol{B}\mathbf{e}}{\|\boldsymbol{B}\mathbf{e}\|_\infty} & \frac{\mathbf{e}}{\|\mathbf{e}\|_\infty} \end{bmatrix}$$

$$\tilde{\boldsymbol{W}}_k = \boldsymbol{D}_u \boldsymbol{W}_k \boldsymbol{D}_v.$$

Here $\boldsymbol{D}_u$ *is a* $(k+1) \times (k+1)$ *diagonal matrix with entries* $(\|\boldsymbol{A}^k\mathbf{u}\|, \|\boldsymbol{A}^{k-1}\mathbf{e}\|, \ldots, \|\boldsymbol{A}\mathbf{e}\|, \|\mathbf{e}\|)$ *on the diagonal, and* $\boldsymbol{D}_v$ *is a diagonal matrix with entries* $(\|\boldsymbol{B}^k\mathbf{v}\|, \|\boldsymbol{B}^{k-1}\mathbf{e}\|, \ldots, \|\boldsymbol{B}\mathbf{e}\|, \|\mathbf{e}\|)$.

The diagonal matrices in Theorem (5.3.1) are designed specifically to satisfy $\boldsymbol{S}_k\boldsymbol{D}_u^{-1} = \tilde{\boldsymbol{U}}_k$, $\boldsymbol{R}_k\boldsymbol{D}_v^{-1} = \tilde{\boldsymbol{V}}_k$, so the equivalence between the scaled and unscaled three factor decompositions is straightforward. Note that the result is still unnormalized. However, we can easily normalize in practice by scaling the matrix $\tilde{\boldsymbol{W}}_k$ as we see fit.

Note that when computing this decomposition in practice, we do not simply construct $\boldsymbol{S}, \boldsymbol{R}$, and $\boldsymbol{W}$ and then scale with $\boldsymbol{D}_u$ and $\boldsymbol{D}_v$. Instead, we form the scaled factors recursively by noting the similarities between each factor at step k and the corresponding factor at step $k+1$. A pseudo-code for our implementation that directly computes these is shown in Algorithm 2.

As we shall see in the next section, we would ultimately like to express $\boldsymbol{X}_k$ in terms of a just a left and a right low-rank factor in order to apply our techniques for

Algorithm 2 The pseudocode of the algorithm to decompose $\boldsymbol{X}$ into two low-rank matrices. Note that $\circ$ refers to the element-wise Hadamard product between two vectors.

```
1: Input: $\boldsymbol{A}, \boldsymbol{B}, \mathbf{u}, \mathbf{v}, c_1, c_2, c_3, k$
2: Output: $\hat{\boldsymbol{U}}$ and $\hat{\boldsymbol{V}}$ such that $\boldsymbol{X}_k = \hat{\boldsymbol{U}}\hat{\boldsymbol{V}}^T$
3: Compute $\tilde{\boldsymbol{U}}_k, \tilde{\boldsymbol{V}}_k$ and save norms in $\mathbf{a}[1], \ldots, \mathbf{a}[k+1]; \mathbf{b}[1], \ldots, \mathbf{b}[k+1]$.
4: Compute the terms necessary to build $\mathbf{r}_1, \ldots, \mathbf{r}_k$ and $\mathbf{h}_1, \ldots, \mathbf{h}_k$.
5: Define vectors $\mathbf{a}_i = \frac{\mathbf{a}[i+1]}{\mathbf{a}[1 \ldots i]}$ for $i = 1, \ldots, k$
6: Define vectors $\mathbf{b}_i = \frac{\mathbf{b}[i+1]}{\mathbf{b}[1 \ldots i]}$ for $i = 1, \ldots, k$
7: Set $\tilde{\boldsymbol{W}}_0 = \mathbf{a}[1]\mathbf{b}[1]$
8: for k=1 to k do
9:     Update $\tilde{\boldsymbol{W}}_i = \begin{bmatrix} c_1\tilde{\boldsymbol{W}}_{i-1} & c_2\tilde{\boldsymbol{W}}_{i-1}(\mathbf{b}_i \circ \mathbf{h}_i) \\ c_2(\mathbf{r}_i \circ \mathbf{a}_i)^T\tilde{\boldsymbol{W}}_{i-1} & c_3(\mathbf{r}_i \circ \mathbf{a}_i)^T\tilde{\boldsymbol{W}}_{i-1}(\mathbf{b}_i \circ \mathbf{h}_i) \end{bmatrix}$
10: end for
11: Compute $\boldsymbol{U}_W, \boldsymbol{S}_W, \boldsymbol{V}_W = \text{SVD}(\tilde{\boldsymbol{W}}_k)$ and set $\boldsymbol{D} = \boldsymbol{S}_W^{1/2}$
12: return $\hat{\boldsymbol{U}} = \tilde{\boldsymbol{U}}_k\boldsymbol{U}_W\boldsymbol{D}, \quad \hat{\boldsymbol{V}} = \tilde{\boldsymbol{V}}_k\boldsymbol{V}_W\boldsymbol{D}$
```

low-rank bipartite matching. It is preferable for our purposes to produce two factors that have roughly equal scaling, so we accomplish this by factorizing $\tilde{\boldsymbol{W}}_k$ using an SVD decomposition and splitting the pieces of $\tilde{\boldsymbol{W}}_k$ into the left and right terms. The last steps of the Algorithm 2 accomplish this goal.

5.4 Low Rank Matching

In this section, we consider the problem of solving a maximum weight bipartite matching problem on a low rank matrix with a useful a-posteriori approximation guarantee. In our network alignment routine, our algorithm will be used on the low-rank matrix from Algorithm 2. In this section, however, we proceed in terms of a general matrix $\boldsymbol{Y}$ with low rank factors $\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{V}^T$. The matrix $\boldsymbol{Y}$ represents the edge-weights of a bipartite graph, and so the max-weight matching problem is:

$$\begin{array}{ll} \underset{\boldsymbol{M}}{\text{maximize}} & \boldsymbol{M} \bullet \boldsymbol{Y} \\ \text{subject to} & M_{i,j} \in \{0,1\} \\ & \sum_i M_{i,j} \leq 1 \text{ for all } j, \sum_{ij} M_{i,j} \leq 1 \text{ for all } i, \end{array} \tag{5.6}$$

where $\bullet$ is the matrix inner-product (see (5.3)). The $M_{i,j}$ entries represent a match between node i on one side of the bipartite graph and node j on the other side. We call any $\boldsymbol{M}$ that satisfies the matching constraints a matching matrix.

5.4.1 Optimal Matching on a Rank 1 Matrix

We begin by considering optimal matchings for a rank-1 matrix $\boldsymbol{Y} = \mathbf{u}\mathbf{v}^T$ where $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ (these results are easily adapted for vectors of different lengths).

Case 1: $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n_{\geq 0}$ or $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n_{\leq 0}$ If $\mathbf{u}$ and $\mathbf{v}$ contain only non-negative entries, *or* both contain only non-positive entries, the procedure for finding the optimal matching is the same: we order the entries of both vectors by magnitude and pair up elements as they appear in the sorted list. If any pair contributes a 0 weight, we do not bother to match that pair since it doesn't improve the overall matching score. The optimality of this matching for these special cases can be seen as a direct result of the rearrangement inequality.

Case 2: General $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ If $\mathbf{u}$ and $\mathbf{v}$ have entries that can be positive, negative, or zero, we require a slightly more sophisticated method for finding the optimal matching on $\boldsymbol{Y}$. In this case, define $\tilde{\boldsymbol{Y}}$ to be the matrix obtained by copying $\boldsymbol{Y}$ and deleting all negative entries. To find the optimal matching of $\boldsymbol{Y}$ we would never pair elements giving a negative weight, so the optimal matching for $\tilde{\boldsymbol{Y}}$ is the same as for

$\boldsymbol{Y}$. Now let $\mathbf{u}_+$ and $\mathbf{u}_-$ be the vectors that contain the strictly positive and negative elements in $\mathbf{u}$ respectively, and define $\mathbf{v}_+$, and $\mathbf{v}_-$ similarly for $\mathbf{v}$. Then,

$$\tilde{\boldsymbol{Y}} = \tilde{\boldsymbol{Y}}_1 + \tilde{\boldsymbol{Y}}_2$$

where $\tilde{\boldsymbol{Y}}_1 = \mathbf{u}_+\mathbf{v}_+^T$ and $\tilde{\boldsymbol{Y}}_2 = \mathbf{u}_-\mathbf{v}_-^T$. Let $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ be the optimal matching matrices for $\tilde{\boldsymbol{Y}}_1$ and $\tilde{\boldsymbol{Y}}_2$ respectively, obtained using the sorting techniques for case 1. Since $\mathbf{u}_+, \mathbf{u}_-, \mathbf{v}_+$ and $\mathbf{v}_-$ will contain some entries that are zero, both $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ may leave certain nodes unmatched. The following lemma shows that combining these matchings yields the optimal result for $\tilde{\boldsymbol{Y}}$: The set of nodes matched by $\boldsymbol{M}_1$ will be disjoint from the set of nodes matched by $\boldsymbol{M}_2$. The matching $\tilde{\boldsymbol{M}}$ defined by combining these two matchings will be optimal for $\boldsymbol{Y}$.

Proof We will prove by contradiction that there are no conflicts between $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$. Assume that $\boldsymbol{M}_1$ contains the match (i, j) and $\boldsymbol{M}_2$ contains a conflicting match (i, k). Since $\boldsymbol{M}_1$ contains the match (i, j), $\tilde{\boldsymbol{Y}}_1(i, j)$ must be nonzero, implying that $\mathbf{u}_+(i)$ and $\mathbf{v}_+(j)$ are both positive. Similarly, $\boldsymbol{M}_2$ contains the pair (i, k), so $\mathbf{u}_-(i)$ and $\mathbf{v}_-(k)$ are both negative. This is a contradiction, since at least one of $\mathbf{u}_+(i)$ and $\mathbf{u}_-(i)$ must be zero.

We just need to show that $\tilde{M}$ is an optimal matching for $\boldsymbol{Y}$. If this were not the case, there would exist some matching $\boldsymbol{M}$ such that $\boldsymbol{M} \bullet \tilde{\boldsymbol{Y}} > \tilde{\boldsymbol{M}} \bullet \tilde{\boldsymbol{Y}}$. If such an $\boldsymbol{M}$ existed, we would have that

$$\boldsymbol{M} \bullet \tilde{\boldsymbol{Y}}_1 + \boldsymbol{M} \bullet \tilde{\boldsymbol{Y}}_2 > \tilde{\boldsymbol{M}} \bullet \tilde{\boldsymbol{Y}}_1 + \tilde{\boldsymbol{M}} \bullet \tilde{\boldsymbol{Y}}_2$$

However, $\tilde{\boldsymbol{M}} \bullet \tilde{\boldsymbol{Y}}_1 = \boldsymbol{M}_1 \bullet \tilde{\boldsymbol{Y}}_1 \geq \boldsymbol{M} \bullet \tilde{\boldsymbol{Y}}_1$, and $\tilde{\boldsymbol{M}} \bullet \tilde{\boldsymbol{Y}}_2 = \boldsymbol{M}_2 \bullet \tilde{\boldsymbol{Y}}_2 \geq \boldsymbol{M} \bullet \tilde{\boldsymbol{Y}}_2$. Thus, a contradiction; $\tilde{\boldsymbol{M}}$ is an optimal matching of $\tilde{\boldsymbol{Y}}$. ■

5.4.2 Matchings on Low Rank Factors

Now we address the problem of finding a good matching for a matrix $\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{V}^T$, where $\boldsymbol{Y} \in \mathbb{R}^{m\times n}$, $\boldsymbol{U} \in \mathbb{R}^{m\times k}$, and $\boldsymbol{V} \in \mathbb{R}^{n\times k}$. Let $\mathbf{u}_i$ and $\mathbf{v}_i$ be the ith columns in $\boldsymbol{U}$ and $\boldsymbol{V}$, and let $\boldsymbol{Y}_i = \mathbf{u}_i\mathbf{v}_i^T$, then $\boldsymbol{Y} = \sum_{i=1}^k \boldsymbol{Y}_i$.

We can find the optimal matching on each $\boldsymbol{Y}_i$ using the results from Section 5.4.1. Let $\boldsymbol{M}_i$ be the matching matrix corresponding to $\boldsymbol{Y}_i$, and let $\boldsymbol{M}^*$ be a matching matrix that achieves an optimal maximum weight on $\boldsymbol{Y}$. Note that $\boldsymbol{M}^* \bullet \boldsymbol{Y}_i \leq \boldsymbol{M}_i \bullet \boldsymbol{Y}_i$, and thus,

$$\boldsymbol{M}^* \bullet \boldsymbol{Y} \leq \textstyle\sum_{i=1}^k \boldsymbol{M}_i \bullet \boldsymbol{Y}_i. \tag{5.7}$$

To analyze how good of a matching each $\boldsymbol{M}_i$ is on the entire matrix $\boldsymbol{Y}$, define the following terms:

$$d_{i,j} = \tfrac{\boldsymbol{M}_i \bullet \boldsymbol{Y}_i}{\boldsymbol{M}_j \bullet \boldsymbol{Y}_i} \quad d_j = \max_i d_{i,j} \quad D = \min_j d_j \tag{5.8}$$

and let $j^* = \text{argmin}_j\, d_j$, i.e. $D = d_{j^*}$. Note that for any fixed indices i, j, we have $d_{i,j} \leq d_j$. Applying this to $j = j^*$ we have that for all i,

$$\tfrac{\boldsymbol{M}_i \bullet \boldsymbol{Y}_i}{\boldsymbol{M}_{j^*} \bullet \boldsymbol{Y}_i} = d_{i,j^*} \leq d_{j^*} = D \implies \boldsymbol{M}_i \bullet \boldsymbol{Y}_i \leq D(\boldsymbol{M}_{j^*} \bullet \boldsymbol{Y}_i) \tag{5.9}$$

By combining (5.7) and (5.9) we have the following result.

Theorem 5.4.1 *We can achieve a D-approximation for the bipartite matching problem by selecting an optimal matching for one of the low-rank factors of $\boldsymbol{Y}$.*

Proof $\boldsymbol{M}^* \bullet \boldsymbol{Y} \leq \sum_{i=1}^k \boldsymbol{M}_i \bullet \boldsymbol{Y}_i \leq \sum_{i=1}^k D\boldsymbol{M}_{j^*} \bullet \boldsymbol{Y}_i = D(\boldsymbol{M}_{j^*} \bullet \boldsymbol{Y})$. ■

This procedure (Algorithm 3) runs in $\mathcal{O}(k^2 n + kn\log n)$ where k is the rank, and $\boldsymbol{U}$ and $\boldsymbol{V}$ have $O(n)$ rows. The space requirement is $\mathcal{O}(nk)$. In practice, the approximation factors D are less than 1.1 for our problems (see Figure 5.5). Algorithm 3 shows pseudocode to implement this matching algorithm.

Algorithm 3 Pseudocode for finding a D-approximate matching from a low rank matrix.

```
1: Input: $\boldsymbol{U}$,$\boldsymbol{V}$ such that $\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{V}^T$
2: Output: Approximate max-weight matching $\boldsymbol{M}$, approx value $D$
3: Find optimal matching $M_i$ for each rank-1 matrix (see §5.4.1)
4: Evaluate the weight of the matching $v_i = M_i \bullet \boldsymbol{Y}_i$
5: Compute $d_{i,j} = v_i/(\boldsymbol{M}_j \bullet \boldsymbol{Y}_i)$ for all i,j = $\{1, \ldots, k\}$
6: Evaluate $d_j = \max_i d_{i,j}$, $D = \text{minimum}(d_j)$, $j^* = \text{argmin}(d_j)$
7: return $\boldsymbol{M} = \boldsymbol{M}_{j^*}, D$
```

5.4.3 Improved practical variations

Our method (Algorithm 3) can be improved without substantially changing its runtime or memory requirement. The key idea is to create a sparse max-weight bipartite matching problem that include the matching $\boldsymbol{M}_{j^*}$ and other helpful edges. By optimally solving these, we will only improve the approximation. These incur the cost of solving those problems optimally, but sparse max-weight matching solvers are practical and fast for problems with millions of edges.

Union of matchings. The simplest improvement is to create a sparse graph based on the full set of matches $\boldsymbol{M}_1, \ldots, \boldsymbol{M}_k$. We can do this by transforming the complete bipartite network defined by $\boldsymbol{Y}$ into a sparsified network $\hat{\boldsymbol{Y}}$ where edge (j, k) is nonzero with weight $\boldsymbol{Y}_{j,k}$ only if nodes (j, k) were matched by some $\boldsymbol{M}_i$. Then, we solve a maximum bipartite matching problem on the sparse matrix $\hat{\boldsymbol{Y}}$ with $O(nk)$ non-zeros or edges. This only improves the approximation because we included the matching $\boldsymbol{M}_{j^*}$.

Expanding non-matchings on rank-1 factors. Since algorithm 3 relies on a sorting procedure when building $\boldsymbol{M}_i$ from the rank-1 factors, and since these numbers may very likely be close to each other, we can choose to expand the set of possible matchings and let each node pair up with c closest values to it. By way of example, if $c = 3$ and we had sorted indices

$$\begin{array}{ll} \text{sorted } \mathbf{u}\text{: } i_1 \; i_2 \; i_3 \; i_4 \; i_5 & \text{then we} \\ \text{sorted } \mathbf{v}\text{: } j_1 \; j_2 \; j_3 \; j_4 \; j_5 & \text{add edges} \end{array} \quad \begin{array}{lll} (i_1, j_1) & (i_1, j_2) & \\ (i_2, j_1) & (i_2, j_2) & (i_2, j_3) \\ (i_3, j_2) & \ldots & \end{array}$$

We add all these edges to the sparse matrix $\hat{\boldsymbol{Y}}$ with their true values from $\boldsymbol{Y}$ and solve a maximum bipartite matching problem on the resulting matrix. Again, this includes all edges from $\boldsymbol{M}_{j^*}$. After adding all the edges from set $\mathbf{u}_i, \mathbf{v}_i$, the final number of edges is $O(kcn)$, and thus, the resulting union of matchings matrix is a sparse matrix when kc is $o(n)$.

5.5 Experiments

To evaluate our method, we first study the relationship between the EigenAlign algorithm and our Low Rank EigenAlign algorithm. The goal of these initial experiments is to show (i) that we need about 8 iterations, which gives a rank 9 matrix, to get equivalent results to EigenAlign (Figure 5.2), (ii) our method performs the same over a variety of graph models (Figure 5.4), (iii) the method scales better (Figure 5.3), and (iv) the computed approximation bounds are better than 1.1 (Figure 5.5). We also compare against other scalable techniques in Figure 5.6, and see that our approach is the best. Next, we use a test-set of networks with known alignments from Biology (Vijayan and Milenković, 2017) to evaluate our algorithms (Section 5.5.2). Finally, we end our experiments with a study on a collaboration network where we seek to align vertex neighborhoods (Section 5.5.3).

Our low-rank EigenAlign In all of these experiments, our low-rank techniques use the expanded matching with $c = 3$ (Section 5.4.3) and uniform rank-1 factors: $\mathbf{v} = \mathbf{u} = \mathbf{e}$. Let $\alpha = 1 + \frac{\text{nnz}(A)\text{nnz}(B)}{\text{nnz}(A)(n_B^2 - \text{nnz}(B)) + \text{nnz}(B)(n_A^2 - \text{nnz}(A))}$. This equals one plus the ratio of possible overlaps divided by possible conflicts. Let $\gamma = 0.001$, then

$s_O = \alpha + \gamma, s_N = 1 + \gamma, s_C = \gamma$. These parameters correspond to those used in (Feizi et al., 2016) as well. Finally, we set the number of iterations to be 8 for all experiments except those where we explicitly vary the number of iterations.

Theoretical runtime. When we combine our low-rank computation and the subsequent expanded low-rank matching, the runtime of our method is

$$O(\underbrace{nk^2}_{\substack{\text{low-rank factors} \\ \\ \text{compute } d_{ij}}} + \underbrace{k^3}_{\text{SVD}} + \underbrace{kn \log n}_{\text{sorting}} + \text{matching with } ckn \text{ edges})$$

and $O(nck)$ memory. (Note that $k = 8$ and $c = 3$ in our experiments.)

EigenAlign baseline. For EigenAlign, we use the same set of parameters s_O, s_N, s_C and use the power method with starting with the all ones vector. We run the power method with normalization as described in (5.4) until we reach an eigenvalue-eigenvector pair that achieves a residual value 10^{-12}. This usually occurs after a 15-20 iterations.

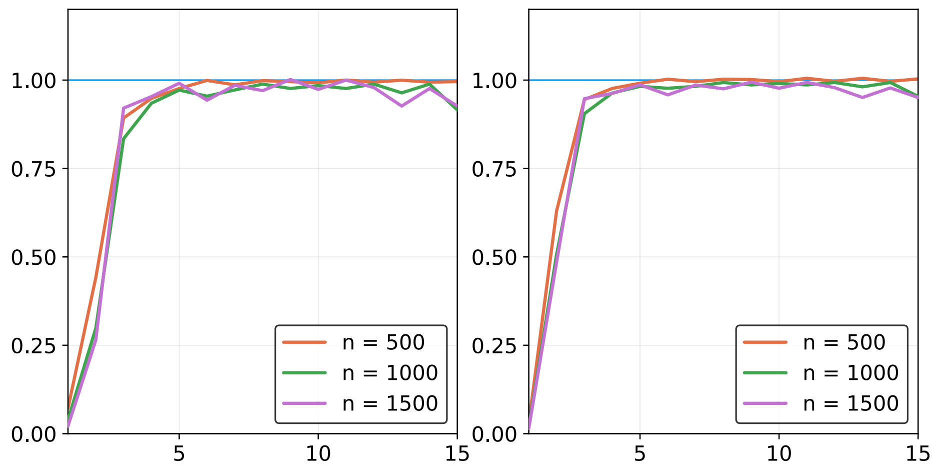
5.5.1 Erdős-Rényi and preferential attachment

The goal of our first experiment is to assess the performance of our method compared to EigenAlign. These experiments are all done with respect to synthetic problems with a known alignment between the graphs. The metric we use to assess the performance is *recovery* (Feizi et al., 2016), where we want *large recovery values.* Recovery is between 0 and 1 and is defined

$$\text{recovery}(\boldsymbol{M}) = 1 - \tfrac{1}{2n} \|\boldsymbol{M} - \boldsymbol{M}_{\text{true}}\|_F. \tag{5.10}$$

In words, recovery is the fraction of correct alignments.

Graph models. To generate the starting undirected network in the problem ($\boldsymbol{G}_A$), we use either Erdős-Rényi with average degree ρ (where the edge probability



Overlapped edges relative to EigenAlign. *Recovery results relative to EigenAlign.*

Figure 5.2. At left, the number of overlapped edges in the alignment computed by our low-rank method relative to EigenAlign's alignment. A value of 1.0 means that we get the same number as EigenAlign's solution. At right, the same ratio but with respect to the recovery. The recovery results stop improving after around 8 iterations, so we fix this value in the rest of our experiments.

is ρ/n) or preferential attachment with a random 6-node initial graph and adding θ edges with each vertex.

Noise Model. Given a network $\boldsymbol{G}_A$, we add some noise to generate our second network $\boldsymbol{G}_B$ (Feizi et al., 2016). With probability p_{e_1}, we remove an edge, and with probability p_{e_2} we add an edge. Then, algebraically, $\boldsymbol{B}$ can be written as $\boldsymbol{A} \circ (1 - \boldsymbol{Q}_1) + (1 - \boldsymbol{A}) \circ \boldsymbol{Q}_2$, where $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ are undirected Erdős-Rényi graphs with density p_{e_1} and p_{e_2} respectively and $\circ$ is the Hadamard (element-wise) product. We fix $p_{e_2} = pp_{e_1}/(1-p)$ where p is the density of $\boldsymbol{G}_A$. Because some algorithms have a bias in the presence of multiple possible solutions, after $\boldsymbol{B}$ is generated, we relabel the nodes in $\boldsymbol{B}$ in reverse order.

Eight iterations are enough. We first study the change in results with the number of iterations. We use Erdős-Rényi graphs with average degree 20 and analyze

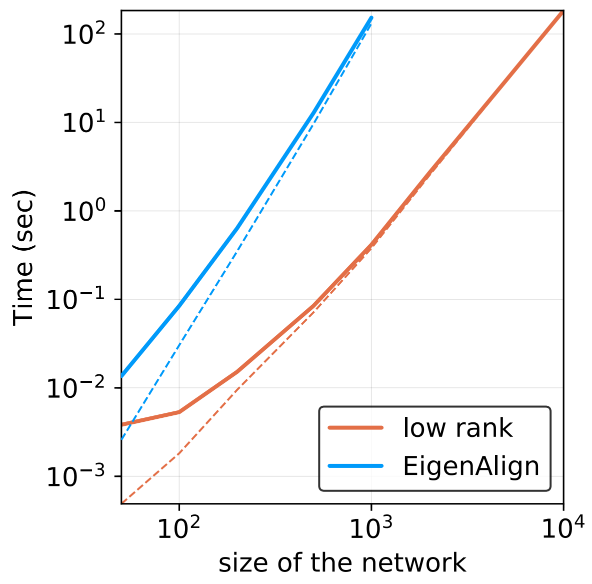
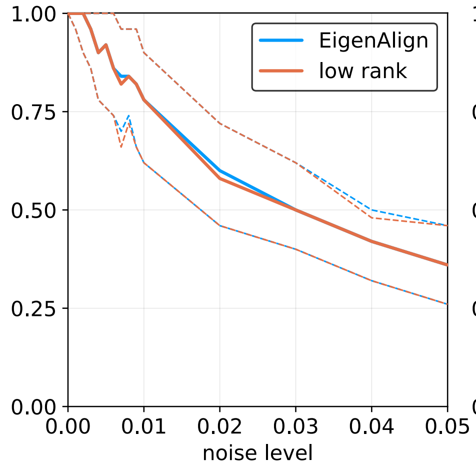


Figure 5.3. These show the average time over 10 trials. The dashed line is the time required for the matching step. The time required for EigenAlign is an order of magnitude larger than our low-rank formulation. Our low rank EigenAlign solves 10,000 node problems in about two minutes whereas EigenAlign requires the same amount of time to solve a 1000 node problem.

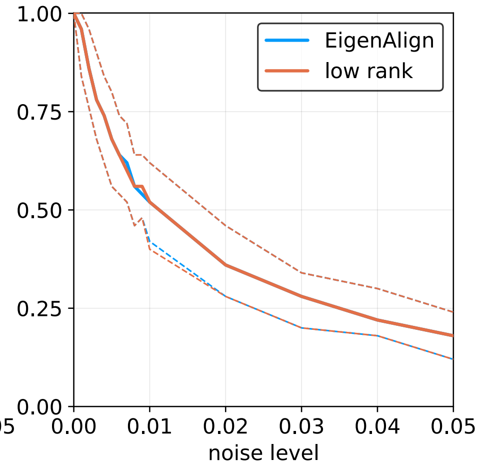
the performance of our method as iterations vary. Figure 5.2 shows the recovery (left) and overlap (right) relative to the EigenAlign result so a value of 1.0 means the same number as EigenAlign. After 8 iterations, the recovery stops increasing, and so we perform the rest of our experiments with only 8 iterations.

Our Low-rank EigenAlign matches EigenAlign for Erdős-Rényi and preferential attachment. We next test a variety of graphs as the noise level p_{e_1} varies. For these experiments, we create Erdős-Rényi graphs with average degree 5 and 20 and preferential attachment graphs with $\theta = 4$ and $\theta = 6$ for graphs with 50 nodes. Figure 5.4 shows these results in terms of the recovery of the true alignment. In the figure, the experimental results over 200 trials are essentially indistinguishable.

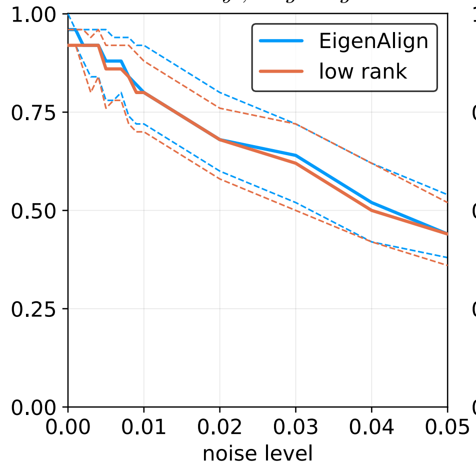
Our low-rank method is far more scalable. We next consider what happens to the runtime of the two algorithms as the graphs get larger. Figure 5.3 shows these results where we let each method run up to two minutes. We look at preferential attachment graphs with $\theta = 4$ and $p_{e_1} = 0.5/n$. EigenAlign requires a little more than two minutes to solve a problem of size 1000, whereas our low rank formulation can solve a problem that is an order of magnitude bigger in the same amount of time.



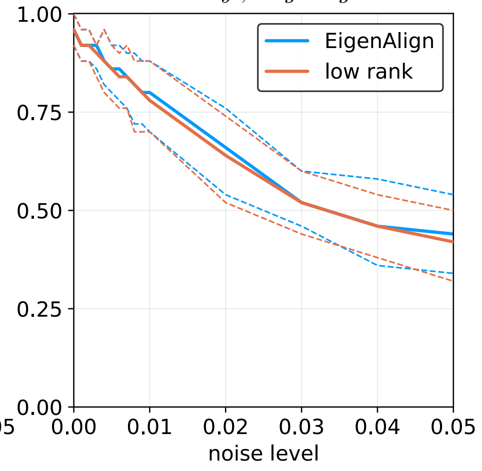
Erdős-Rényi, avg. deg. 5



Erdős-Rényi, avg. deg. 20



Preferential attachment $\theta = 4$



Preferential attachment $\theta = 6$

Figure 5.4. Thick lines are the median recovery fractions over 200 trials and dashed lines are the 20th and 80th percentiles. These figures show that there appears to be small and likely insignificant differences between the alignment quality of EigenAlign and our low rank method.

Our matching approximations are high quality. We also evaluate the effectiveness of our D-approximation computed in Section 5.4.2. Here, we compare the computed bound D we get to the actual approximation value of our algorithm and to the actual approximation ration of a greedy matching algorithm. The greedy algorithm *can* be implemented in a memory scalable fashion with a $O(n^3)$ runtime (or $O(n^2 \log n)$ with quadratic memory) and guarantees a 2-approximation whereas our D

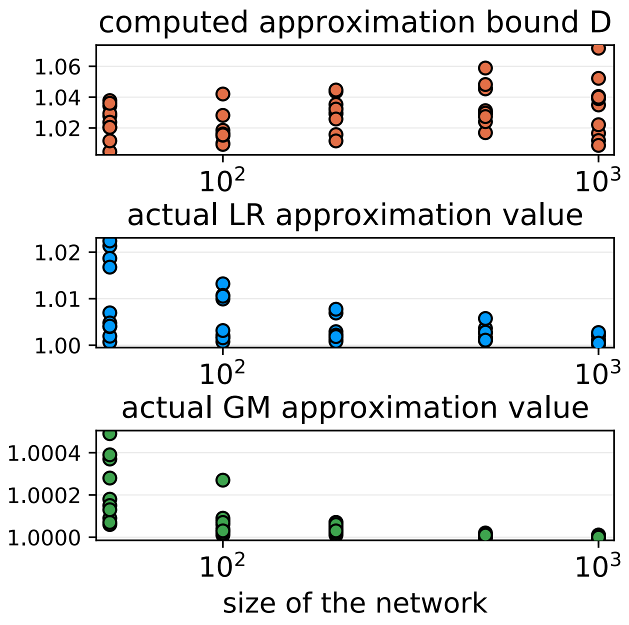


Figure 5.5. Over the experiments from Figure 5.3, the top figure shows the guaranteed D-approximation value computed by our algorithm. The bound appears to be strong and gives a better-than 1.1 approximation. The middle figure shows the true approximation value after solving the optimal matching using Low-rank EigenAlign (LR), and the bottom figure shows the true approximation value for a greedy matching (GM) strategy, which guarantees a 2-approximation.

value gives better theoretical bounds. Figure 5.5 shows these results. Our guaranteed approximation factors are always less than 1.1 when the low-rank factors arise from the problems in Figure 5.3. Surprisingly, greedy matching does exceptionally well in terms of approximation, prompting our next experiment.

Our matching greatly outperforms greedy matching and other low-rank techniques. NSD (Kollias et al., 2012) is another network alignment algorithm which solves the network alignment problem via low-rank factors. In the previous experiment, we saw that greedy matching consistently gave better than expected approximation ratios. Here, we compare the low-rank EigenAlign formulations with our low-rank matching scheme to greedy matching in terms of *recovery*. The results are shown in Figure 5.6 and show that the low-rank EigenAlign strategy with our low-rank matching outperforms the other scalable alternatives.

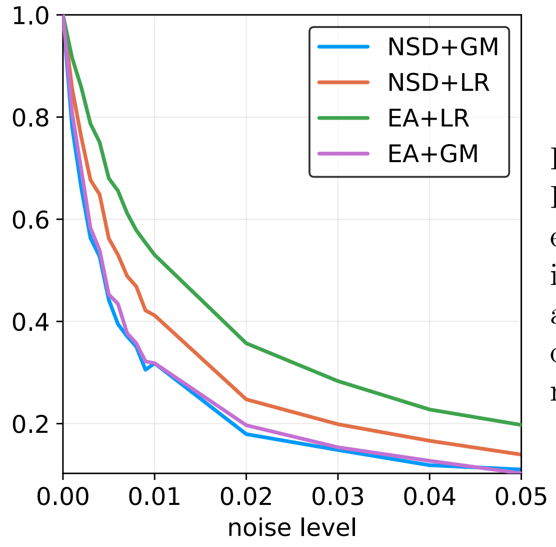


Figure 5.6. Recovery scores achieved by Low Rank EigenAlign and NSD (Kollias et al., 2012). We use a greedy matching (GM) and our low rank matching algorithm (LR) on the low rank factors of the similarity matrix from both algorithms.

5.5.2 Biological networks

The MultiMagna dataset is a test case in bioinformatics that involves network alignment (Vijayan and Milenković, 2017; Meng et al., 2016). It consists of a base yeast network that has been modified in different ways to produce five related networks, which we can think of as different edge sets on the same set of 1004 nodes. This results in 15 pairs of networks to align (6 choose 2). One unique aspect of this data is that there is no side information provided to guide the alignment process, which is exactly where our methods are most useful. In Figure 5.7, we show results for aligning MultiMagna networks using low-rank EigenAlign, EigenAlign, belief propagation (BP) (Bayati et al., 2013), and Klau's method (Klau, 2009) in terms of two biologically relevant measures:

F-Node Correctness (F-NC). This is the F-score (harmonic mean) of the precision and recall of the alignment.

NCV-Generalized S3. This shows how well the network structure correlates. Let $\boldsymbol{M}$ be a matching matrix for graphs with n_A and n_B nodes. The node coverage value of an alignment is NCV $= 2\text{nnz}(\boldsymbol{M})/(n_A + n_B)$, where $\text{nnz}(\boldsymbol{M})$ counts the number of nonzero entries in $\boldsymbol{M}$. Let E_O be the set of overlapping edges for an alignment M and E_C be the set of conflicts, and define GS3 $= |E_O|/(|E_O| + |E_C|)$. The NCV-GS3 score is the geometric mean of NCV and GS3.

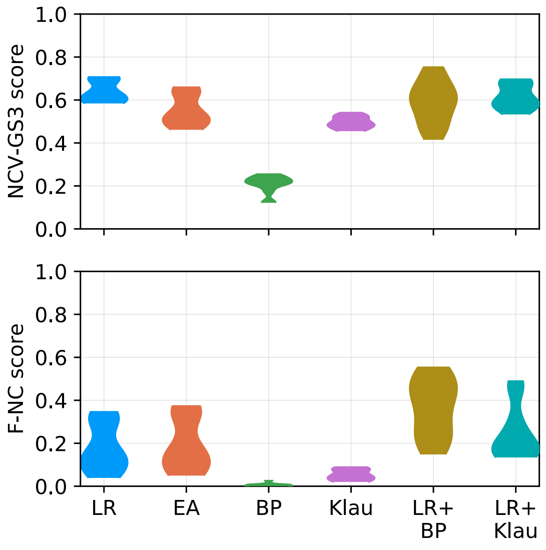


Figure 5.7. These are violin plots over the results of 15 problems. Klau and BP are strong algorithms for network alignment but they only perform well when given a sparsified set of possible matches from our expanded low-rank matchings (LR+BP, LR+Klau). Larger scores are better.

In this experiment, we find that standard network alignment algorithms (BP and Klau) perform dreadfully (F-NC) without any guidance about which nodes might be good matches. Towards that end, we can take the output from our expanded matchings from the low-rank factors and run the Klau and BP methods *on this restricted set of matchings*. This enables them to run in a reasonable amount of time with improved results. The idea here is that we are treating Klau and BP as the matching algorithm rather than using bipartite matching for this step. This picks a matching that also yields a good alignment. Our results are comparable with the results in Meng et al. (2016), which is a recent paper that uses a number of other algorithms on the same data. The timing results from these experiments are shown in Table 5.1.

5.5.3 Collaboration network

We now use Low Rank EigenAlign to perform a study on a collaboration network to understand what would be possible in terms of a fully anonymized network problem. We show that we could use our network alignment technique to identify edges

Algorithm	Time (sec)		
	min	median	max
LR	1.9553	2.1971	2.9173
EA	83.677	96.9938	194.36
BP	1985.2	2216.3	2744.3
Klau	3031.4	3856.0	4590.2
LR+BP	174.06	182.58	190.44
LR+Klau	257.59	301.86	318.83

Table 5.1. Time required for methods on the MultiMagna data.

where the endpoints have a high Jaccard similarity. We do so by aligning the node neighborhoods of each of the end points of each edge and observe that a high overlap implies a high Jaccard similarity score.

In more detail, recall that the Jaccard similarity of two nodes (a and b) is defined as $\frac{|N(a) \cap N(b)|}{|N(a) \cup N(b)|}$, where $N(a)$ are the neighboring nodes of a. The vertex neighborhood of node a is the induced subgraph of the node and all of its neighbors. Given an edge (i, j), we then compute the Jaccard similarity between i and j, and also align the vertex neighborhood of i to the vertex neighborhood of j using our technique.

We use the DBLP collaboration network from Esfandiar et al. (2010) and consider pairs of nodes that have a sufficiently big neighborhood and are connected by an edge. Specifically, we consider nodes that have 100 or more neighbors. In total, we end up with 15187 such pairs. This is an easy experiment with our fast codes and it takes less than five minutes. The results are in Figure 6.7. We score the network alignments in terms of normalized overlap, which is the number of overlapped edges to the maximum possible number for a pair of neighborhoods. What we observe is that large Jaccard similarities and large overlap scores are equivalent. This means we *could* have identified these results without any information on the actual identity of the vertices.

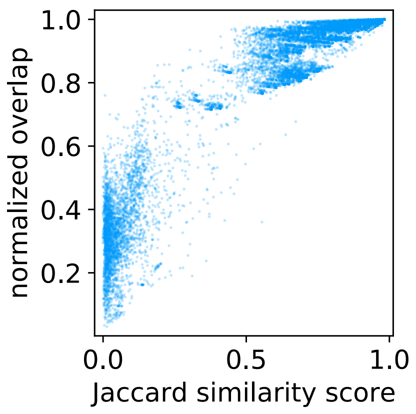


Figure 5.8. The edge overlap (normalized so the maximum value is 1.0) of alignments between vertex neighborhoods of nodes in the DBLP dataset. This shows that the Jaccard score of two connected nodes in the network is correlated to the overlap size.

5.6 Conclusion & Discussion

The low-rank spectral network alignment framework we introduce here offers a number of exciting possibilities in new uses of network alignment methods. First, it enables a new level of high-quality results with a scalable, principled method as illustrated by our experiments. This is because it has near-linear runtime and memory requirements in the size of the input networks. Second, in the course of this application, we developed a novel matching routine with high-quality a-posteriori approximation guarantees that will likely be useful in other areas as well.

That said, there are a number of areas that merit further exploration. First, the resulting low-rank factorization uses the matrix $\boldsymbol{S}_k$, which is related to graph diffusions. There are results in computational geometry that prove rigorous results about using diffusions to align manifolds (Ovsjanikov et al., 2010). There are likely to be useful connections to further explore here. Second, there are strong relationships between our low-rank methods and fast algorithms for Sylvester and multi-term matrix equations (Simoncini, 2016) of the form $\boldsymbol{C}_1\boldsymbol{X}\boldsymbol{D}_1 + \boldsymbol{C}_2\boldsymbol{X}\boldsymbol{D}_2 + \cdots = \boldsymbol{F}$. These connections offer new possibilities to improve our methods.

6 MULTIPLE NETWORK ALIGNMENT

So far in this thesis, we have discussed the network alignment problem in the context of aligning two networks and in this chapter we focus on the network alignment problem in the presence of three or more networks. This chapter shows our results from our paper "Low rank methods for multiple network alignment" (Nassar et al., 2018).

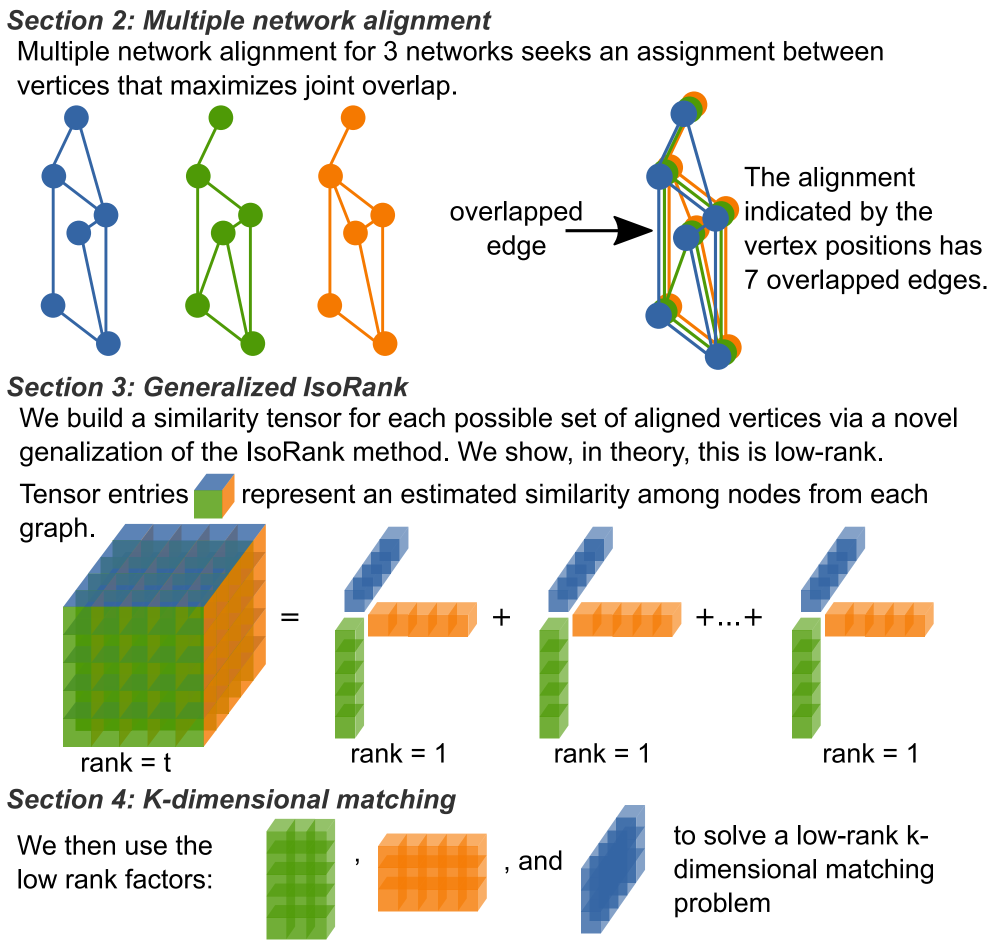
Multiple network alignment arises when we want to identify similar or related regions in more than two networks. While there are a large number of effective techniques for pairwise problems with two networks that scale in terms of edges (Bayati et al., 2013; Feizi et al., 2016; Klau, 2009), these cannot be readily extended to align multiple networks as the computational complexity will tend to grow exponentially with the number of networks. In this chapter we introduce a new multiple network alignment algorithm and framework that is effective at aligning thousands of networks with thousands of nodes. The key enabling technique of our algorithm is identifying an exact and easy to compute low-rank tensor structure inside of a generalized version of IsoRank. This can be combined with a new algorithm for k-dimensional matching problems on low-rank tensors to produce the alignment. At the end of this chapter, we demonstrate results on synthetic and real-world problems that show our technique (i) is as good or better in terms of quality as existing methods, when they work on small problems, while running considerably faster and (ii) is able to scale to aligning a number of networks unreachable by current methods. This chapter demonstrates that our method is the realistic choice for aligning multiple networks when no prior information is present.

6.1 Introduction

A more general problem to the pairwise network network alignment (PNA) is that of multiple global network alignment (MNA)) (Gligorijevic et al., 2016; Liao et al., 2009; Malmi et al., 2017), where we are interested in finding a common subgraph present in more than two input networks (illustrated in the top panel of Figure 6.1). Applications of this problem arise in comparative proteomics (where the networks are protein interactions from multiple species), entity resolution (where the networks reflect different records), subject registration (where the networks reflect multiple measured views), and other applied machine learning tasks.

Both PNA and MNA are related to the subgraph isomorphism problem which is known to be NP-hard, with MNA being a harder problem in practice due to the combinatorial explosion of possible aligned pairs. As an illustration of this point, consider a common strategy in PNA algorithms (Klau, 2009; Kollias et al., 2012; Feizi et al., 2016; Nassar et al., 2018; Bayati et al., 2013; Patro and Kingsford, 2012): (i) score each potential matched pair of nodes between the graphs based on a topological similarity measure; and (ii) perform a maximum weight bipartite matching (or a closely related algorithm) on the set of scores. Simple extensions of these principled procedures to MNA with k networks cannot easily scale to more than a handful of networks because the set of data in step (i) becomes $O(n^k)$ when each network has $O(n)$ nodes, and the obvious generalization of max weight bipartite matching (step (ii)) is k-dimensional matching, which is NP-complete for $k \geq 3$ (Karp, 1972). As an alternative, there are approximation algorithms for k-dimensional matching (Kann, 1991).

Despite the computational difficulty, there are a few algorithms that navigate the computational and memory requirements. A straightforward solution is to consider sequences of pairwise network alignment problems, or to use pairwise network alignment data to infer multi-network alignments. Another straightforward solution is to restrict the set of possible alignments to those inferred through prior information or



Section 2: Multiple network alignment
Multiple network alignment for 3 networks seeks an assignment between vertices that maximizes joint overlap.
overlapped edge
The alignment indicated by the vertex positions has 7 overlapped edges.
Section 3: Generalized IsoRank
We build a similarity tensor for each possible set of aligned vertices via a novel genalization of the IsoRank method. We show, in theory, this is low-rank.
Tensor entries represent an estimated similarity among nodes from each graph.
=
+
+...+
rank = t
rank = 1
rank = 1
rank = 1
Section 4: K-dimensional matching
We then use the low rank factors:
, and
to solve a low-rank k-dimensional matching problem

Figure 6.1. A visulaization of the pipeline in this chapter. The multiple network alignment problem is illustrated in the top panel and discussed in Section 6.2. Our solution is to form a low rank representation of a high dimensional tensor (middle panel, Section 6.3), and then perform $k - dimensional$ matching on the low rank factors (bottom panel, Section 6.4).

.

metadata about the nodes. Such information often speeds up the computation drastically and guides the algorithm to a meaningful solution (Malmi et al., 2017). In this chapter as well, we focus on the case when such information is not present and there is no reduction to pairwise alignments. To the best of our knowledge, the highest number of networks an existing MNA method aligns is 10 networks (Malmi et al., 2017), and ours is the first multiple network alignment algorithm that can scale to thousands of networks with thousands of nodes in a reasonable runtime (about 3 hours for 1000 networks with 1000 nodes, Section 6.5.7). In this regime existing techniques take too long, run out of memory, or give bad results. We note that a number of important applications, e.g. aligning protein/gene interaction networks from specific tissues or pathologies lie in this regime.

The two main technical innovations we present in this chapter are (i) a specific multi-network generalization of the pairwise network alignment algorithm IsoRank (Singh et al., 2008) that enables us to compute a representation of the $O(n^k)$, k-way alignment data efficiently, and (ii) an extremely efficient k-dimensional matching algorithm which generalizes our results from the previous chapter to low-rank tensors. In summary, this chapter addresses the following components.

- Section 6.3 generalizes the IsoRank algorithm to multiple networks and shows that the solution can be represented by a multidimensional tensor that can be explicitly written in terms of low-rank nonnegative factors that are easy to compute.

- Section 6.4 presents a new k-dimensional matching algorithm for low-rank tensors with an a-posteriori approximation bound.

- Section 6.5.3 shows experimentally that multiple network alignment is faster and higher-quality compared to performing multiple pairwise alignments when the number of networks grows.

- Section 6.5.5 extends the case study from the previous chapter on anonymized data from the DBLP collaboration network, where we show that aligning anonymized triplets of egonets can identify those triples with high Jaccard similarity, which can only be accurately computed from the de-anonymized data.

6.2 Related work

Existing MNA algorithms can be viewed in two classes. Biologically motivated algorithms are often designed to align protein-protein interaction networks, whereas topological algorithms are more generic and try to exploit the network structure.

6.2.1 Biological algorithms

In biology, there is a need to discover new relationships between proteins, and MNA can be used as a tool to study these connections (Singh et al., 2008). The networks to be aligned are often protein protein interaction networks (PPIs) of different species. In these cases, there are several measures to compare the proteins independently of network interaction structure, such as by evaluating their sequence similarity of their genetic codings. Biological algorithms are designed with this piece of information in mind, such as PrimAlign (Kalecky and Cho, 2018). Another algorithm, MultiMagna++ (Vijayan and Milenković, 2017), uses a genetic algorithm that works directly with the multi-way alignment permutations and uses objective or fitness functions that utilize the biological information. IsoRank (Singh et al., 2008) and IsoRankN (Liao et al., 2009), which are two of the earliest MNA algorithms, computed pairwise topological similarity scores between each pair of networks and then assembled the result into a multiple alignment in a variety of ways. They can be related back to a complete k-partite network representation of all the pairwise alignment information. Another recent algorithm, FUSE (Gligorijevic et al., 2016) uses protein similarity to build the k-partite representation of the problem and then uses non-negative matrix trifactorization to incorporate network structure into the overall alignment.

6.2.2 Topological algorithms

There are two state of the art algorithms introduced in (Malmi et al., 2017): *FLAN* and *ProgNatalie++*. The FLAN method is based on generalizing the concept of the facility location problem and is a good way to utilize prior information about possible relationships (such as in entity resolution in their case). We compare against ProgNatalie++ in this chapter, which extends the PNA algorithm *Natalie* that was proposed by Klau et al. (Klau, 2009). ProgNatalie++ proceeds by solving the multiple network alignment problem progressively, by aligning the first two networks, and then folding in the third network using the existing match, etc. This involves solving $k-1$ PNA problems.

6.2.3 The need for new methods

To run these algorithms on networks where no prior similarity measures are available, one can assume that all pairs of nodes are similar and assign them the same score. Such an approach empirically fails in producing meaningful results for the algorithms FUSE, IsoRankN, and MultiMagna++ (Gligorijevic et al., 2016; Liao et al., 2009; Vijayan and Milenković, 2017). To motivate the need for new algorithms that can run without prior known similarity, we demonstrate a simple MNA problem with one of the most recent MNA methods, MultiMagna++ (Vijayan and Milenković, 2017). In the problem setup, we use three graphs of size $n = 500$. We generate the three graphs as instances from an initial preferential attachement graph of size 500 and average degree 8 by randomly removing edges with a probability $0.5/n$ (more detail on synthetic graph generation can be found in Section 6.5.3). MultiMagna++ does not produce any meaningful results, and the relative number of overlapped edges (illustrated in the top panel of Figure 6.1) to the ground truth alignment's overlapped edges is consistently less than 1%. In contrast, our method, consistently produces a relative overlap higher than 80%. ProgNatalie++ and FLAN are more resistant to the absence of this information, but the running time of these algorithms is extreme, as they are using a Lagrangian relaxation approach to solve an NP-hard problem at each step.

6.3 Multiple Network Alignment and An Exact Low Rank Method

The multiple network alignment problem can be formulated for three undirected networks as:

$$\begin{array}{ll} \text{maximize} & \sum_{i,j,k} \sum_{r,s,t} A_{ir} B_{js} C_{kt} \underline{X}_{ijk} \underline{X}_{rst} \\ \text{subject to} & \sum_{j,k} \underline{X}_{ijk} \leq 1 \text{ for all } i; \sum_{i,k} \underline{X}_{ijk} \leq 1 \text{ for all } j; \sum_{i,j} \underline{X}_{ijk} \leq 1 \text{ for all } k \\ & \underline{X}_{i,j,k} \in \{0,1\} \text{ for all } i,j,k. \end{array} \tag{6.1}$$

Here $\underline{X}_{ijk} = 1$ indicates that node i in network A matches to node j in network B and node k of network C, $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ are adjacency matrices for the three networks, and the number of vertices of these networks give the summation limits in the above expression. The objective function can be read as nodes i, j, k are matched and we have edges (i, r) in A, (j, s) in B, and (k, t) in C which are all simultaneously preserved if we also match r, s, t. That is, the product of all of these expressions is 1 when all the edges exist and they match, and 0 otherwise. The extension to k networks will be straightforward once we introduce some notation.

If we write $\mathbf{x} = \text{vec}(\underline{\boldsymbol{X}})$, i.e $\mathbf{x}$ is the large vector representation of the tensor data $\underline{\boldsymbol{X}}$, then the objective function is: $\mathbf{x}^T(\boldsymbol{C} \otimes \boldsymbol{B} \otimes \boldsymbol{A})\mathbf{x}$. (This is an instance of the mixed-product property for Kronecker products and tensors, see, e.g. equation 12.4.19 in Golub and van Loan (2013).) The constraints can be written in terms of the tensor *flattening or unfolding* operator $\flat_j$ that turns $\underline{\boldsymbol{X}}$ into a matrix by unfolding along dimension j (see Golub and van Loan (2013, Section 12.4.5) and Draisma and Kuttler (2014)). Then, we have the three and k-network problems:

$$\begin{aligned}
&\text{maximize} && \mathbf{x}^T(\boldsymbol{C} \otimes \boldsymbol{B} \otimes \boldsymbol{A})\mathbf{x} \\
&\text{subject to} && \flat_1(\underline{\boldsymbol{X}})\mathbf{e} \le \mathbf{e}; \flat_2(\underline{\boldsymbol{X}})\mathbf{e} \le \mathbf{e}; \flat_3(\underline{\boldsymbol{X}})\mathbf{e} \le \mathbf{e} \\
& && \underline{X}_{i,j,k} \in \{0,1\} \text{ for all } i,j,k. \\
&\text{maximize} && \mathbf{x}^T(\boldsymbol{A}_k \otimes \cdots \otimes \boldsymbol{A}_1)\mathbf{x} \\
&\text{subject to} && \flat_1(\underline{\boldsymbol{X}})\mathbf{e} \le \mathbf{e}; \ldots; \flat_k(\underline{\boldsymbol{X}})\mathbf{e} \le \mathbf{e} \\
& && \underline{X}_{i,j,\ldots,k} \in \{0,1\} \text{ for all indices.}
\end{aligned} \tag{6.2}$$

Here $\mathbf{e}$ is the vector of all ones of appropriate dimension. Throughout, we frequently interchange between tensor representations of data $\underline{\boldsymbol{X}}$ and their vectorized representations $\mathbf{x} = \text{vec}(\underline{\boldsymbol{X}})$.

Note that, if we were to relax to real-values and heuristically change the constraints to $\|\mathbf{x}\|_2 = 1$, then the solution is the eigenvector of $\boldsymbol{C} \otimes \boldsymbol{B} \otimes \boldsymbol{A}$ with largest eigenvalue. This eigenvector could then be reshaped and input to a 3d matching routine to produce a multiple network alignment. In practice, this technique needs a number of improvements even for the pairwise case (Feizi et al., 2016), and these are non-trivial to adapt to the multiple network case, which is discussed further in the conclusion. Instead, we adapt the IsoRank methodology, and specifically, the network similarity decomposition (NSD) method (Kollias et al., 2012) to compute IsoRank, which will easily scale to multiple networks; we briefly review NSD below and generalize it to multiple networks.

6.3.1 Network Similarity Decomposition (NSD)

The NSD method specializes IsoRank in the case when the prior similarity matrix is a low-rank matrix (Kollias et al., 2012), such as when we are using the uniform personalization term, i.e., $\boldsymbol{L} = \frac{1}{n_A n_B}\text{ones}(n_A, n_B)$ (where $\boldsymbol{A}$ has n_A vertices and $\boldsymbol{B}$ has n_B vertices and $\mathbf{e}$ the vector of all ones of appropriate size). Thus, the relevant case for us is when $\boldsymbol{L}$ is rank-1. Then NSD computes an exact low-rank representation of $\boldsymbol{X}$. Let $\boldsymbol{D}_A$ and $\boldsymbol{D}_B$ be the diagonal degree matrices for graphs $\boldsymbol{A}$ and $\boldsymbol{B}$ and suppose we initialize a fixed-point iteration for the PageRank linear system with $\boldsymbol{X}^{(0)} = \boldsymbol{L} = \mathbf{u}\mathbf{v}^T$ (because it is rank-1), and then t^{th} iterate is given by:

$$\begin{aligned}\boldsymbol{X}^{(t)} = (1-\alpha)\sum_{i=0}^{t-1} \alpha^i [(\boldsymbol{A}\boldsymbol{D}_A^{-1})^i \mathbf{u}][(\boldsymbol{B}\boldsymbol{D}_B^{-1})^i \mathbf{v}]^T \\ + \alpha^t [(\boldsymbol{A}\boldsymbol{D}_A^{-1})^t \mathbf{u}][(\boldsymbol{B}\boldsymbol{D}_B^{-1})^t \mathbf{v}]^T.\end{aligned}$$

With some reorganization, this can be written: $\boldsymbol{X}^{(t+1)} = \boldsymbol{U}\boldsymbol{V}^T$ for an n-by-$(t+1)$ matrix $\boldsymbol{U}$ and an m-by-$(t+1)$ matrix $\boldsymbol{V}$. The PageRank solution converges fast in the regime $\alpha \in [0.7, 0.9]$ and usually only 10 iterations are enough. We now generalize this insight to multiple networks to handle multiple network alignment.

For multiple networks, the above formulation extends straightforwardly. We need to compute the PageRank vector on the network $\boldsymbol{A}_k \otimes \boldsymbol{A}_{k-1} \otimes \cdots \otimes \boldsymbol{A}_2 \otimes \boldsymbol{A}_1$. Since we have k networks, the analogue of the matrix $\boldsymbol{Y}$ is now a k dimensional tensor $\underline{\boldsymbol{Y}}$ that stores the PageRank measure between every possible combination k of nodes coming from k distinct networks. In words, we have $\underline{Y}(i_1, i_2, \ldots, i_k)$ denote the PageRank measure for the "node" representing an alignment between nodes i_1 from the first graph, i_2 from the second, …, and node i_k from the k^{th} graph. Assume now that we have k column stochastic adjacency matrices corresponding to k networks. Call them $\boldsymbol{P}_1 = \boldsymbol{A}_1\boldsymbol{D}_1^{-1}, \boldsymbol{P}_2 = \boldsymbol{A}_2\boldsymbol{D}_2^{-1}, \ldots, \boldsymbol{P}_k = \boldsymbol{A}_k\boldsymbol{D}_k^{-1}$. (Where $\boldsymbol{D}_i$ is the diagonal degree matrix for the ith network.) The massive PageRank vector we are interested in is given by:

$$\mathbf{y} = \alpha(\boldsymbol{P}_k \otimes \boldsymbol{P}_{k-1} \otimes \cdots \otimes \boldsymbol{P}_2 \otimes \boldsymbol{P}_1)\mathbf{y} + (1-\alpha)\mathbf{h}. \tag{6.3}$$

We note that a similar formulation is used in (Li et al., 2018), where the focus is on the sparsity of the vector $\mathbf{h}$ and the solution. In our formulation, we note that the matrix $(\boldsymbol{P}_k \otimes \cdots \otimes \boldsymbol{P}_1)$, and the *vector* $\mathbf{y}$ are never formed explicitly.

We study the case that $\mathbf{h} = \mathbf{u}_k \otimes \mathbf{u}_{k-1} \otimes \cdots \otimes \mathbf{u}_1$, which corresponds to assuming that the tensor representation $\underline{\boldsymbol{H}}$ would be rank 1. In this instance, we can proceed akin to the NSD scenario. We also start the iteration with $\mathbf{y}^{(0)} = \mathbf{h} = \mathbf{u}_k \otimes \mathbf{u}_{k-1} \otimes \cdots \otimes \mathbf{u}_1$. Then, the first iterate is:

$$\begin{aligned}\mathbf{y}^{(1)} &= \alpha(\boldsymbol{P}_k \otimes \cdots \otimes \boldsymbol{P}_1)\mathbf{y}^{(0)} + (1-\alpha)\mathbf{y}^{(0)} \\ &= \alpha(\boldsymbol{P}_k\mathbf{u}_k \otimes \ldots \otimes \boldsymbol{P}_1\mathbf{u}_1) + (1-\alpha)(\mathbf{u}_k \otimes \mathbf{u}_{k-1} \otimes \ldots \otimes \mathbf{u}_1)\end{aligned}$$

At step t, $\mathbf{y}^{(t)}$ can be expressed as follows

$$\mathbf{y}^{(t)} = (1-\alpha)\sum_{i=0}^{t-1} \alpha^i(\boldsymbol{P}_k^i\mathbf{u}_k \otimes \ldots \otimes \boldsymbol{P}_1^i\mathbf{u}_1) + \alpha^t \boldsymbol{P}_k^t\mathbf{u}_k \otimes \ldots \otimes \boldsymbol{P}_1^t\mathbf{u}_1$$

Next, we can decompose the above equation. Form k matrices $\boldsymbol{U}_i$, such that

$$\boldsymbol{U}_i = \begin{bmatrix} c_0\boldsymbol{P}_i^0\mathbf{u}_i & c_1\boldsymbol{P}_i^1\mathbf{u}_i & \ldots & c_{t-1}\boldsymbol{P}_i^{t-1}\mathbf{u}_i & c_t\boldsymbol{P}_i^t\mathbf{u}_i \end{bmatrix} \tag{6.4}$$

where each c_j is $((1-\alpha)\alpha^j)^{1/k}$ when $j \leq t-1$, and $c_t = \alpha^{t/k}$. Hence, $\mathbf{y}^t$ can be rewritten as follows:

$$\mathbf{y}^{(t)} = \sum_{i=0}^{t} \boldsymbol{U}_k(:,i) \otimes \boldsymbol{U}_{k-1}(:,i) \otimes \ldots \otimes \boldsymbol{U}_1(:,i) = \sum_{i=0}^{t} \hat{\mathbf{y}}^{(i)}$$

where $\hat{\mathbf{y}}^{(i)} = \boldsymbol{U}_k(:,i) \otimes \boldsymbol{U}_{k-1}(:,i) \otimes \ldots \otimes \boldsymbol{U}_1(:,i)$, and the notation $\boldsymbol{F}(:,i)$ corresponds to the i^{th} column of a matrix $\boldsymbol{F}$. If we reshape into a tensor with $\text{vec}(\underline{\boldsymbol{Y}}_i) = \hat{\mathbf{y}}^{(i)}$, and $\text{vec}(\underline{\boldsymbol{Y}}^{(t)}) = \mathbf{y}^{(t)}$, then $\underline{\boldsymbol{Y}}^{(t)} = \sum_{i=0}^{t} \underline{\boldsymbol{Y}}_i$. We can thus deduce that $\underline{\boldsymbol{Y}}^{(t)}$ is a sum of $t+1$ rank-1 tensors. (Formally, the matrices $\boldsymbol{U}_1, \ldots, \boldsymbol{U}_k$ are the CP factors of $\underline{\boldsymbol{Y}}^{(t)}$ (Golub and van Loan, 2013, Section 12.5.4).) What remains in our procedure is a way to turn this low rank representation into an alignment by running a matching algorithm (Section 6.4).

6.3.2 Optimizing hyper-permutations

We now show that the formulation in (6.2) is closely related to optimizing over hyper-permutation matrices. Consider aligning three networks. The idea is to permute symmetric $\boldsymbol{C}$ to match symmetric $\boldsymbol{B}$ and $\boldsymbol{A}$, and permute $\boldsymbol{B}$ and $\boldsymbol{A}$ to match $\boldsymbol{C}$. This yields the objective:

$$\begin{aligned}&\min_{\underline{\boldsymbol{P}}} \|\underline{\boldsymbol{P}} \times_1 \boldsymbol{C} - \underline{\boldsymbol{P}} \times_2 \boldsymbol{B} \times_3 \boldsymbol{A}\|_F^2 = \\ &\min_{\underline{\boldsymbol{P}}} \|(\boldsymbol{C} \otimes \boldsymbol{I} \otimes \boldsymbol{I})\mathbf{p} - (\boldsymbol{I} \otimes \boldsymbol{B} \otimes \boldsymbol{A})\mathbf{p}\|_2^2\end{aligned} \tag{6.5}$$

where $\mathbf{p} = \text{vec}(\underline{\boldsymbol{P}})$ and $\underline{\boldsymbol{P}}$ is a hyper-permutation tensor and we use the vec equivalences from (Golub and van Loan, 2013, Section 12.4.11). This can be reworked into the objective $-2\mathbf{p}^T(\boldsymbol{C} \otimes \boldsymbol{B} \otimes \boldsymbol{A})\mathbf{p} + \mathbf{p}^T(\boldsymbol{C} \otimes \boldsymbol{I} \otimes \boldsymbol{I})^2\mathbf{p} + \mathbf{p}^T(\boldsymbol{I} \otimes \boldsymbol{B} \otimes \boldsymbol{A})^2\mathbf{p}$. Minimizing this objective can be related to the maximization problem in (6.2), specially when the networks have a similar number of nodes and edges because the additive terms $\mathbf{p}^T(\boldsymbol{C} \otimes \boldsymbol{I} \otimes \boldsymbol{I})^2\mathbf{p} + \mathbf{p}^T(\boldsymbol{I} \otimes \boldsymbol{B} \otimes \boldsymbol{A})^2\mathbf{p}$ will be nearly constant. And thus, we get a near equivalence between the formulation in (6.5) over hyper-matrices and ours if we neglect these terms. This analysis extends to a variety of other ways to partition the set of networks into two groups.

6.4 K-Dimensional Matching with Low-Rank Factors

In this section, we discuss two approaches to solve the k-dimensional matching problem:

$$\begin{array}{ll}\text{maximize} & \sum_{i,j,\ldots,\ell} \underline{T}(i,j,\ldots,\ell)\underline{X}(i,j,\ldots,\ell) \\ \text{subject to} & \flat_1(\underline{\boldsymbol{X}})\mathbf{e} \le \mathbf{e}; \ldots; \flat_k(\underline{\boldsymbol{X}})\mathbf{e} \le \mathbf{e}; \underline{X}(i,j,\ldots,\ell) \in \{0,1\}\end{array} \tag{6.6}$$

when $\underline{\boldsymbol{T}}$ is given by a non-negative rank-t representation:

$$\begin{aligned}\underline{T}(i,j,\ldots,\ell) &= \textstyle\sum_{t=1}^{r} U_1(i,t)U_2(j,t)\cdots U_k(\ell,t) \\ \Leftrightarrow \underline{\boldsymbol{T}} &= \textstyle\sum_{i=1}^{t} \underline{\boldsymbol{T}}_i \text{ where } \operatorname{vec}(\underline{\boldsymbol{T}}_i) = \boldsymbol{U}_k(:,i)\otimes \boldsymbol{U}_{k-1}(:,i)\otimes\cdots\otimes \boldsymbol{U}_1(:,i)\end{aligned} \tag{6.7}$$

The first builds on an algorithm for low-rank bipartite matchings from Nassar et al. (2018). The second builds on algorithms for progressive alignment (Malmi et al., 2017) and k-partite alignment problems (Gligorijevic et al., 2016; He et al., 2000).

6.4.1 An a-posteriori approximation bound from the best single-rank alignment

We proceed to show a new k-dimensional matching algorithm that can be applied on tensors represented as low-rank factors. The idea is that we use each rank-1 factor $\underline{\boldsymbol{T}}_i$ to generate a single k-dimensional matching. Then we provide an a-posteriori bound on the best alignment in this set. This extends our work on bipartite matching from the previous chapter. In practice, these bounds are very good and provide approximation factors around 1.08. Figure 6.2 is an illustration of the histogram distribution of the values of the approximatin bound found.

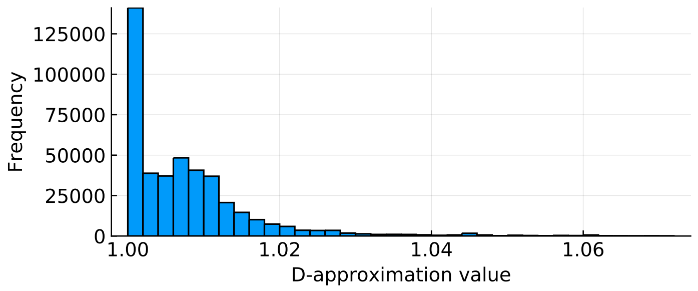


Figure 6.2. This figure shows the D approximation values from the experiment in section 6.5.6. These numbers show that the approximation bound D is very close to 1 in practice.

To do so, we first need a specific generalized rearrangement inequality for k sequences. Generalized forms of the rearrangement inequality are often posed as a homework problem as their proof follows an extension to the proof by induction of the inequality for two sequences. For completeness, we provide a full proof here.

Generalized Rearrangement Inequality.. Assume that we have k sequences of numbers that are all positive. Let $x_i^{(j)}$ denote the i^{th} element in the j^{th} sequence and assume that $x_1^{(j)} \leq x_2^{(j)} \leq \ldots x_n^{(j)}$ for all sequences. The generalized rearrangement inequality guarantees that:

$$\sum_{i=1}^{n} \prod_{j=1}^{k} x_i^{(j)} \geq \sum_{i=1}^{n} x_i^{(1)} \prod_{j=2}^{k} x_{\sigma_j(i)}^{(j)}$$

where σ_j is any permutation function corresponding to the j^{th} sequence.

Proof The proof follows a similar strategy as the proof of rearrangement inequality on two sequences and we extend it here. We prove this by induction.

We first assume that we have k sequences with 2 elements each. We claim that

$$\prod_{i=1}^{k} x_1^{(i)} + \prod_{i=1}^{k} x_2^{(i)} \geq x_1^{(1)} \prod_{i=2}^{k} x_{\sigma_i(1)}^{(i)} + x_2^{(1)} \prod_{i=2}^{k} x_{\sigma_i(2)}^{(i)}$$

We prove this by contradiction. Assume that there exists a permutation σ such that the above formula is incorrect. Let us expand the right hand side.

$$x_1^{(1)} \prod_{i=2}^{k} x_{\sigma(1)}^{(i)} + x_2^{(1)} \prod_{i=2}^{k} x_{\sigma(2)}^{(i)} =$$
$$x_1^{(1)} x_{\sigma_2(1)}^{(2)} \ldots x_{\sigma_k(1)}^{(k)} + x_2^{(1)} x_{\sigma_2(2)}^{(2)} \ldots x_{\sigma_k(2)}^{(k)}$$

Let $P_1 = x_1^{(1)} \prod x_{\sigma_j(1)}^{(j)} \forall j$ such that $\sigma_j(1) = 1$. Let $P_2 = \prod x_{\sigma_j(1)}^{(j)} \forall j$ such that $\sigma_j(1) = 2$. Similarly, let $Q_1 = \prod x_{\sigma_j(2)}^{(j)} \forall j$ such that $\sigma_j(2) = 1$, and $Q_2 = x_2^{(1)} \prod x_{\sigma_j(2)}^{(j)} \forall j$ such that $\sigma_j(2) = 2$. Then,

$$x_1^{(1)} \prod_{i=2}^{k} x_{\sigma(1)}^{(i)} + x_2^{(1)} \prod_{i=2}^{k} x_{\sigma(2)}^{(i)} = P_1 P_2 + Q_1 Q_2$$

Now observe that $P_2 \geq Q_1$ and $Q_2 \geq P_1$ by definition. Thus, using the rearrangement inequality on two sequences, $P_2 Q_2 + P_1 Q_1$ must be $\geq P_1 P_2 + Q_1 Q_2$, thus, a contradiction and the best way to arrange the terms in these sequences is by grouping all the bigger elements together and the smaller elements together.

Next, we assume that the property holds for k sequences with $n-1$ elements each. We now prove it for k sequences with n elements each. We prove this by contradiction and assume that there exists $k-1$ permutations $(\sigma_2, \ldots \sigma_k)$ of size n each which achieves the maximal pairing, i.e.

$$\sum_{i=1}^{n} x_i^{(1)} \prod_{j=1}^{k} x_{\sigma_j(i)}^{(j)}$$

If this sum is the maximal sum, then there are two terms in this summation of the form:

$$x_1^{(1)} x_{i_2}^{(2)} x_{i_3}^{(3)} \ldots x_{i_k}^{(k)} + x_{j_1}^{(1)} x_1^{(2)} x_{j_3}^{(3)} \ldots x_{j_k}^{(k)}$$

We can now apply the base case on the following k sequences with 2 elements:

$$\{x_1^{(1)}, x_{j_1}^{(1)}\}, \{x_1^{(2)}, x_{i_2}^{(2)}\}, \{x_{i_3}^{(3)}, x_{j_3}^{(3)}\}, \ldots, \{x_{i_k}^{(3)}, x_{j_k}^{(3)}\}$$

Without loss of generality, assume that $x_{i_l}^{(l)} \leq x_{j_l}^{(l)}$ for all $3 \leq l \leq k$. Thus, by using the base case, we know that $x_1^{(1)} x_1^{(2)} x_{i_3}^{(3)} \ldots x_{i_k}^{(k)} + x_{j_1}^{(1)} x_{i_2}^{(2)} x_{j_3}^{(3)} \ldots x_{j_k}^{(k)} \geq x_1^{(1)} x_{i_2}^{(2)} x_{i_3}^{(3)} \ldots x_{i_k}^{(k)} + x_{j_1}^{(1)} x_1^{(2)} x_{j_3}^{(3)} \ldots x_{j_k}^{(k)}$. This means that we have just found a rearrangement of two terms in the maximal summation that can be rearranged to achieve a higher weight, which is a contradiction. We proceed with the same strategy to show that the permutations should indeed be the identity permutations. Next, we consider the following two terms:

$$x_1^{(1)} x_1^{(2)} x_{i_3}^{(3)} \ldots x_{i_k}^{(k)} + x_{j_1}^{(1)} x_{j_2}^{(2)} x_1^{(3)} \ldots x_{j_k}^{(k)}$$

Similarly, and assuming that $i_l \leq j_l$, we can conclude that

$$x_1^{(1)} x_1^{(2)} x_1^{(3)} \ldots x_{i_k}^{(k)} + x_{j_1}^{(1)} x_{j_2}^{(2)} x_{j_3}^{(3)} \ldots x_{j_k}^{(k)} \geq$$
$$x_1^{(1)} x_1^{(2)} x_{i_3}^{(3)} \ldots x_{i_k}^{(k)} + x_{j_1}^{(1)} x_{j_2}^{(2)} x_1^{(3)} \ldots x_{j_k}^{(k)}.$$

If we proceed with the same strategy for all the remaining terms, we will achieve a summation of the form

$$x_1^{(1)} x_1^{(2)} \dots x_1^{(k)} + \sum_{i=2}^{n} x_i^{(1)} \prod_{j=1}^{k} x_{\sigma_j(i)}^{(j)}$$

By the inductive hypothesis, we know that $\sum_{i=2}^{n} x_i^{(1)} \prod_{j=1}^{k} x_i^{(j)} \geq \sum_{i=2}^{n} x_i^{(1)} \prod_{j=1}^{k} x_{\sigma_j(i)}^{(j)}$, and hence each of the permutations must be the identity permutation.

$$\sum_{i=1}^{n} x_i^{(1)} \prod_{j=1}^{k} x_{\sigma_j(i)}^{(j)}$$

And with that, the rearrangement inequality for k sequences of size n each is proved. For cases when the number of elements in each sequence is different, we set n to be the size of the smallest sequence and pick the top n elements of each of the other sequences. The reason we pick the top n can be viewed as a direct application of the rearrangement inequality. ■

Now, assume that we have a *k-dimensional* tensor $\underline{\boldsymbol{T}}$ of the form (6.7). For each rank-1 tensor $\underline{\boldsymbol{T}}_i$, the generalized rearrangement inequality guarantees the best matching on it can be computed by sorting the vectors $\boldsymbol{U}_1(:,i), \dots, \boldsymbol{U}_k(:,i)$ in decreasing order and aligning the elements. (We find it helpful to think of the pairwise, matrix, case where $\boldsymbol{T}_i = \mathbf{u}\mathbf{v}^T$ and the sorting is simple to see.) Let the binary-valued tensors $\underline{\boldsymbol{M}}_i$ of size $n_1 \times n_2 \times \dots n_k$ store the matching corresponding to $\underline{\boldsymbol{T}}_i$ tensor, i.e., $\underline{M}(j_1, j_2, \dots, j_k) = 1$ if $(j_1, j_2, \dots, j_k)$ is a match, and 0 otherwise. Next, we prove the following:

Result. *Consider the best k-dimensional matching from the set* $\underline{\boldsymbol{M}}_1, \dots, \underline{\boldsymbol{M}}_t$, *then this is a D-approximation to the best k-dimensional matching, where D is an aposterori computable bound.*

Define $\underline{\boldsymbol{M}}_i \bullet \underline{\boldsymbol{T}}_i = \text{vec}(\underline{\boldsymbol{M}}_i)^T \, \text{vec}(\underline{\boldsymbol{T}}_i)$ to be the weight of the matching $\underline{\boldsymbol{M}}_i$ applied on the tensor $\underline{\boldsymbol{T}}_i$. Also, let $\underline{\boldsymbol{M}}^*$ to be the matching that achieves the maximum possible weight on $\underline{\boldsymbol{T}}$.

Define $d_{i,j} = \frac{\underline{\boldsymbol{M}}_i \bullet \underline{\boldsymbol{T}}_i}{\underline{\boldsymbol{M}}_j \bullet \underline{\boldsymbol{T}}_i}$, and $d_j = \max_i d_{i,j}$. Let $j^* = \text{argmin}_j d_j$. Set $D = d_{j^*}$. Then, $\underline{\boldsymbol{M}}^* \bullet \underline{\boldsymbol{T}} \leq D \underline{\boldsymbol{M}}_{j^*} \bullet \underline{\boldsymbol{T}}$. The proof of this statement follows:

$$\begin{aligned} \underline{\boldsymbol{M}}^* \bullet \underline{\boldsymbol{T}} = \underline{\boldsymbol{M}}^* \bullet \textstyle\sum_{i=1}^t \underline{\boldsymbol{T}}_i &\leq \textstyle\sum_{i=1}^t \underline{\boldsymbol{M}}_i \bullet \underline{\boldsymbol{T}}_i \\ &\leq D \textstyle\sum_{i=1}^t (\underline{\boldsymbol{M}}_{j^*} \bullet \underline{\boldsymbol{T}}_i) \leq D(\underline{\boldsymbol{M}}_{j^*} \bullet \underline{\boldsymbol{T}}), \end{aligned}$$

where we used $\underline{\boldsymbol{M}}_i \bullet \underline{\boldsymbol{T}}_i \leq D \underline{\boldsymbol{M}}_{j^*} \bullet \underline{\boldsymbol{T}}_i$ by the definition of the quantities. Therefore, the matching $\boldsymbol{M}_{j^*}$ achieves a $D-$approximation on the tensor $\underline{\boldsymbol{T}}$.

6.4.2 A progressive alignment

The bounds given by the low-rank matching algorithm (above) are often very good (around 1.08). In practice we found the following procedure to give better results in terms of the overall multiple network alignment objective. The inspiration for this algorithm is the progressive nature of both *ProgNatalie++* and *FUSE* (Malmi et al., 2017; Gligorijevic et al., 2016), and a progressive algorithm for the k-partite matching problem (He et al., 2000). For three networks (a three-mode tensor), the idea is: align (via bipartite matching) the first two modes (networks). Then, use the alignment between the first two modes to produce a new bipartite alignment problem to fold in the third mode. That is, if we know that node i_1 in network 1 matches to i_2 in network 2, then we can look at the entries $\underline{\boldsymbol{T}}(i_1, i_2, :)$ to determine the best match for (i_1, i_2) in the third network. These entries also have low-rank structure. This can be done via k bipartite matching calls in our low-rank framework, and it is easiest to state the overall procedure as an algorithm. We briefly studied optimizing the ordering of alignment, but this did not seem to yield large differences.

6.5 Experiments

To evaluate our proposed algorithm, we perform a series of experiments (i) on synthetically generated networks, where we can easily vary parameters to understand

Algorithm 4 Pseudocode for the progressive k-dimensional matching algorithm.

```
1: **Input**: $\boldsymbol{U}_1, \boldsymbol{U}_2, \ldots, \boldsymbol{U}_k$
2: **Output**: Matching $\boldsymbol{M}$ with $k$ columns and matches in rows
3: a,b = bipartitematching($(\boldsymbol{U}_1 \boldsymbol{U}_2^T)$)   ▷ match first two modes and return matches a and b.
4: M[:,1:2] = [a,b]
5: **for** i = 3 to k **do**
6:     $\boldsymbol{U} = \boldsymbol{U}_1[\boldsymbol{M}[:,1],:] \odot \boldsymbol{U}_2[M[:,2],:] \odot \cdots \odot$ U$_{i-1}$[M[:,i],:]   ▷ generate the matching information with $i-1$ modes matched
7:     a,b = bipartitematching($\boldsymbol{U}\boldsymbol{U}_i^T$)   ▷ using element-wise/Hadamard product $\odot$
8:     M = M[a,:];
9:     M[:,i] = b   ▷ permute and extend the matching
10: **end for**
```

how the algorithms behave, (ii) on the problem of aligning snapshots of a temporally evolving network of internet routers, and (iii) on inferring high triangle Jaccard similarity in anonymized egonets, and (iv) a case study on large networks with about a million nodes.

6.5.1 Methods

We precisely state the parameters of the various methods we consider here, including some obvious baseline measures such as a random method and intuitive extensions to pairwise methods. We also tried two software packages *IsoRankN* and *FUSE* for these problems. These methods all returned empty alignments, which we believe is due to our lack of *prior* or *biological* information to guide the method.

Pairwise. A simple way to align multiple networks is to run a pairwise network alignment for all pairs of networks and extract any consistent alignment. For instance, if the following three pairs appeared while aligning the three networks $\boldsymbol{G}_A$, $\boldsymbol{G}_B$, $\boldsymbol{G}_C$, (a_1, b_3), (b_3, c_9), (a_1, c_9), we treat the triplet (a_1, b_3, c_9) as a match. For choosing the right pairwise method to employ in this paradigm, we wanted a pairwise method that does not rely on prior similarity scores, thus we chose the recent low rank spectral network alignment by Nassar et al. (2018).

By degree. This method is intuitive since we would expect that high degree nodes match to each other. For each network, sort the nodes according to their degrees, and then match the top degree nodes with each other until no more nodes are left in one of the networks.

Progressive EigenAlign. We mention the recent pairwise network alignment algorithm EigenAlign in Feizi et al. (2016), and its low rank formulation from Nassar et al. (2018) to be strong pairwise network alignment algorithm when no prior information about node similarity is present. Here, we suggest a simple extension to this algorithm to adapt it to align multiple networks and we follow a progressive approach. We start with two networks to align them using the low rank formulation of EigenAlign from Nassar et al. (2018). After the first two networks are aligned, we fold them on top of each other by using the new matches to form a new network. Then, we use this network to align it to the next network. For k networks, the pairwise procedure would occur $k-1$ times.

Random. Another method we choose to compare our existing methods to is a random alignment. This is more of a sanity check experiment to make sure that the algorithms we are using do not generate arbitrary matchings and that indeed a random matching would not outperform any of the existing methods.

MultiLR-D. This is our algorithm, where we compute the matrices $\boldsymbol{U}_i$ from (6.4) with 8 iterations and $\alpha = 0.8$ then the final alignment is extracted by our D-approximation (Section 6.4.1). We support the choice of 8 iterations with a small experiment shown in Figure 6.3.

MultiLR-Prog. This is our algorithm, where the $\boldsymbol{U}_i$ are from (6.4) with 8 iterations and $\alpha = 0.8$ and the final match is determined by the progressive method (Algorithm 4). The bipartite matching problems are themselves solved via a low-rank bipartite matching procedure from Nassar et al. (2018) (with parameter $b = 10$).

MultiLR-Prog+. This is the same as MultiLR-Prog, but where we replace the element-wise multiplication from Algorithm 4 (line 6) with a mixture model for $\boldsymbol{U}$. Specifically we use $\boldsymbol{U} = (1/2)\boldsymbol{U}^*/\text{sum}(\boldsymbol{U}^*) + (1/2)\boldsymbol{U}^+/\text{sum}(\boldsymbol{U}^+)$ where $\boldsymbol{U}^*$ is the matrix computed on line 6 and $\boldsymbol{U}^+$ is the matrix computed on line 6 with element-wise multiplication replaced with element-wise addition. Empirically (and by accident), we found that this strategy performed more consistently with large numbers of networks; theoretically, it is more akin to treating the alignment data as finding a combination of k-dimensional matches and dense k-partite regions as in Gligorijevic et al. (2016); Liao et al. (2009).

ProgNatalie++ and ProgNatalie++ with prior. We use *ProgNatalie++* from Malmi et al. (2017) using a uniform prior for small problems. This does not scale with a reasonable runtime (we ran problems with 100 nodes and 5 networks for a day without completing), and so we also consider using the union of alignments produced by our low-rank factors (Section 6.4.1) as the prior. In this case, the algorithms complete in a reasonable amount of time (an hour for 5 networks with 100 nodes) because of the constrained matching space.

In all experiments, we use 8 iterations, and below we run a small example to conclude that 8 iterations are enough.

6.5.2 Evaluation

We use a few evaluation metrics to discuss the resulting alignments. When there is a true alignment known among the set of networks, then we compute ***degree weighted recovery***, which is the number of correct pairs, scaled by the degrees of the nodes in the network. We often found that the algorithms would align large portions of the network well, but make mistakes on regions of ambiguous degree-1 nodes (or other automorphic regions of the graphs). Consequently, this measure places more emphasis on high degree regions. The pairwise nature also protects against a single mistake in, say, 100 networks ruining the other 99 correctly aligned results. The

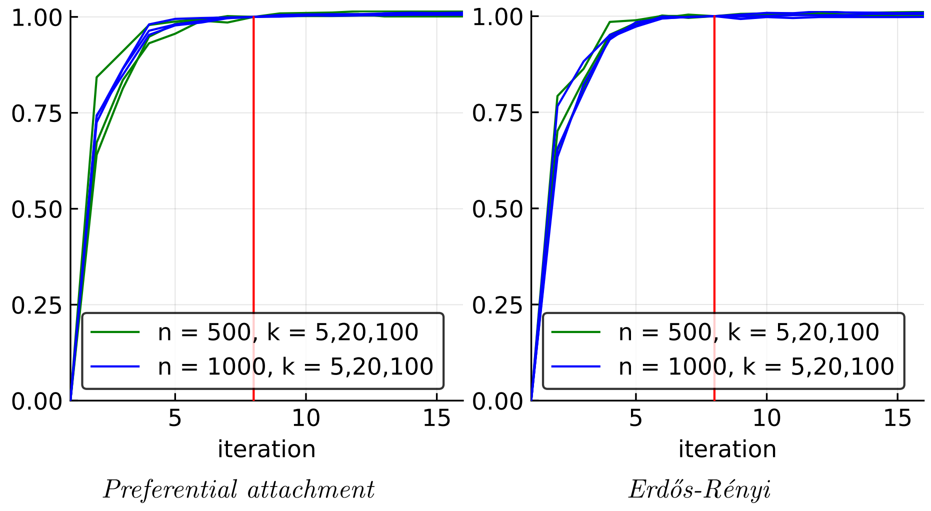


Figure 6.3. To make sure that 8 iterations of the power method are enough to achieve a good result, we run our algorithm MultiLR-D for other iteration values and discover that after 8 iterations essentially nothing changes. The y-axis in these plots is the degree weighted recovery value relative to the value at iteration 8; there are 6 curves for three different settings in terms of the number of networks and the size of the networks that are all indistinguishable.

formal measure involves some ancillary notation. Let D_j be the sum of all degrees in network j. The weight of a pair of vertices in network j and k in a pair of networks is $w(v_j, v_k) = (\text{degree}(v_j) + \text{degree}(v_k))/(D_j + D_k)$; the expression $\text{correct}(v_j, v_k)$ is one if node v from network j should be aligned to node v from network k; the *score* of a single alignment of vertices between all networks is:

$$\text{score}(v_1, \ldots, v_k) = \binom{k}{2}^{-1} (\sum_{j=1}^{k} \sum_{h=j+1}^{k} w(v_j, v_h)\text{correct}(v_j, v_h)) \tag{6.8}$$

The overall degree weighted recovery score is simply the sum of scores for each alignment set. (These scores are scaled to sum to 1 for a perfect alignment of isomorphic networks.)

The ***normalized overlap*** of a set of networks $A_1, \dots, A_k$ is the number of edges in the conserved region after alignment scaled by the number of edges of the largest graph. (Again, normalized overlap scores are between 0 and 1). If $\tilde{\boldsymbol{A}}_1, \dots \tilde{\boldsymbol{A}}_k$ are the adjacency matrices permuted via the alignment, then this is $\text{nnz}(\tilde{\boldsymbol{A}}_1 \odot \cdots \odot \tilde{\boldsymbol{A}}_k)/\max(\text{nnz}(\boldsymbol{A}_i), \dots, \text{nnz}(\boldsymbol{A}_k))$ where $\odot$ is the element-wise product.

6.5.3 Aligning Erdős-Rényi and preferential attachment graphs

In this first experiment, our goal is to study how well our algorithm recovers solutions in a planted problem as we add more noise and how this changes as we vary the number of networks to be aligned. We consider Erdős-Rényi and preferential attachment graphs with average degree 8 as reference graphs, and then randomly delete edges to generate k instances of the networks to align. In this case, the ground-truth alignment is known (even though there are symmetries in these graphs, and thus the ground-truth may be ambigious).

Graph Generation. For Erdős-Rényi, we set the edge probability such that we achieve the expected degree $d = 8$ and n nodes. For preferential attachement, to generate a graph with n edges, we start with a 5-node clique graph and add θ edges from each new vertex following the preferential attachment model. The expected degree is 2θ because each new edge gets counted twice in the average degree computation. Then to generate k instances of these graphs, we generate one reference graph, and then we then pick an edge deletion probability p_e, and generate k instances of the base graph, we allow each edge to be deleted according the the probability p_e. We repeat this process k times to reach k networks.

For our first experiment, we consider using 5 networks with 500 nodes and vary the edge-deletion probability. The results from our methods and the baselines are shown in Figure 6.4 (top two panels). In both types of graphs, both of our progressive low-rank methods achieved the best results, whereas MultiLR-D did not perform well as more edges were deleted. Although this method is the least expensive (see runtime discussion in section 6.5.7) and provides a theoretically strong bound on the matches (here, the highest value of D was 1.07) the method relies on a sorting procedure, which may mislead the matching when there are many numbers close to each other.

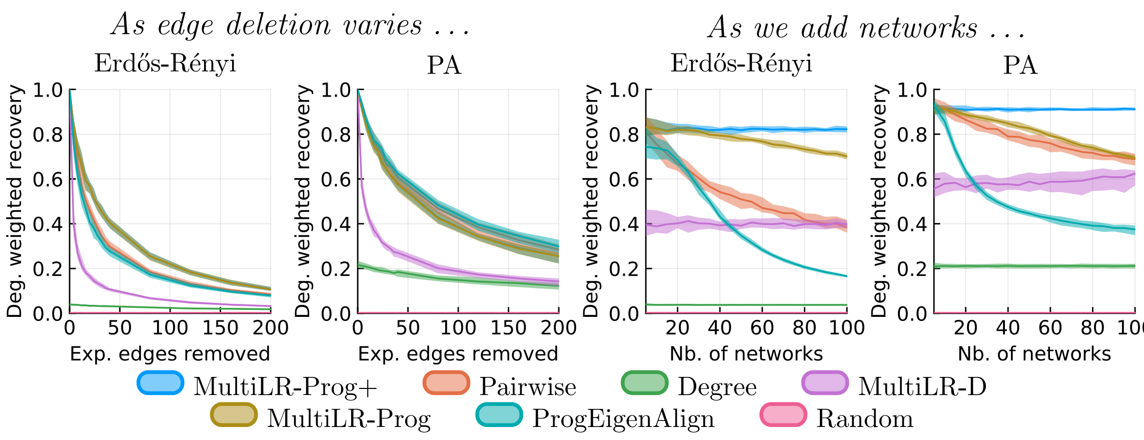


Figure 6.4. (First two panels) As we increase the expected number of edges removed while aligning 5 networks, all methods recover fewer true matches and MultiLR-Prog and MultiLR-Prog+ are consistently the best where MultiLR-D does not do well. Note that MultiLR-Prog and MultiLR-Prog+ are overlapping here. (Last two panels) As we vary the number of networks to be aligned, all methods decay in quality except for MultiLR-Prog+ and MultiLR-D, with MultiLR-Prog+ consistently achieving the best result (the two right most figures). In all figures, the shaded areas represent the 20^{th} and the 80^{th} percentiles with these experiments run for 50 trials.

For the second experiment, we also consider 500 node networks again and consider aligning a growing number of networks with a fixed edge deletion probability $0.5/n$. This corresponds to the case where we expect good accuracy. The results are shown in Figure 6.4 (lower two panels) and show that MultiLR-Prog+ and MultiLR-D are the only methods that are not sensitive to the number of networks. Because of this, MultiLR-D becomes a competitive method for large numbers of networks. Figure 6.4 shows the results for these two experiments.

As the network sizes vary. In this experiment, we are interested in observing how the alignment quality varies as we change the sizes of the networks to be aligned.

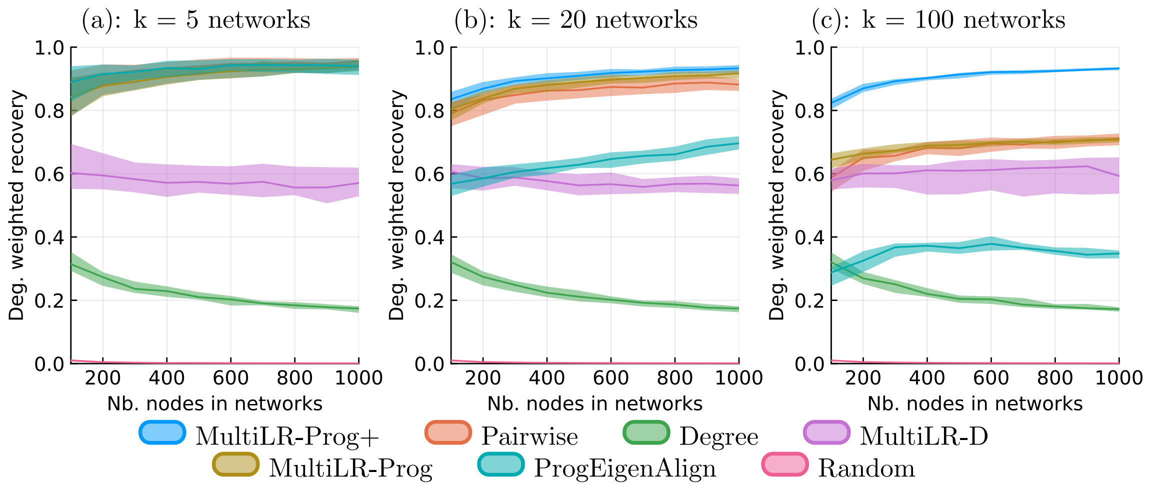


Figure 6.5. These figures show the weighted recovery scores on preferential attachment graphs as we vary the sizes of the networks, and the number of networks to be aligned. We observe that when the number of networks is small enough (5 networks) pairwise and multiple alignment methods achieve similar results. Whereas when we increase the number of networks to be aligned, multiple alignment sustains its result whereas the pairwise method fails to do so. In all figures, the shaded areas represent the 20^{th} and the 80^{th} percentiles with these experiments run for 50 trials.

Here, we use preferential attachment graphs and we fix the edge deletion probability to $p_e = 0.5/n$, as we vary n. We observe that all methods are essentially resistant to the change in the network sizes whereas this behavior is not true when the number of networks become much bigger (such as 100). From figure 6.5, we can conclude that MultiLR-Prog+ is resistant to both changes in the network sizes, as well as the number of networks to be aligned. Interestingly, MultiLR-D is also resistant to such changes but with a worse recovery score.

6.5.4 Aligning real-world temporal graph snapshots

A representative use of our methods would be to align a set of snapshots of a real-world graph over time. Here we consider a dataset from Leskovec et al. (2005) which consists of snapshots of an Internet routers network at 733 time points. We

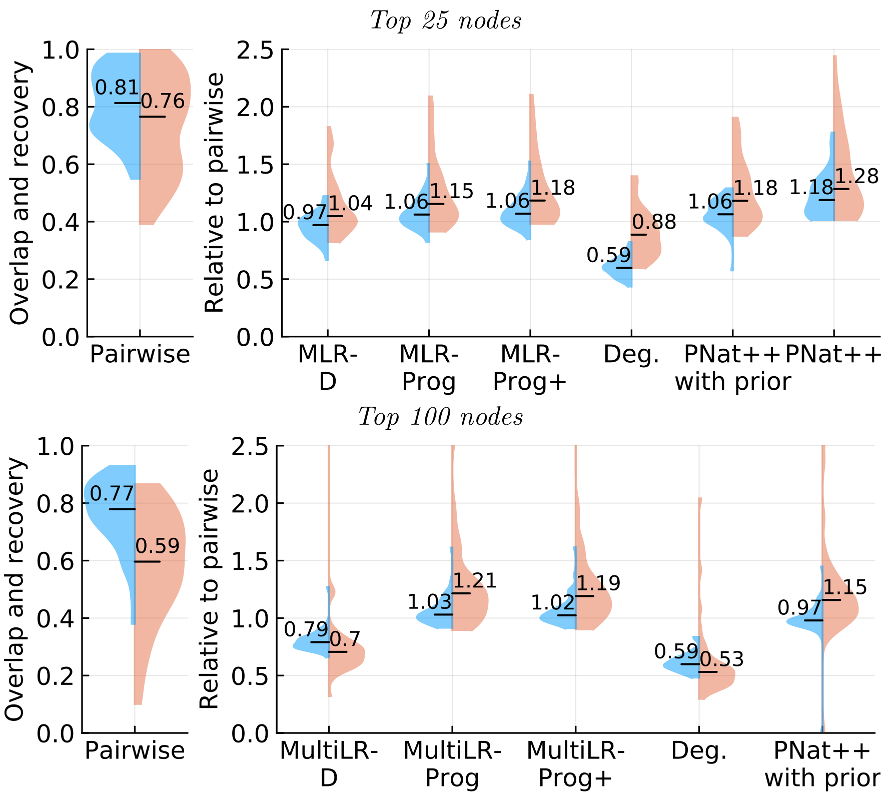


Figure 6.6. We consider 50 trials of aligning 5 real-world router graphs and show a violin plot (with the median flagged) of the degree weighted recovery (blue) and normalized overlap (red) side by side for the pairwise method. For the other methods, we show values relative to the pairwise scores. These results show that we are almost as good as the existing state of the art method ProgNatalie++ on the small problems, whereas our methods run faster, and we can scale to larger problems.

consider two problems: aligning 5 random snapshots with the 25 highest degree nodes (where we are able to run existing methods) and aligning 5 random snapshots with the 100 highest degree nodes (where we can still run ProgNatalie++ with our low-rank generated prior).

In Figure 6.6, we show a violin plot of the distribution of our results in terms of overlap and degree weighted recovery over 50 trials of 5 random snapshots. For the small run, we get comparable results to ProgNatalie++, while running in less than 2 seconds vs. 40 minutes (see more timing in section 6.5.7). For the larger run, MultiLR-Prog and MultiLR-Prog+ achieve results that consistently outperform the pairwise baseline in terms of overlap.

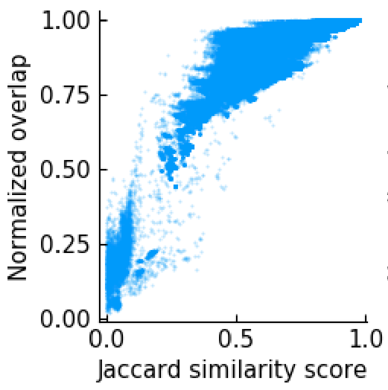
6.5.5 Aligning anonymized egonets

Next, we use our multiple network alignment algorithm to align anonymized egonets of the collaboration network DBLP (Esfandiar et al., 2010). This experiment is inspired by the one in the previous chapter for pairs of networks (and from (Nassar et al., 2018)). In DBLP, the nodes are authors, and edges represent coauthorship (Leskovec et al., 2005). We consider whether or not multiple alignment could infer whether a group of three mutual coauthors (i.e. a triangle in the network) has high Jaccard similarity when we only know anonymized egonets from the original network.

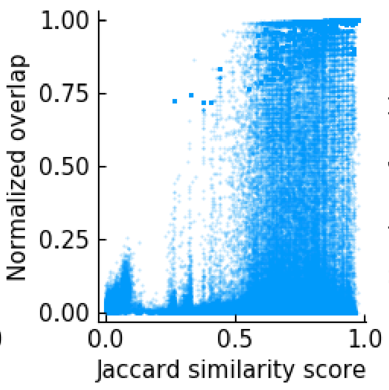
We use Jaccard$(a, b, c) = \frac{N(a) \cap N(b) \cap N(c)}{N(a) \cup N(b) \cup N(c)}$ where $N(a)$ is the set of neighbors of node a. For each triple of three coauthors with at least 100 other co-authors, we align the egonets using the MultiLR-D method and measure the normalized overlap. The results in Figure 6.7 show that we can easily infer high-Jaccard similarity whereas pairwise techniques cannot. This experiment entails aligning 425,388 triplets of networks and MultiLR-D runs in about 1.5 hours whereas the pairwise method takes a little over 4 hours to finish. To ensure that we could be confident that high-overlap implies high-Jaccard, we show that random triples are unlikely to have high normalized overlap in the final figure panel.

6.5.6 A case study on large graphs

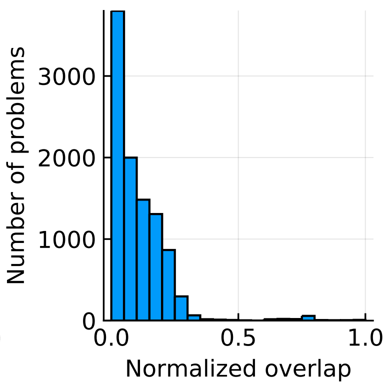
In the previous sections, we note the scalability of our methods as we increase the number of networks; here we want to study our methods on large graphs, specifically graphs with more than one million nodes. We used 4 base networks from the SNAP database that satisfy this criteria, namely three road networks *roadNet-CA, roadNet-PA, and roadNet-TX* from Leskovec et al. (2008), and an Internet topology graph



(a) Using MNA



(b) Using pairwise



(c) MNA on random networks

Figure 6.7. Figures a and b show the normalized overlap of the aligned three egonets using multiple network alignment and pairwise respectively. These two figures show that when using multiple network alignment, normalized overlap track the Jaccard similarity scores whereas the pairwise method fails to show that. Figure c shows that the opposite is true as well. For a random set of three networks, the normalized overlap is is less than 0.25 in the majority of experiments.

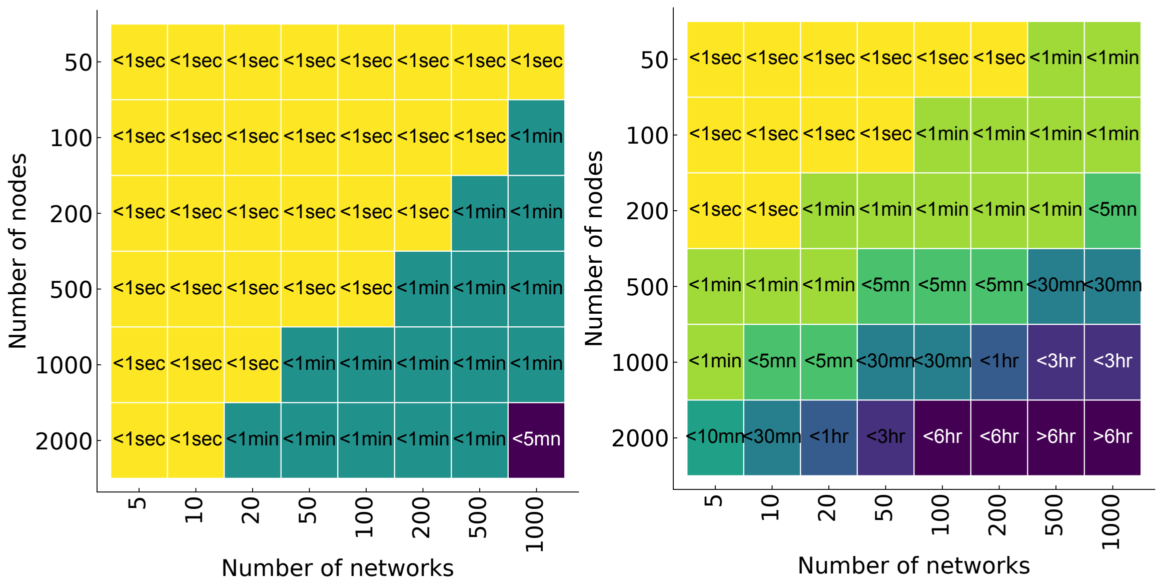
as-Skitter from Leskovec et al. (2005). The pipeline of the experiment is similar to that in Section 6.5.3; we use the edge deletion probablity p_e/n with $p_e = \{0.1, 0.5\}$ and generate 5 instances of the graph.

Methods. All existing MNA methods render themselves infeasible for such kinds of problems, mainly because of an intermediate step that involves solving a huge bipartite matching procedure. From our methods, MultiLR-D is the most interesting comparison as it runs extremely fast and only relies on a sorting procedure. We also use the *By Degree* method described in Section 6.5, as it is the only other feasible method, and provides a sanity check of how well MultiLR-D is performing.

Results and conclusions. We show the average results from 20 runs of this experiment in Table 6.1. These problems were extremely challenging in practice, and we account the main challenge to symmetries in these graphs (i.e. areas in the base network looking topologically similar to other areas in the network). Nevertheless, we note here that while existing methods can only scale to five or ten networks with 100s of nodes, our method is the first feasible option for aligning large graphs specially

Table 6.1.
Degree weighted recovery for large graphs from Section 6.5.6.

Dataset	Nodes	Edges	MultiLR-D		Degree	
			0.1	0.5	0.1	0.5
as-Skitter	1.6M	11M	0.90	0.78	0.15	0.15
roadNet-TX	1.3M	1.8M	0.98	0.43	10^{-5}	10^{-5}
roadNet-CA	1.9M	2.7M	0.67	0.42	10^{-5}	10^{-5}
roadNet-PA	1M	1.5M	0.66	0.31	10^{-5}	10^{-5}



MultiLR-D *MultiLR-Prog+*

Figure 6.8. The runtime as we run MultiLR-D (top) and MultiLR-Prog+ (bottom) on the synthetic experiments on a wide variety of problem sizes with Erdős-Rényi graphs.

when very little perturbation occured. We believe that the result in this challenging scenario is not ideal, but is a demonstation that such low-rank methods expand the scalability envelope for multiple network alignment problems.

6.5.7 Runtime information

In Figure 6.8, we show runtime information for MultiLR-D and MultiLR-Prog+ for synthetic networks as we increase size and the number of networks up to thousands. Then in Table 6.2, we show runtimes for the methods on the routers alignment problems.

Table 6.2.
Runtimes for the 25-node 5-network router problem (columns 2-4), and the 100-node 5-network router problem (columns 5-7).

Algorithm	Time (sec) 25-node problem		100-node problem	
	median	max	median	max
MultiLR-D	0.27	0.40	0.37	0.46
MultiLR-Prog	0.37	0.48	0.34	0.50
MultiLR-Prog+	0.32	0.51	0.38	0.48
Degree	0.02	0.04	0.02	0.04
Random	0.01	0.02	0.01	0.02
ProgEigenAlign	1.39	1.48	2.44	2.59
Pairwise	5.86	6.21	5.04	5.37
ProgNatalie++ & prior	23.09	241.1	649.4	1451
ProgNatalie++	852.0	2823	-	-

6.6 Further discussion

Having a method that accurately and scalably aligns large numbers of networks opens a number of new dimensions in applied machine learning. In ongoing work, we are studying how to use this in terms of aligning graphs derived from functional MRI data. In terms of the current method, we wish to gain a better understanding for why MultiLR-Prog+ outperformed MultiLR-Prog. Our working hypothesis is that the element-wise addition (compared with multiplication) gives the method resilience to mistakes made early in the progressive process. More broadly, the EigenAlign framework (Feizi et al., 2016) is superior to the IsoRank framework for pairwise alignment. The ideas here apply to a multi-network generalization of EigenAlign, however, the analogous tensor $\underline{\boldsymbol{Y}}$ would have a Tucker-style factorization instead of the CP-factorization we get for MultiLR. Crucially, the Tucker factorization needs a t^k-element core that would limit scalability to small k, and we need new k-dimensional matching methods for these.

7 SUMMARY AND FUTURE DIRECTIONS

The work we introduce in this thesis addresses the network alignment problem when no prior knowledge about node similarity is available. This is the first class of work that produces multiple frameworks that handle this scenario—which is known to be the hardest setting in network alignment problems. Througout the work in this thesis, we were able to discover a number of findings about network alignment. We first summarize the network alignment problems addressed.

7.1 Summary of network alignment problems addressed in this thesis

We approached three types of network alignment problem in this thesis. Here, we standardize notation and summarize the three optimization problems in table 7.1. Recall that all of these problems are formulated in terms of maximizing the global overlap of the overlayed networks after relabeling the nodes—the pairwise network alignment problem seeks to find a relabeling of the nodes in one graph, multimodal network alignment seeks to find *one* relabeling in all modes of one multimodal graph, and multiple network alignment seeks to find a relabeling of nodes in $k-1$ networks (when aligning k) networks. In all our network alignment settings, we used no prior known similarity about the node alignment and this setting is traditionally the hardest setting in network alignment algorithms. The key enabling technique for our methods was solving the problems in terms of low rank factors and developing multiple approximate low rank matching algorithms.

Nevertheless, the absence of this prior knowledge can be viewed as a drawback of our methods. For instance, we develop the first scalable multiple network alignment method that is able to align 100s of networks when no prior is provided, but this does not immediately extend to the case when prior knowledge is provided. One way to remedy this is by imposing restrictions on the matchings after the scoring procedure, but this is a heuristic and does not incorporate the prior known similarities in the scoring step. A natural idea to explore under this setting is to use a low rank approximation of the prior similarity matrix as this will still guarantee a low rank structure of the similarity matrix in the pairwise case. Tensor low rank approximations can be a little more tricky but there are a host of methods to explore (Friedland and Tammali, 2015).

Table 7.1.
Summary of the network alignment problems that we addressed in this thesis. All of these problems are formulated in terms of maximizing the edge overlap. One way of solving these problems is by relaxing the constraints to allow $\boldsymbol{P}$ and $\underline{\boldsymbol{X}}$ to be non binary and dense, and then applying a matching procedure. The key idea explored in all problems is provably showing that the solution is low rank and providing low rank matching procedures to accomplish this task.

Pairwise network alignment (Chapter 5)	$\underset{\boldsymbol{P}}{\text{maximize}}$	$\sum_{ij} \sum_{rs} \boldsymbol{A}_{ij} \boldsymbol{B}_{rs} \boldsymbol{P}_{ir} \boldsymbol{P}_{js}$
	subject to	$\sum_i \boldsymbol{P}_{ij} \leq 1$ for all j
		$\sum_j \boldsymbol{P}_{ij} \leq 1$ for all i
		$\boldsymbol{P}_{ij} \in \{0, 1\}$.
Multimodal network alignment (Chapter 4)	$\underset{\boldsymbol{P}}{\text{maximize}}$	$\sum_{k=1}^{m} \sum_{ij} \sum_{rs} \boldsymbol{A}_{ij}^{(k)} \boldsymbol{B}_{rs}^{(k)} \boldsymbol{P}_{ir} \boldsymbol{P}_{js}$
	subject to	$\sum_i \boldsymbol{P}_{ij} \leq 1$ for all j
		$\sum_j \boldsymbol{P}_{ij} \leq 1$ for all i
		$\boldsymbol{P}_{ij} \in \{0, 1\}$.
Multiple network alignment (Chapter 6)	$\underset{\underline{\boldsymbol{X}}}{\text{maximize}}$	$\sum_{i_1, i_2, \ldots, i_k} \sum_{j_1, j_2, \ldots, j_k} \boldsymbol{A}_{i_1 j_1}^{(1)} \boldsymbol{A}_{i_2 j_2}^{(2)} \ldots \boldsymbol{A}_{i_k j_k}^{(k)} \underline{X}_{i_1, i_2, \ldots, i_k} \underline{X}_{j_1, j_2, \ldots, j_k}$
	subject to	$\sum_{i_2, \ldots, i_k} \underline{\boldsymbol{X}}_{i_1, i_2, \ldots, i_k} \leq 1$ for all i_1
		$\sum_{i_1, i_3, \ldots, i_k} \underline{\boldsymbol{X}}_{i_1, i_2, \ldots, i_k} \leq 1$ for all i_2
		$\ldots$
		$\sum_{i_1, i_2, \ldots, i_{k-1}} \underline{\boldsymbol{X}}_{i_1, i_2, \ldots, i_k} \leq 1$ for all i_k
		$\underline{\boldsymbol{X}}_{i_1, i_2, \ldots, i_k} \in \{0, 1\}$.

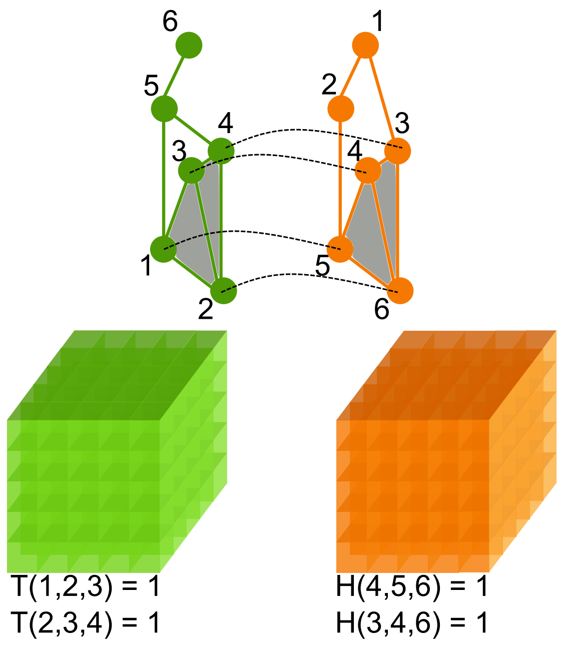


Figure 7.1. The TAME algorithm solves the network alignment problem while aiming at maximizing the number of triangles overlapped when the aligned graphs are overlayed on top of each other. It does so by forming a tensor for each graph that encodes the triangles in the graph. For instance, in this toy example $\underline{\boldsymbol{T}}[1,2,3]$ because nodes 1, 2, and 3 form a triangle in the green graph.

7.2 TAME is another algorithm that fits our pipeline

TAME by Mohammadi et al. (2016) introduces an innovative definition on the network alignment problem, and that is, maximizing the number of triangles to be aligned (in contrast to classic network alignment algorithms that maximize the number of edges aligned). Figure 7.2 illustrates the key goal of the TAME algorithm. We have new theory that supports that in TAME as well, when the starting prior knowledge is uniform (rank-1), the output similarity matrix is low rank as well. We briefly go through an overview of why this holds. We use the notation used in Table 7.1 to restate the TAME optimization problem.

$$\begin{array}{ll} \underset{\boldsymbol{P}}{\text{maximize}} & \sum_{ij}\sum_{rs}\sum_{uv} \underline{\boldsymbol{T}}_{iru}\underline{\boldsymbol{H}}_{jsv}\boldsymbol{P}_{ij}\boldsymbol{P}_{rs}\boldsymbol{P}_{uv} \\ \text{subject to} & \sum_i \boldsymbol{P}_{ij} \leq 1 \text{ for all } j \\ & \sum_j \boldsymbol{P}_{ij} \leq 1 \text{ for all } i \\ & \boldsymbol{P}_{ij} \in \{0,1\}. \end{array}$$

where $\underline{\boldsymbol{T}}$ and $\underline{\boldsymbol{H}}$ are the tensors that encode the triangles in each of the graphs. Relaxing this problem to allow $\boldsymbol{P}$ to be a similarity matrix translates to an eigenvector problem on the Kronecker product of these tensors. That is, the goal becomes to find $\mathbf{x}$ such that

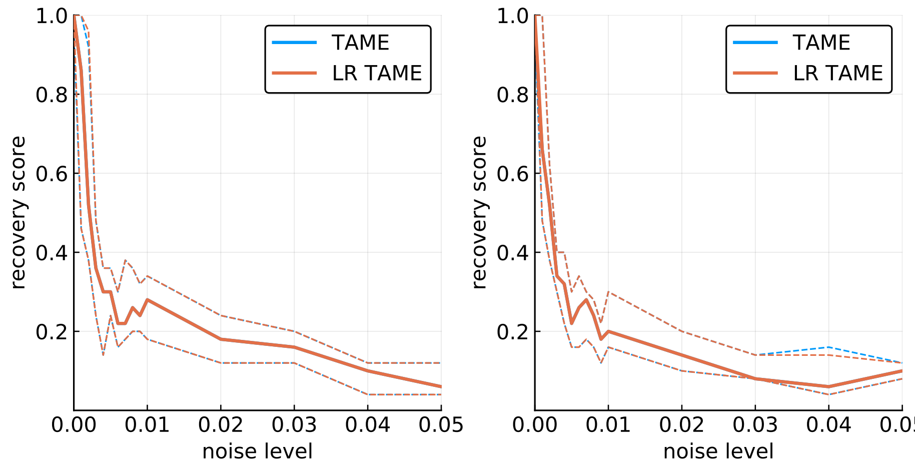
$$\begin{aligned} \text{maximize} \quad & (\underline{\boldsymbol{H}} \otimes \underline{\boldsymbol{T}})\mathbf{x}^3 \\ \text{subject to} \quad & \|\mathbf{x}\| = 1. \end{aligned}$$

A standard way to solve this eigenvector problem on tensors is via the *SS-HOPM* method (Kolda and Mayo, 2014). The main iteration of the SS-HOPM method is similar to the power method on matrices, with a regularization factor β,

$$\begin{aligned} \hat{\mathbf{x}}_{k+1} &= (\underline{\boldsymbol{H}} \otimes \underline{\boldsymbol{T}})\mathbf{x}_k^2 + \beta \mathbf{x}_k \\ \mathbf{x}_{k+1} &= \frac{\hat{\mathbf{x}}_{k+1}}{\|\hat{\mathbf{x}}_{k+1}\|} \end{aligned}$$

Some of our recent work on tensors shows that the eigenvector of $\underline{\boldsymbol{H}} \otimes \underline{\boldsymbol{T}}$ associated with the largest eigenvalue is rank-1, and in fact is, the Kronecker product of the dominant eigenvectors of $\underline{\boldsymbol{H}}$ and $\underline{\boldsymbol{T}}$. Using the SS-HOPM method with $\beta = 0$ and combining it with our recent theory allows us to deduce that a low rank version of TAME is possible and that the similarity matrix is in fact low rank. Further explorations with $\beta \neq 0$ are possible too. For instance, Mohammadi et al. (2016) recommend running TAME with three iterations only, which can be theoretically shown to have a rank at most 10. We have preliminary work on the low rank version of TAME with $\beta = 0$, and figure 7.2 shows these results.

In this experiment, we run 50 random experiments where we generate instances of 500 node graphs of Erdős-Rényi graphs, and then modify the current instance by either randomly removing edges only or by adding and removing randomly. Then, we run the TAME algorithm with $\beta = 0$ and the low rank formulation which states that the eigenvector of $\underline{\boldsymbol{H}} \otimes \underline{\boldsymbol{T}}$ is the kronecker product of the eigenvectors of $\underline{\boldsymbol{H}}$ and $\underline{\boldsymbol{T}}$ when $\beta = 0$. Then, we measure the recovery results since we have the groundtruth alignment. The results are shown in figure 7.2. The TAME and Low-Rank TAME algorithms achieves almost the same exact results, but the low rank version runs



Remove edges only *Add and remove edges*

Figure 7.2. As we align pairs of 500 node graphs of Erdős-Rényi, the low rank version of TAME achieves the same results in terms of node recovery.

considerably faster (the low rank version runs in less than one second whereas the TAME algorithm requires around 400 seconds).

7.3 New low rank formulations

One more observation to point out is that three state of the art algorithms for network alignment mentioned in this thesis—IsoRank (Singh et al., 2008), EigenAlign (Feizi et al., 2016), and TAME (Mohammadi et al., 2016)—produce a similarity scoring matrix that is provably low rank when the starting prior knowledge matrix is uniform. This is independently interesting because each row or column in the low rank factors can be interepreted as a *signature* or *embedding* of nodes in the graphs, and these signatures often represent a diffusion procedure on the graphs independently. This property, coupled with our class of algorithms to perform low rank matching can open the space for even more network alignment algorithms that rely on using signatures of nodes from both graphs (or equivalently, finding a similarity matrix that is low rank). This translates to a network alignment problem, where the binary matrix $\boldsymbol{P}$ is relaxed and restricted to be low rank, that can be stated as:

$$\begin{array}{ll} \underset{\boldsymbol{W}}{\text{maximize}} & \sum_{ij}(\boldsymbol{AXB})_{ij}\boldsymbol{X}_{ij} \\ \text{subject to} & \boldsymbol{X} = \boldsymbol{U_A}\boldsymbol{W}\boldsymbol{V_B}. \end{array}$$

where $\boldsymbol{U_A}$ and $\boldsymbol{V_B}$ are embeddings of nodes in each of the graphs.

7.4 Matching

In addition to our low rank network alignment methods, we introduced the idea of incorporating the several matchings we obtain while solving the low rank matching problem with existing state of the art methods that are highly effective when a prior knowledge is given (such as Klau's method Natalie (Klau, 2009)). Also, we saw in this thesis (Chapters 4 and 6) that adapting existing state of the art network alignment algorithms on multimodal networks or multiple networks can often produce worse results. We demonstrated this via a set of synthetic experiments where adapted standard network alignment algorithms to multimodal network alignment or multiple network alignment degrades in quality as the modes or networks present increases. This supported the need for solving the multimodal network alignment problem and the multiple network alignment problem via new methods that are specifically designed for these tasks.

In the context of matching, we presented a number of matching algorithms with theoretically grounded approximation bounds and our bounds were empirically very close to 1. These methods can be used beyond network alignment scores, whenever a low rank structure in the matrix is present. Since our simple low rank methods produced high quality matchings, we didn't dive into more detailed algorithms. Furthermore, bipartite matching algorithms such as the Path Growing Algorithm (Vinkemeier and Hougardy, 2005) or the greedy method (which both achieve a theoretical 2-approximation) can be adapted to use linear storage as well, but both would require a runtime larger than our low rank methods ($\mathcal{O}(n^2k)$ for PGA when the rank is k and the number of nodes is $\mathcal{O}(n)$, and $\mathcal{O}(n^3$ for the greedy method). Nevertheless, these methods do not immediately extend to the k-dimensional matching scenario.

7.5 To end

To end, the work presented in this thesis was predominantly enabled by our low rank techniques that were fast and scalable. And thus, our new low rank methods for bipartite matching and k-dimensional matching opens the space for further explorations of low rank structures in the network alignment problem. In this thesis, we introduced the first formulation and solution for multimodal network alignment and introduced the first multiple network alignment algorithm that can scale to 100s of networks with 1000s of nodes, when the previous state of the art could only scale to up to five networks.

REFERENCES

Altschul, S. F., W. Gish, W. Miller, E. W. Myers, and D. J. Lipman (1990). Basic local alignment search tool. *Journal of Molecular Biology 215*(3), 403 – 410.

Atias, N. and R. Sharan (2012, May). Comparative analysis of protein networks: hard problems, practical solutions. *Commun. ACM 55*(5), 88–97.

Battiston, F., V. Nicosia, and V. Latora (2014). Structural measures for multiplex networks. *Phys. Rev. E 89*, 032804.

Bayati, M., D. F. Gleich, A. Saberi, and Y. Wang (2013). Message-passing algorithms for sparse network alignment. *ACM Trans. Knowl. Discov. Data 7*(1), 3:1–3:31.

Burkard, R., M. Dell'Amico, and S. Martello (2012). *Assignment Problems* (Revised ed.). SIAM.

Cardillo, A., J. Gómez-Gardeñes, M. Zanin, M. Romance, D. Papo, F. d. Pozo, and S. Boccaletti (2013, 02). Emergence of network features from multiplexity. *Sci. Rep. 3*, 1344 EP.

Conte, D., P. Foggia, C. Sansone, and M. Vento (2004). Third years of graph matching in pattern recognition.

Conte, D., P. P. Foggia, C. Sansone, and M. Vento (2004, May). Thirty years of graph matching in pattern recognition. *Int. J. Pattern Recogn. 18*(3), 265–298.

Draisma, J. and J. Kuttler (2014, January). Bounded-rank tensors are defined in bounded degree. *Duke Mathematical Journal 163*(1), 35–63.

Esfandiar, P., F. Bonchi, D. F. Gleich, C. Greif, L. V. S. Lakshmanan, and B.-W. On (2010). *Fast Katz and Commuters: Efficient Estimation of Social Relatedness in Large Networks*, pp. 132–145. Berlin, Heidelberg: Springer Berlin Heidelberg.

Feizi, S., G. Quon, M. R. Mendoza, M. Médard, M. Kellis, and A. Jadbabaie (2016). Spectral alignment of networks. *arXiv cs.DS*, 1602.04181.

Fraikin, C. and P. van Dooren (2007). Graph matching with type constraints on nodes and edges. Number 07071 in Dagstuhl Proceedings.

Friedland, S. and V. Tammali (2015). Low-rank approximation of tensors. In *Numerical Algebra, Matrix Theory, Differential-Algebraic Equations and Control Theory*, pp. 377–411. Springer.

Gligorijevic, V., N. Malod-Dognin, and N. Przulj (2016). Fuse: multiple network alignment via data fusion. *Bioinformatics 32*(8), 1195–1203.

Golub, G. H. and C. van Loan (2013). *Matrix Computations* (4th ed.). Johns Hopkins University Press.

He, G., J. Liu, and C. Zhao (2000). Approximation algorithms for some graph partitioning problems. *Journal of Graph Algorithms and Applications 4*(2), 1–11.

Hu, W., Y. Qu, and G. Cheng (2008, October). Matching large ontologies: A divide-and-conquer approach. *Data Knowl. Eng. 67*(1), 140–160.

Kalecky, K. and Y.-R. Cho (2018, 06). PrimAlign: PageRank-inspired Markovian alignment for large biological networks. *Bioinformatics 34*(13), i537–i546.

Kann, V. (1991). Maximum bounded 3-dimensional matching is max snp-complete. *Information Processing Letters 37*(1), 27 – 35.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, pp. 85–103.

Kelley, B. P., B. Yuan, F. Lewitter, R. Sharan, B. R. Stockwell, and T. Ideker (2004). PathBLAST: a tool for alignment of protein interaction networks. *Nucl. Acids Res. 32*, W83–88.

Klau, G. (2009). A new graph-based method for pairwise global network alignment. *BMC Bioinfor. 10*(Suppl 1), S59.

Knossow, D., A. Sharma, D. Mateus, and R. Horaud (2009). Inexact matching of large and sparse graphs using laplacian eigenvectors. Number 5534 in LNCS, pp. 144–153.

Kolda, T. and B. Bader (2009). Tensor decompositions and applications. *SIAM Review 51*(3), 455–500.

Kolda, T. G. and J. R. Mayo (2014). An adaptive shifted power method for computing generalized tensor eigenpairs. *SIAM Journal on Matrix Analysis and Applications 35*(4), 1563–1581.

Kollias, G., S. Mohammadi, and A. Grama (2012, December). Network similarity decomposition (nsd): A fast and scalable approach to network alignment. *IEEE Trans. on Knowl. and Data Eng. 24*(12), 2232–2243.

Kollias, G., M. Sathe, O. Schenk, and A. Grama (2014). Fast parallel algorithms for graph similarity and matching. *J. Parallel Dist. Comp. 74*(5), 2400 – 2410.

Korula, N. and S. Lattanzi (2014, January). An efficient reconciliation algorithm for social networks. *Proc. VLDB Endow. 7*(5), 377–388.

Koutra, D., H. Tong, and D. Lubensky (2013, Dec). Big-align: Fast bipartite graph alignment. In *2013 IEEE 13th International Conference on Data Mining*, pp. 389–398.

Kuchaiev, O., T. Milenković, V. Memišević, W. Hayes, and N. Pržulj (2010). Topological network alignment uncovers biological function and phylogeny. *J. R. Soc. Interface*.

Kuchaiev, O. and N. Pržulj (2011). Integrative network alignment reveals large regions of global network similarity in yeast and human. *Bioinformatics 27*(10), 1390–1396.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly 2*(12), 83–97.

Leskovec, J., J. Kleinberg, and C. Faloutsos (2005). Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, New York, NY, USA, pp. 177–187. ACM.

Leskovec, J., K. J. Lang, A. Dasgupta, and M. W. Mahoney (2008). Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *CoRR abs/0810.1355*.

Li, Z., R. Petegrosso, S. Smith, D. Sterling, G. Karypis, and R. Kuang (2018). Scalable label propagation for multi-relational learning on tensor product graph. *CoRR abs/1802.07379*.

Liao, C.-S., K. Lu, M. Baym, R. Singh, and B. Berger (2009). Isorankn: spectral methods for global alignment of multiple protein networks. *Bioinformatics 25*(12), i253–i258.

Liben-Nowell, D. and J. Kleinberg (2007). The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology 58*(7).

Liu, X. and S.-H. Teng (2016, March). Maximum bipartite matchings with low rank data. *Theor. Comput. Sci. 621*(C), 82–91.

Lü, L. and T. Zhou (2011). Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications 390*(6).

Malmi, E., S. Chawla, and A. Gionis (2017, September). Lagrangian relaxations for multiple network alignment. *Data Min. Knowl. Discov. 31*(5), 1331–1358.

Malmi, E., A. Gionis, and E. Terzi (2017). Active network alignment: A matching-based approach. In *Proceedings of the International Conference on Information and Knowledge Management*, pp. In presss.

Malod-Dognin, N. and N. Pržulj (2015). L-graal: Lagrangian graphlet-based network aligner. *Bioinformatics 31*(13), 2182–2189.

Memisevic, V. and N. Pržulj (2012, 07). C-graal: Common-neighbors-based global graph alignment of biological networks. *Integrative biology : quantitative biosciences from nano to macro 4*(7), 734–43.

Meng, L., A. Striegel, and T. Milenkovi (2016). Local versus global biological network alignment. *Bioinformatics. 32*(20), 3155–3164.

Milano, M., P. H. Guzzi, and M. Cannataro (2017). Using multi network alignment for analysis of connectomes. *Procedia Computer Science 108*, 1155 – 1164. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

Milano, M., O. Tymofiyeva, D. Xu, C. Hess, M. Cannataro, and P. H. Guzzi (2016). Using network alignment for analysis of connectomes: Experiences from a clinical dataset. In *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pp. 649–656. ACM.

Milenkovic, T., W. L. Ng, W. Hayes, and N. Pržulj (2010, 06). Optimal network alignment with graphlet degree vectors. *Cancer Informatics 9*, 121–37.

Mohammadi, S., D. F. Gleich, T. G. Kolda, and A. Grama (2016). Triangular alignment (tame): A tensor-based approach for higher-order network alignment. *IEEE/ACM transactions on computational biology and bioinformatics Online*, 1–14.

Mucha, P. J., T. Richardson, K. Macon, M. A. Porter, and J.-P. Onnela (2010). Community structure in time-dependent, multiscale, and multiplex networks. *Science 328*(5980), 876–878.

Nassar, H. and D. F. Gleich (2017). Multimodal network alignment. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pp. 615–623. SIAM.

Nassar, H., G. Kollias, A. Grama, and D. F. Gleich (2018). Low rank methods for multiple network alignment. *CoRR abs/1809.08198*.

Nassar, H., N. Veldt, S. Mohammadi, A. Grama, and D. F. Gleich (2018). *Low Rank Spectral Network Alignment*, pp. 619–628. WWW '18.

Ni, J., H. Tong, W. Fan, and X. Zhang (2014). Inside the atoms: Ranking on a network of networks. In *KDD*, pp. 1356–1365.

Nicosia, V. and V. Latora (2015, Sep). Measuring and modeling correlations in multiplex networks. *Phys. Rev. E 92*, 032805.

Ovsjanikov, M., Q. Mérigot, F. Mémoli, and L. Guibas (2010). One point isometric matching with the heat kernel. *Computer Graphics Forum 29*(5), 1555–1564.

Patro, R. and C. Kingsford (2012). Global network alignment using multiscale spectral signatures. *Bioinformatics 28*(23), 3105–3114.

Schellewald, C. and C. Schnörr (2005). Probabilistic subgraph matching based on convex relaxation. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, Berlin, Heidelberg, pp. 171–186. Springer Berlin / Heidelberg.

Simoncini, V. (2016). Computational methods for linear matrix equations. *SIAM Review 58*(3), 377–441.

Singh, R., J. Xu, and B. Berger (2008). Global alignment of multiple protein interaction networks with application to functional orthology detection. *PNAS 105*(35), 12763–12768.

Szell, M., R. Lambiotte, and S. Thurner (2010). Multirelational organization of large-scale social networks in an online world. *PNAS 107*(31), 13636–13641.

Vijayan, V. and T. Milenković (2017). Multiple network alignment via multimagna++. *IEEE/ACM Transactions on Computational Biology and Bioinformatics PP*(99), 1–1.

Vijayan, V., V. Saraph, and T. Milenković (2015). Magna++: Maximizing accuracy in global network alignment via both node and edge conservation. *Bioinformatics 31*(14), 2409–2411.

Vinkemeier, D. E. D. and S. Hougardy (2005, July). A linear-time approximation algorithm for weighted matchings in graphs. *ACM Trans. Algorithms 1*(1), 107–122.