2019

Network Alignment by Propagating Reliable Similarities

Zirou Qiu

Ruslan Shaydulin

Xiaoyuan Liu

Yuri Alexeev

Christopher S. Henry

See next page for additional authors

Authors
Zirou Qiu, Ruslan Shaydulin, Xiaoyuan Liu, Yuri Alexeev, Christopher S. Henry, and Ilya Safro

Network Alignment by Propagating Reliable Similarities

Zirou Qiu* Ruslan Shaydulin * Xiaoyuan Liu * Yuri Alexeev †
Christopher S. Henry † Ilya Safro *

Abstract

Networks model a variety of complex phenomena across different domains. In many applications, one of the central questions is how to align two or more networks to infer the similarities between nodes and discover potential correspondence. In this paper, we propose a network alignment algorithm that relies exclusively on the underlying graph structure. Under the guidance of rules which we defined, we compute the similarity between a pair of cross-network vertices iteratively based on their neighborhood. The resulting cross-network similarity matrix is then used to infer a permutation matrix which encodes the alignment. We improve the performance of a commonly used post-processing step (local search) by introducing a novel selection rule based on the levels of mismatching of vertices. Through extensive numerical experiments, we show that our alignment algorithm outperforms the state-of-the-art alignment methods in terms of alignment accuracy at the cost of an extra logarithmic factor.
Reproducibility: Our source code, documentation, and data sets are available at `https://tinyurl.com/y6qlsyh2`

1 Background and Motivation

Network alignment has applications across various domains. Given two networks, many basic analysis tasks include quantification of structural similarities and discovering potential correspondences between cross-network vertices. For example, aligning protein-protein interaction networks could help us discover functionally conserved components and identify proteins that play similar roles in networked biosystems [10]. In the context of marketing, it is often useful for companies to link similar users across different networks in order to recommend products to potential customers [15]. On top of that, network alignment problems also exist in fields such as computer vision [3], chemistry [8], social network mining [15], and economy [16].

In general, network alignment aims to map vertices from one network to another such that some cost function is optimized and pairs of mapped vertices are similar [3]. While the exact definitions of similarities are problem dependent, they often reveal some resemblance between structures of two networks or additional domain information such as DNA sequences [10]. We define the network alignment problem as follows.

Network Alignment Problem: Consider two networks with underlying undirected, unweighted graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ with $|\mathcal{V}_1| = |\mathcal{V}_2|$ (condition is satisfied by adding dummy 0-degree nodes to the smaller network)[1]. Let $\mathbf{A}$ and $\mathbf{B}$ be the adjacency matrices of $\mathcal{G}_1$ and

*Clemson University, Clemson, **Email**:{zirouq, rshaydu, xiaoyu3, isafro}@clemson.edu
†Argonne National Laboratory, Lemont, **Email**:{yuri, chenry}@anl.gov

[1]Note that the requirement $|\mathcal{V}_1| = |\mathcal{V}_2|$ is introduced only to make $\mathbf{P}$ square for simple computation of the objective as 0-degree dummy nodes do not contribute to it. In later discussion, we do not require $|\mathcal{V}_1| = |\mathcal{V}_2|$.

$\mathcal{G}_2$, respectively. The goal of network alignment is to find the permutation matrix $\mathbf{P}$ that minimizes the cost function:

$$\min_{\mathbf{P}} -trace(\mathbf{P}^T\mathbf{A}\mathbf{P}\mathbf{B}^T), \tag{1}$$

where the permutation matrix $\mathbf{P}$ encodes the bijective mappings between $\mathcal{V}_1$ and $\mathcal{V}_2$ for which $\mathbf{P}_{i,u} = 1$ if $i \in \mathcal{V}_1$ is aligned with $u \in \mathcal{V}_2$ (or $\mathbf{P}_{i,u} = 0$ otherwise). An equivalent problem is to maximize the number of ***conserved*** edges, where an edge $(i, j) \in \mathcal{E}_1$ is conserved if $\mathbf{P}_{i,u} = 1$, $\mathbf{P}_{j,v} = 1$ and $(u, v) \in \mathcal{E}_2$.

The above problem is a special case of the quadratic assignment problem which is known to be NP-hard [14]. Therefore, many iterative algorithms have been developed to solve the problem by relaxing the integrality constraints. Typically, they first compute similarity between every pair of cross-network vertices iteratively by accumulating similarities between pairs of cross-network neighbors, then infer the alignments between cross-network nodes by solving variants of maximum weight matching problem [4]. The existing approaches exhibit limitations. First, computing similarity between $i \in \mathcal{V}_1$ and $u \in \mathcal{V}_2$ is a process of accumulating the similarities between **all** pairs of their cross-network neighbors. This leads to an unwanted case where i has a high similarity score with u simply because u is a high-degree node so they have many pairs of cross-network neighbors that can contribute similarities to (i, u). This setting also makes it difficult to effectively penalize the degree difference between i and u. Second, previous approaches accumulate similarities between cross-network neighbors indiscriminately which dilutes the result after normalization.

Our contribution: To address the limitations, we propose a novel network alignment algorithm based on identifying *globally most similar* pairs of vertices with growing *contribution threshold* (both defined in the later section). Such a threshold is used to determine which pairs of cross-network vertices can contribute similarities. Local search can be used to further enhance the alignment quality and we introduce a new selection method for local search procedure which narrows the search space by locating mismatched vertices. The experimental results show that our network alignment algorithm (before applying local search) already significantly outperforms all the baseline methods. At the same time, the local search scheme equipped with the proposed selection method drastically decreases the number of iterations it takes to reach a optimum.

2 Related Works.

Extensive research has been conducted in solving the network alignment problem. The underlying intuition is that *two cross-network vertices are similar if their cross-network neighbors are similar.*

IsoRank [13] is a classic alignment algorithm which is equivalent to PageRank on the Kronecker product of two networks. Koutra et al. [9] formulate the bipartite network alignment problem and propose an iterative improvement algorithm to find the local/global optimal. Klau formulates the problem based on maximum weight trace and suggests a lagrangian relaxation approach [8].

Zhang and Tong [15] tackle the attribute network alignment problem for which vertices have different labels. They drop the topological consistency assumption and solve the problem by using attributes as alignment guidance. In another paper, Zhang et al. [16] consider multilevel network alignment problems based on the coarsening and uncoarsening scheme. They not only discover the node-level correspondence but also cluster-level correspondence.

Hashemifar and Xu's [6] approach involves computing topological importance for each node, and a pair of cross network vertices have similar score if they play similar roles in the corresponding

networks. NETAL [10] introduces the concept of interaction scores between each pair of cross-network vertices which are estimations of the number of conserved edges. ModuleAlign [5] combines the topological information with non-network information such as protein sequence for each vertex and produces alignments that resemble both topological similarities and sequence similarities.

3 Proposed Network Alignment Algorithm

By relaxing the combinatorial constraints, our proposed algorithm is a two-step procedure: (1) compute the cross-network similarity matrix which encodes similarities between cross-network vertices; (2) extract the alignments based on the similarity matrix.

Given two undirected networks $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ with $|\mathcal{V}_1| = n_1$ and $|\mathcal{V}_2| = n_2$. Without loss of generality, assume $n_1 \leq n_2$, and two networks have comparable number of vertices such that $O(n_1) = O(n_2)$. Nodes in each network are labeled with consecutive integers starting from 1. Throughout the paper, we use bold uppercase letters to represent matrices and bold lowercase letters to represent vectors. We use subscript over a node to refer the network it belongs to, for example, $i_{(1)} \in \mathcal{V}_1$. We use superscript over vectors/matrices to denote number of iterations. Let $\mathcal{N}(i_{(1)})$ denote the set of neighbors of vertex $i_{(1)}$. Let $\mathbf{S}$ be the $n_1 \times n_2$ cross-network similarity matrix where $\mathbf{S}_{i,u}$ encodes the similarity score between $i_{(1)}$ and $u_{(2)}$. Note that $i_{(1)}$ and $u_{(2)}$ do not carry superscripts in matrix / vector indexing. Let $f : \mathcal{V}_1 \to \mathcal{V}_2$ denote the bijective alignment function for which $f(i_{(1)}) = u_{(2)}$ if $\mathbf{P}_{i,u} = 1$. Let t_{max} denote the maximum number of iterations of our algorithm which equals to the larger diameter of the two networks.

3.1 Similarity Computation

We introduce three rules which serve as the design guidance of the proposed algorithm. In general, the proposed algorithm improves the similarities between cross-network vertices by updating $\mathbf{S}$ itertivley. Given a vertex $i_{(1)}$ and its aligned vertex $u_{(2)} = f(i_{(1)})$, a neighbor $j_{(1)} \in \mathcal{N}(i_{(1)})$ is ***conserved*** if $f(j_{(1)}) \in \mathcal{N}(u_{(2)})$.

Definition 3.1 (Best Matching)**.** *A vertex $u_{(2)}$ is the **best matching** of a vertex $i_{(1)}$ if aligning $i_{(1)}$ to $u_{(2)}$ maximizes the number of conserved neighbors of $i_{(1)}$ in comparison with aligning $i_{(1)}$ to other nodes in $\mathcal{V}_2$. The best matching of $u_{(2)}$ is defined in the same fashion.*

Definition 3.2 (Globally Most Similar)**.** *At the k^{th} iteration, a vertex $u_{(2)}$ is **globally most similar** to a vertex $i_{(1)}$ if $\mathbf{S}^{(k)}_{iu} \geq \mathbf{S}^{(k)}_{iv}$, $\forall v_{(2)} \in \mathcal{V}_2$. Likewise, $i_{(1)}$ is globally most similar to $u_{(2)}$ if $\mathbf{S}^{(k)}_{iu} \geq \mathbf{S}^{(k)}_{ju}$, $\forall j_{(1)} \in \mathcal{V}_1$.*

Note that both definitions are one-way. In other words, if $i_{(1)}$ is globally most similar to $u_{(2)}$, it does not imply that $u_{(2)}$ is also globally most similar to $i_{(1)}$.

Theorem 3.1. *Given a permutation matrix $\mathbf{P}$ for which all nodes are aligned to their best matchings, $\mathbf{P}$ is the optimal solution of Eq. (1).*

Proof. For the sake of contradiction, suppose there exist a permutation matrix $\bar{\mathbf{P}} \neq \mathbf{P}$ such that

$$trace(\bar{\mathbf{P}}^T \mathbf{A} \bar{\mathbf{P}} \mathbf{B}^T) > trace(\mathbf{P}^T \mathbf{A} \mathbf{P} \mathbf{B}^T) \tag{2}$$

Let $\mathbf{B}'$ and $\mathbf{B}''$ denote $\bar{\mathbf{P}}^T\mathbf{A}\bar{\mathbf{P}}$ and $\mathbf{P}^T\mathbf{A}\mathbf{P}$, respectively. Inequality (2) implies that

$$\mathbf{B}'_{i,*}\mathbf{B}^T_{*,i} > \mathbf{B}''_{i,*}\mathbf{B}^T_{*,i},\ \exists\ i \in \mathcal{V}_1 \tag{3}$$

where $\mathbf{B}^T_{i,*}$ and $\mathbf{B}^T_{*,i}$ denotes the i^{th} row and column of $\mathbf{B}^T$, respectively. However, the inequality (3) implies that there exists a vertex $i_{(1)}$ who is not aligned with its best matching which is a contradiction □

By Theorem 3.1, nodes are desired to aligned with their best matchings. Our hope is that the node who is globally most similar to $i_{(1)}$, provided by the similarity matrix, should correspond to the best matching of $i_{(1)}$. Given a pair $(i_{(1)}, u_{(2)})$, consider computing their similarity as a process of aligning their neighbors. A pair of cross-network neighbors $(j_{(1)}, v_{(2)})$ can contribute its similarity to $\mathbf{S}_{i,u}$ if $j_{(1)}$ can be aligned with $v_{(2)}$. As a result, $i_{(1)}$ and $u_{(2)}$ have a higher similarity if they have more neighbors that can be aligned. The one-to-one nature of alignments leads to the first rule:

> **Rule 1.** Given a pair of cross-network vertices $(i_{(1)}, u_{(2)})$, a neighbor $j_{(1)}$ of $i_{(1)}$ can contribute its similarity (with some neighbor of $u_{(2)}$) to $\mathbf{S}_{i,u}$ **at most once**. Similarly, a neighbor $v_{(2)}$ of $u_{(2)}$ can contribute its similarity (with some neighbor of $i_{(1)}$) to $\mathbf{S}_{i,u}$ **at most once**.

Rule 1 provides effective ways to penalize the degree differences as shown in the later section. Ideally, a pair of cross-network neighbors can be aligned if at least one of them is globally most similar to the other, however during the first several iterations of the algorithm, the computed similarities are less reliable. In other words, we are less certain about whether the node that is globally most similar to $j_{(1)}$ is indeed its best matching. However, as we proceed with more iterations, the reliability of similarities increases. To model this, we define vectors **b1** and **b2** for two networks respectively such that $\mathbf{b1}_i^{(k)} = \max_{u\in\mathcal{V}_2} \mathbf{S}_{i,u}^{(k)}$ and $\mathbf{b2}_u^{(k)} = \max_{i\in\mathcal{V}_1} \mathbf{S}_{i,u}^{(k)}$. Also, we define *contribution-threshold* vectors, **c1** and **c2**, for two networks. Given a pair of vertices $(i_{(1)}, u_{(2)})$, a pair of their cross-network neighbors $(j_{(1)}, v_{(2)})$ can only contribute similarity to $\mathbf{S}_{i,u}$ if $\mathbf{S}_{j,v}^{(k)} \geq \min\{\mathbf{c1}_j^{(k)}, \mathbf{c2}_v^{(k)}\}$. At the same time, such thresholds grows as the algorithm proceeds with more iterations.

Computing the similarity between $(i_{(1)}, u_{(2)})$ iteratively can be seen as a process of gathering information (regarding the similarity) from other nodes in a breadth-first search (BFS) manner such that $\mathbf{S}_{i,u}^{(k)}$ is computed based on similarities between cross-network nodes that are within distance k away from $i_{(1)}$ and $u_{(u)}$. A node $j_{(1)}$ has been *visited* by $i_{(1)}$ at the iteration k if $j_{(1)}$ is within distance k away from $i_{(1)}$. We use the fraction of visited nodes after each iteration as a simple measure to model the increase of the contribution threshold.

Definition 3.3 (Contribution Threshold)**.** *Given $\mathcal{G}_1$ with size n_1, let* **T1** *be the $n_1 \times t_{max}$ matrix for which $\mathbf{T1}_{i,k}$ is the fraction of nodes that $i_{(1)}$ has visited after the k^{th} iteration. Then the* ***contribution threshold*** *of $i_{(1)}$, denoted by $\mathbf{c1}_i$, after the k^{th} iteration is defined as*

$$\mathbf{c1}_i^{(k)} = \mathbf{b1}_i^{(k)} \times \mathbf{T1}_{i,k} \tag{4}$$

$\mathbf{T1}_{i,0} = 1/n_1$ for all $i_{(1)} \in \mathcal{V}_1$. As k approaches t_{max}, $\mathbf{T1}_{i,k}$ approaches 1 and $\mathbf{c}_i^{(k)}$ approaches $\mathbf{b}_i^{(k)}$. **c2** and **T2** are defined for $\mathcal{G}_2$ in the same fashion. The pseudocode for computing **T1** and **T2** is shown in Algorithm 1.

Algorithm 1: Contribution Threshold

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, t_{max}
Output: Contribution threshold matrix $\mathbf{T}$

$\mathbf{T} \leftarrow n \times t_{max}$ empty matrix ▷ $|\mathcal{V}| = n$
for i ***in*** $\mathcal{V}$ **do**
  $\mathbf{d} \leftarrow n \times 1$ vector with all entries equal to 0
  $\mathbf{d}_i = 1$
  $num\_of\_visited\_node \leftarrow 1$
  $frontier \leftarrow$ empty list.
  $frontier$.**insert**(i)
  for $k \leftarrow 1$ ***to*** t_{max} **do**
    $new\_frontier \leftarrow$ empty list.
    for j ***in*** $frontier$ **do**
      for q ***in*** $\mathcal{N}(j)$ **do**
        if $\boldsymbol{d}_q == 0$ **then**
          $num\_of\_visited\_node += 1$
          $new\_frontier$.**insert**(q)
          $\mathbf{d}_q = 1$
    $\mathbf{T}_{i,k} = \frac{num\_of\_visited\_node}{n}$
    $frontier = new\_frontier$
return $\boldsymbol{T}$

Finally, let $(j_{(1)}, v_{(2)})$ be a pair of cross-network neighbors of $(i_{(1)}, u_{(2)})$. Without loss of generality, suppose $\mathbf{S}_{j,v}^{(k)} \geq \mathbf{c1}_j^{(k)}$ but $\mathbf{S}_{j,v}^{(k)} < \mathbf{c2}_v^{(k)}$. This implies that there must exists a better alignment with higher similarity (than $\mathbf{S}_{j,v}^{(k)}$) for $v_{(2)}$ at the k^{th} iteration. We still want to accumulate the similarity between $j_{(1)}$ and $v_{(2)}$ to $\mathbf{S}_{i,u}^{(k)}$, at the same time, we should also consider the loss of similarity by aligning $j_{(1)}$ to $v_{(2)}$ (recall that we model similarity accumulation as a process of aligning neighbors). We introduce a simple measure called *net similarity*:

$$\mathbf{N}_{jv}^{(k)} = 2\mathbf{S}_{j,v}^{(k)} - \left[\frac{\mathbf{S}_{j,v}^{(k)} - \mathbf{c1}_j^{(k)}}{\mathbf{b1}_j^{(k)} - \mathbf{c1}_j^{(k)}} \cdot (\mathbf{b2}_v^{(k)} - \mathbf{c2}_v^{(k)}) + \mathbf{c2}_v^{(k)} \right] \tag{5}$$

and if $\mathbf{S}_{j,v}^{(k)} \geq \mathbf{c2}_v^{(k)}$ but $\mathbf{S}_{j,v}^{(k)} < \mathbf{c1}_j^{(k)}$:

$$\mathbf{N}_{jv}^{(k)} = 2\mathbf{S}_{j,v}^{(k)} - \left[\frac{\mathbf{S}_{j,v}^{(k)} - \mathbf{c2}_v^{(k)}}{\mathbf{b2}_v^{(k)} - \mathbf{c2}_v^{(k)}} \cdot (\mathbf{b1}_j^{(k)} - \mathbf{c1}_j^{(k)}) + \mathbf{c1}_j^{(k)} \right] \tag{6}$$

which leads to our second rule:

Rule 2. Given a pair of cross-network vertices $(j_{(1)}, v_{(2)})$, under rule 1, the amount of

similarity they can contribute to pair of neighbors $(i_{(1)}, u_{(2)})$ is :

$$\begin{cases} \mathbf{S}_{j,v}^{(k)} & \text{if } \mathbf{S}_{j,v}^{(k)} \geq \max\{\mathbf{c1}_j^{(k)},\ \mathbf{c2}_v^{(k)}\} \\ \mathbf{N}_{j,v}^{(k)} & \text{if } \min\{\mathbf{c1}_j^{(k)},\ \mathbf{c2}_v^{(k)}\} \leq \mathbf{S}_{j,v}^{(k)} \leq \max\{\mathbf{c1}_j^{(k)},\ \mathbf{c2}_v^{(k)}\} \\ 0 & \text{otherwise} \end{cases}$$

Given the first two rules, it is possible that a neighbor of $u_{(2)}$ is globally most similar to multiple neighbors of $i_{(1)}$, but with different similarities. For example, consider a sample graph shown in Figure 1 where $v_{(2)}$ is globally most similar to both $j_{(1)}$ and $k_{(1)}$. Dashed lines indicate the similarities between two vertices.

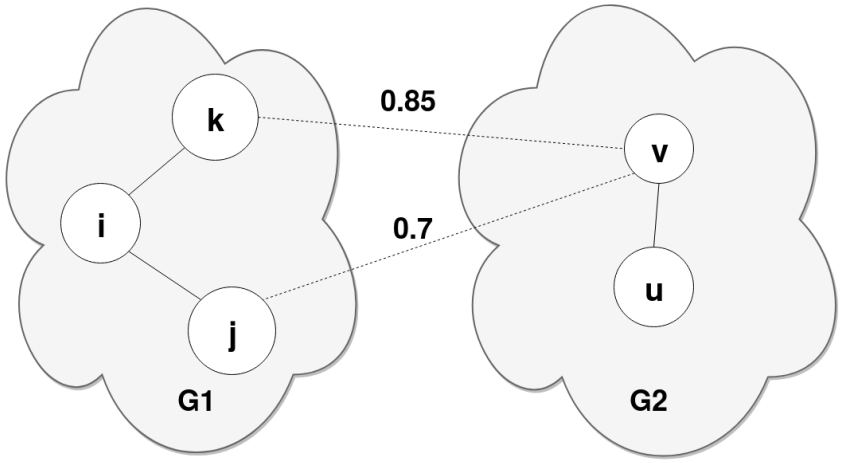

Figure 1: Example of rule 3

In this case, we ought to consider the pair $(k_{(1)}, v_{(2)})$ and contribute its similarity to $\mathbf{S}_{i,u}$ which leads to the final rule:

> **Rule 3.** *To compute the similarity between $i_{(1)}$ and $u_{(2)}$, a pair of cross-network neighbors with a* ***higher similarity should be given the prior consideration****.*

The Similarity Computation Algorithm The general idea of the algorithm is to update **S**, **b1** and **b2** iterativley based on their values in the previous iteration. The initial similarities between each pair of cross-network vertices are uniformly distributed. We set them all equal to 1.

In each iteration, for each pair of cross-network vertices $(i_{(1)}, u_{(2)})$, we first check all pairs of their cross-network neighbors against Rule 2 to determine which pairs are qualified such that the similarity is greater than the contribution threshold of at least one node in the pair. Then we sort those pairs by similarities in descending order which is needed to follow Rule 3. After sorting, we go over each $(j_{(1)}, v_{(2)})$ in the sorted order, check $j_{(1)}$ and $v_{(2)}$ against rule 1. If none of them has contributed to $\mathbf{S}_{i,u}$ before, we accumulate the similarity between $(j_{(1)}, v_{(2)})$ based on Rule 2 and mark $j_{(1)}$ and $v_{(2)}$ as *selected* which indicates that they can no longer be considered. This step enforces Rule 1. Note that the *selected* neighbors will no longer be selected after we are done computing $\mathbf{S}_{i,u}$. At last, we update **S**, **b1** and **b2**.

After accumulating similarities from neighbors, we normalize it by:

$$\mathbf{S}_{iu}^{(k+1)} = \frac{\text{accumulated similarity}}{\max\{\sum_{j\in\mathcal{N}(i)} \mathbf{b1}_j^{(k)}, \sum_{v\in\mathcal{N}(u)} \mathbf{b2}_v^{(k)}\}}$$

This normalization also penalizes the degree discrepancy between $i_{(1)}$ and $u_{(2)}$ because the maximum number of pairs of cross-network neighbors that can contribute similarity to $(i_{(1)}, u_{(2)})$ is upper bounded by the smaller degree between $i_{(1)}$ and $u_{(2)}$. The detailed algorithm is summarized in Algorithm 2.

Algorithm 2: Similarity Computation

Input: $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$, $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$, t_{max}, $\mathbf{T1}, \mathbf{T2}$
Output: The similarity matrix $\mathbf{S}$

for $k \leftarrow 1$ ***to*** t_{max} **do**
  $\mathbf{S}^{(k)} \leftarrow n_1 \times n_2$ similarity matrix
  $\mathbf{b1}^{(k)} \leftarrow n_1 \times 1$ vector with all entries equal to -1
  $\mathbf{b2}^{(k)} \leftarrow n_2 \times 1$ vector with all entries equal to -1
  Update $\mathbf{c1}^{(k-1)}$ and $\mathbf{c2}^{(k-1)}$ based on equation 4
  for i ***in*** $\mathcal{V}_1$ **do**
    for u ***in*** $\mathcal{V}_2$ **do**
      $\mathbf{e} \leftarrow$ empty associative array
      $sum \leftarrow 0$
      for j ***in*** $\mathcal{N}(i)$ **do**
        for v ***in*** $\mathcal{N}(u)$ **do**
          if $\mathbf{S}_{jv}^{(k-1)} \geq \min\{\mathbf{c1}_j^{(k-1)}, \mathbf{c2}_v^{(k-1)}\}$ **then**
            $\mathbf{e}[(j, v)] \leftarrow \mathbf{S}_{jv}^{(k-1)}$
      $\mathbf{e} \leftarrow$ sort by ***value*** in descending order
      for (j, v) ***in*** $\mathbf{e}$.***keys*** **do**
        if *j and v are not selected* **then**
          Accumulate sum based on rule 2
          Mark j and v as selected
      $\mathbf{S}_{iu}^{(k)} \leftarrow \frac{sum}{\max\{\sum_{j\in\mathcal{N}(i)} \mathbf{b1}_j^{(k-1)}, \sum_{v\in\mathcal{N}(u)} \mathbf{b2}_v^{(k-1)}\}}$
      if $\mathbf{S}_{iu}^{(k)} > \mathbf{b1}_i^{(k)}$ **then**
        $\mathbf{b1}_i^{(k)} = \mathbf{S}_{iu}^{(k)}$
      if $\mathbf{S}_{iu}^{(k)} > \mathbf{b2}_u^{(k)}$ **then**
        $\mathbf{b2}_u^{(k)} = \mathbf{S}_{iu}^{(k)}$
return ***S***

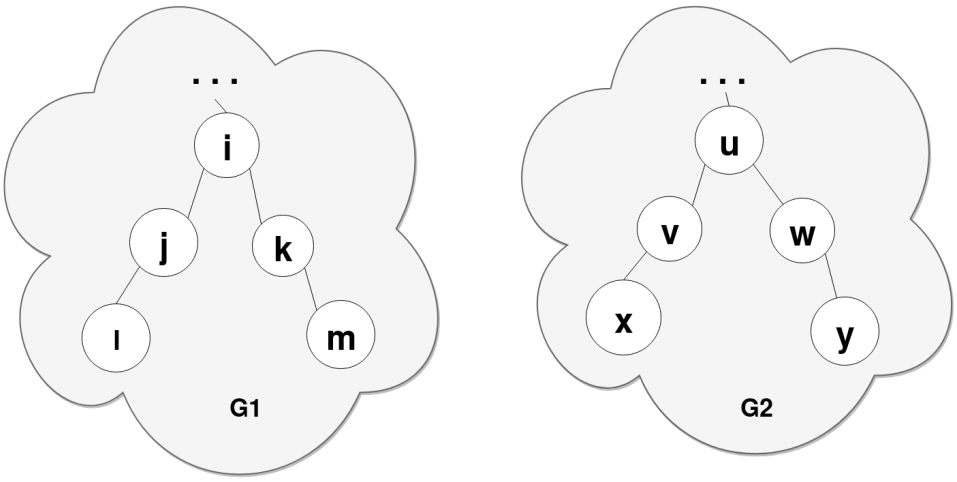

Figure 2: Naive alignment fails to distingush symmetric nodes

3.2 Building Alignments

We use two methods to extract the mappings between vertices from two networks, namely, naive and seed-and-extend alignments.

3.2.1 Naive alignment

Suggested by [15], we sort all pairs of cross-network vertices by similarities in descending order. Then iteratively align the next pair of unaligned vertices.

While this methods gives alignments with good qualities, we observe that it fails to distinguish nodes that are symmetric. As an example shown in Figure 2 where $\mathcal{G}_1$ and $\mathcal{G}_2$ are isomorphic. Under this circumstance, $k_{(1)}$ is equally similar to $v_{(2)}$ and $w_{(2)}$. At the same time, $m_{(1)}$ is equally similar to $x_{(2)}$ and $y_{(2)}$. If we break ties randomly, then it is possible that $k_{(1)}$ and $m_{(1)}$ are mapped to vertices on different sides of the tree which causes the loss of the number of conserved edges.

3.2.2 Seed-and-extend approach

Aligned nodes could serve as guidance for aligning other nodes. Back to the Figure 2 example, aligning $k_{(1)}$ to $w_{(2)}$ should imply that $m_{(1)}$ ought to be aligned with $y_{(2)}$ rather than $x_{(2)}$. To fulfill this, we first find the pair of unaligned nodes with the highest alignment score. Then align them and increase similarity scores between every pair of their unaligned cross-network neighbors by some constant. Iterations proceed until all nodes in the smaller network are aligned. For efficiency, we use red-black tree to store pairs of nodes.

3.3 Time complexity

Let $O(n_1) = O(n_2) = O(n)$ and $O(m_1) = O(m_2) = O(m)$. Let t_{max} denote the total number of iterations. It is easy to see that Algorithm 1 runs in $O(n^2 + mn)$ time and the naive alignment method runs in $O(n^2 \log n)$ time.

Lemma 3.1. *The time complexity of Algorithm 2 is* $O(t_{max}m^2 \log n)$

Proof. Let d_i denote the degree of vertex $i_{(1)}$. Operations on lines 12 to 13 and lines 19 to 23 take constant time thus the nested for loops on lines 10 to 13 take $\Theta(d_i d_u)$ time for every pair of $i_{(1)}$ and $u_{(2)}$. Sorting on line 14 takes $\Theta(d_i d_u \log d_i d_u)$ time and the for loop on lines 15 to 18 takes $\Theta(d_i d_u)$ time. We observe that $O(\log d_i d_u) = O(\log n)$, therefore, the outer nested for loop on lines 6 to 23 takes:

$$\begin{aligned} O(\sum_{i\in\mathcal{V}_1}\sum_{u\in\mathcal{V}_2} d_i d_u log(d_i, d_u)) &= O(\log n \sum_{i\in\mathcal{V}_1} d_i \sum_{u\in\mathcal{V}_2} d_u) \\ &= O(m^2 \log n) \end{aligned} \tag{7}$$

Finally, the time complexity of the Algorithm 2 is $O(t_{max}m^2 \log n)$. □

The time complexity of the seed-and-extend alignment method is $O((n^2 + mn)\log(n^2 + mn))$. Please see the supplementary data for proof.

t_{max} equals to the diameter of the larger network. Without loss of generality, we treat t_{max} as a constant. At the same time, we assume $O(m) = O(n \log n)$ which is a fair assumption for the networks that are used in the experiments. Therefore, the overall running time of our proposed algorithm is $O(n^2(\log n)^3)$.

4 Proposed Selection Rule for Local Search

In this section, we first discuss the naive local search then introduce a novel way to quantify the mismatching of vertices for a better selection rule.

4.1 Baseline

Local search algorithms are well-studied in solving combinatorial optimization problems [1]. Given the permutation matrix produced by an alignment algorithm, we construct its neighborhood by selecting a subset of vertices from the smaller network and generate all permutations of their alignments while fixing the alignments of all other vertices that are not in the subset [12].

Given $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$, we transform the alignment matrix $\mathbf{P}$ to the $n_1 \times 1$ alignment vector $\widetilde{\Pi}$ for which $\widetilde{\Pi}_i = u_{(2)}$ implies vertex $i_{(1)}$ is aligned to vertex $u_{(2)}$.

For each iteration, we select a subset of vertices $\mathcal{V}_1' \subset \mathcal{V}_1$ with a fixed cardinality. Let $\mathcal{V}_2' = \{\widetilde{\Pi}_i : i \in \mathcal{V}_1'\}$. Local search explores all neighbors and attempts to find a new alignment vector Π with a lower objective:

$$\begin{aligned} \min_{\Pi}\{-\sum_{i,j\in\mathcal{V}_1'}(\mathbf{A}_{ij}\mathbf{B}_{\Pi_i\Pi_j} + \sum_{k\in\mathcal{N}(i)|k\notin\mathcal{V}_1'}\mathbf{A}_{ik}\mathbf{B}_{\Pi_i\widetilde{\Pi}_k} + \\ \sum_{p\in\mathcal{N}(j)|p\notin\mathcal{V}_1'}\mathbf{A}_{jp}\mathbf{B}_{\Pi_j\widetilde{\Pi}_p})\} \\ \text{s.t. } \Pi_i \in \mathcal{V}_2' \ \ \forall i \in \mathcal{V}_1' \end{aligned} \tag{8}$$

4.2 Quantify the mismatching

Depending on the quality of the initial solution, it is possible that only a small fraction of vertices are not mapping optimally and constructing the subset $\mathcal{V}_1'$ with random selections from the entire vertex set is not efficient. Therefore, locating vertices that are mismatched becomes important.

Start from computing the levels of mismatching for vertices based on the neighborhoods.

Definition 4.1 (Violation)**.** *The number of **violation** of a vertex $i_{(1)} \in \mathcal{G}_1$ is defined as*

$$\begin{aligned} \mathbf{o}_i &= \sum_{j \in \mathcal{N}(i)} (1 - \mathbf{B}_{\widetilde{\Pi}_i, \widetilde{\Pi}_j}) \\ &= |\mathcal{N}(i)| - \sum_{j \in \mathcal{N}(i)} \mathbf{B}_{\widetilde{\Pi}_i, \widetilde{\Pi}_j} \end{aligned} \tag{9}$$

Let $\bar{\mathcal{V}}_2 = \{u_{(2)} \in \mathcal{V}_2 \ : \ \widetilde{\Pi}_i = u_{(2)} \ \exists \ i_{(1)} \in \mathcal{V}_1\}$ be the subset of $\mathcal{V}_2$ consisting all aligned vertices. Let $\widetilde{\Pi}^{-1} : \bar{\mathcal{V}}_2 \to \mathcal{V}_1$ be the inverse mapping, then the total number of violations of vertex $u \in \mathcal{V}_2$ is defined in the same fashion:

$$\begin{aligned} \mathbf{o}_u &= \sum_{v \in \mathcal{N}(u)} (1 - \mathbf{A}_{\widetilde{\Pi}_u^{-1}, \widetilde{\Pi}_v^{-1}}) \\ &= |\mathcal{N}(u)| - \sum_{v \in \mathcal{N}(u)} \mathbf{A}_{\widetilde{\Pi}_u^{-1}, \widetilde{\Pi}_v^{-1}} \end{aligned} \tag{10}$$

We normalize violations of vertices by the degrees. For two real world networks where isomorphism does not exist, we expect many vertices to have nonzero violations. Obviously, a nonzero violation does not imply a non-optimal mapping. It it worth noting that a zero violation does not always implies a optimal mapping as shown in the Figure 3. where dashed lines indicate mappings. Note that $i_{(1)}$ has zero violation. This suggests the insufficiency of local violation values.

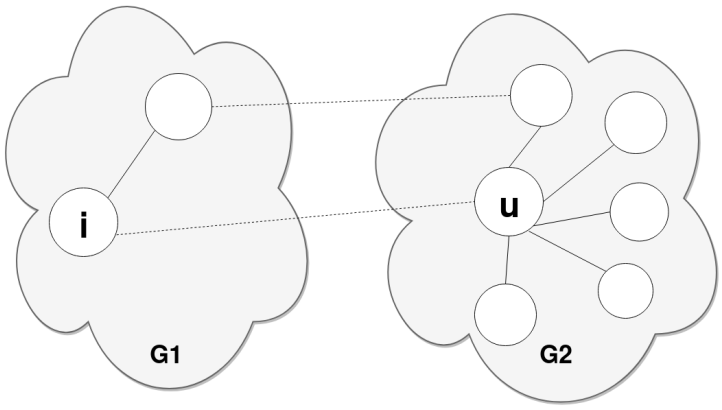

Figure 3: Zero violation of vertex i

To adapt the network information into this model, we use a iterative approach based on the intuition that *a vertex is more mismatched if its neighbors are more mismatched.* One immediate

approach is to propagate violations via edges. Let $\mathcal{G}_2' = (\bar{\mathcal{V}}_2, \bar{\mathcal{E}}_2)$ be the subgraph induced by $\bar{\mathcal{V}}_2$. To start with, we merge $\mathcal{G}_1$ and $\mathcal{G}_2'$ by adding edges to connect aligned cross-network vertices. For simplicity, suppose nodes in $\mathcal{V}_1$ are label with consecutive integers starting from 1, and nodes in $\mathcal{V}_2$ are label with consecutive integers starting from $|\mathcal{V}_1|+1$. Denote the newly constructed undirected graph $\mathcal{G}_3 = (\mathcal{V}_3, \mathcal{E}_3)$ where $\mathcal{V}_3 = \mathcal{V}_1 \cup \bar{\mathcal{V}}_2$ and $\mathcal{E}_3 = \mathcal{E}_1 \cup \bar{\mathcal{E}}_2 \cup \{(i_{(1)}, u_{(2)}) : i_{(1)} \in \mathcal{V}_1, u_{(2)} = \widetilde{\Pi}_i\}$. Let $\mathbf{C}$ denote the adjacency matrix of $\mathcal{G}_3$. Let $\mathbf{R}$ be the vector which encode the level of mismatching for each vertices. Let $\mathbf{D}$ denote the diagonal degree matrix such that $\mathbf{D}_{i,i} = |\mathcal{N}(i)|$. We propagate violations in a PageRank fashion [13]:

$$\mathbf{R}_i^{(k)} = \alpha \sum_j \frac{\mathbf{C}_{ij}}{\mathbf{D}_{j,j}} \mathbf{R}_j^{(k-1)} + (1-\alpha)\mathbf{o}_i. \tag{11}$$

In matrix notation:

$$\mathbf{R}^{(k)} = \alpha \mathbf{C}\mathbf{D}^{-1}\mathbf{R}^{(k-1)} + (1-\alpha)\mathbf{o} \tag{12}$$

By normalizing the violation vector $\mathbf{o}$, we can rewrite eq. (12):

$$\mathbf{R}^{(k)} = \left[\alpha \mathbf{C}\mathbf{D}^{-1} + (1-\alpha)\mathbf{o}\mathbf{1}^T\right] \mathbf{R}^{(k-1)} \tag{13}$$

where $\mathbf{1}$ is the vector with all entries equal to 1.

The Eq. (13) encodes a eigenvalue problem which can be approximated by power iteration. Let $\mathbf{E} = \alpha \mathbf{C}\mathbf{D}^{-1} + (1-\alpha)\mathbf{o}\mathbf{1}^T$. By the undirected nature of $\mathcal{G}_3$, the transition matrix $\mathbf{C}\mathbf{D}^{-1}$ is *irreducible*, therefore $\mathbf{E}$ is a left-stochastic matrix with leading eigenvalue equal to 1 and the solution of Eq. (13) is the principle eigenvector of $\mathbf{E}$. On top of that, $\mathbf{E}$ is also primitive therefor the leading eigenvalue of $\mathbf{E}$ is unique and the corresponding principle eigenvector can be chosen to be strictly positive. As a result, the power iteration converge to its principle eigenvector.

Violation vector $\mathbf{o}$ plays the role of teleportation distribution which encodes external influences on the importance of vertices. We set the damping factor $\alpha = 0.5$ by experiments.

The converged $\mathbf{R}$ gives the levels of mismatching of vertices and a vertex with a higher value is more mismatched.

4.3 Proposed Local Search

We first sort the vertices in $\mathcal{G}_1$ by their level of mismatching computed in $\mathbf{R}$ in descending order, then use a sliding window over sorted vertices to narrow search space. Let N be the size of the window with the tail lying at the highest-ranked vertices. We construct $\mathcal{V}_1'$ by randomly selecting vertices within the window. If the objective has not been improved for k iterations, we move the window ℓ nodes forward. The local search process terminates if the objective has not been improved for k_{max} number of iterations. In our experiments, we set $|\mathcal{V}_1'| = 6$.

5 Experimental Results

We evaluate the performance of our algorithms in comparison to IsoRank [13], NETAL [10] and HubAlign [6] on synthetic networks and real-world networks. To quantify the alignment quality, we use the *edge correctness* (EC) which is defined as [10]:

Table 1: Data set description

Domain	$\lvert\mathcal{V}_1\rvert$	$\lvert\mathcal{E}_1\rvert$	Title
BA random	400	2751	barabasi
HK random	400	2732	homle
Coauthorships	379	914	co-auth
Gene functional association	993	1300	bio 1
Economic network	1258	7513	econ
Network of routers	2113	6632	router
Gene functional association	2831	4562	bio 2
Twitter retweet network	4171	7059	retweet
Erdos collaboration	5019	7536	erdos
Reality mining network	6809	7680	reality

$$EC = \frac{|\{(i_{(1)}, j_{(1)}) \in \mathcal{E}_1 \ : \ ((f(i_{(1)}), f(j_{(1)})) \in \mathcal{E}_2\}|}{|\mathcal{E}_1|} \tag{14}$$

Note that the numerator of EC equals to the total number of conserved edges which is also the objective that we try to maximize.

5.1 Self-alignment under noise

Simple perturbation of the original network $\mathcal{G}_1$ is not a difficult task for the existing state of the art network alignment algorithms. The most interesting test cases that typically demonstrate their weak sides are networks augmented with noisy edges. They also reflect many real-life network alignment scenarios [3]. Given the original network $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$, a noisy permutation of $\mathcal{G}_1$ with noise level p, denoted by $\mathcal{G}_2^{(p)} = (\mathcal{V}_2, \mathcal{E}_2)$ is created with two steps:

1. Permute $\mathcal{G}_1$ with some randomly generate permutation matrix.
2. Add $p|\mathcal{E}_1|$ edges to $\mathcal{G}_1$ uniformly at random by randomly choosing nonadjacent pairs of vertices.

Note that under this model, the highest EC any algorithm can achieve by aligning $\mathcal{G}_1$ and $\mathcal{G}_2$ is always 1. *In this experiment we demonstrate the results without local search, i.e., we demonstrate how our initial solution outperforms state of the art methods.*

We first solve the alignment problems on random networks generated by Baràbasi-Albert preferential attachment (BA model) [2] and Holme-Kim model (HK model) [7], see Table 1. The HK model reinforces BA model with an additional probability q of creating a triangle after connecting a new node to an existing node. For our experiment we set $q = 0.4$. For each network, we generate 12 noisy permutations with increasing noise level p from 0 to 0.21. We then align $\mathcal{G}_1$ with each of its permutations using the proposed and baseline algorithms. The results are summarized in the first row of Figure 6. The proposed algorithm has two versions (`Naive alignment` and `Seed alignment`) that differ by the alignment method we use.

Next, we solve the alignment problem on eight real-world networks [11] from various domains. Information of networks are shown in Table 1. For each network, we use the same model to generate 14 noisy permutations with increasing noise level p from 0 to 0.25. We then align $\mathcal{G}_1$ with each of its permutation using the proposed and baseline algorithms. The results are summarized in Figure 6.

Noisy edges change the degrees of vertices by making them more uniformly distributed, i.e., performs a process that can be viewed as network anonymization. The observed results clearly demonstrate significant superiority of the proposed algorithm using seed and naive alignment methods over other methods under additional noisy conditions even when the noise levels are low. Iteratively accumulated neighborhood information with three rules makes our methods more robust to noises.

5.2 Alignment of Pairs of Real Networks

We evaluate the performance of the proposed algorithm on two pairs of real networks [11]. The first pair consists two subnetworks, $\mathcal{G}_1$ and $\mathcal{G}_2$, of a larger DBLP coauthorship network. $\mathcal{G}_1$ has 3134 nodes and 7829 edges, and $\mathcal{G}_2$ has 3875 nodes and 10594 edges. We extract $\mathcal{G}_1$ and $\mathcal{G}_2$ in a way that they have a common subgraph with 2455 nodes and 5162 edges. The second pairs of real networks consists of two gene functional associations networks of Caenorhabditis elegans. The first network has 2194 nodes and 2688 edges, and the second network has 2831 nodes and 4562 edges. The experimental results are summarized in Figure 4.

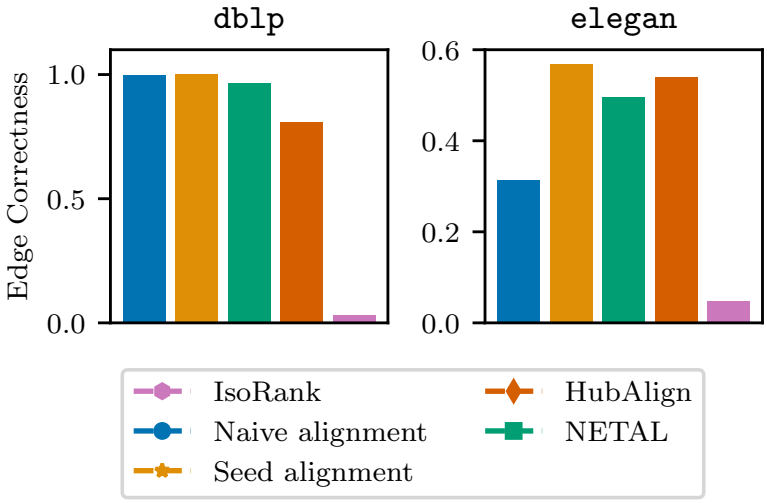

Figure 4: DBLP and Elegan

Our proposed algorithm with seed-and-extend alignment method outperforms all other methods. For the pair of DBLP networks, even though they only share a common subgraph (instead of being isomorphic), the proposed algorithm using two different alignment methods still identify the alignments with EC equals to 0.9907 and 0.9973, respectively. This suggests that this pair of DBLP networks have very similar underlying topology.

5.3 Running time

A better alignment quality comes with a price tag of the logarithmic factor to the total complexity. In the table of Figure 5, we demonstrate the running time in seconds when the algorithms attempt to solve the alignment on isomorphic networks (column $p = 0$), on and networks augmented with noisy edges (column $p = 0.25$). In spite of the logarithmic factor in our algorithms and 25% increase in the number of edges in one of the networks, we observe a significant increase in relative running

time of other state of the art algorithms except IsoRank whose quality is significantly worse. In the plot of Figure 5, we present the running time (in seconds) of our algorithm as a function of the size of both networks.

Algorithm	$p = 0$	$p = 0.25$
Naive	42	53 (+26%)
Seed	97	137 (+ 41%)
NETAL	10	25 (+ 150%)
Hubalign	3	53 (+ 1667%)
IsoRank	9	10 (+ 11%)

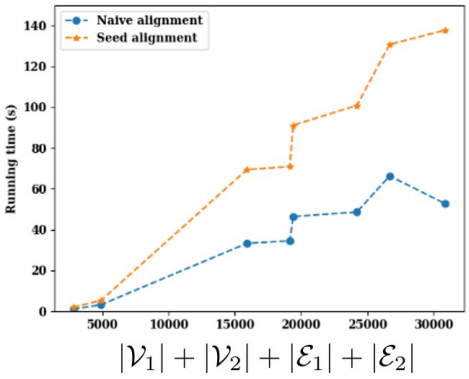

Figure 5: Running time comparison.

5.4 Experimental results on local search

We test the proposed local search scheme against the baseline for which the subset $\mathcal{V}_1'$ is generated by randomly selecting vertices from $\mathcal{V}_1$. We perform the experiments on a pair of isomorphic BA networks with 200 vertices and 1348 edges. We create two initial solutions with EC equal 0.6017 and 0.9614 respectively. We then run the proposed local search and baseline 20 times on each initial solution. On average, for the first initial solution, the proposed local search took 247 iterations to reach the global optimum whereas the baseline too 3145 iterations. For the second initial solution, the proposed local search takes 39 iterations to reach global optimum where native baseline takes 2132 iterations.

6 Conclusion and future work

We propose an iterative network alignment algorithm based on propagation of similarities. During each iteration, a pair of cross-network vertices accumulate similarities from cross-network neighbors by following three rules that are suggested in the paper. We also propose a novel selection rule for local search scheme which decreases the number of iterations it takes to reach local/global optimum. Experimental results show the superiority of our algorithm in terms of quality and scalability. Advanced local search schemes to reduce the number of iterations and faster similarity propagation are promising research directions.

References

[1] E Aarts, E. HL Aarts, and J.K Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[2] A. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[3] M. Bayati, M. Gerritsen, D. Gleich, A. Saberi, and Y. Wang. Algorithms for large, sparse network alignment problems. In *2009 Ninth IEEE International Conference on Data Mining*, pages 705–710, 2009.

[4] P. Hiram Guzzi and T. Milenković. Survey of local and global biological network alignment: the need to reconcile the two sides of the same coin. *Briefings in bioinformatics*, 19(3):472–481, 2017.

[5] S. Hashemifar, J. Ma, H. Naveed, S. Canzar, and J. Xu. Modulealign: module-based global alignment of protein–protein interaction networks. *Bioinformatics*, 32(17):i658–i664, 2016.

[6] S. Hashemifar and J. Xu. Hubalign: an accurate and efficient method for global alignment of protein–protein interaction networks. *Bioinformatics*, 30(17):i438–i444, 2014.

[7] P. Holme and B. Kim. Growing scale-free networks with tunable clustering. *Physical review E*, 65(2):026107, 2002.

[8] G W. Klau. A new graph-based method for pairwise global network alignment. *BMC bioinformatics*, 10(1):S59, 2009.

[9] D. Koutra, H. Tong, and D. Lubensky. Big-align: Fast bipartite graph alignment. In *2013 IEEE 13th ICDM*, pages 389–398, 2013.

[10] B. Neyshabur, A. Khadem, S. Hashemifar, and S. Arab. Netal: a new graph-based method for global alignment of protein–protein interaction networks. *Bioinformatics*, 29(13):1654–1662, 2013.

[11] Ryan A. Rossi and Nesreen K. Ahmed. An interactive data repository with visual analytics. *SIGKDD Explor.*, 17(2):37–41, 2016.

[12] Ruslan Shaydulin, Hayato Ushijima-Mwesigwa, Ilya Safro, Susan Mniszewski, and Yuri Alexeev. Network community detection on small quantum computers. *Advanced Quantum Technologies*, page 1900029, 2019.

[13] R. Singh, J. Xu, and B. Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proceedings of the National Academy of Sciences*, 105(35):12763–12768, 2008.

[14] J. Vogelstein, J. Conroy, V. Lyzinski, L. Podrazik, S. Kratzer, E. Harley, D. Fishkind, R. Vogelstein, and C. Priebe. Fast approximate quadratic programming for graph matching. *PLOS one*, 10(4):e0121002, 2015.

[15] S. Zhang and H. Tong. Final: Fast attributed network alignment. In *Proceedings of the 22nd ACM SIGKDD*, pages 1345–1354. ACM, 2016.

[16] S. Zhang, H. Tong, R. Maciejewski, and T. Eliassi-Rad. Multilevel network alignment. In *WWW*, pages 2344–2354. ACM, 2019.

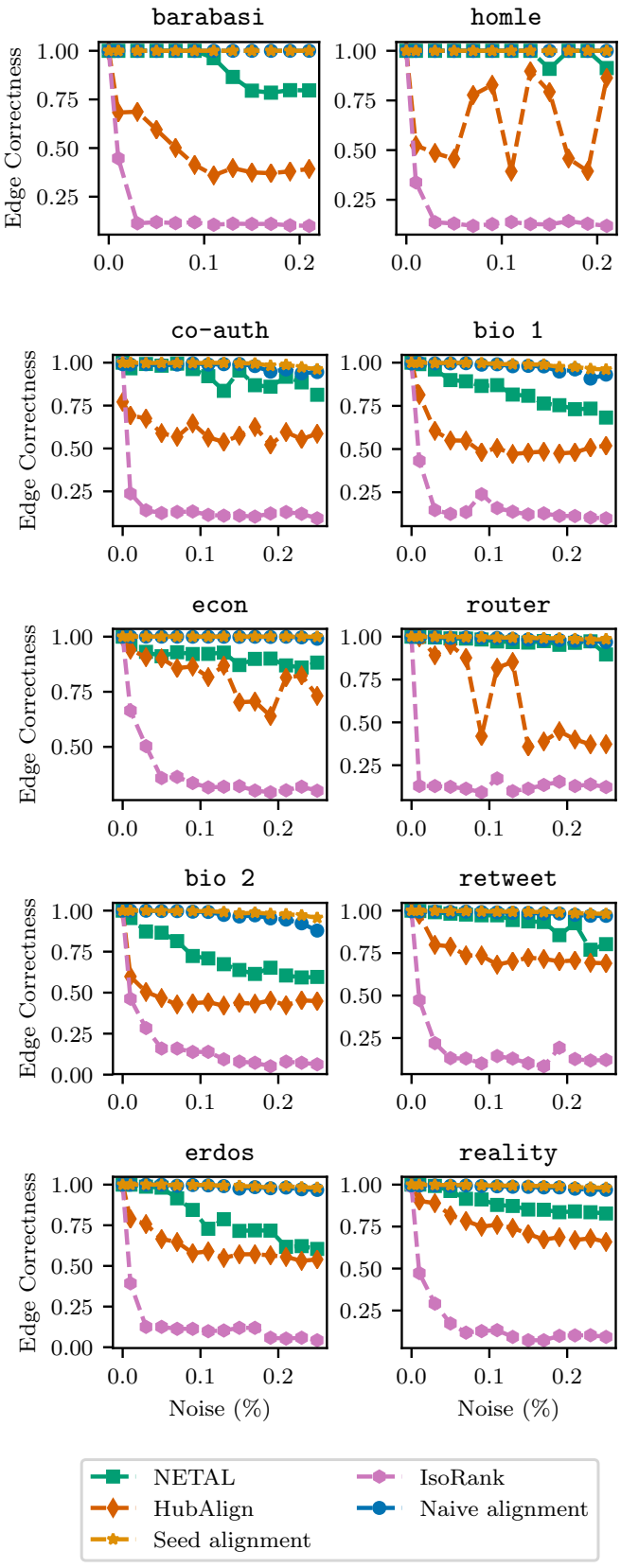

Figure 6: Alignment quality on random and real networks. The horizontal axis indicated the amount of introduced noise. The vertical axis is *EC*.