SIAM J. SCI. COMPUT.
Vol. 43, No. 5, pp. S592–S611

SCALABLE ALGORITHMS FOR MULTIPLE NETWORK ALIGNMENT*

HUDA NASSAR†, GEORGIOS KOLLIAS‡, ANANTH GRAMA§, AND DAVID F. GLEICH§

Abstract. Multiple network alignment is the problem of identifying similar and related regions in a given set of networks. While there are a large number of effective techniques for pairwise problems with two networks that scale in terms of edges, 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 manuscript 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 principled heuristic procedure for pairwise network alignment called IsoRank. This can be combined with a new algorithm for k-dimensional matching problems on low-rank tensors to produce the alignment. We demonstrate results on synthetic and real-world problems that show our technique (i) is as good as or better than, in terms of quality, 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.

Key words. multiple network alignment, network alignment, low-rank tensor, k-partite matching, k-dimensional matching

AMS subject classifications. 05C85, 90B10, 68R10

DOI. 10.1137/20M1345876

1. Introduction.

Pairwise global network alignment (PNA) is the problem of matching pairs of nodes in two input graphs such that the pairing identifies common structures in both graphs. Algorithms for and applications of this problem are extensively discussed in the literature [8, 18, 25, 27, 1, 29, 2, 16, 26, 19, 13]. A more general problem is that of multiple global network alignment (MNA) [10, 23, 24, 30], where we are interested in finding a common subgraph present in more than two input networks (illustrated in the top panel of Figure 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 [5]. That said, MNA is 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 [16, 17, 8, 26, 2, 27]: (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

*Received by the editors June 17, 2020; accepted for publication (in revised form) April 19, 2021; published electronically August 5, 2021.
https://doi.org/10.1137/20M1345876
Funding: The authors were supported by NSF CCF-1149756, IIS-1422918, IIS-1546488, CCF-0939370, CCF-1909528, DARPA SIMPLEX, and the Sloan Foundation.
†RelationalAI, Inc., Berkeley, CA 94704 USA (huda.nassar@relational.ai).
‡IBM, New York, NY 10022 USA (gkollias@us.ibm.com).
§Department of Computer Science, Purdue University, West Lafayette, IN 47906 USA (ayg@purdue.edu, dgleich@purdue.edu).

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 also NP-complete for $k \geq 3$ [15]. There are approximation algorithms for k-dimensional matching [14], but they still require exponential data from step (i).

Despite the computational difficulty, there are only a few existing algorithms that navigate the computational and memory requirements. In order to do so, they often make strong assumptions or use heuristic ideas that compromise general-purpose quality. For instance, a straightforward solution is to consider sequences of pairwise network alignment problems, or to use pairwise network alignment data to infer multinetwork alignments. This can be problematic if there is an ordering of the graphs that will produce poor alignment results initially and will end up biasing the overall alignment. Another straightforward solution is to restrict the set of possible alignments to those inferred through prior information or metadata about the nodes. Such information often speeds up the computation drastically and guides the algorithm to a meaningful solution [24]; however, when such information is not available, the algorithms often run out of memory or fail to return any useful information.

In this paper, we focus on the case when such information is not present and there is no reduction to pairwise alignments. In this regime existing techniques take too long, run out of memory, or give bad results as explained below. We note that a number of important applications, e.g., aligning protein/gene interaction networks from specific tissues or pathologies, lie in this regime. We return to this point that motivates the need for new methods (section 2.1) after a brief introduction to multiple network alignment (section 2).

2. Multiple network alignment. The multiple network alignment problem can be idealized in many different ways. The formulation of the problem we assume is that there is some *core* graph shared among all the different given networks and that finding this *shared* graph is the alignment task. This core graph will have its edges present in *all* of the different realizations. The resulting formulation of the multiple network alignment problem we use for three networks is

$$\begin{array}{ll} \text{maximize} & \sum_{i,j,k} \sum_{r,s,t} A_{ir} B_{js} C_{kt} \underline{X}_{ijk} \underline{X}_{rst} \\ \text{subject to} & \sum_{j,k} \underline{X}_{ijk} \leq 1 \text{ for all } i; \sum_{i,k} \underline{X}_{ijk} \leq 1 \text{ for all } j; \\ & \sum_{i,j} \underline{X}_{ijk} \leq 1 \text{ for all } k; \underline{X}_{i,j,k} \in \{0,1\} \text{ for all } i,j,k. \end{array} \tag{2.1}$$

Throughout the paper, we use the notation of bold underlined capital letters (e.g., $\underline{\boldsymbol{X}}$) to denote tensors, and 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 the adjacency matrices for the three undirected and unweighted networks, and the number of vertices of these networks gives the summation limits in the above expression. The objective function can be read as follows: nodes i, j, k are matched and nodes r, s, t are matched; the edges (i, r) in A, (j, s) in B, and (k, t) in C are all simultaneously preserved. That is, the product of all of these expressions is 1 when all the edges exist and they match, and 0 otherwise. The constraints express the fact that any node in any of the three networks can be part of at most one matching triplet. The extension to k networks will be straightforward once we introduce some notation.

If we use the notation $\mathbf{x} = \text{vec}(\underline{\boldsymbol{X}})$ to denote the vectorization procedure of a tensor $\underline{\boldsymbol{X}}$, as used in [11, section 12.4.11], i.e., vectorize the tensor column by column, 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

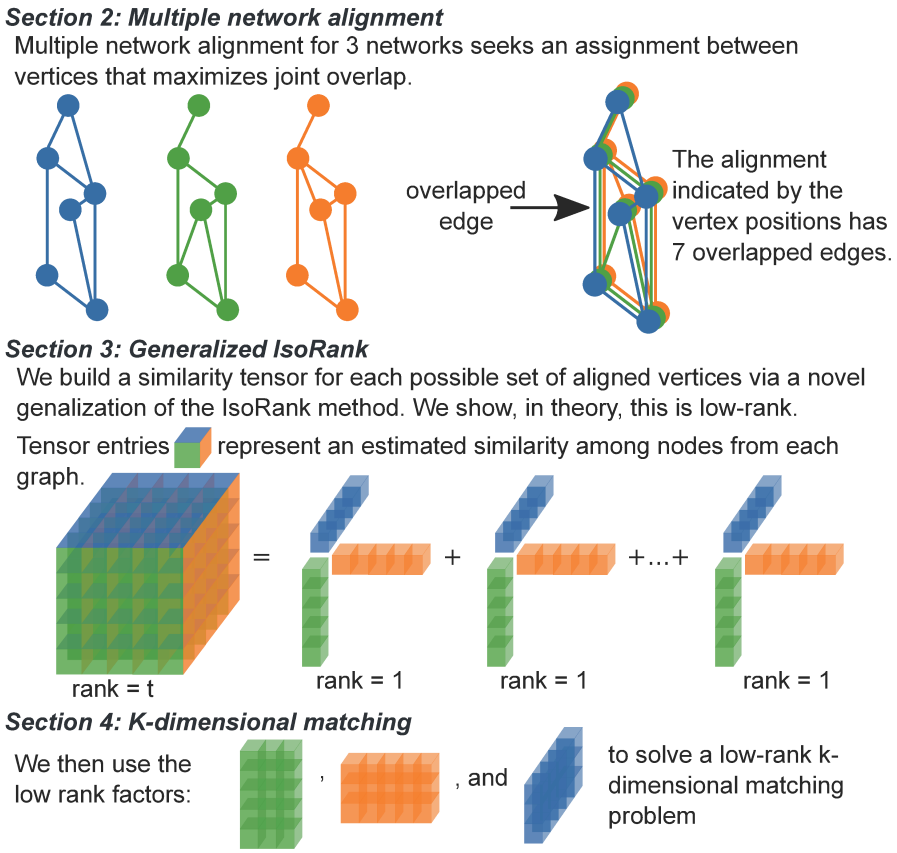


FIG. 1. *A visualization of the pipeline of this paper. The multiple network alignment problem is illustrated in the top panel and discussed in section 2. Our solution is to form a low-rank representation of a high dimensional tensor (middle panel, section 3), and then perform k-dimensional matching on the low-rank factors (bottom panel, section 4).*

12.4.19 in [11].) 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 [11, section 12.4.5] and [6]). Then, we have the three and k-network problems:

$$\begin{aligned} &\text{maximize} && \mathbf{x}^T(\boldsymbol{C}\otimes\boldsymbol{B}\otimes\boldsymbol{A})\mathbf{x} \\ &\text{subject to} && \flat_1(\underline{\boldsymbol{X}})\mathbf{e}\le\mathbf{e};\flat_2(\underline{\boldsymbol{X}})\mathbf{e}\le\mathbf{e};\flat_3(\underline{\boldsymbol{X}})\mathbf{e}\le\mathbf{e} \\ &&& \underline{X}_{i,j,k}\in\{0,1\} \text{ for all } i,j,k. \end{aligned} \tag{2.2}$$

$$\begin{aligned} &\text{maximize} && \mathbf{x}^T(\boldsymbol{A}_k\otimes\cdots\otimes\boldsymbol{A}_1)\mathbf{x} \\ &\text{subject to} && \flat_1(\underline{\boldsymbol{X}})\mathbf{e}\le\mathbf{e};\ldots;\flat_k(\underline{\boldsymbol{X}})\mathbf{e}\le\mathbf{e} \\ &&& \underline{X}_{i,j,\ldots,k}\in\{0,1\} \text{ for all indices.} \end{aligned} \tag{2.3}$$

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 $\mathbf{x} = \text{vec}(\underline{\boldsymbol{X}})$.

2.1. The need for new methods. Existing MNA methods include FUSE [10], IsoRankN [23], MultiMagna++ [30], FLAN [24], and ProgNatalie++ [24]. These are

discussed in detail in section 5. To the best of our knowledge, the highest number of networks an existing MNA method aligns is 10, from the ProgNatalie++ method. What we propose 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; see details in section 7.2).

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++ [30]. In the problem setup, we use three graphs of size $n = 500$. We generate the three input graphs as random perturbations of a single preferential attachment graph of size 500 and average degree 8. Each perturbation is formed by randomly removing edges with a probability $0.5/n$ (more detail on synthetic graph generation can be found in section 6.3). When we give this problem as input to MultiMagna++, it does not produce any meaningful results, and the relative number of overlapped edges (illustrated in the top panel of Figure 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% (a detailed analysis of our method's performance is provided in the Experiments; see section 6). ProgNatalie++ and FLAN are more resistant to the absence of prior known similarity due to these algorithms using the input similarity information as topological similarity and not biological similarity, but the running time of these algorithms is extreme (for example, they would take multiple hours for aligning 5 small networks of size 100 nodes each), and the reason for their slow speed is that they are using a Lagrangian relaxation approach to solve an NP-hard problem at each step.

2.2. Technical overview of our improvements. The two main technical innovations are (i) a specific multinetwork generalization of the pairwise network alignment algorithm IsoRank [29] 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 with an a posteriori approximation bound when the matching information is given by a low-rank tensor. We summarize our contributions here:

- We generalize the IsoRank algorithm to multiple networks and show 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 3).
- We present a new k-dimensional matching algorithm for low-rank tensors with an a posteriori approximation bound (section 4.1).
- We experimentally show that multiple network alignment is faster and higher-quality compared to performing multiple pairwise alignments when the number of networks grows (section 6.3).
- We perform a case study on anonymized data from a collaboration network, where we show that aligning anonymized triplets of egonets can identify those triplets with high Jaccard similarity, which can only be accurately computed from the deanonymized data (section 6.5).

3. A low-rank heuristic method. We now discuss algorithms for (2.2), (2.3). 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 the 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 [8], and these are nontrivial 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 [17] to compute IsoRank, which will easily scale to multiple networks; we explain these now.

3.1. IsoRank. IsoRank is a pairwise network alignment method [29] that uses the PageRank vector $\mathbf{y}$ of the graph with adjacency matrix $\boldsymbol{B} \otimes \boldsymbol{A}$ (see [9] for more on this relationship) as a principled heuristic analogue of what we informally think of as a "matching-biased eigenvector" of $\boldsymbol{B} \otimes \boldsymbol{A}$. Formally, let $\boldsymbol{D}_A$ and $\boldsymbol{D}_B$ be the diagonal degree matrices for graphs A and B; then $\mathbf{y}$ is given by the solution of the equations

$$\begin{aligned} \mathbf{y} &= \alpha(\boldsymbol{B}\boldsymbol{D}_B^{-1} \otimes \boldsymbol{A}\boldsymbol{D}_A^{-1})\mathbf{y} + (1-\alpha)\mathbf{h} \\ \Updownarrow \mathbf{y} &= \mathrm{vec}(\boldsymbol{Y}), \mathbf{h} = \mathrm{vec}(\boldsymbol{H}), \\ \boldsymbol{Y} &= \alpha \boldsymbol{A}\boldsymbol{D}_A^{-1}\boldsymbol{Y}\boldsymbol{D}_B^{-1}\boldsymbol{B} + (1-\alpha)\boldsymbol{H}. \end{aligned}$$

The value of α is typically chosen to be somewhere between 0.7 and 0.9 following [29], and the data $\mathbf{h}$ or $\boldsymbol{H}$ is either uniform (if there is no prior information about what might be a match) or chosen to represent some prior information. These equations can be solved without ever forming the Kronecker matrix, although the data involved is still $O(n^2)$ for two $O(n)$ node graphs. Once we have the solution $\boldsymbol{Y}$, this can be turned into an alignment by solving a bipartite matching problem with $\boldsymbol{Y}$.

3.2. Network similarity decomposition (NSD). The NSD method specializes IsoRank in the case when $\boldsymbol{H}$ is a low-rank matrix [17], such as when we are using the uniform personalization term $\mathbf{h} = \frac{1}{mn}\mathbf{e}$, i.e., $\boldsymbol{H} = \frac{1}{mn}\mathrm{ones}(m,n)$ (where A has n vertices and B has m vertices and $\mathbf{e}$ the vector of all ones of appropriate size). Thus, the relevant case for us is when $\boldsymbol{H}$ is rank-1. Then there is an extremely efficient procedure to compute an exact low-rank representation of $\boldsymbol{Y}$. Suppose we initialize a fixed-point iteration for the PageRank linear system with $\boldsymbol{Y}^{(0)} = \boldsymbol{H} = \mathbf{u}\mathbf{v}^T$ (because it is rank-1), and then tth iterate is given by

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

With some reorganization, this can be written as $\boldsymbol{Y}^{(t+1)} = \boldsymbol{U}\boldsymbol{V}^T$ 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.

3.3. Our method: Multiple network similarity decomposition. 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, $\ldots$, and node i_k from the kth 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

$$\mathbf{y} = \alpha(\boldsymbol{P}_k \otimes \boldsymbol{P}_{k-1} \otimes \cdots \otimes \boldsymbol{P}_2 \otimes \boldsymbol{P}_1)\mathbf{y} + (1-\alpha)\mathbf{h}. \tag{3.1}$$

We note that a similar formulation is used in [22], 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 similarly 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

$$\boldsymbol{U}_i = \begin{bmatrix} c_0\boldsymbol{P}_i^0\mathbf{u}_i & c_1\boldsymbol{P}_i^1\mathbf{u}_i & \ldots & c_{t-1}\boldsymbol{P}_i^{t-1}\mathbf{u}_i & c_t\boldsymbol{P}_i^t\mathbf{u}_i \end{bmatrix}, \tag{3.2}$$

where each c_j is $((1-\alpha)\alpha^j)^{1/k}$ when $j \le t-1$, and $c_t = \alpha^{t/k}$, and we summarize this decomposition in the algorithm shown in Figure 2.

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 ith column of a matrix $\boldsymbol{F}$. If we reshape this into a tensor with $\mathrm{vec}(\underline{\boldsymbol{Y}}_i) = \hat{\mathbf{y}}^{(i)}$, and $\mathrm{vec}(\underline{\boldsymbol{Y}}^{(t)}) = \mathbf{y}^{(t)}$, 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 $\underline{\boldsymbol{Y}}^{(t)}$ [11, 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 4).

4. k-dimensional matching with low-rank factors. In this section, we discuss two approaches to solving the k-dimensional matching problem given here:

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

when $\underline{\boldsymbol{T}}$ is given by a nonnegative rank-t representation (illustrated in the middle panel of Figure 1):

$$\begin{aligned}&\underline{T}(i,j,\ldots,\ell) = \textstyle\sum_{t=1}^{r} U_1(i,t)U_2(j,t)\cdots U_k(\ell,t) \Leftrightarrow \underline{\boldsymbol{T}} = \sum_{i=1}^{t} \underline{\boldsymbol{T}}_i \\ &\text{where } \mathrm{vec}(\underline{\boldsymbol{T}}_i) = \boldsymbol{U}_k(:,i) \otimes \boldsymbol{U}_{k-1}(:,i) \otimes \cdots \otimes \boldsymbol{U}_1(:,i).\end{aligned} \tag{4.1}$$

The first builds on an algorithm for low-rank bipartite matchings from [26]. The second builds on algorithms for progressive alignment [24] and k-partite alignment problems [10, 12].

```
Input: u_1, u_2, ..., u_k; P_1, P_2, ..., P_k; t;
Output: U_1, U_2, ..., U_k
for i = i to k
   c_0 = (1 - α)^{1/k}
   v_0 = u_i
   U_i = [c_0 v_0]
   for j = 1 to t-1
      c_j = ((1 - α)α^j)^{1/k}
      v_j = P_i v_{j-1}
      U_i = [U_i   c_j v_j]
   end
   c_t = α^{t/k}
   v_t = P_i v_{t-1}
   U_i = [U_i   c_t v_t]
end
```

FIG. 2. *Pseudocode for the the multiple network similarity decomposition discussed in section* 3. *Here,* $\boldsymbol{P}_i$ *is the column stochastic adjacency matrix corresponding to network* i*, and* $\mathbf{u}_i$ *is the corresponding factor from the starting low-rank tensor decomposition, and* t *is the number of iterations.*

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. In practice, these bounds are very good and provide approximation factors around 1.08 (see Figure 10).

The techniques extend [26] for the bipartite matching case. To do so, we first need a specific generalized rearrangement inequality for k sequences. We use the result from [28] and provide the statement of the inequality below.

GENERALIZED REARRANGEMENT INEQUALITY. *Assume 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.*

Now, assume that we have a *k-dimensional* tensor $\underline{\boldsymbol{T}}$ of the form shown in (4.1). 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 $\boldsymbol{U}_1(:,i),\ldots,\boldsymbol{U}_k(:,i)$ 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). We now prove the following result (which extends the proof of the 2-dimensional case presented in [26]).

RESULT. *Let the binary-valued tensors* $\underline{\boldsymbol{M}}_i$ *of size* $n_1 \times n_2 \times \ldots n_k$ *store the matching corresponding to* $\underline{\boldsymbol{T}}_i$ *tensor, i.e.,* $\underline{M}(j_1, j_2, \ldots, j_k) = 1$ *if* $(j_1, j_2, \ldots, j_k)$ *is a match, and* 0 *otherwise. Then, there is a best matching in this set* $\underline{\boldsymbol{M}}_1, \ldots, \underline{\boldsymbol{M}}_t$ *that solves the* k*-dimensional matching problem while attaining a* D*-approximation bound, where* D *is an a posterori computable bound.*

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

possible weight on $\underline{\boldsymbol{T}}$.

Define

$$d_{i,j} = \frac{\underline{\boldsymbol{M}}_i \bullet \underline{\boldsymbol{T}}_i}{\underline{\boldsymbol{M}}_j \bullet \underline{\boldsymbol{T}}_i}, \text{ and } d_j = \max_i d_{i,j}.$$

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 $\underline{\boldsymbol{T}}$, where

$$D = \min_j \max_i \frac{\underline{\boldsymbol{M}}_i \bullet \underline{\boldsymbol{T}}_i}{\underline{\boldsymbol{M}}_j \bullet \underline{\boldsymbol{T}}_i}.$$

∎

4.2. A progressive alignment. The bounds given by the low-rank matching algorithm (above) are often very good (Figure 10). In practice we found the procedure in Figure 3 to give better results in terms of the overall multiple network alignment objective. The inspiration for this algorithm is the progressive nature of both *Prog-Natalie++* and *FUSE* [24, 10], and a progressive algorithm for the k-partite matching problem [12]. For three networks (a three-mode tensor), the idea is as follows: 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. When $\underline{\boldsymbol{T}}$ is low rank, these entries also have low-rank structure, and thus we use the routine from [26] to solve each of these low-rank bipartite matching problems and state the overall procedure as an algorithm below. We briefly studied optimizing the ordering of alignment, but this did not seem to yield large differences. Note that this is different from progressive pairwise network alignment because the scores $\underline{\boldsymbol{T}}$ incorporate information from all networks, and our progressive operations simply distill this information into a matching. That said, this method loses quality as the number of networks increases (Figure 5), and our study of this behavior led to improvements in the next subsection.

```
1  Input: U_1, U_2, ..., U_k;
2  Output: Matching M with k columns and matches in rows
3  a,b = bipartitematching(U_1 U_2^T) # match first two modes.
4  M[:,1:2] = [a,b]
5  for i = 3 to k
6    # generate matching information with i-1 modes matched
7    U = U_1[M[:,1],:] ⊙ U_2[M[:,2],:] ⊙···⊙ U_{i-1}[M[:,i-1],:]
8    # using element-wise/Hadamard product ⊙
9    a,b = bipartitematching(U U_i^T) # match in the ith mode
10   M = M[a,:]; M[:,i] = b # permute and extend the matching
11 end
```

FIG. 3. *Pseudocode for the progressive k-dimensional matching algorithm (section 4.2).*

4.3. Further improving the alignment for large numbers of networks. The key step of the previous progressive method is a folding procedure on all the previous alignments (line 7 of Figure 3) and we perform this folding by an element-wise multiplication of all the previous low-rank factors. We notice experimentally that this approach is reasonable for a handful of networks but becomes skewed by any small entries close to zero when we increase the number of networks to align. For this reason, we introduce a new additive term to the matrix $\boldsymbol{U}$ in line 8 that balances the small entries with a sum and show the improved algorithm in Figure 4. 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 7 and $\boldsymbol{U}^+$ is the matrix computed on line 8 of the algorithm in Figure 4. Empirically, 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 [10, 23]. These operations retain the low-rank structure in each bipartite matching problem.

```
1  Input: U_1, U_2, ..., U_k;
2  Output: Matching M with k columns and matches in rows
3  a,b = bipartitematching(U_1 U_2^T) # match first two modes.
4  M[:,1:2] = [a,b]
5  for i = 3 to k
6    # generate matching information with i-1 modes matched
7    U* = U_1[M[:,1],:] ⊙ U_2[M[:,2],:] ⊙ ··· ⊙ U_{i-1}[M[:,i-1],:]
8    U+ = U_1[M[:,1],:] + U_2[M[:,2],:] + ··· + U_{i-1}[M[:,i-1],:]
9    U = 1/2 U*/sum(U*) + 1/2 U+/sum(U+)
10   a,b = bipartitematching(U U_i^T) # match in the ith mode
11   M = M[a,:]; M[:,i] = b # permute and extend the matching
12 end
```

FIG. 4. *Pseudocode for the improved progressive k-dimensional matching algorithm, which is more reasonable to use for a large number of networks (section 4.3).*

5. 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.

5.1. 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 [29]. The networks to be aligned are often protein-protein interaction networks (PPIs) of different species, and the idea is to use the alignment to learn new information about the less studied 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 [13]. Another algorithm, MultiMagna++ [30], uses a genetic algorithm that works directly with the multiway alignment permutations and uses objective or fitness functions that utilize the biological information.

IsoRank [29] and IsoRankN [23] were some of the earliest MNA algorithms. These methods 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 [10], uses protein

similarity to build the k-partite representation of the problem and then uses nonnegative matrix trifactorization to incorporate network structure into the overall alignment.

5.2. Topological algorithms. There are two state-of-the-art algorithms introduced in [24]: *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++ below, which extends the PNA algorithm *Natalie* that was proposed by Klau [16]. 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. 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 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.

6.1. 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 [26].

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 [8] and its low-rank formulation from [26] to be a 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 [26]. 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 (3.2) with 8 iterations and $\alpha = 0.8$; then the final alignment is extracted by our D-approximation (section 4.1). We support the choice of 8 iterations in a study on Erdős–Rényi and preferential attachment graphs (Figure 7).

MultiLR-Prog. This is our algorithm where we use the procedure from section 4.2 and Figure 3 for the matching, where the $\boldsymbol{U}_i$ are from (3.2) with 8 iterations and $\alpha = 0.8$. The bipartite matching problems are themselves solved via a low-rank bipartite matching procedure from [26] (with parameter $b = 10$).

MultiLR-Prog+. This is our algorithm with the improved progressive matching explained in section 4.3 and Figure 4 and is the same as MultiLR-Prog, where we replace the elementwise multiplication from Figure 3 (line 7) with a mixture model for $\boldsymbol{U}$. Otherwise it is identical to MultiLR-Prog.

ProgNatalie++ and ProgNatalie++ with prior. We use ProgNatalie++ from [24] 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 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.

6.2. Evaluating multiple alignment algorithms. Recall that the multiple network alignment problem can be idealized in many different ways. The formulation of the problem we assume is that there is some *core* graph shared among all the different given networks and that finding this *shared* graph is the alignment task. This core graph will have its edges present in *all* of the different realizations. This assumption, however, is not shared across all the different perspectives on multiple network alignment. Hence, when we think about evaluating multiple network alignment methods, we use a few evaluation metrics to discuss the resulting alignments.

The *normalized overlap* of a set of networks $A_1, \ldots, A_k$ 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 $\tilde{\boldsymbol{A}}_1, \ldots \tilde{\boldsymbol{A}}_k$ are the adjacency matrices permuted via the alignment, then this is $\text{nnz}(\tilde{\boldsymbol{A}}_1 \odot \cdots \odot \tilde{\boldsymbol{A}}_k)/\max(\text{nnz}(\boldsymbol{A}_i), \ldots, \text{nnz}(\boldsymbol{A}_k))$ where $\odot$ is the elementwise product and nnz is the number of nonzeros. This evaluation measure is exactly what our multiple network alignment method is designed to optimize.

Alternatively, 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. This setting fixes an answer to the problem and then checks how much of that answer we recover. Because of the degree weighting, this measure places more emphasis on high-degree regions. We regard these as more important to align as low-degree regions often admit automorphic equivalences. The pairwise nature of the evaluation measures also protects against a single mistake in, say, 100 networks ruining the other 99 correctly aligned results. The 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 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_j from network j should be aligned to node v_k from network k; the *score* of a single

alignment of vertices between all networks is

$$\text{score}(v_1, \ldots, v_k) = \binom{k}{2}^{-1} \left(\sum_{j=1}^{k} \sum_{h=j+1}^{k} w(v_j, v_h)\text{correct}(v_j, v_h) \right). \tag{6.1}$$

The overall degree weighted recovery score is simply the sum of the scores for each alignment set. (These scores are scaled to sum to 1 for a perfect alignment of isomorphic networks.) Note that this method cannot be directly optimized as it presumes knowledge of a correct answer.

6.3. 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 ambiguous).

Graph generation. For Erdős–Rényi, we set the edge probability such that we achieve the expected degree $d = 8$ and n nodes. For preferential attachment, to generate a graph with n nodes, 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, and we allow each edge to be deleted according to the probability p_e. We repeat this process k times to reach k networks.

As edge deletion varies. 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 5 (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 7.2) 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. Note that this issue was addressed in [26] by allowing each node to *match* to more than one node (following a sorted ordering), then creating a sparse matrix where each row/column had a very small number of nonzeros, and then solving the bipartite matching problem on the matrix. This approach is not feasible here as k-dimensional matching is NP-complete for $k \geq 3$.

As we add networks. 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 with only 2 edges deleted from each network (in expectation). The results are shown in Figure 5 (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 5 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.

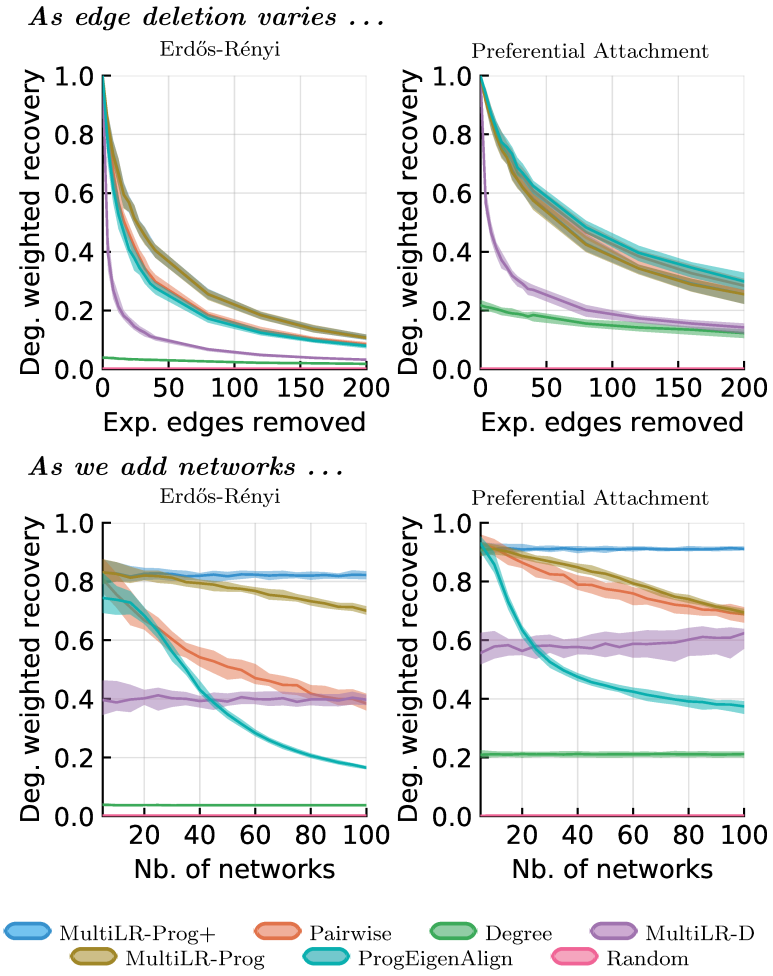
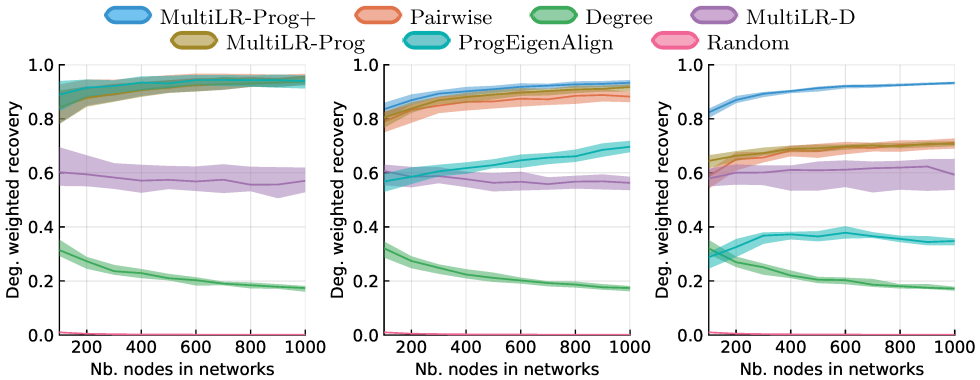


FIG. 5. *(Top two panels) As we increase the expected number of edges removed while aligning* 5 *networks, all methods recover fewer true matches and MultiLR-Prog and MultiLR-Prog+ are consistently the best where MultiLR-D does not do well. Note that MultiLR-Prog and MultiLR-Prog+ are overlapping here. (Lower 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 lower figures). In all figures, the shaded areas represent the* 20*th and the* 80*th percentiles with these experiments run for* 50 *trials.*

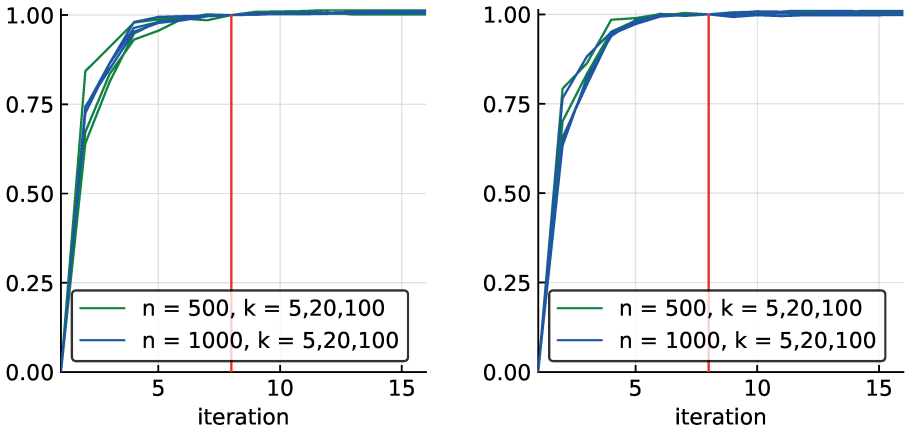
Here, we use preferential attachment graphs, and we fix the edge deletion probability to $p_e = 0.5/n$, as we vary n. We observe that all methods are essentially resistant to the change in the network sizes, whereas this behavior is not true when the number of networks become much bigger (such as 100). From Figure 6, we can conclude that MultiLR-Prog+ is resistant to changes in both the network sizes and the number of networks to be aligned. Interestingly, MultiLR-D is also resistant to such changes but with a worse recovery score.

Eight iterations are enough. We now show a case study to support our choice of 8 iterations. We run our algorithm on several different problems with varying k and varying n. We plot the degree weighted recovery normalized by the value on iteration



(a) $k = 5$ *networks* (b) $k = 20$ *networks* (c) $k = 100$ *networks*

FIG. 6. *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 (MultiLR-Prog+) is the only method that sustains its result. In all figures, the shaded areas represent the 20th and the 80th percentiles with these experiments run for 50 trials.*



Preferential attachment *Erdős–Rényi*

FIG. 7. *To make sure that 8 iterations of the power method are enough, 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 6 different settings on two types of graphs, and they are all indistinguishable.*

8. These results show that the quality of the result does not change considerably after iteration 8 (see Figure 7). It might be helpful here to note that the vectors we produce for each network are PageRank iterates (up to a constant scaling), and we know that especially in the regime of $\alpha = [0.7, 0.9]$, PageRank will converge quickly [4, 3].

6.4. 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 [20] which consists of snapshots of an Internet router network at 733 time points. We consider two problems: aligning 5 random snapshots

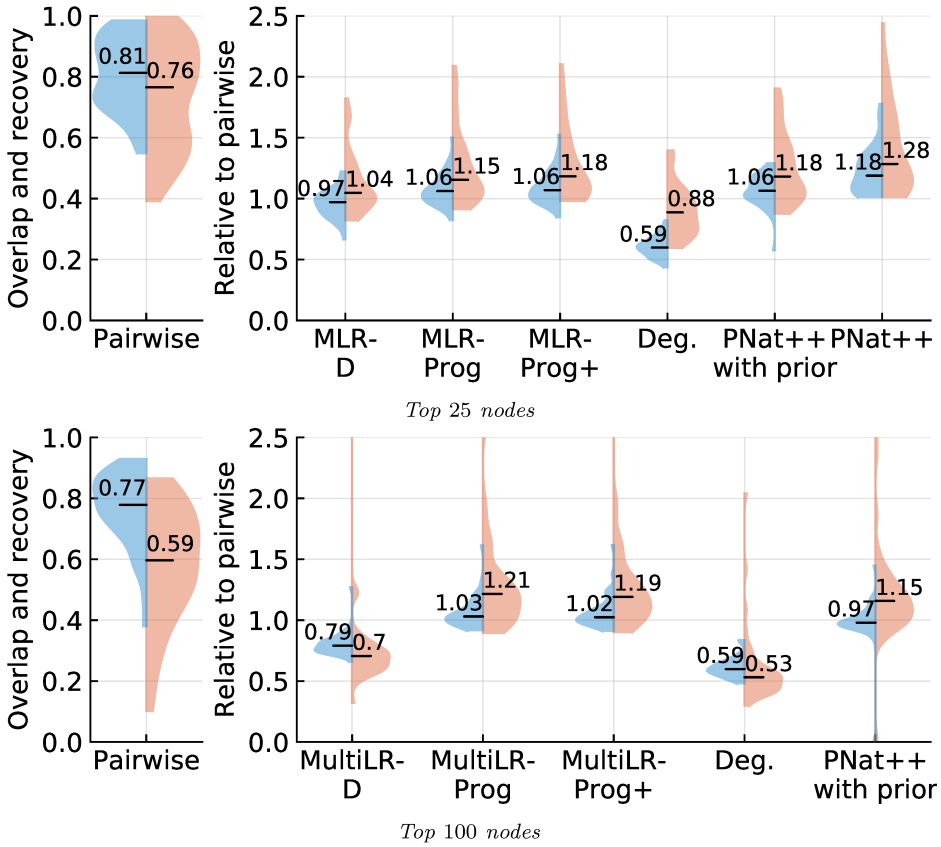


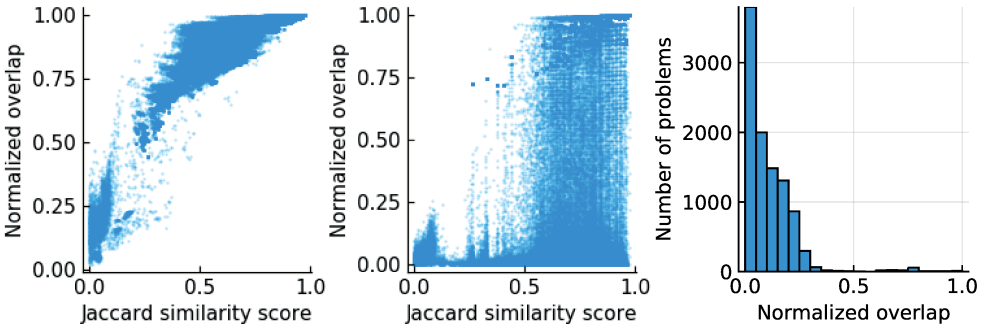
FIG. 8. *We consider* 50 *trials of aligning* 5 *real-world router graphs and show a violin plot (with the median flagged) of the degree weighted recovery (blue) and normalized overlap (red) side by side for the pairwise method. For the other methods, we show values relative to the pairwise scores. These 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.*

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 8, 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 versus 40 minutes (see more timing in section 7.2). For the larger run, MultiLR-Prog and MultiLR-Prog+ achieve results that consistently outperform the pairwise baseline in terms of overlap.

6.5. Aligning anonymized egonets. Next, we use our multiple network alignment algorithm to align anonymized egonets of the collaboration network DBLP [7]. This experiment is inspired by one in [26]. In DBLP, the nodes are authors, and edges represent coauthorship. 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 $\text{Jaccard}(a,b,c) = \frac{N(a)\cap N(b)\cap N(c)}{N(a)\cup N(b)\cup N(c)}$ where $N(a)$ is the set of neighbors of node a. For each triple of three coauthors with at least 100 other coauthors, we align



(a) *Using MNA* (b) *Using pairwise* (c) *MNA on random networks*

FIG. 9. *Figures* (a) *and* (b) *show the normalized overlap of the aligned three egonets using multiple network alignment and the pairwise method, respectively. These two figures show that when using multiple network alignment, normalized overlap tracks the Jaccard similarity scores, whereas the pairwise method fails to show that. Figure* (c) *shows that for random triplets, the normalized overlap is less than* 0.25 *in the majority of experiments. This means that random triplets have small normalized overlap as well, which allows us to infer high-Jaccard similarity with high normalized overlap.*

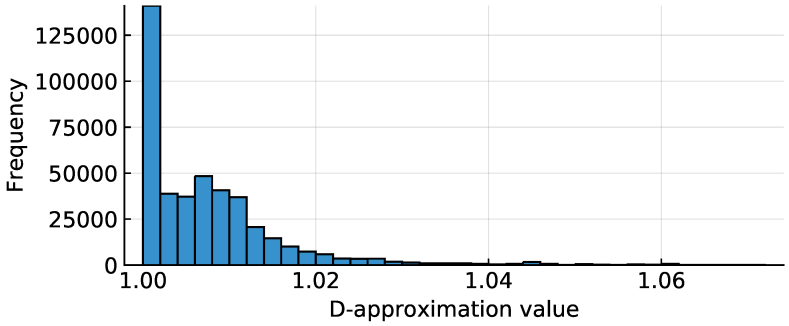


FIG. 10. *This figure shows the D approximation values from the experiment of aligning multiple egonets in the DBLP network. These numbers show that the approximation bound D is very close to* 1 *in practice.*

the egonets using the MultiLR-D method and measure the normalized overlap. The results in Figure 9 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.

We now show the quality of the posterior bound D we get from this method. We plot a histogram of these values here (Figure 10) and observe that a striking number of them are very close to 1, and even the maximum of them is still less than 1.1. For other cases when we ran our MulitLR-D algorithm, the values were comparable as well with the maximum less than 1.07 or 1.08.

TABLE 1
Degree weighted recovery for large graphs from section 7.1.

Dataset	Nodes	Edges	MultiLR-D		Degree	
			p_e=0.1	p_e=0.5	p_e=0.1	p_e=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}

7. Scalability. In this section we evaluate our methods as we grow the networks' sizes and the number of networks to be aligned, and we show running time results from previous experiments. The highest number of networks an existing MNA method aligns is 10 (ProgNatalie++), and this is in the setting when prior similarity knowledge is present. In this section, we show that two of our methods can scale to hundreds of networks.

7.1. 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 [21], and an Internet topology graph *as-Skitter* from [20]. The pipeline of the experiment is similar to that in section 6.3; we use the edge deletion probability p_e/n with $p_e = \{0.1, 0.5\}$ and generate 5 instances of the graph.

Methods. We are unaware of any methods besides ours that will scale to networks of this size (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, as it is the only other feasible method, and provides a useful point of comparison for how MultiLR-D performs.

Results and conclusions. We show the average results from 20 runs of this experiment in Table 1. These problems were extremely challenging in practice, and we attribute 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, especially when very little perturbation occurred. We believe that the result in this challenging scenario is not ideal but is a demonstration that such low-rank methods expand the scalability envelope for multiple network alignment problems.

7.2. Scalability in terms of running time. In Table 2, we show running times for the methods on the routers' alignment problems. Then in Figure 11, 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. Figure 11 is a demonstration that our low-rank methods are the first reasonable choice for aligning many networks (beyond 5 networks), and especially when no prior similarity scores are present.

8. Discussion and future work. Having a method that accurately and scalably aligns large numbers of networks opens a number of new dimensions in data

TABLE 2
Runtimes in seconds for the 25-node 5-network router problem (columns 2–3) and the 100-node 5-network router problem (columns 4–5). We show the median and maximum times.

Algorithm	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	-	-

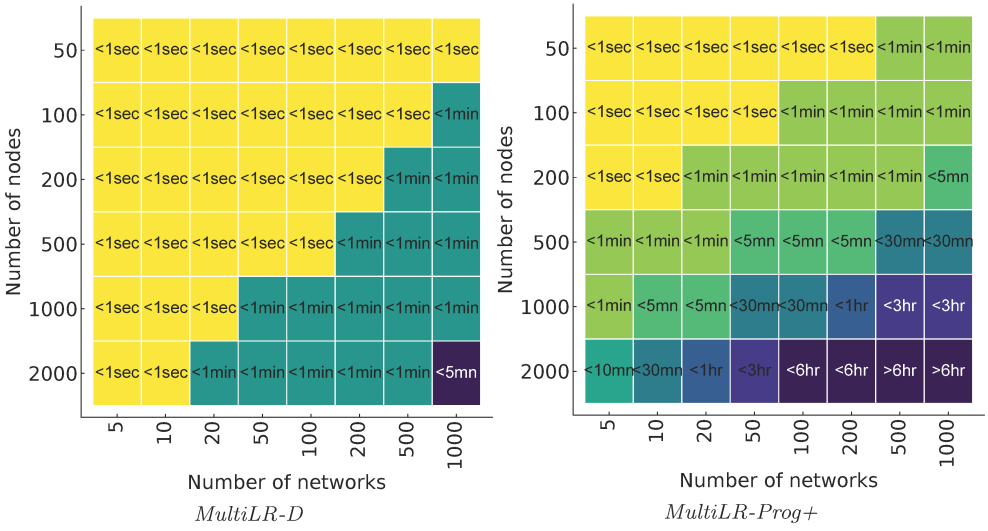


FIG. 11. *The runtime as we run MultiLR-D (top) and MultiLR-Prog+ (bottom) on the synthetic experiments on a wide variety of problem sizes with Erdős–Rényi graphs.*

science. 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 of why MultiLR-Prog+ outperformed MultiLR-Prog. Our working hypothesis is that the elementwise addition (compared with multiplication) gives the method resilience to mistakes made early in the progressive process. More broadly, the EigenAlign framework [8] is superior to the IsoRank framework for pairwise alignment. The ideas here apply to a multinetwork 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.

REFERENCES

[1] N. Atias and R. Sharan, *Comparative analysis of protein networks: Hard problems, practical solutions*, Commun. ACM, 55 (2012), pp. 88–97, https://doi.org/10.1145/2160718.2160738.

[2] M. Bayati, D. F. Gleich, A. Saberi, and Y. Wang, *Message-passing algorithms for sparse network alignment*, ACM Trans. Knowl. Discov. Data, 7 (2013), 3, https://doi.org/10.1145/2435209.2435212.

[3] P. Berkhin, *A survey on PageRank computing*, Internet Math., 2 (2005), pp. 73–120, https://doi.org/10.1080/15427951.2005.10129098.

[4] M. Bianchini, M. Gori, and F. Scarselli, *Inside PageRank*, ACM Trans. Internet Tech., 5 (2005), pp. 92–128, https://doi.org/10.1145/1052934.1052938.

[5] S. A. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the Third Annual ACM Symposium on Theory of Computing, ACM, New York, 1971, pp. 151–158.

[6] J. Draisma and J. Kuttler, *Bounded-rank tensors are defined in bounded degree*, Duke Math. J., 163 (2014), pp. 35–63, https://doi.org/10.1215/00127094-2405170.

[7] P. Esfandiar, F. Bonchi, D. F. Gleich, C. Greif, L. V. S. Lakshmanan, and B.-W. On, *Fast Katz and Commuters: Efficient Estimation of Social Relatedness in Large Networks*, Springer, Berlin, Heidelberg, 2010, pp. 132–145, https://doi.org/10.1007/978-3-642-18009-5_13.

[8] S. Feizi, G. Quon, M. R. Mendoza, M. Médard, M. Kellis, and A. Jadbabaie, *Spectral Alignment of Networks*, preprint, https://arxiv.org/abs/1602.04181, 2016.

[9] D. F. Gleich, *PageRank beyond the web*, SIAM Rev., 57 (2015), pp. 321–363, https://doi.org/10.1137/140976649.

[10] V. Gligorijevic, N. Malod-Dognin, and N. Przulj, *Fuse: Multiple network alignment via data fusion*, Bioinformatics, 32 (2016), pp. 1195–1203, https://doi.org/10.1093/bioinformatics/btv731.

[11] G. H. Golub and C. van Loan, *Matrix Computations*, 4th ed., Johns Hopkins University Press, Baltimore, MD, 2013.

[12] G. He, J. Liu, and C. Zhao, *Approximation algorithms for some graph partitioning problems*, J. Graph Algorithms Appl., 4 (2000), pp. 1–11, https://doi.org/10.7155/jgaa.00021.

[13] K. Kalecky and Y.-R. Cho, *PrimAlign: PageRank-inspired Markovian alignment for large biological networks*, Bioinformatics, 34 (2018), pp. i537–i546, https://doi.org/10.1093/bioinformatics/bty288.

[14] V. Kann, *Maximum bounded* 3*-dimensional matching is MAX SNP-complete*, Inform. Process. Lett., 37 (1991), pp. 27–35, https://doi.org/10.1016/0020-0190(91)90246-E.

[15] R. M. Karp, *Reducibility among combinatorial problems*, in Proceedings of a Symposium on the Complexity of Computer Computations, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1972, pp. 85–103, https://doi.org/10.1007/978-1-4684-2001-2_9.

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

[17] G. Kollias, S. Mohammadi, and A. Grama, *Network similarity decomposition (NSD): A fast and scalable approach to network alignment*, IEEE Trans. Knowl. Data Engrg., 24 (2012), pp. 2232–2243, https://doi.org/10.1109/TKDE.2011.174.

[18] O. Kuchaiev, T. Milenković, V. Memišević, W. Hayes, and N. Pržulj, *Topological network alignment uncovers biological function and phylogeny*, J. Roy. Soc. Interface, 7 (2010), https://doi.org/10.1098/rsif.2010.0063.

[19] G. Langs, P. Golland, Y. Tie, L. Rigolo, and A. Golby, *Functional geometry alignment and localization of brain areas*, in Proceedings of the 24th Annual Conference on Neural Information Processing Systems, NeurIPS, San Diego, CA, 2010, pp. 1225–1233, https://papers.nips.cc/paper/2010/hash/35f4a8d465e6e1edc05f3d8ab658c551-Abstract.html.

[20] J. Leskovec, J. Kleinberg, and C. Faloutsos, *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, ACM, New York, 2005, pp. 177–187, https://doi.org/10.1145/1081870.1081893.

[21] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, *Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters*, preprint, https://arxiv.org/abs/0810.1355, 2008.

[22] Z. Li, R. Petegrosso, S. Smith, D. Sterling, G. Karypis, and R. Kuang, *Scalable Label Propagation for Multi-Relational Learning on Tensor Product Graph*, preprint, https://arxiv.org/abs/1802.07379, 2018.

[23] C.-S. Liao, K. Lu, M. Baym, R. Singh, and B. Berger, *IsoRankN: Spectral methods for global alignment of multiple protein networks*, Bioinformatics, 25 (2009), pp. i253–i258,

https://doi.org/10.1093/bioinformatics/btp203.

[24] E. Malmi, S. Chawla, and A. Gionis, *Lagrangian relaxations for multiple network alignment*, Data Min. Knowl. Discov., 31 (2017), pp. 1331–1358, https://doi.org/10.1007/s10618-017-0505-2.

[25] N. Malod-Dognin and N. Pržulj, *L-graal: Lagrangian graphlet-based network aligner*, Bioinformatics, 31 (2015), pp. 2182–2189, https://doi.org/10.1093/bioinformatics/btv130.

[26] H. Nassar, N. Veldt, S. Mohammadi, A. Grama, and D. F. Gleich, *Low rank spectral network alignment*, in Proceedings of the 2018 Web Conference, WWW '18, 2018, pp. 619–628, https://doi.org/10.1145/3178876.3186128.

[27] R. Patro and C. Kingsford, *Global network alignment using multiscale spectral signatures*, Bioinformatics, 28 (2012), pp. 3105–3114, https://doi.org/10.1093/bioinformatics/bts592.

[28] H. D. Ruderman, *Two new inequalities*, Amer. Math. Monthly, 59 (1952), pp. 29–32, https://doi.org/10.2307/2307185.

[29] R. Singh, J. Xu, and B. Berger, *Global alignment of multiple protein interaction networks with application to functional orthology detection*, Proc. Natl. Acad. Sci. USA, 105 (2008), pp. 12763–12768, https://doi.org/10.1073/pnas.0806627105.

[30] V. Vijayan and T. Milenković, *Multiple network alignment via multiMAGNA++*, IEEE/ACM Trans. Comput. Biol. Bioinformatics, 15 (2018), pp. 1669–1682, https://doi.org/10.1109/TCBB.2017.2740381.