Low Rank Spectral Network Alignment

Huda Nassar
Computer Science Department,
Purdue University
West Lafayette, Indiana, USA
hnassar@purdue.edu

Nate Veldt
Mathematics Department,
Purdue University
West Lafayette, Indiana, USA
lveldt@purdue.edu

Shahin Mohammadi
CSAIL MIT &
Broad Institute
Cambridge, Massachusetts, USA
mohammadi@broadinstitute.org

Ananth Grama
Computer Science Department,
Purdue University
West Lafayette, Indiana, USA
ayg@cs.purdue.edu

David F. Gleich
Computer Science Department,
Purdue University
West Lafayette, Indiana, USA
dgleich@purdue.edu

ABSTRACT

Network alignment or graph matching is the classic problem of finding matching vertices between two graphs with applications in network de-anonymization and bioinformatics. There exist a wide variety of algorithms for it, but a challenging scenario for all of the algorithms is aligning two networks without any information about which nodes might be good matches. In this case, the vast majority of principled algorithms demand quadratic memory in the size of the graphs. We show that one such method—the recently proposed and theoretically grounded EigenAlign algorithm—admits a novel implementation which requires memory that is linear in the size of the graphs. 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.

KEYWORDS

network alignment; graph matching; low rank matrix; low-rank bipartite matching

ACM Reference Format:
Huda Nassar, Nate Veldt, Shahin Mohammadi, Ananth Grama, and David F. Gleich. 2018. Low Rank Spectral Network Alignment. In *WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3178876.3186128

WWW 2018, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5639-8/18/04.
https://doi.org/10.1145/3178876.3186128

1 INTRODUCTION AND MOTIVATION

Network alignment is the problem of pairing nodes across two different graphs in a way that preserves edge structure and highlights similarities between the networks. The node pairings can either be one-to-one or many-to-many. While the methods we propose are amenable to both settings with some modification, we focus on the one-to-one case as it has the most extensive literature. Applications of network alignment include (i) finding similar nodes in social networks, which uncovers information about one or both of the paired nodes, and can help with tailoring advertisements and suggesting activities for similar users in a network; (ii) social-network de-anonymization [10]; and (iii) pattern matching in graphs [3]. One very popular example of this problem is the alignment of protein-protein interaction networks in biology [6, 14, 27]. 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.

There are two major approaches to network alignment problems [1]: local network alignment, where the goal is to find local regions of the graph that are similar to any given node, and global network alignment, where the goal is to understand how two large graphs would align to each other. 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. Here, we focus on global alignment with one-to-one matches between the two graphs.

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 [2]. 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 instance, methods such as the Lagrangian relaxation method of Klau et al. [7] require at least quadratic memory. There do exist

memory-scalable heuristics for solving network alignment problems with no prior, including the GHOST procedure of Patro et al. [24], or the GRAAL algorithm of Kuchaieve et al [12] and its variants. However, these 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).

One principled approach that avoids the quadratic memory requirement is the Network Similarity Decomposition (NSD) [8, 9, 22], which provides a useful low-rank decomposition of a specific similarity matrix based on the IsoRank method [27]. This method enables alignments to be computed between extremely large networks. However, there have been many improvements to network alignment methods since the publication of IsoRank.

A recent innovation is a method based on eigenvectors called EigenAlign. The EigenAlign method 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 [5]. 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 [27]. In contrast, a 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.*

In our manuscript, 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 the low-rank information provided by our new method is that there are only a few ideas on how to use the low-rank structure of the similarity scores in the matching step [15, 22]. We contribute a new analysis of a simple idea to use the low-rank structure that gives a computable a-posteriori approximation guarantee. In practice, this approximation guarantee is extremely good: around 1.1. Such a procedure should enable further low-rank applications beyond just network alignment.

Our contributions.

- An explicit expression for the solution of the EigenAlign eigenvector as a low-rank matrix (Theorem 3.1).
- 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 4.2). In practice, these approximation guarantees are better than 1.1 for our experiments (Figure 7). This improves recent work in [22], which gave a simple k-approximation algorithm, where k is the rank.
- 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 5.1), and our methods are more scalable.
- A demonstration that our low-rank methods can be combined with existing network alignment methods to yield better quality results (Section 5.2).
- 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 10).

2 NETWORK ALIGNMENT FORMULATIONS AND CURRENT TECHNIQUES

We now review the state of network alignment algorithms and our specific setting and objective. A helpful illustration is shown in Figure 1.

2.1 The canonical network alignment problem

For the network alignment problem, we are given two graphs G_A and G_B with adjacency matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. The goal is to produce a one-to-one mapping between nodes of G_A and G_B that preserves topological similarities between the networks [3]. In some cases we additionally receive information about which nodes in one network can be paired with nodes in the other. This additional information is presented in the form of a bipartite graph whose edge weights are stored in a matrix $\boldsymbol{L}$; if $\boldsymbol{L}_{uv} > 0$, this indicates outside evidence that node u in G_A should be matched to node v in G_B. We call this outside evidence *a prior* on the alignment. When a prior is present, the prior and topological information are taken together to determine an alignment.

More formally, we seek a binary matrix $\boldsymbol{P}$ that encodes a matching between the nodes of the networks and maximizes one of a few possible objective functions discussed below. The matrix $\boldsymbol{P}$ encodes a matching when it satisfies the constraints

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

The inequality constraints guarantee that a node in the first network is only matched with one or zero nodes in the other network.

2.2 Objective functions for network alignment

The classic formulation of the problem seeks a matrix $\boldsymbol{P}$ that maximizes the number of overlapping edges between G_A and G_B, i.e. the number of adjacent node pairs (i_A, j_A) in G_A that are mapped to an adjacent node pair (i'_B, j'_B) in G_B. This results in the following integer quadratic program:

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

Recent variations of this objective include an extension to overlapping triangles [21], an extension that combines edge overlapping with prior similarity scores [2, 27], as well as an extension specific to bipartite graphs [11].

2.3 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.

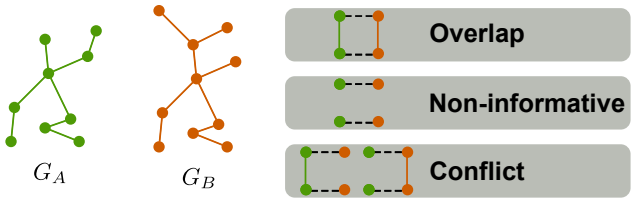

Figure 1: Our setup for network alignment follows Feizi et. al. [5], where we seek to align two networks without any other metadata. Possible alignments between pairs of any nodes (i,j) in G_A and (i',j') in G_B are scored based on one of three cases and assembled into a massive, but highly structured, alignment matrix M.

edges in G_A that are mapped to non-edges in G_B or vice versa. Neither do these objective functions consider the case where non-edges in G_A are mapped to non-edges in G_B. The first problem was recognized in [25] which proposed an SDP-based method to minimize the number of conflicting matches. More recently, the EigenAlign objective [5] included explicit terms for these three cases: overlaps, non-informative matches, and conflicts, see Figure 1. The alignment score corresponding to $\boldsymbol{P}$ in this case is

$$\text{AlignmentScore}(\boldsymbol{P}) = s_O(\#\text{ overlaps}) + s_N(\#\text{ non-informatives}) + s_C(\#\text{ conflicts}) \quad (2)$$

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_A and all pairs i'_B, j'_B in G_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_A and G_B are undirected. (There are directed extensions discussed in [5], but we don't consider them here.)

Maximizing the alignment score (2) is then equivalent to the following quadratic assignment problem:

$$\begin{aligned} \underset{\mathbf{y}}{\text{maximize}} \quad & \mathbf{y}^T \boldsymbol{M} \mathbf{y} \\ \text{subject to} \quad & y_i \in \{0,1\} \\ & \textstyle\sum_u y[u,v] \le 1 \text{ for all } v \in V_B \\ & \textstyle\sum_v y[u,v] \le 1 \text{ for all } u \in V_A \end{aligned} \quad (3)$$

where V_A and V_B are the node sets of G_A and G_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 (1).

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_A and G_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_A and G_B.*

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

Our contribution. 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.

2.4 Summary of other techniques

Our work shares a number of similarities with the Network Similarity Decomposition (NSD) [8], a technique based on a low-rank factorization of a different similarity matrix, the matrix used by the IsoRank algorithm [27]. The authors of [8] show that this decomposition can be obtained by performing calculations separately on the two graphs, which significantly speeds up the calculation of similarity scores between nodes. Another procedure designed for aligning networks without prior information is the the Graph Alignment tool (GRAAL) [12]. 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 [13, 17, 18, 20]. The seed-and-extend alignment procedure was also employed by the GHOST algorithm [24], which computes topological similarity scores based on a novel spectral signature for each node. Recently, [21] 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.

Alternative approaches to improve network alignment include active methods that allow users to select matches from a host of potential near equal matches [16].

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 $\boldsymbol{X}$ [5]. Because of the Kronecker structure in $\boldsymbol{M}$, this can equivalently be formulated directly as the matrix $\boldsymbol{X}$ that satisfies:

$$\begin{array}{ll}\underset{\boldsymbol{X}}{\text{maximize}} & \boldsymbol{X} \bullet (c_1\boldsymbol{A}\boldsymbol{X}\boldsymbol{B}^T + c_2\boldsymbol{A}\boldsymbol{X}\boldsymbol{E}^T + c_2\boldsymbol{E}\boldsymbol{X}\boldsymbol{B}^T + c_3\boldsymbol{E}\boldsymbol{X}\boldsymbol{E}^T) \\ \text{subject to} & \|\boldsymbol{X}\|_F = 1, \boldsymbol{X} \in \mathbb{R}^{n_A \times n_B}.\end{array} \tag{4}$$

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.

3.1 A Four Factor Low-Rank Decomposition

In the matrix form of problem (4), one step of the power method corresponds to the iteration:

$$\boldsymbol{X}_{k+1} = c_1\boldsymbol{A}\boldsymbol{X}_k\boldsymbol{B}^T + c_2\boldsymbol{A}\boldsymbol{X}_k\boldsymbol{E}^T + c_2\boldsymbol{E}\boldsymbol{X}_k\boldsymbol{B}^T + c_3\boldsymbol{E}\boldsymbol{X}_k\boldsymbol{E}^T. \tag{5}$$

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

$$\boldsymbol{X}_k = \boldsymbol{U}_k\boldsymbol{V}_k^T \tag{6}$$

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) and (6) holds up to k and we will prove the equivalence for $k+1$. We begin with equation (5) and plug in the decomposition of $\boldsymbol{X}_k$ from (6):

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

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:

$$\begin{aligned}\boldsymbol{S}_k &= [\boldsymbol{A}^k\mathbf{u} \mid \boldsymbol{A}^{k-1}\mathbf{e} \mid \ldots \mid \boldsymbol{A}\mathbf{e} \mid \mathbf{e}] \\ \boldsymbol{R}_k &= [\boldsymbol{B}^k\mathbf{v} \mid \boldsymbol{B}^{k-1}\mathbf{e} \mid \ldots \mid \boldsymbol{B}\mathbf{e} \mid \mathbf{e}] \\ \boldsymbol{C}_k &= \begin{bmatrix} c_1\boldsymbol{C}_{k-1} & \mathbf{0} & c_2\boldsymbol{C}_{k-1} & \mathbf{0} \\ \mathbf{0}^T & c_2\mathbf{r}_k^T\boldsymbol{C}_{k-1} & \mathbf{0}^T & c_3\mathbf{r}_k^T\boldsymbol{C}_{k-1} \end{bmatrix} \\ \boldsymbol{T}_k &= \begin{bmatrix} \boldsymbol{T}_{k-1} & \boldsymbol{T}_{k-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0}^T & \mathbf{0}^T & \mathbf{h}_k^T\boldsymbol{T}_{k-1} & \mathbf{h}_k^T\boldsymbol{T}_{k-1} \end{bmatrix}.\end{aligned}$$

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:

$$\begin{aligned}\mathbf{r}_k &= [\ \mathbf{e}^T\boldsymbol{A}^{k-1}\mathbf{u} \quad \mathbf{e}^T\boldsymbol{A}^{k-2}\mathbf{e} \quad \cdots \quad \mathbf{e}^T\boldsymbol{A}^1\mathbf{e} \quad \mathbf{e}^T\boldsymbol{A}^0\mathbf{e}\ ]^T \\ \mathbf{h}_k &= [\ \mathbf{e}^T\boldsymbol{B}^{k-1}\mathbf{v} \quad \mathbf{e}^T\boldsymbol{B}^{k-2}\mathbf{e} \quad \cdots \quad \mathbf{e}^T\boldsymbol{B}^1\mathbf{e} \quad \mathbf{e}^T\boldsymbol{B}^0\mathbf{e}\ ]^T\end{aligned}$$

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,

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

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

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

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

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):

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

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

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

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 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) *permits the following low-rank factorization:*

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

where

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

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

$$\tilde{W}_k = D_u W_k D_v.$$

Here $\boldsymbol{D}_u$ is a $(k+1) \times (k+1)$ diagonal matrix with diagonal entries $(\|\boldsymbol{A}^k \mathbf{u}\|, \|\boldsymbol{A}^{k-1} \mathbf{e}\|, \ldots, \|\boldsymbol{A}\mathbf{e}\|, \|\mathbf{e}\|)$ 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 (3.1) are designed specifically to satisfy $S_k D_u^{-1} = \tilde{U}_k$, $R_k D_v^{-1} = \tilde{V}_k$, 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 Figure 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 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 Figure 2 accomplish this goal.

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 Figure 2. In this section, however, we proceed in terms of a general matrix Y with low rank factors $Y = UV^T$. The matrix Y represents the edge-weights of a bipartite graph, and so the max-weight matching problem is:

$$\begin{aligned} \underset{\boldsymbol{M}}{\text{maximize}} \quad & \boldsymbol{M} \bullet \boldsymbol{Y} \\ \text{subject to} \quad & M_{i,j} \in \{0, 1\} \\ & \textstyle\sum_i M_{i,j} \leq 1 \text{ for all } j, \sum_{ij} M_{i,j} \leq 1 \text{ for all } i, \end{aligned} \tag{7}$$

where $\bullet$ is the matrix inner-product (see (4)). 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.

4.1 Optimal Matching on a Rank 1 Matrix

We begin by considering optimal matchings for a rank-1 matrix $Y = \mathbf{u}\mathbf{v}^T$ 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 Y. In this case, define $\tilde{Y}$ to be the matrix obtained by copying Y and deleting all negative entries. To find the optimal matching of Y we would never pair elements giving a negative weight, so the optimal matching for $\tilde{Y}$ is the same as for 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,

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

where $\tilde{Y}_1 = \mathbf{u}_+ \mathbf{v}_+^T$ and $\tilde{Y}_2 = \mathbf{u}_- \mathbf{v}_-^T$. Let $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ be the optimal matching matrices for $\tilde{Y}_1$ and $\tilde{Y}_2$ 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 $\tilde{Y}$:

Lemma 4.1. *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 $\tilde{\boldsymbol{Y}}$. □

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 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,

$$\boldsymbol{M}^* \bullet \boldsymbol{Y} \leq \textstyle\sum_{i=1}^{k} \boldsymbol{M}_i \bullet \boldsymbol{Y}_i. \tag{8}$$

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

$$d_{i,j} = \frac{\boldsymbol{M}_i \bullet \boldsymbol{Y}_i}{\boldsymbol{M}_j \bullet \boldsymbol{Y}_i} \quad d_j = \max_i d_{i,j} \quad D = \min_j d_j \tag{9}$$

and let $j^* = \operatorname{argmin}_j d_j$, 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,

$$\frac{\boldsymbol{M}_i \bullet \boldsymbol{Y}_i}{\boldsymbol{M}_{j^*} \bullet \boldsymbol{Y}_i} = d_{i,j^*} \leq d_{j^*} = D \implies \boldsymbol{M}_i \bullet \boldsymbol{Y}_i \leq D(\boldsymbol{M}_{j^*} \bullet \boldsymbol{Y}_i) \tag{10}$$

By combining (8) and (10) we have the following result.

Theorem 4.2. *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 (Figure 3) runs in $O(k^2 n + kn \log n)$ where k is the rank, and $\boldsymbol{U}$ and $\boldsymbol{V}$ have $O(n)$ rows. The space requirement is $O(nk)$. In practice, the approximation factors D are less than 1.1 for our problems (see Figure 7). Figure 3 shows pseudocode to implement this matching algorithm.

```
Input: U, V such that Y = U V^T
Output: Approximate max-weight matching M, approx value D
Find optimal matching M_i for each rank-1 matrix (see §4.1)
Evaluate the weight of the matching v_i = M_i • Y_i
Compute d_{i,j} = v_i/(M_j • Y_i) for all i,j = {1,...,k}
Evaluate d_j = max_i d_{i,j}, D = minimum(d_j), j* = argmin(d_j)
return M = M_{j*}, D
```

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

4.3 Improved practical variations

Our method (Figure 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

sorted $\mathbf{u}$:	i_1	i_2	i_3	i_4	i_5	then we	(i_1,j_1) (i_1,j_2)
sorted $\mathbf{v}$:	j_1	j_2	j_3	j_4	j_5	add edges	(i_2,j_1) (i_2,j_2) (i_2,j_3)
							(i_3,j_2) ...

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 EXPERIMENTS

To evaluate our method, we first study the relationship between Low Rank EigenAlign and the original 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 4), (ii) our method performs the same over a variety of graph models (Figure 5), (iii) the method scales better (Figure 6), and (iv) the computed approximation bounds are better than 1.1 (Figure 7). We also compare against other scalable techniques in Figure 8, and see that our approach is the best. Next, we use a test-set of networks with known alignments from biology [28] to evaluate our algorithms (Section 5.2). Finally, we end

our experiments with a study on a collaboration network where we seek to align vertex neighborhoods (Section 5.3).

Our low-rank EigenAlign In all of these experiments, our low-rank techniques use the expanded matching with $c = 3$ (Section 4.3) and set the initial rank-1 factors to be all uniform: $\mathbf{v} = \mathbf{e}, \mathbf{u} = \mathbf{e}$. 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 [5] 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) until we reach an eigenvalue-eigenvector pair that achieves a residual value 10^{-12}. This usually occurs after a 15-20 iterations.

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* [5], where we want *large recovery values*. Recovery is between 0 and 1 and is defined

$$\text{recovery}(\boldsymbol{M}) = 1 - \tfrac{1}{2n}\|\boldsymbol{M} - \boldsymbol{M}_{\text{true}}\|_F. \tag{11}$$

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 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 $\boldsymbol{G}_B$ [5]. 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 the performance of our method as iterations vary. Figure 4 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.

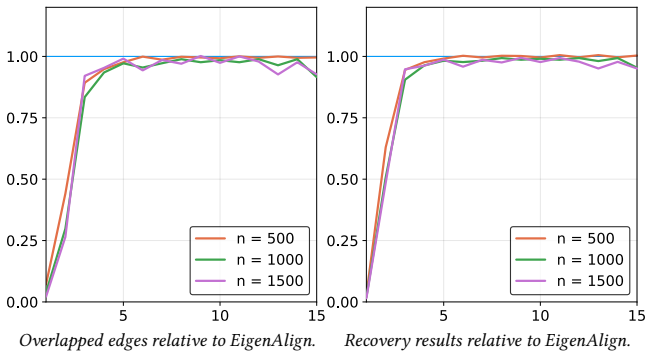

Figure 4: At left, the number of overlapped edges in the alignment computed by our low-rank method relative to EigenAlign's alignment. A value of 1.0 means that we get the same number as EigenAlign's solution. At right, the same ratio but with respect to the recovery. The recovery results stop improving after around 8 iterations, so we fix this value in the rest of our experiments.

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 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 6 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.

Our matching approximations are high quality. We also evaluate the effectiveness of our D-approximation computed in Section 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 value gives better theoretical bounds. Figure 7 shows these results. Our guaranteed approximation factors are always less than 1.1 when the low-rank factors arise from the problems in Figure 6. 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 [8] is another network alignment algorithm which solves the network alignment problem via low-rank factors. In the previous experiment, we saw that greedy

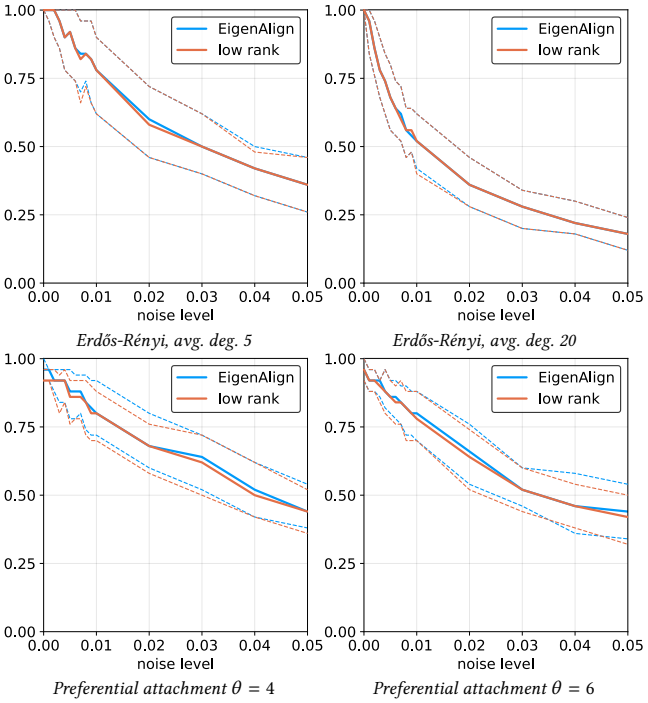

Figure 5: Thick lines are the median recovery fractions over 200 trials and dashed lines are the 20th and 80th percentiles. These figures show that there appears to be small and likely insignificant differences between the alignment quality of EigenAlign and our low rank method.

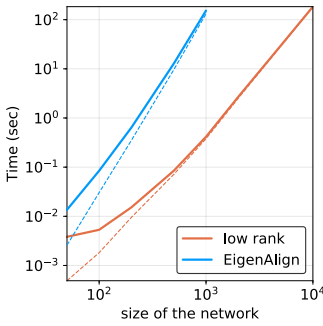

Figure 6: These show the average time over 10 trials. The dashed line is the time required for the matching step. The time required for EigenAlign is an order of magnitude larger than our low-rank formulation. Our low rank EigenAlign solves 10,000 node problems in about two minutes whereas EigenAlign requires the same amount of time to solve a 1000 node problem.

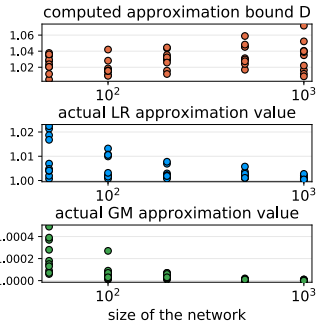

Figure 7: Over the experiments from Figure 6, the top figure shows the guaranteed D-approximation value computed by our algorithm. The bound appears to be strong and gives a better-than 1.1 approximation. The middle figure shows the true approximation value after solving the optimal matching using Low-rank EigenAlign (LR), and the bottom figure shows the true approximation value for a greedy matching (GM) strategy, which guarantees a 2-approximation.

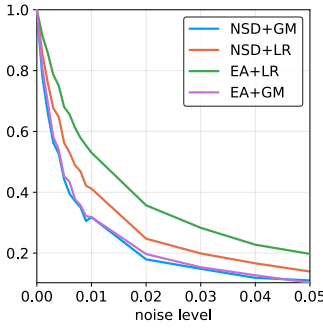

Figure 8: Recovery scores achieved by Low Rank EigenAlign and NSD [8]. We use a greedy matching (GM) and our low rank matching algorithm (LR) on the low rank factors of the similarity matrix from both algorithms.

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 8 and show that the low-rank EigenAlign strategy with our low-rank matching outperforms the other scalable alternatives.

5.2 Biological networks

The MultiMagna dataset is a test case in bioinformatics that involves network alignment [19, 28]. 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 9, we show results for aligning MultiMagna networks using low-rank EigenAlign, EigenAlign , belief propagation (BP) [2], and Klau's method [7] 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.

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

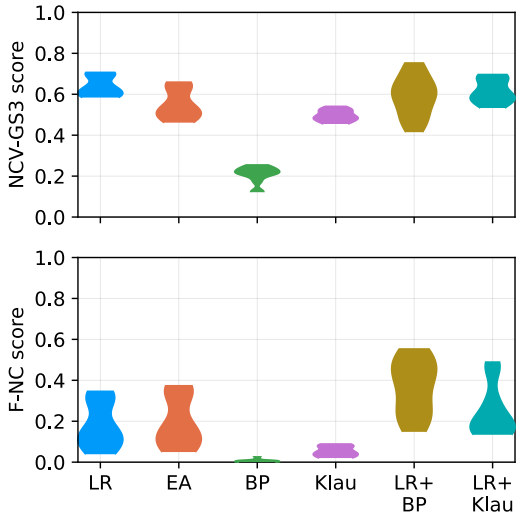

Figure 9: These are violin plots over the results of 15 problems. Klau and BP are strong algorithms for network alignment but they only perform well when given a sparsified set of possible matches from our expanded low-rank matchings (LR+BP, LR+Klau). Larger scores are better.

Table 1: Time required for methods on the MultiMagna data.

Algorithm	Time (sec)		
	min	median	max
LR	1.9553	2.1971	2.9173
EA	83.6777	96.9938	194.363
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

matching for this step. This picks a matching that also yields a good alignment. Our results are comparable with the results in [19], 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 1.

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 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

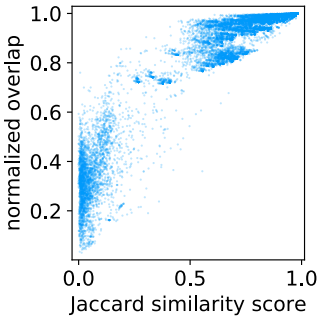

Figure 10: The edge overlap (normalized so the maximum value is 1.0) of alignments between vertex neighborhoods of nodes in the DBLP dataset. This shows that the Jaccard score of two connected nodes in the network is correlated to the overlap size.

align the vertex neighborhood of i to the vertex neighborhood of j using our technique.

We use the DBLP collaboration network from [4] 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 10. 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.

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 S_k, which is related to graph diffusions. There are results in computational geometry that prove rigorous results about using diffusions to align manifolds [23]. 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 multiterm matrix equations [26] of the form $C_1XD_1 + C_2XD_2 + \cdots = F$. These connections offer new possibilities to improve our methods.

ACKNOWLEDGEMENTS

The authors were supported by NSF CCF-1149756, IIS-1422918, IIS-1546488, CCF-0939370, DARPA SIMPLEX, and the Sloan Foundation.

REFERENCES

[1] Nir Atias and Roded Sharan. 2012. Comparative analysis of protein networks: hard problems, practical solutions. *Commun. ACM* 55, 5 (May 2012), 88–97. https://doi.org/10.1145/2160718.2160738

[2] Mohsen Bayati, David F. Gleich, Amin Saberi, and Ying Wang. 2013. Message-Passing Algorithms for Sparse Network Alignment. *ACM Trans. Knowl. Discov. Data* 7, 1, Article 3 (March 2013), 31 pages. https://doi.org/10.1145/2435209.2435212

[3] D. Conte, P. Foggia, C. Sansone, and M. Vento. 2004. Third Years of Graph Matching in Pattern Recognition. (2004), 265-298 pages. arXiv:http://www.worldscientific.com/doi/pdf/10.1142/S0218001404003228 http://www.worldscientific.com/doi/abs/10.1142/S0218001404003228

[4] Pooya Esfandiar, Francesco Bonchi, David F. Gleich, Chen Greif, Laks V. S. Lakshmanan, and Byung-Won On. 2010. *Fast Katz and Commuters: Efficient Estimation of Social Relatedness in Large Networks.* Springer Berlin Heidelberg, Berlin, Heidelberg, 132–145. https://doi.org/10.1007/978-3-642-18009-5_13

[5] Soheil Feizi, Gerald Quon, Mariana Recamonde Mendoza, Muriel Médard, Manolis Kellis, and Ali Jadbabaie. 2016. Spectral Alignment of Networks. *arXiv* cs.DS (2016), 1602.04181. http://arxiv.org/abs/1602.04181

[6] Brian P. Kelley, Bingbing Yuan, Fran Lewitter, Roded Sharan, Brent R. Stockwell, and Trey Ideker. 2004. PathBLAST: a tool for alignment of protein interaction networks. *Nucl. Acids Res.* 32 (2004), W83–88. https://doi.org/10.1093/nar/gkh411

[7] Gunnar W Klau. 2009. A new graph-based method for pairwise global network alignment. *BMC Bioinformatics* 10, 1 (2009), S59.

[8] Giorgos Kollias, Shahin Mohammadi, and Ananth Grama. 2012. Network Similarity Decomposition (NSD): A Fast and Scalable Approach to Network Alignment. *IEEE Trans. on Knowl. and Data Eng.* 24, 12 (December 2012), 2232–2243. https://doi.org/10.1109/TKDE.2011.174

[9] Giorgos Kollias, Madan Sathe, Olaf Schenk, and Ananth Grama. 2014. Fast parallel algorithms for graph similarity and matching. *J. Parallel and Distrib. Comput.* 74, 5 (2014), 2400 – 2410. https://doi.org/10.1016/j.jpdc.2013.12.010

[10] Nitish Korula and Silvio Lattanzi. 2014. An Efficient Reconciliation Algorithm for Social Networks. *Proc. VLDB Endow.* 7, 5 (January 2014), 377–388. https://doi.org/10.14778/2732269.2732274

[11] D. Koutra, H. Tong, and D. Lubensky. 2013. BIG-ALIGN: Fast Bipartite Graph Alignment. In *2013 IEEE 13th International Conference on Data Mining*. 389–398. https://doi.org/10.1109/ICDM.2013.152

[12] Oleksii Kuchaiev, Tijana Milenković, Vesna Memišević, Wayne Hayes, and Nataša Pržulj. 2010. Topological network alignment uncovers biological function and phylogeny. *Journal of The Royal Society Interface* 7, 50 (2010), 1341–1354. https://doi.org/10.1098/rsif.2010.0063

[13] Oleksii Kuchaiev and Nataša Pržulj. 2011. Integrative network alignment reveals large regions of global network similarity in yeast and human. *Bioinformatics* 27, 10 (2011), 1390–1396. https://doi.org/10.1093/bioinformatics/btr127

[14] Chung-Shou Liao, Kanghao Lu, Michael Baym, Rohit Singh, and Bonnie Berger. 2009. IsoRankN: spectral methods for global alignment of multiple protein networks. *Bioinformatics* 25, 12 (2009), i253–i258. https://doi.org/10.1093/bioinformatics/btp203

[15] Xingwu Liu and Shang-Hua Teng. 2016. Maximum Bipartite Matchings with Low Rank Data. *Theor. Comput. Sci.* 621, C (March 2016), 82–91. https://doi.org/10.1016/j.tcs.2016.01.033

[16] Eric Malmi, Aristides Gionis, and Evimaria Terzi. 2017. Active Network Alignment: A Matching-Based Approach. In *Proceedings of the International Conference on Information and Knowledge Management*. In presss. https://arxiv.org/abs/1610.05516

[17] Noël Malod-Dognin and Nataša Pržulj. 2015. L-GRAAL: Lagrangian graphlet-based network aligner. *Bioinformatics* 31, 13 (2015), 2182–2189. https://doi.org/10.1093/bioinformatics/btv130

[18] Vesna Memisevic and Nataša Pržulj. 2012. C-GRAAL: Common-neighbors-based global GRAph ALignment of biological networks. *Integrative biology : quantitative biosciences from nano to macro* 4, 7 (07 2012), 734–43.

[19] Lei Meng, Aaron Striegel, and Tijana MilenkoviÄĞ. 2016. Local versus global biological network alignment. *Bioinformatics* 32, 20 (2016), 3155–3164. https://doi.org/10.1093/bioinformatics/btw348

[20] Tijana Milenkovic, Weng Leong Ng, Wayne Hayes, and Nataša Pržulj. 2010. Optimal Network Alignment with Graphlet Degree Vectors. *Cancer Informatics* 9 (06 2010), 121–37.

[21] Shahin Mohammadi, David F Gleich, Tamara G Kolda, and Ananth Grama. 2016. Triangular alignment (TAME): A tensor-based approach for higher-order network alignment. *IEEE/ACM transactions on computational biology and bioinformatics* Online (2016), 1–14. https://doi.org/10.1109/TCBB.2016.2595583

[22] Huda Nassar and David F. Gleich. 2017. Multimodal Network Alignment. In *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 615–623. https://doi.org/10.1137/1.9781611974973.69

[23] Maks Ovsjanikov, Quentin Mérigot, Facundo Mémoli, and Leonidas Guibas. 2010. One Point Isometric Matching with the Heat Kernel. *Computer Graphics Forum* 29, 5 (2010), 1555–1564. https://doi.org/10.1111/j.1467-8659.2010.01764.x

[24] Rob Patro and Carl Kingsford. 2012. Global network alignment using multiscale spectral signatures. *Bioinformatics* 28, 23 (2012), 3105–3114.

[25] Christian Schellewald and Christoph Schnörr. 2005. Probabilistic Subgraph Matching Based on Convex Relaxation. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*. Springer Berlin / Heidelberg, Berlin, Heidelberg, 171–186. https://doi.org/10.1007/11585978_12

[26] V. Simoncini. 2016. Computational Methods for Linear Matrix Equations. *SIAM Rev.* 58, 3 (2016), 377–441. https://doi.org/10.1137/130912839 arXiv:https://doi.org/10.1137/130912839

[27] Rohit Singh, Jinbo Xu, and Bonnie Berger. 2008. Global alignment of multiple protein interaction networks with application to functional orthology detection. *PNAS* 105, 35 (2008), 12763–12768. https://doi.org/10.1073/pnas.0806627105

[28] V. Vijayan and T. Milenković. 2017. Multiple network alignment via multiMAGNA++. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* PP, 99 (2017), 1–1. https://doi.org/10.1109/TCBB.2017.2740381