TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Graph Alignment Algorithms

by

Papamanolis Anastasios

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DIPLOMA DEGREE OF

ELECTRICAL AND COMPUTER ENGINEERING

March 2023

THESIS COMMITTEE

Professor Athanasios P. Liavas, *Thesis Supervisor*
Professor George N. Karystinos
Professor Michalis Zervakis

Abstract

Graph alignment is a computational problem which aims to find a correspondence between vertices of graphs that minimizes their node and edge disagreements. This problem arises in many fields, such as computational biology, where finding conserved functional components between species can lead to gene-disease associations or drug discoveries, social sciences, where unveiling unique users on different platforms can remove bots, and computer vision for recognising objects. Compared with the exact graph (sub)isomorphism problem often considered in a theoretical setting, the inexact graph alignment problem is often cast as a Quadratic Assignment Problem (QAP), which has attracted significant research interest. This thesis presents a comprehensive review of the recent research activity concerning the global pairwise one-to-one alignment, detailing the methodologies and formulations of four state-of-the-art graph alignment algorithms.

Specifically, Umeyama's method solves the orthogonal relaxation of the QAP by using spectral embeddings, IsoRank applies random walks with restarts on the normalized Kronecker product graph, LowRank-Align solves a rank-k approximation of the orthogonal relaxation of the QAP using an alternating optimization framework, and CONE-Align uses embeddings obtained with the NetMF method and aligns their subspaces using a Wasserstein-Procrustes framework.

To evaluate these algorithms, a standard experimental setup is adopted. A graph, G_1, is aligned with a "noisy" and permuted version of itself, G_2, in order to create an instance of the graph alignment problem. The noise level is the percentage of extra edges in G_2 with respect to the total edges in G_1. Only simple (unweighted, undirected, with no loops or multiple edges) graphs are considered. Several widely-used quality measures are used, such as the Node Correctness (NC), the Edge Correctness (EC), the Induced Conserved Structure (ICS), the Symmetric Substructure Score (S^3), the Matched Neighborhood Consistency (MNC), and the wall time.

The experiments suggest that CONE-Align performs well for both synthetic and real-world networks, LowRank-Align performs well in some occasions, Umeyama's method has moderate performance, but steady over datasets, and IsoRank is the overall worst. As for the quality measures, MNC and S^3 seem to be the most fair.

Table of Contents

List of Figures

Chapter 1

Introduction

Networks are intuitive and powerful structures that capture relationships between different entities in many domains, such as electric networks, social networks, biological networks, citation networks, and others. Studying and comparing the structural representations in the form of complex networks has been a very fruitful and fascinating research area [1]. The term graph alignment (or network alignment) includes several distinct but related problem variants [2]. In general, graph alignment aims to find a bijective mapping across two (or more) graphs so that, if two nodes are connected in one graph, their images are also connected in the other graph(s). The case of an exact graph alignment is known as the graph isomorphism problem [3]. However, in general, an exact alignment scheme may not be feasible. In such cases, graph alignment aims to find a mapping with the minimum error and/or the maximum overlap, i.e. the subgraph isomorphism, where a part of a graph is matched to a part of another graph. In its most general form, the graph alignment problem is equivalent to a Quadratic Assignment Problem (QAP) [4].

Network alignment can be used in numerous application domains. In bioinformatics, aligning different Protein-Protein Interaction (PPI) networks can result in identifying functionally similar regions across different species and benefit studying more sophisticated problems such as gene-disease associations [5]. In computer vision, object recognition can be attained by matching similar images [6]. In social analysis, network alignment is used to unveil unique users on different social platforms [7]. For knowledge completion, entities in knowledge graphs are aligned to construct a unified knowledge base [8].

While all graph alignment algorithms aim to find corresponding nodes in different networks, they may begin from different objective formulations. Typically, the optimization problem is to find a correspondence that maximizes a similarity score or minimizes the cost of transforming the second graph to the first. The most common input data are the adjacency matrices of the graphs to be aligned. Some algorithms may require additional information as input, such as preferred or known prior alignments. In the biology domain, such pre-processed input, e.g. BLAST scores, is needed as nodes typically have roles and semantics beyond structural properties.

This thesis is organized as follows. Chapter 2 provides basic definitions on the graph alignment problem and some useful mathematical formulations. Chapter 3 illustrates the connection of the Linear Assignment Problem (LAP) to the graph alignment problem. Chapter 4 includes a detailed analysis of four recent methods used for graph alignment. Chapter 5 presents the experimental setup and the performance of each studied algorithm for real-world and synthetic networks. The thesis concludes in Chapter 6.

Chapter 2

The Graph Alignment Problem

Graph theory has existed for almost three centuries as a principal branch in mathematics. It has been used for a long time in computer science, biology, chemistry, social sciences, engineering, and linguistics. The beginning of graph theory is considered the paper of Leonhard Euler on the Seven Bridges of Königsberg [9], published in 1736. Recently, graph theory has been a rapidly developing application and research area with many open problems. The focus of this thesis is the fundamental problem of graph alignment. A short review of the basic graph definitions and the graph alignment problem will be presented in this chapter.

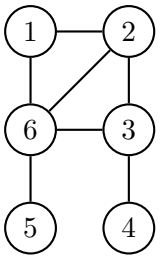
2.1 Graphs

Graphs are a widely used data structure for describing complex systems. In the most general setting, a graph is a set of objects (i.e., nodes), along with a set of interactions (i.e., edges) between pairs of these objects [3]. For example, a graph can represent a social network, by encoding individuals as nodes and their friendships as edges.

The abstraction of graphs gives them a powerful formalism. The same graph formalism can be used to represent, for example, social networks, co-authorship networks, interactions between proteins, or the connections between terminals in a telecommunications network. In addition to providing an elegant theoretical framework, graphs offer a mathematical foundation for analyzing, understanding, and learning from real-world complex systems [10]. With the recent increase in the quantity and quality of graph-structured data that led to the advent of large-scale social networking platforms, billions of interconnected web-enabled devices, massive scientific initiatives to model the interactome, and databases of molecule graph structures, there is no shortage of meaningful graph data for researchers to analyze.

Before discussing the graph alignment problem, we give a formal description of "graph data," following the basic definitions of [11]. Formally, a graph $G = (\mathcal{V}, \mathcal{E})$ is defined by a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ between these nodes. We denote an edge going from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$ as $(u, v) \in \mathcal{E}$. A graph can be directed or undirected. In the case of undirected graphs, it is assumed that the edge connecting the vertex u to the vertex v also connects the vertex v to the vertex u. This means that if $(u, v) \in \mathcal{E}$ then $(v, u) \in \mathcal{E}$. In general, this property does not hold for directed graphs. Undirected graphs can be considered as a special case of directed graphs.

For a given set of vertices and edges, the graph can be represented by an adjacency matrix $\mathbf{A}$. If there are n vertices, then $\mathbf{A}$ is an $n \times n$ matrix. The elements A_{uv} of the



$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Figure 2.1: A toy example of a simple graph and its adjacency matrix $\mathbf{A}$.

adjacency matrix $\mathbf{A}$ take values $A_{uv} \in \{0, 1\}$. More specifically,

$$A_{uv} = \begin{cases} 1, & \text{if } (u, v) \in \mathcal{E}, \\ 0, & \text{if } (u, v) \notin \mathcal{E}. \end{cases} \tag{2.1}$$

If the graph is undirected, then its adjacency matrix is symmetric, $\mathbf{A} = \mathbf{A}^T$, but if the graph is directed (i.e., edge direction matters), then $\mathbf{A}$ is not necessarily symmetric.

In general, the edges can be weighted. If the weights of the edges are defined, a weighted graph is obtained. The set of weights $\mathcal{W}$ corresponds to the set of edges $\mathcal{E}$. A weighted graph is a more general object than an unweighted graph. Commonly, it is assumed that the edge weights are nonnegative real numbers. If weight 0 is associated with the nonexisting edges, then the graph can be described with a weight matrix $\mathbf{W}$ similar to the adjacency matrix $\mathbf{A}$. A nonzero element W_{uv} determines the weight of the edge between vertices u and v. The value $W_{uv} = 0$ means that there is no edge between vertices u and v.

A degree matrix of a graph, denoted by $\mathbf{D}$, is a diagonal matrix where the main diagonal elements D_{uu} are equal to the sum of weights of all edges connected with the vertex u,

$$D_{uu} = \sum_{v \in \mathcal{V}} W_{uv}. \tag{2.2}$$

In the case of an unweighted and undirected graph, the value of D_{uu} is equal to the number of edges connected to the u-th vertex.

The normalized adjacency matrix is defined as

$$\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}. \tag{2.3}$$

The normalized adjacency matrix captures the local structure of a graph by scaling the entries of the adjacency matrix based on the degrees of the nodes in the graph. This normalization process ensures that each entry in the normalized adjacency matrix reflects not only the presence of a connection between two nodes, but also the relative importance of that connection based on the degrees of the nodes.

The Laplacian matrix captures structural properties of a graph because it encodes information about the connectivity patterns and the degree distribution of the nodes in the graph and is defined as

$$\mathbf{L} = \mathbf{D} - \mathbf{A}. \tag{2.4}$$

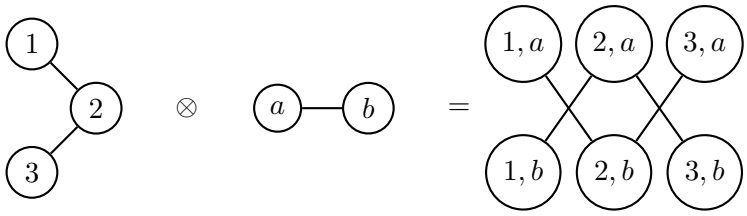


Figure 2.2: Kronecker (tensor) product of two graphs.

The diagonal entries of the Laplacian matrix are equal to the degree of each node in the graph, while the off-diagonal entries represent the strength of the connections between the nodes. Thus, the Laplacian matrix captures the degree distribution of the nodes in the graph, which is an important structural property. Furthermore, the Laplacian matrix reflects the connectivity patterns of the nodes in the graph by indicating which nodes are connected and the strength of the connections. This information can be used to identify clusters of nodes that are densely connected to each other, as well as nodes that are sparsely connected to the rest of the graph. This reflects the local structural properties of the graph.

The normalized Laplacian matrix is a variant of the Laplacian matrix that captures additional structural properties of a graph. The normalized Laplacian matrix is defined as

$$\tilde{\mathbf{L}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}. \tag{2.5}$$

where $\mathbf{I}$ is the identity matrix of appropriate size. The normalized Laplacian matrix captures additional structural properties of the graph that are not captured by the standard Laplacian matrix, such as the geometry, by normalizing the weights of the edges based on the degrees of the nodes, and the clustering of the nodes, by normalizing the diagonal entries of the Laplacian matrix based on the degree of the nodes.

In this thesis, only simple graphs are considered, where there are no self-loops and no multiple edges, and the edges have no weights and are all undirected. A toy example of a simple graph is illustrated in Figure 2.1.

Kronecker (tensor) product of two disjoint simple graphs $G_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $G_2 = (\mathcal{V}_2, \mathcal{E}_2)$ is a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{V}_1 \times \mathcal{V}_2$ and $((u_1, v_1), (u_2, v_2)) \in \mathcal{E}$ only if $(u_1, u_2) \in \mathcal{E}_1$ and $(v_1, v_2) \in \mathcal{E}_2$. The adjacency matrix of G, $\mathbf{A} \in \{0, 1\}^{n^2 \times n^2}$, is equal to the Kronecker product of the adjacency matrices $\mathbf{A}_1$ and $\mathbf{A}_2$,

$$\mathbf{A} = \mathbf{A}_1 \otimes \mathbf{A}_2. \tag{2.6}$$

The tensor product operator is commutative, associative, and has many other interesting and useful properties [12]. Performing a random walk on the tensor product graph $G = G_1 \otimes G_2$ is equivalent to performing two simultaneous random walks on G_1 and G_2. This means that the tensor product graph G can encode the commonalities between the two input graphs G_1 and G_2. Figure 2.2 shows an illustrative example of the tensor product between two simple graphs.

2.2 Embeddings

Node proximities in a graph refer to measures that quantify the similarity or relatedness between pairs of nodes in the graph. Formally, given a graph $G = (\mathcal{V}, \mathcal{E})$, a node proximity function is a mapping $p : \mathcal{V} \times \mathcal{V} \to \mathbb{R}^d$, where $d \ll |\mathcal{V}|$, that assigns a real-valued proximity score to each pair of nodes (i, j) in the graph. The proximity score can be interpreted as a measure of how "close" or "similar" the nodes i and j are in the graph. Edge weights W_{uv} are also called first-order proximities between nodes u and v, since they are the primary measures of similarity between nodes [13]. Likewise, the second-order proximity between a pair of nodes describes the similarity of the pair's neighborhood structure. Higher-order proximities are a family of measures that capture the similarity between nodes in a graph based on their higher-order relationships, rather than just their direct connections. Higher-order relationships refer to relationships between nodes that involve more than two hops or degrees of separation in the graph. The concept of higher-order proximities is closely related to the idea of graph transitivity, which refers to the tendency of nodes to form triangles in a graph. Higher-order proximities extend the concept of transitivity to capture relationships between nodes that involve more than just triangles. There are several types of higher-order proximities, including Rooted PageRank, Common Neighbors, Katz Index, Adamic Adar, etc. (for more details see [14]).

Graph embedding and node embedding are techniques used to transform a graph into a low-dimensional vector space. Graph embedding involves learning a low-dimensional representation of the entire graph, while node embedding involves learning a low-dimensional representation for each individual node in the graph. Formally, given a graph $G = (\mathcal{V}, \mathcal{E})$, graph embedding aims to learn a function $f : G \to \mathbb{R}^d$ that maps the entire graph to a d-dimensional vector space. The goal is to capture the global structure of the graph, including its communities, hubs, and other structural features. Node embedding, on the other hand, aims to learn a function $g : \mathcal{V} \to \mathbb{R}^d$ that maps each individual node in the graph to a d-dimensional vector space. The goal is to capture the structural properties of each node, including its neighbors, degree, and other properties. The functions f and g preserve some proximity measure defined on graph G [13].

In Chapter 4, the NetMF method [15], which is described in Algorithm 5, is used to obtain node embeddings that capture both the local and global structural properties of the graph. To achieve this, NetMF constructs a high-order proximity matrix that captures the structural information of the graph, and factorizes it using truncated Singular Value Decomposition (SVD) to obtain a low-dimensional representation of the graph. At first, NetMF approximates the normalized adjacency matrix $\tilde{\mathbf{A}}$ with its top-h eigenpairs $\mathbf{U}_h \mathbf{\Lambda}_h \mathbf{U}_h^T$. Then, given a window size T, the algorithm constructs a proximity matrix $\mathbf{M}$ by adding T different powers of matrix $\mathbf{\Lambda}_h$. Each power of matrix $\mathbf{\Lambda}_h$ captures different levels of structural information in the graph, and adding them together produces a proximity matrix that filters all the negative and small positive eigenvalues of matrix $\tilde{\mathbf{A}}$. The factorization of the DeepWalk matrix $\log \mathbf{M}$, where $\left[\log \mathbf{M}\right]_{ij} = \log M_{ij}$, is equivalent to the maximization of the objective function of LINE [16] and presents computational challenges due to the element-wise matrix logarithm. The matrix is not only

ill-defined (since $\log 0 = -\infty$), but also dense. The NetMF algorithm factorizes $\log \mathbf{M}'$, where $M'_{ij} = \max(M_{ij}, 1)$, by using SVD and constructs a node embedding matrix by using its top-d singular values/vectors.

2.3 Problem Definition

Graph[1] alignment (or matching) is the process of finding a correspondence between the nodes and the edges of two graphs that satisfies some (more or less strict) constraints, ensuring that similar substructures in one graph are mapped to similar substructures in the other. The problem can be classified [17] as:

- **Local or Global alignment**
  The target of local alignment is to identify the closely mapping subgraphs between different graphs. Typically, local alignment reports multiple subgraphs across graphs, which may be mutually inconsistent (i.e., a node might be mapped differently under each alignment). On the other hand, global alignment tries to match different graphs as a whole, and the output result is a single mapping between the nodes of the graphs.

- **Pairwise or Multiple alignment**
  Pairwise alignment compares two graphs, while multiple alignment considers more than two graphs at the same time. The computational complexity increases exponentially in the number of graphs [17].

- **One-to-one, One-to-many or Many-to-many alignment**
  One-to-one alignment produces one-to-one (or injective) node mappings, where a node from a given graph can be mapped to at most one node from another graph. Following the same logic, one-to-many alignment maps a node from a given graph to multiple nodes from another graph, while a node from the later graph can be mapped to at most one node from the former graph. A many-to-many alignment produces a many-to-many node mapping, where a node from a given graph can be mapped to several nodes from another graph.

2.4 Problem Formulation for Global Pairwise One-to-one alignment

For simplicity, this work considers a pair of simple graphs with the same number of nodes n. Let $G_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $G_2 = (\mathcal{V}_2, \mathcal{E}_2)$ be this pair, with adjacency matrices $\mathbf{A}_1 \in \{0,1\}^{n \times n}$ and $\mathbf{A}_2 \in \{0,1\}^{n \times n}$, respectively. Let their eigendecompositions be $\mathbf{A}_1 = \mathbf{U}_1 \mathbf{\Lambda}_1 \mathbf{U}_1^T$ and $\mathbf{A}_2 = \mathbf{U}_2 \mathbf{\Lambda}_2 \mathbf{U}_2^T$, respectively.

[1] **Graph or network?** Both terms are used interchangeably in this thesis, but a distinction should be made between the usage of these terms. According to their current popular usage, the term "graph" describes an abstract data structure, while the term "network" describes specific, real-world instantiations of this data structure (e.g., social networks). Network analysis is generally concerned with the properties of real-world data, whereas graph theory is concerned with the theoretical properties of the mathematically abstract graph.

The graph alignment problem can be divided into two broad categories: exact and inexact (for more information see [18] and [19]). The case of exact graph alignment is known as the graph isomorphism problem. Inexact graph alignment refers to problems where exact matching is impossible, which is the most common case for real-world applications.

2.4.1 Graph Isomorphism Problem

An isomorphism from G_1 to G_2 is a one-to-one mapping π, from the vertices of the first graph $\mathcal{V}_1$ onto the vertices of the second graph $\mathcal{V}_2$, that preserves adjacency and nonadjacency, that is, $(u, v) \in \mathcal{E}_1$ iff $(\pi(u), \pi(v)) \in \mathcal{E}_2$ for all pairs (u, v) of vertices in $\mathcal{V}_1$.

In matrix forms, G_1 and G_2 are isomorphic iff there exists a permutation matrix $\mathbf{P}$ such that $\mathbf{A}_1 = \mathbf{P}\mathbf{A}_2\mathbf{P}^T$ [20]. The computational problem of determining whether two finite graphs are isomorphic is called the graph isomorphism problem. Moreover, given two isomorphic graphs G_1 and G_2, in the graph isomorphism problem one aims to find the permutation matrix $\mathbf{P}$ such that $\mathbf{A}_1 = \mathbf{P}\mathbf{A}_2\mathbf{P}^T$.

The graph isomorphism problem remains one of the few natural problems in NP that is suspected to be neither in P nor NP-complete. This class is called NP-intermediate and exists iff P $\neq$ NP [21].

2.4.2 Quadratic Assignment Problem

Following the above definition of graph isomorphism, inexact graph alignment can be formulated as the optimization problem

$$\underset{\mathbf{P}\in\mathcal{P}}{\text{minimize}} \quad \left\|\mathbf{A}_1 - \mathbf{P}\mathbf{A}_2\mathbf{P}^T\right\|_F^2, \tag{2.7}$$

where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix, while the feasible set $\mathcal{P}$ denotes the set of all $n \times n$ permutation matrices, which can be compactly expressed as

$$\mathcal{P} := \left\{\mathbf{P} \in \{0,1\}^{n\times n} \mid \mathbf{P}^T\mathbf{1}_n = \mathbf{1}_n,\ \mathbf{P}\mathbf{1}_n = \mathbf{1}_n\right\}. \tag{2.8}$$

Note that the cost function of (2.7) can be expressed as

$$\begin{aligned}\left\|\mathbf{A}_1 - \mathbf{P}\mathbf{A}_2\mathbf{P}^T\right\|_F^2 &= \left\|\mathbf{A}_1\right\|_F^2 + \left\|\mathbf{A}_2\right\|_F^2 - 2\left\langle\mathbf{A}_1, \mathbf{P}\mathbf{A}_2\mathbf{P}^T\right\rangle_F &(2.9)\\ &= \left\|\mathbf{A}_1\right\|_F^2 + \left\|\mathbf{A}_2\right\|_F^2 - 2\,\mathrm{trace}\left(\mathbf{A}_1^T\mathbf{P}\mathbf{A}_2\mathbf{P}^T\right). &(2.10)\end{aligned}$$

Hence, problem (2.7) is equivalent to

$$\underset{\mathbf{P}\in\mathcal{P}}{\text{maximize}} \quad \mathrm{trace}\left(\mathbf{A}_1\mathbf{P}\mathbf{A}_2\mathbf{P}^T\right), \tag{2.11}$$

which corresponds to an NP-hard problem; the Quadratic Assignment Problem (QAP) [22].

2.4.3 Embedding alignment

An interesting approach towards the solution of an approximation to the graph alignment problem is based on learned representations of the nodes, i.e. node embeddings. The advantage of using node embedding methods for graph alignment is that, in contrast to the NP–hard problem (2.11), if $\mathbf{E}_1, \mathbf{E}_2 \in \mathbb{R}^{n\times d}$ denote d-dimensional node embeddings (with $d \ll n$) of G_1 and G_2, respectively, then the alignment problem can be solved by the following LAP

$$\underset{\mathbf{P}\in\mathcal{P}}{\text{minimize}} \quad \|\mathbf{E}_1 - \mathbf{P}\mathbf{E}_2\|_F^2 \quad \Longleftrightarrow \quad \underset{\mathbf{P}\in\mathcal{P}}{\text{maximize}} \quad \text{trace}\left(\mathbf{P}\mathbf{E}_2\mathbf{E}_1^T\right), \tag{2.12}$$

which, as we shall see in the sequel, is a subproblem which is solved by every algorithm of graph alignment that is studied in this thesis. The next chapter provides a detailed description on this problem and its connection with graph alignment.

Chapter 3

The Linear Assignment Problem

The Linear Assignment Problem (LAP) is a fundamental optimization problem. In its most general form, the problem consists of finding the best way to assign a number of tasks to a number of agents, given the cost of each task for each agent.

A simple instance of the problem has n agents and n tasks. It is required to perform all tasks by assigning exactly one agent to each task and exactly one task to each agent. Let the assignment of task i to agent j, represented by a binary variable X_{ij}, incur a cost C_{ij}. In order to complete all tasks in the cheapest possible way, the problem can be formulated as a 0-1 linear program [23]:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}X_{ij} \\ \text{subject to} \quad & \sum_{i} X_{ij} = 1, \quad \forall j = 1, 2, \ldots, n \\ & \sum_{j} X_{ij} = 1, \quad \forall i = 1, 2, \ldots, n \\ & x_{ij} \in \{0, 1\}, \quad \forall i, j = 1, 2, \ldots, n. \end{aligned} \tag{3.1}$$

Alternatively, as shown in Figure 3.1, an assignment can be viewed as a bijective mapping of a finite set into itself, i.e., a permutation. Every permutation π of the set $\mathcal{N} = \{1, \ldots, n\}$ corresponds in a unique way to a permutation matrix $\mathbf{X}_\pi$ with $X_{ij} = 1$ for $j = \pi(i)$ and $X_{ij} = 0$ for $j \neq \pi(i)$. Matrix $\mathbf{X}_\pi$ can be viewed as the adjacency matrix of a bipartite graph $G_\pi = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$, where the vertex sets $\mathcal{V}_1$ and $\mathcal{V}_2$ have n vertices, i.e., $|\mathcal{V}_1| = |\mathcal{V}_2| = n$, and there is an edge $(i, j) \in \mathcal{E}$ iff $j = \pi(i)$.

A similarity score matrix $\mathbf{M}$ can be viewed as an adjacency matrix of a weighted bipartite graph $G = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{W})$, where $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$, $\mathcal{W} \subseteq \mathcal{V}_1 \times \mathcal{V}_2$. Each row of matrix $\mathbf{M}$ represents a vertex in $\mathcal{V}_1$ and each column a vertex in $\mathcal{V}_2$. A non-zero entry, M_{ij}, of matrix $\mathbf{M}$ is interpreted as an edge between the i-th row and j-th column vertices, with edge weight M_{ij}. A matching in the bipartite graph is defined as a subset $\mathcal{M} \subseteq \mathcal{W}$ such that no pair of edges of $\mathcal{M}$ are incident on the same vertex. A maximum (cardinality) matching contains the largest possible number of edges. In a perfect matching, all vertices of the graph are matched and the matching is maximum. Finding a perfect matching, in which the sum of the edge weights is maximum, is called Maximum Weight Matching (MWM). The total weight of the matching is defined as $\sum_{(i,j)\in\mathcal{M}} M_{ij}$ and corresponds to the matching pairs with the highest cumulative similarity score.

If its weights of matrix $\mathbf{M}$ are ignored, e.g. by setting them equal to 1, the adjacency matrix of the matching $\mathcal{M}$ yields the permutation matrix $\mathbf{X}_\pi$. Thus, the MWM can be

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 1 & 3 \end{pmatrix}$$

$$\mathbf{X}_\pi = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

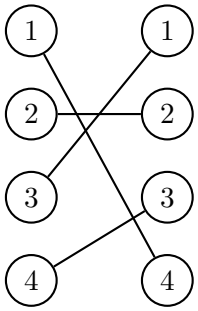


Figure 3.1: Different representations of assignments

considered as the projection of similarity score matrix $\mathbf{M}$ onto the set of permutation matrices $\mathcal{P}$ and is equivalent to the LAP, as proved in the sequel. The permutation matrix $\mathbf{P}$ of the alignment can be computed by solving the LAP:

$$\underset{\mathbf{P}\in\mathcal{P}}{\text{minimize}} \quad \|\mathbf{M}-\mathbf{P}\|_F^2 \quad \Longleftrightarrow \quad \underset{\mathbf{P}\in\mathcal{P}}{\text{maximize}} \quad \text{trace}\left(\mathbf{M}\mathbf{P}^T\right). \tag{3.2}$$

Using properties of the trace operator, the objective function can be expressed as follows:

$$\text{trace}\left(\mathbf{M}\mathbf{P}^T\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}P_{ij} = \sum_{(i,j)\in\mathcal{M}} M_{ij}. \tag{3.3}$$

Using (3.3), the equivalence of (3.1) and (3.2) is clear. The intuition behind the computation of matrix $\mathbf{P}$ is that:

> If node $i \in \mathcal{V}_1$ has high similarity M_{ij} with node $j \in \mathcal{V}_2$ in $\mathbf{M}$, then node $j = \pi(i)$ is likely to be a permutation of node i.

There exists an amazing amount of algorithms for the LAP, ranging from the simple Hungarian method [24] to more recent developments, involving sophisticated data structures and including parallel algorithms [25]. The majority of the algorithms studied in Chapter 4 use the exact Hungarian method, running in $\mathcal{O}(n^3)$ time, which is acceptable for small networks, but for the networks used in Chapter 5, which have over 10^3 nodes, this method is inadequate. Hence, in this work, a greedy heuristic algorithm is used, namely GreedyMatching [26, 27], with $\mathcal{O}(n^2 \log n)$ complexity, which has been shown to have good performance in practice [28]. Using the similarity matrix $\mathbf{M}$, the highest score M_{ij} is located, the pairing (i, j) is recorded, i.e. $P_{ij} = 1$, and all scores involving either i or j are deleted until all nodes are paired. The GreedyMatching algorithm is described in Algorithm 1.

Algorithm 1 Greedy Matching

Input: bipartite graph $G = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{W})$
Output: a maximum weight matching $\mathcal{M} \subset \mathcal{W}$

$\mathcal{M} \leftarrow \emptyset$
sort edges in $\mathcal{W}$ in descending order
while $\mathcal{W} \neq \emptyset$ **do**
    take the edge $(u, v) \in \mathcal{W}$ with the highest weight
    $\mathcal{M} \leftarrow \mathcal{M} \cup (u, v)$
    remove all edges incident to u or v from $\mathcal{W}$
end while

Chapter 4

Algorithms for the graph alignment problem

The combinatorial nature of graph alignment implies approximate solutions, using various optimization approaches (see [29, 30, 31]). Using linearization methods, the QAP can be reformulated as a Mixed Integer Linear Program (MILP), which introduces a large number of new variables and linear constraints [32]. This makes the efficient solution of the resulting MILP a practical challenge. Various relaxations have been applied to the combinatorial constraints of the QAP, such as convex and non-convex quadratic programming relaxations [33], [34], Lagrangian-based relaxations [35], and semidefinite relaxations [36], which are, in general, computationally intensive and there is no guarantee that the obtained solution will be feasible for the original QAP. In order to obtain a feasible solution, a post-processing discretization step often has to be performed on the result, which may increase complexity. These considerations limit the application of such methods to small problem instances.

Spectral methods [4, 37, 38, 39] are scalable approximation methods for graph alignment with relatively low complexity. They compute a matching based on the spectral content of the input graph adjacency matrices. The main motivation behind this approach is that the eigenvalues and the eigenvectors of the adjacency matrix of a graph are invariant with respect to node permutations, meaning that, if two graphs are isomorphic, then their adjacency matrices have the same eigenvalues and eigenvectors (the converse is not true) [40]. There is a great interest in using eigenvalues/eigenvectors for graph alignment, since their computation is a well studied problem, that can be solved in polynomial time [41].

As shown in Chapter 2, the graph alignment problem can be expressed in terms of node embeddings of a given pair of graphs and, as a result, may potentially lead to more tractable formulations of the graph alignment problem. Graph embedding approaches can be usually viewed as finding mappings that embed nodes, subgraphs, or even whole graphs, as points in a low-dimensional vector space [42, 43]. The main goal of these approaches is to capture valuable information and properties of a graph in geometric relationships in the embedding space, where several tasks can be performed naturally and at lower complexity. One example of embedding based methods is REGAL [44], which computes degree-based features from a node's neighborhood at different hop-lengths, and then uses an implicit factorization of the resulting feature matrix to form embeddings that are suitable for network alignment. Recently, the same authors proposed CONE-Align [45], which uses the NetMF node embeddings [15] of the two graphs and produces a matching based on aligning the representations of the embedding subspaces. GWL [46] solves a Gromov-Wasserstein optimization problem to jointly find node embeddings and the graph matching.

A general algorithmic idea behind most graph aligners is the two-step approach. The first step is to compute similarities between nodes in different networks with respect to some cost function. Then, the second step is to identify, with respect to the node similarities, a high-scoring alignment from all possible alignments, i.e. solving the Linear Assignment Problem.

4.1 Umeyama's Method

A pioneering work on spectral methods is the method proposed by Umeyama [4] in 1998. This work proposed an algorithm for the Weighted Graph Matching Problem (WGMP), which is equivalent to (2.7).

In order to construct a similarity matrix between the vertex sets of the two graphs, the algorithm obtains the solution to the relaxation of the QAP over the set of orthogonal matrices, in closed form, by using the eigendecomposition of both adjacency matrices $\mathbf{A}_1$ and $\mathbf{A}_2$. Then, the similarity matrix is used to instantiate a LAP, the solution of which corresponds to the final estimate of the alignment. The method is presented below.

Theorem 4.1.1. *[47] If* $\mathbf{A}$ *and* $\mathbf{B}$ *are Hermitian matrices (*$\mathbf{A} = \mathbf{A}^*$ *and* $\mathbf{B} = \mathbf{B}^*$*) with eigenvalues* $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_n$ *and* $\beta_1 \geq \beta_2 \geq \ldots \geq \beta_n$*, respectively, then*

$$\left\|\mathbf{A} - \mathbf{B}\right\|_F^2 \geq \sum_{i=1}^{n} (\alpha_i - \beta_i)^2. \tag{4.1}$$

Theorem 4.1.2. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be* $n \times n$ *real symmetric matrices with* n *distinct eigenvalues* $\alpha_1 > \alpha_2 > \ldots > \alpha_n$ *and* $\beta_1 > \beta_2 > \ldots > \beta_n$*, respectively, and their eigendecompositions be given by*

$$\mathbf{A} = \mathbf{U}_A \mathbf{\Lambda}_A \mathbf{U}_A^T, \tag{4.2}$$

$$\mathbf{B} = \mathbf{U}_B \mathbf{\Lambda}_B \mathbf{U}_B^T, \tag{4.3}$$

where $\mathbf{U}_A$ *and* $\mathbf{U}_B$ *are orthogonal matrices and* $\mathbf{\Lambda}_A = \operatorname{diag}(\alpha_i)$ *and* $\mathbf{\Lambda}_B = \operatorname{diag}(\beta_i)$*. Then, the following problem*

$$\underset{\mathbf{Q} \in \mathcal{O}}{\textit{minimize}} \quad \left\|\mathbf{A} - \mathbf{Q}\mathbf{B}\mathbf{Q}^T\right\|_F^2, \tag{4.4}$$

where the feasible set $\mathcal{O}$ *denotes the set of all* $n \times n$ *orthogonal matrices, which can be compactly expressed as*

$$\mathcal{O} := \{\mathbf{Q} \in \mathbb{R}^{n \times n} \mid \mathbf{Q}^T\mathbf{Q} = \mathbf{Q}\mathbf{Q}^T = \mathbf{I}_n\}, \tag{4.5}$$

has closed form solution, given by

$$\begin{aligned} \mathbf{Q} &= \mathbf{U}_A \mathbf{S} \mathbf{U}_B^T, \\ \mathbf{S} \in \mathcal{S}_1 &= \{\operatorname{diag}(s_1, s_2, \ldots, s_n) \mid s_i = 1 \text{ or } -1\}. \end{aligned} \tag{4.6}$$

The minimum value achieved is $\sum_{i=1}^{n} (\alpha_i - \beta_i)^2$.

Proof. [4] From Theorem 4.1.1, we have that, for any orthogonal matrix $\mathbf{R}$, the following hold:

$$\left\|\mathbf{A}-\mathbf{R}\mathbf{B}\mathbf{R}^T\right\|_F^2 \geq \sum_{i=1}^{n}(\alpha_i-\beta_i)^2, \tag{4.7}$$

since eigenvalues of $\mathbf{R}\mathbf{B}\mathbf{R}^T$ are the same as those of $\mathbf{B}$.

On the other hand, if we use matrix $\mathbf{Q}$ given in (4.6), we have

$$\begin{aligned}\left\|\mathbf{A}-\mathbf{Q}\mathbf{B}\mathbf{Q}^T\right\|_F^2 &\overset{(*1)}{=} \left\|\mathbf{U}_A\mathbf{\Lambda}_A\mathbf{U}_A^T-\mathbf{U}_A\mathbf{S}\mathbf{U}_B^T\mathbf{U}_B\mathbf{\Lambda}_B\mathbf{U}_B^T\mathbf{U}_B\mathbf{S}\mathbf{U}_A^T\right\|_F^2\\ &= \left\|\mathbf{U}_A(\mathbf{\Lambda}_A-\mathbf{S}\mathbf{\Lambda}_B\mathbf{S})\mathbf{U}_A^T\right\|_F^2\\ &\overset{(*2)}{=} \left\|\mathbf{\Lambda}_A-\mathbf{S}\mathbf{\Lambda}_B\mathbf{S}\right\|_F^2\\ &\overset{(*3)}{=} \left\|\mathbf{\Lambda}_A-\mathbf{\Lambda}_B\right\|_F^2\\ &= \sum_{i=1}^{n}(\alpha_i-\beta_i)^2,\end{aligned} \tag{4.8}$$

where we used

$^{(*1)}$: $\mathbf{A}=\mathbf{U}_A\mathbf{\Lambda}_A\mathbf{U}_A^T$, $\mathbf{B}=\mathbf{U}_B\mathbf{\Lambda}_B\mathbf{U}_B^T$ and $\mathbf{Q}=\mathbf{U}_B\mathbf{S}\mathbf{U}_A^T$,

$^{(*2)}$: $\left\|\mathbf{U}\mathbf{X}\right\|_F=\left\|\mathbf{X}\mathbf{U}^T\right\|_F=\left\|\mathbf{X}\right\|_F$, for any orthogonal matrix $\mathbf{U}$,

$^{(*3)}$: $\mathbf{S}\mathbf{\Lambda}_B\mathbf{S}=\mathbf{S}^2\mathbf{\Lambda}_B=\mathbf{\Lambda}_B$, since $\mathbf{S}$ and $\mathbf{\Lambda}_B$ are both diagonal matrices and $\mathbf{S}^2=\mathbf{I}$.

Thus, the minimum value of $\left\|\mathbf{A}-\mathbf{Q}\mathbf{B}\mathbf{Q}^T\right\|_F^2$ is attained for $\mathbf{Q}$'s given in (4.6). ■

In the sequel, we assume that G_1 and G_2 are isomorphic, meaning that $\exists\mathbf{P}\in\mathcal{P}\subset\mathcal{O}$ such that $\mathbf{A}_1=\mathbf{P}\mathbf{A}_2\mathbf{P}^T$, where $\mathbf{A}_1=\mathbf{U}_1\mathbf{\Lambda}_1\mathbf{U}_1^T$ and $\mathbf{A}_2=\mathbf{U}_2\mathbf{\Lambda}_2\mathbf{U}_2^T$ are the eigendecompositions of the corresponding adjacency matrices. Thus, there exists $\hat{\mathbf{S}}\in\mathcal{S}_1$ which renders an optimal orthogonal matrix $\mathbf{Q}$ of problem (4.4) a permutation matrix, expressed as $\hat{\mathbf{P}}=\mathbf{U}_1\hat{\mathbf{S}}\mathbf{U}_2^T$, corresponding to the permutation $\hat{\pi}$. The following holds:

$$\mathrm{trace}\Big(\hat{\mathbf{P}}^T\mathbf{U}_1\hat{\mathbf{S}}\mathbf{U}_2^T\Big)=\mathrm{trace}\Big(\hat{\mathbf{P}}^T\hat{\mathbf{P}}\Big)=\mathrm{trace}\Big(\mathbf{I}_n\Big)=n. \tag{4.9}$$

Also, we have

$$\mathrm{trace}\Big(\hat{\mathbf{P}}^T\mathbf{U}_1\hat{\mathbf{S}}\mathbf{U}_2^T\Big)=\sum_{i=1}^{n}\sum_{j=1}^{n}\hat{s}_j\mathbf{U}_1[i,j]\mathbf{U}_2[\hat{\pi}(i),j]. \tag{4.10}$$

The following holds:

$$\begin{aligned}\sum_{j=1}^{n}\hat{s}_j\mathbf{U}_1[i,j]\mathbf{U}_2[\hat{\pi}(i),j] &\leq \left|\sum_{j=1}^{n}\hat{s}_j\mathbf{U}_1[i,j]\mathbf{U}_2[\hat{\pi}(i),j]\right|\\ &\leq \sum_{j=1}^{n}\Big|\hat{s}_j\mathbf{U}_1[i,j]\mathbf{U}_2[\hat{\pi}(i),j]\Big|\\ &= \sum_{j=1}^{n}\Big|\mathbf{U}_1[i,j]\Big|\Big|\mathbf{U}_2[\hat{\pi}(i),j]\Big|.\end{aligned} \tag{4.11}$$

Thus,

$$\text{trace}\Big(\hat{\mathbf{P}}^T\mathbf{U}_1\hat{\mathbf{S}}\mathbf{U}_2^T\Big) \leq \sum_i^n\sum_j^n|\mathbf{U}_1[i,j]||\mathbf{U}_2[\pi(i),j]| = \text{trace}\Big(\hat{\mathbf{P}}^T\overline{\mathbf{U}}_1\overline{\mathbf{U}}_2^T\Big), \tag{4.12}$$

where $\overline{\mathbf{U}}_1$ and $\overline{\mathbf{U}}_2$ are the matrices whose elements are the absolute values of the corresponding elements of $\mathbf{U}_1$ and $\mathbf{U}_2$, respectively. Since the matrices $\mathbf{U}_1$ and $\mathbf{U}_2$ are orthogonal, the following hold:

$$\Big\|\mathbf{U}_1[i,:]\Big\|_2 = \Big\|\mathbf{U}_1[:,i]\Big\|_2 = \Big\|\overline{\mathbf{U}}_1[i,:]\Big\|_2 = \Big\|\overline{\mathbf{U}}_1[:,i]\Big\|_2 = 1,\ \forall i = 1,\ldots,n, \tag{4.13}$$

and

$$\Big\|\mathbf{U}_2[j,:]\Big\|_2 = \Big\|\mathbf{U}_2[:,j]\Big\|_2 = \Big\|\overline{\mathbf{U}}_2[j,:]\Big\|_2 = \Big\|\overline{\mathbf{U}}_2[:,j]\Big\|_2 = 1,\ \forall j = 1,\ldots,n. \tag{4.14}$$

Obviously, for all vectors $\mathbf{x} \in \mathbb{R}^n$ holds that

$$\|\mathbf{x}\|_2 = \|\overline{\mathbf{x}}\|_2, \tag{4.15}$$

where $\overline{\mathbf{x}}$ is the vector whose elements are the absolute values of the corresponding elements of $\mathbf{x}$.

The Cauchy-Schwarz inequality yields:

$$\overline{\mathbf{U}}_1[i,:]\overline{\mathbf{U}}_2^T[j,:] \leq \Big\|\overline{\mathbf{U}}_1[i,:]\Big\|_2\Big\|\overline{\mathbf{U}}_2^T[j,:]\Big\|_2 = 1. \tag{4.16}$$

Thus, the following holds for each element M_{ij} of $\mathbf{M} = \overline{\mathbf{U}}_1\overline{\mathbf{U}}_2^T$:

$$0 \leq M_{ij} \leq 1. \tag{4.17}$$

Thus, we have

$$\text{trace}\Big(\mathbf{P}^T\overline{\mathbf{U}}_1\overline{\mathbf{U}}_2^T\Big) \leq n, \tag{4.18}$$

for any permutation matrix $\mathbf{P}$.
Thus, from (4.9), (4.12), and (4.18), we have

$$n \leq \text{trace}\Big(\hat{\mathbf{P}}^T\overline{\mathbf{U}}_1\overline{\mathbf{U}}_2^T\Big) \leq n. \tag{4.19}$$

This means that $\hat{\mathbf{P}}$ maximizes $\text{trace}\Big(\mathbf{P}^T\overline{\mathbf{U}}_1\overline{\mathbf{U}}_2^T\Big)$, for any permutation matrix $\mathbf{P}$. Therefore, when G_1 and G_2 are isomorphic, the optimum permutation matrix can be obtained as a permutation matrix $\mathbf{P}$ which maximizes $\text{trace}(\mathbf{P}^T\overline{\mathbf{U}}_1\overline{\mathbf{U}}_2^T)$; this is an instance of the LAP, with similarity matrix $\mathbf{M} = \overline{\mathbf{U}}_1\overline{\mathbf{U}}_2^T$, that is,

$$\underset{\mathbf{P}\in\mathcal{P}}{\text{maximize}} \quad \text{trace}\Big(\mathbf{M}\mathbf{P}^T\Big). \tag{4.20}$$

This method computes an optimal permutation matrix when the two graphs are isomorphic and a suboptimal permutation matrix if the graphs are nearly isomorphic. However, the

Algorithm 2 Umeyama's method

Input: adjacency matrices $\mathbf{A}_1$ and $\mathbf{A}_2$
Output: an estimated permutation matrix $\mathbf{P}$

1: Eigenvalue Decomposition $\mathbf{U}_1\mathbf{\Lambda}_1\mathbf{U}_1^T \leftarrow \mathbf{A}_1$, $\mathbf{U}_2\mathbf{\Lambda}_2\mathbf{U}_2^T \leftarrow \mathbf{A}_2$
2: Absolute value of each element $\overline{\mathbf{U}}_1 \leftarrow \text{abs}(\mathbf{U}_1)$, $\overline{\mathbf{U}}_2 \leftarrow \text{abs}(\mathbf{U}_2)$
3: Compute $\mathbf{M} = \overline{\mathbf{U}}_1\overline{\mathbf{U}}_2^T$
4: Estimate $\mathbf{P} \leftarrow$ GreedyMatching($\mathbf{M}$)

most general case is when the graphs might even not be nearly isomorphic. In these cases, this method may produce poor results. Umeyama's method is presented in Algorithm 2.

4.2 IsoRank

IsoRank is a spectral method, proposed by Singh et al. [48], for pairwise global alignment of protein-protein interaction (PPI) networks, which considers a regularized form of the QAP (2.11). In order to obtain the solution, the algorithm applies random walks with restarts to compute the PageRank eigenvector of the normalized Kronecker product graph $\tilde{\mathbf{C}} = \mathbf{A}_1 \tilde{\otimes} \mathbf{A}_2 = \tilde{\mathbf{A}}_1 \otimes \tilde{\mathbf{A}}_2$ [49]. The solution, which is essentially a cross-network node similarity matrix, is used to instantiate a LAP to obtain the final alignment. For the case of undirected, unattributed graphs, IsoRank is a special case of the more general FINAL algorithm [50].

The key idea behind IsoRank lies in the assumption that a pair of nodes has high similarity if their immediate neighbors have high similarities (topological similarity). In order to compute the similarity matrix, IsoRank solves an eigenvalue problem, which is related to the original QAP. If, instead of the adjacency matrices $\mathbf{A}_1$ and $\mathbf{A}_2$, the normalized ones are used $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$, respectively, then the objective function of the QAP (2.11) can be expressed as:

$$\text{trace}\Big(\tilde{\mathbf{A}}_1\mathbf{P}\tilde{\mathbf{A}}_2\mathbf{P}^T\Big) = \text{vec}\big(\mathbf{P}\big)^T\big(\tilde{\mathbf{A}}_1 \otimes \tilde{\mathbf{A}}_2\big)\,\text{vec}\big(\mathbf{P}\big), \tag{4.21}$$

where we used the property $\text{trace}\big(\mathbf{ABCD}\big) = \text{vec}\big(\mathbf{A}\big)^T\big(\mathbf{D}\otimes\mathbf{B}\big)\,\text{vec}\big(\mathbf{C}\big)$ and the symmetry of $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$.

Defining the $m := n^2$ dimensional vector $\mathbf{s} = \text{vec}(\mathbf{P})$ and the $m \times m$ matrix $\tilde{\mathbf{C}} = \mathbf{A}_1 \tilde{\otimes} \mathbf{A}_2 = \tilde{\mathbf{A}}_1 \otimes \tilde{\mathbf{A}}_2$, the objective function of (2.11) can be expressed as the quadratic function $\mathbf{s}^T\tilde{\mathbf{C}}\mathbf{s}$. The maximization of this function is equivalent with the minimization of $\mathbf{s}^T(\mathbf{I} - \tilde{\mathbf{C}})\mathbf{s}$. Putting everything together, the optimization problem of IsoRank can be stated as follows.

$$\underset{\mathbf{s}}{\text{argmin}} \quad J(\mathbf{s}) = \alpha\mathbf{s}^T(\mathbf{I} - \tilde{\mathbf{C}})\mathbf{s} + (1-\alpha)\left\|\mathbf{s} - \mathbf{h}\right\|_F^2, \tag{4.22}$$

where α is the regularization parameter. The regularization term $\|\mathbf{s} - \mathbf{h}\|_F^2$ reflects the prior alignment preference, where $\mathbf{h} = \text{vec}(\mathbf{H})$, with element H_{ij} of matrix $\mathbf{H}$ corresponding to the elemental similarity score between node $i \in \mathcal{V}_2$ and $j \in \mathcal{V}_1$. The norm of vector $\mathbf{h}$ is normalized to unity. Topological similarity and elemental similarity of nodes are scaled by

factors $\alpha \leq 1$ and $1-\alpha$, respectively. In the specific application context of [48], $\mathbf{h}$ encodes protein sequence similarity scores (BLAST scores), and protein interaction networks G_1 and G_2 are undirected.

Using the fact that the objective function in (4.22) is quadratic, a solution can be found [50] by setting its derivative to be zero, that is,

$$\frac{\partial J(\mathbf{s})}{\partial \mathbf{s}} = 2(\mathbf{I} - \alpha\tilde{\mathbf{C}})\mathbf{s} + 2(1-\alpha)\mathbf{h} = \mathbf{0}. \tag{4.23}$$

This equation is difficult to solve directly, due to the very large dimension of $\tilde{\mathbf{C}}$. Instead, we use the following iterative procedure:

$$\mathbf{s}_{k+1} = \alpha\tilde{\mathbf{C}}\mathbf{s}_k + (1-\alpha)\mathbf{h}, \tag{4.24}$$

which computes a PageRank eigenvector

$$\mathbf{s}_{k+1} = \alpha\mathbf{G}\mathbf{s}_k + (1-\alpha)\mathbf{v}, \tag{4.25}$$

where the Google matrix $\mathbf{G}$ for one graph is replaced by the $\tilde{\mathbf{C}}$ matrix (the Kronecker product of the normalized adjacency matrices of two graphs) and the personalization vector $\mathbf{v}$ (user preferences in browsing) is replaced by $\mathbf{h}$ (prior alignment preferences).

IsoRank is equivalent to FINAL-P (proof in A.1), which is a variant of FINAL with unavailable node and edge attributes and uses the iteration step

$$\mathbf{s}_{k+1} = \alpha\mathbf{D}_U^{-\frac{1}{2}}(\mathbf{A}_1 \otimes \mathbf{A}_2)\mathbf{D}_U^{-\frac{1}{2}}\mathbf{s}_k + (1-\alpha)\mathbf{h}, \tag{4.26}$$

where $\mathbf{D}_U = \mathbf{D}_1 \otimes \mathbf{D}_2$, $\mathbf{D}_1$ and $\mathbf{D}_2$ are the degree matrices of $\mathbf{A}_1$ and $\mathbf{A}_2$, respectively. FINAL avoids the costly computation of the Kronecker product between $\mathbf{A}_1$ and $\mathbf{A}_2$ by using the property $\mathrm{vec}(\mathbf{ABC}) = (\mathbf{C}^T \otimes \mathbf{A})\,\mathrm{vec}(\mathbf{B})$. Thus, the iteration step becomes

$$\mathbf{s}_{k+1} = \alpha\mathbf{D}_U^{-\frac{1}{2}}(\mathbf{A}_2\mathbf{Q}\mathbf{A}_1) + (1-\alpha)\mathbf{h}, \tag{4.27}$$

where $\mathbf{Q}$ is an $n \times n$ matrix reshaped by $\mathbf{q} = \mathbf{D}_U^{-\frac{1}{2}}\mathbf{s}_k$ in column order, i.e., $\mathbf{Q} = \mathrm{mat}(\mathbf{q}, n, n)$.

In order to obtain the final alignment, the last step of the algorithm entails the projection of similarity matrix $\mathbf{S} = \mathrm{mat}(\mathbf{s}, n, n)$ onto the set of permutation matrices $\mathcal{P}$, which can be done by solving the LAP

$$\underset{\mathbf{P}\in\mathcal{P}}{\text{maximize}} \quad \mathrm{trace}\left(\mathbf{S}\mathbf{P}^T\right). \tag{4.28}$$

IsoRank is described in Algorithm 3.

Equation (4.24) describes a random walk on the normalized Kronecker product graph of $G_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $G_2 = (\mathcal{V}_2, \mathcal{E}_2)$. The standard way to define a random walk on a graph is to allow a random walker to take off on an arbitrary node and then successively visit new nodes by randomly selecting one of the outgoing edges, according to a Markov transition kernel of the graph. In this case, the transition matrix is $\tilde{\mathbf{C}}$. In every iteration, the random

Algorithm 3 IsoRank (FINAL-P)

Input: adjacency matrices $\mathbf{A}_1$ and $\mathbf{A}_2$, prior alignment preference $\mathbf{H}$, the regularization parameter α, and the maximum iteration number k_{max}

Output: an estimated permutation matrix $\mathbf{P}$

Construct degree matrix $\mathbf{D}_U$
Initiate the alignment $\mathbf{s} = \mathbf{h} = \text{vec}(\mathbf{H})$, and $k = 1$
while $k \leq k_{max}$ **do**
    Compute vector $\mathbf{q} = \mathbf{D}_U^{-\frac{1}{2}}\mathbf{s}$
    Reshape $\mathbf{q}$ as $\mathbf{Q} = \text{mat}(\mathbf{q}, n, n)$
    Update $\mathbf{s}_k \leftarrow \alpha\mathbf{D}_U^{-\frac{1}{2}} \text{vec}(\mathbf{A}_2\mathbf{Q}\mathbf{A}_1^T) + (1-\alpha)\mathbf{h}$
    if $\mathbf{s}_k$ has converged to $\mathbf{s}_{k-1}$ **then**
        Set $k \leftarrow k_{max}$
    end if
    Set $k \leftarrow k + 1$
end while
Reshape $\mathbf{s}$ as $\mathbf{S} = \text{mat}(\mathbf{s}, n, n)$
Estimate $\mathbf{P} \leftarrow$ GreedyMatching($\mathbf{S}$)

walker has a probability $\tilde{C}_{ij}$ to go from node j to node i and a probability s_i to be in node i. There is also a probability of going back to the starting node, i.e. restart. So, the task is to find the value of $\mathbf{s}$ such that further iterations will not affect the probabilities. Thus, IsoRank can be considered as a random walk with restarts.

4.3 Low-Rank Align (LRA)

LRA is a spectral alignment method for approximately solving the QAP (2.11) on undirected graphs, recently proposed by Feizi et al. [39]. In the first step, LRA solves a rank-k approximation of the QAP over the relaxed feasible set of orthogonal matrices. At the second step, an alternating optimization framework is adopted to solve a variation of the QAP with two optimization variables. The update of one variable entails solving a discrete optimization problem, which is heuristically solved by searching in a reduced space (determined by the solution of the first step), while the second variable is updated by solving a LAP.

Relaxing the QAP (2.11) as follows:

$$\underset{\mathbf{Q}\in\mathcal{O}}{\text{maximize}} \quad \text{trace}\left(\mathbf{A}_1\mathbf{Q}\mathbf{A}_2\mathbf{Q}^T\right), \tag{4.29}$$

where the feasible set $\mathcal{O}$ denotes the set of all $n \times n$ orthogonal matrices, the problem has closed form solutions given by Theorem 4.1.2. Other relaxations can be considered as well. The feasible set of (4.29) can be any superset of permutation matrices $\mathcal{P}$, but let us consider the set of orthogonal matrices $\mathcal{O}$. Recall that optimal solutions can be found

using the eigendecomposition of matrices $\mathbf{A}_1 = \mathbf{U}_1\mathbf{\Lambda}_1\mathbf{U}_1^T$ and $\mathbf{A}_2 = \mathbf{U}_2\mathbf{\Lambda}_2\mathbf{U}_2^T$ as follows:

$$\begin{aligned} \mathbf{Q} = \mathbf{U}_1\mathbf{S}\mathbf{U}_2^T = \sum_{i=1}^{n} s_i\mathbf{U}_1[:,i]\mathbf{U}_2[:,i]^T, \\ \mathbf{S} \in \mathcal{S}_1 = \{\text{diag}(s_1, s_2, \ldots, s_n) \mid s_i = 1 \text{ or } -1\}. \end{aligned} \tag{4.30}$$

Let the set that represents multiple optimal solutions of problem (4.29) be expressed as

$$\mathcal{P}_0 := \{\mathbf{Q} \in \mathcal{O} \mid \mathbf{Q} = \sum_{i=1}^{n} s_i\mathbf{U}_1[:,i]\mathbf{U}_2[:,i]^T,\ \mathbf{s} \in \{-1,1\}^n\}, \tag{4.31}$$

where s_i is the i-th component of vector $\mathbf{s}$. If $\mathbf{v}$ is an eigenvector of a matrix corresponding to the eigenvalue λ, $-\mathbf{v}$ is also an eigenvector of the same matrix with the same eigenvalue. $\mathcal{P}_0$ can have at most 2^n distinct elements.

The authors of LRA proposed the following two variable variation of the QAP (2.11)

$$\begin{aligned} \text{maximize} \quad & \text{trace}\left(\mathbf{A}_1\mathbf{Q}\mathbf{A}_2\mathbf{P}^T\right) \\ \text{subject to} \quad & \mathbf{P} \in \mathcal{P}, \\ & \mathbf{Q} \in \mathcal{P}_0, \end{aligned} \tag{4.32}$$

which considers all the optimal solutions of set $\mathcal{P}_0$.

By a slight abuse of notation, let

$$\mathbf{Q}(r) = \sum_{i=1}^{n} \mathbf{s}_r[i]\mathbf{\Lambda}_1[i,i]\mathbf{\Lambda}_2[i,i]\mathbf{U}_1[:,i]\mathbf{U}_2[:,i]^T \tag{4.33}$$

be the solution to problem (4.29) in terms of vertex $\mathbf{s}_r$ of the hypercube $\mathcal{S} = \{\mathbf{s}_1, \ldots, \mathbf{s}_R\}$, where $\mathbf{s}_r \in \{-1,1\}^n$ and $R = 2^n$. $\mathbf{Q}(r)$ can be considered a similarity matrix using simultaneous alignment of eigenvectors whose contributions in the overall alignment score are weighed by their corresponding eigenvalues.

Following an alternating optimization framework, for a fixed $\mathbf{Q}(\mathbf{s}_r)$, $\forall \mathbf{s}_r$, (4.32) is a LAP

$$\begin{aligned} \text{maximize} \quad & \text{trace}\left(\left(\sum_{i=1}^{n} \mathbf{s}_r[i]\mathbf{\Lambda}_1[i,i]\mathbf{\Lambda}_2[i,i]\mathbf{U}_1[:,i]\mathbf{U}_2[:,i]^T\right)\mathbf{P}^T\right) \\ \text{subject to} \quad & \mathbf{P} \in \mathcal{P}, \\ & \mathbf{s}_r \in \{-1,1\}^n, \quad \forall 1 \le r \le n. \end{aligned} \tag{4.34}$$

The alignment with the highest score (maximum value of the objective function) is considered as the final alignment of the algorithm.

However, there are possibly exponentially many problems of the form of (4.34) and corresponding optimal solutions and obtaining the resulting permutation matrices would be computationally infeasible. Eigenvectors with small eigenvalues have no significant contribution to the objective function of (4.34), meaning that a low rank approximation

Algorithm 4 Low-Rank Align (LRA)

Input: adjacency matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ and the approximation rank k
Output: an estimated permutation matrix $\mathbf{P}$

1: Eigenvalue Decomposition using the largest k eigenvalues and their corresponding eigenvectors $\mathbf{U}_1\mathbf{\Lambda}_1\mathbf{U}_1^T \leftarrow \mathbf{A}_1$, $\mathbf{U}_2\mathbf{\Lambda}_2\mathbf{U}_2^T \leftarrow \mathbf{A}_2$
2: Form all the $R = 2^k$ possible combinations of sign vectors, as the vertex set of the hypercube $\mathcal{S} = \{\mathbf{s}_1, \ldots, \mathbf{s}_R\}$
3: **for** $r = \{0, \ldots, R\}$ **do**
4: Compute $\mathbf{Q}(r) = \sum_{i=1}^{k} \mathbf{s}_r[i]\mathbf{\Lambda}_1[i,i]\mathbf{\Lambda}_2[i,i]\mathbf{U}_1[:,i]\mathbf{U}_2[:,i]^T$
5: Estimate $\mathbf{P}(r) \leftarrow$ GreedyMatching($\mathbf{Q}(r)$)
6: Compute $g(r) = \text{trace}\left(\mathbf{A}_1\mathbf{P}(r)\mathbf{A}_2\mathbf{P}(r)^T\right)$
7: **end for**
8: Find r_{opt} where $g(r)$ is maximum
9: **return** matrix $\mathbf{P} \leftarrow \mathbf{P}(r_{opt})$

could be used as a good heuristic:

$$\begin{aligned} \text{maximize} \quad & \text{trace}\left(\left(\sum_{i=1}^{k} \mathbf{s}_r[i]\mathbf{\Lambda}_1[i,i]\mathbf{\Lambda}_2[i,i]\mathbf{U}_1[:,i]\mathbf{U}_2[:,i]^T\right)\mathbf{P}^T\right) \\ \text{subject to} \quad & \mathbf{P} \in \mathcal{P}, \\ & \mathbf{s}_r \in \{-1,1\}^k, \quad \forall 1 \le r \le k. \end{aligned} \tag{4.35}$$

where k is a constant that determines the rank of the affinity matrix. LRA is described in Algorithm 4. The number of the problems of the form (4.35) that are solved is equal to $2^k \ll 2^n$.

4.4 CONE-Align

CONE is a joint embedding and alignment algorithm proposed recently by Chen et al. [45], and has three stages. In the first stage, the NetMF method is used to obtain node embeddings of the two graphs. Then, the embedding subspaces are aligned by adopting a Wasserstein-Procrustes framework [51]. In the last step, the final alignment is obtained by employing *kd*-trees for fast nearest node embedding search.

The name of the algorithm stands for CONsistent Embedding-based network Alignment. The authors defined the principle of Matched Neighborhood Consistency (MNC):

> Let $\mathcal{N}_{G_1}(i)$ be the neighbors of node $i \in \mathcal{V}_1$, i.e. nodes that share an edge with i. Node i's "mapped neighborhood" in G_2 is defined as the set of nodes onto which π maps i's neighbors: $\tilde{\mathcal{N}}^{\pi}_{G_2}(i) = \{j \in \mathcal{V}_2 : \exists k \in \mathcal{N}_{G_1}(i) \text{ s.t. } \pi(k) = j\}$. The Matched Neighborhood Consistency (MNC) of node $i \in \mathcal{V}_1$ and $j \in \mathcal{V}_2$ is defined as the Jaccard similarity of the two sets:

$$\text{MNC}(i,j) = \frac{\left|\tilde{\mathcal{N}}^{\pi}_{G_2}(i) \cap \mathcal{N}_{G_2}(j)\right|}{\left|\tilde{\mathcal{N}}^{\pi}_{G_2}(i) \cup \mathcal{N}_{G_2}(j)\right|}. \tag{4.36}$$

Algorithm 5 NetMF

Input: the adjacency matrix $\mathbf{A}$, the approximation rank h of eigenpairs, the embedding dimension d, the window size T and the number of negative samples α
Output: node embedding $\mathbf{E}$

1: Eigenvalue Decomposition using the largest h eigenvalues and their corresponding eigenvectors $\mathbf{U}_h\mathbf{\Lambda}_h\mathbf{U}_h^T \leftarrow \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$
2: Approximate $\hat{\mathbf{M}} = \frac{\text{nnz}(A)}{\alpha}\mathbf{D}^{-\frac{1}{2}}\mathbf{U}_h\left(\frac{1}{T}\sum_{r=1}^{T}\mathbf{\Lambda}_h^r\right)\mathbf{U}_h^T\mathbf{D}^{-\frac{1}{2}}$
3: Compute $\hat{\mathbf{M}}' = \max\left(\hat{\mathbf{M}}, 1\right)$
4: Rank-d approximation of element-wise logarithm by SVD $\mathbf{U}_d\mathbf{\Sigma}_d\mathbf{V}_d^T \leftarrow \log\left(\hat{\mathbf{M}}'\right)$
5: **return** $\mathbf{E} = \mathbf{U}_d\sqrt{\mathbf{\Sigma}_d}$

In the first step, the normalized node embeddings $\mathbf{E}_1, \mathbf{E}_2 \in \mathbb{R}^{n\times d}$ are computed, using any popular embedding method that preserves intra-graph node proximity. This means that, if neighboring nodes in each graph have similar embeddings, then they will be mapped closely by using embedding similarity, which preserves the MNC; even when nodes are not neighbors due to missing edges [52], many node embedding algorithms preserve any higher-order proximities they share. The authors used NetMF, which is described in Algorithm 5, with the default settings: approximation of the normalized graph Laplacian with 256 eigenpairs, embedding dimension $d = 128$, context window size $w = 10$ and $\alpha = 1$ negative samples.

The two node embeddings $\mathbf{E}_1$ and $\mathbf{E}_2$ may be translated, rotated, or rescaled relative to each other, due to the invariance of the embedding objective. Thus, in order to compare them, in the second step, embedding subspaces should be aligned. Inspired by [51], an alternating optimization framework is adopted to solve the two variable problem of Procrustes in Wasserstein distance, which is described in the sequel.

Procrustes analysis learns a linear transformation between two sets of matched points $\mathbf{E}_1$ and $\mathbf{E}_2$. If the node correspondences were known, then a linear embedding transformation $\mathbf{Q}$ could be recovered from the set of orthogonal matrices $\mathcal{O}$. $\mathbf{Q}$ aligns the columns of the node embedding matrices, i.e. the embedding spaces, and can be obtained by solving an orthogonal Procrustes problem (column permutation)

$$\underset{\mathbf{Q}\in\mathcal{O}}{\text{minimize}} \quad \|\mathbf{E}_1\mathbf{Q} - \mathbf{E}_2\|_F^2. \tag{4.37}$$

As shown by Schönemann [53], the orthogonal Procrustes problem has a closed form solution equal to $\mathbf{Q}^* = \mathbf{U}\mathbf{V}^T$, where $[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}] = \text{svd}(\mathbf{E}_1^T\mathbf{E}_2)$ is the Singular Value Decomposition of $\mathbf{E}_1^T\mathbf{E}_2$, i.e. $\mathbf{E}_1^T\mathbf{E}_2 = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$.

The Wasserstein distance is used to measure the distance between sets of points. If the embedding space transformation were known, then the optimal node correspondence $\mathbf{P}$ could be recovered from the set of permutation matrices $\mathcal{P}$. $\mathbf{P}$ aligns the rows of the node embedding matrices, i.e. the nodes. It can be obtained by minimizing the squared Wasserstein distance (row permutation)

$$\underset{\mathbf{P}\in\mathcal{P}}{\text{minimize}} \quad \|\mathbf{E}_1 - \mathbf{P}\mathbf{E}_2\|_F^2, \tag{4.38}$$

Algorithm 6 Frank-Wolfe

Input: convex, differentiable real-valued function $f(\mathbf{x})$ with convex domain $\mathcal{D}$
Output: approximate solution $\mathbf{x}^{(n_0)} = \operatorname{argmin}_{\mathbf{x}\in\mathcal{D}} f(\mathbf{x})$

Let $\mathbf{x}^{(0)} \in \mathcal{D}$
for $k = \{0, \dots, n_0\}$ **do**
    Compute the direction, via SINKHORN, $\mathbf{s} = \operatorname{argmin}_{\mathbf{s}\in\mathcal{D}} \langle \mathbf{s}, \nabla f(\mathbf{x}^{(k)}) \rangle$
    Set the step size $\gamma = \frac{2}{k+2}$
    Update $\mathbf{x}^{(k+1)} = (1-\gamma)\mathbf{x}^{(k)} + \gamma\mathbf{s}$
end for

Algorithm 7 Sinkhorn (LOT)

Input: transportation cost matrix $\mathbf{M}$ and transportation polytope $\mathcal{U}_{r,c} = \{\mathbf{P} \in \mathbb{R}_+^{n\times n} \mid \mathbf{P}\mathbf{1}_n = \mathbf{r}, \mathbf{P}^T\mathbf{1}_n = \mathbf{c}\}$
Output: a transportation matrix $\mathbf{P} = \operatorname{argmin}_{\mathbf{P}\in\mathcal{U}_{r,c}} \langle \mathbf{P}, \mathbf{M} \rangle_F$

Compute the element-wise exponential $\mathbf{K} = \exp(-\lambda\mathbf{M})$
Set $\mathbf{u}^{(0)} = \mathbf{1}$
while u changes **do**
    Compute element-wise division $\mathbf{v}^{(i)} = \mathbf{c} \oslash \mathbf{K}^T\mathbf{u}^{(i-1)}$
    Compute element-wise division $\mathbf{u}^{(i)} = \mathbf{r} \oslash \mathbf{K}\mathbf{v}^{(i)}$
end while
return $\mathbf{P} = \operatorname{diag}(\mathbf{u})\mathbf{K}\operatorname{diag}(\mathbf{v})$

which is an instance of the LAP.

However, neither the correspondences nor the transformation are known. Formally, the goal is to learn an orthogonal matrix $\mathbf{Q} \in \mathcal{O}$, such that the set of points $\mathbf{E}_1$ is close to the set of points $\mathbf{E}_2$ and one-to-one correspondences can be inferred. Using the Wasserstein distance defined in (4.38), as the measure of distance between the two sets of points and combining it with the orthogonal Procrustes defined in (4.37), leads to the problem of Procrustes in Wasserstein distance

$$\underset{\mathbf{Q}\in\mathcal{O},\mathbf{P}\in\mathcal{P}}{\text{minimize}} \quad \|\mathbf{E}_1\mathbf{Q} - \mathbf{P}\mathbf{E}_2\|_F^2 \quad \Longleftrightarrow \underset{\mathbf{Q}\in\mathcal{O},\mathbf{P}\in\mathcal{P}}{\text{maximize}} \quad \operatorname{trace}\left(\mathbf{Q}^T\mathbf{E}_1^T\mathbf{P}\mathbf{E}_2\right). \tag{4.39}$$

The authors solve this problem with a stochastic optimization scheme, alternating between the Wasserstein and Procrustes problems. For T iterations, a temporary embedding transformation $\mathbf{Q}$ is used to find a matching $\mathbf{P}_t$ for mini-batches $\mathbf{E}_{1t}$ and $\mathbf{E}_{2t}$ of size $b \times d$ embeddings each, using GREEDYMATCHING. The gradient of the Wasserstein Procrustes distance $\|\mathbf{E}_{1t}\mathbf{Q} - \mathbf{P}\mathbf{E}_{2t}\|_F^2$, with respect to $\mathbf{Q}$, for fixed $\mathbf{E}_{1t}^T, \mathbf{E}_{2t}$ and $\mathbf{P}_t$, is equal to $2\mathbf{E}_{1t}^T\mathbf{P}_t\mathbf{E}_{2t}$ and is used for the update of $\mathbf{Q}$ in gradient descent.

To initialize the above nonconvex procedure, a convex relaxation to the original QAP (2.7) can be used

$$\underset{\mathbf{P}\in\mathcal{B}}{\text{minimize}} \quad \|\mathbf{A}_1\mathbf{P} - \mathbf{P}\mathbf{A}_2\|_F^2, \tag{4.40}$$

where $\mathcal{B}$ is the convex hull of $\mathcal{P}$, the set of doubly stochastic matrices, namely the Birkhoff polytope, which can be compactly expressed as

$$\mathcal{B} := \{\mathbf{P} \in \mathbb{R}^{n\times n} \mid \mathbf{P}^T\mathbf{1}_n = \mathbf{1}_n,\ \mathbf{P}\mathbf{1}_n = \mathbf{1}_n\}. \tag{4.41}$$

Algorithm 8 CONE-Align

Input: the adjacency matrices $\mathbf{A}_1$ and $\mathbf{A}_1$
Output: an estimated permutation matrix $\mathbf{P}$

Get the node embeddings $\mathbf{E}_1 \leftarrow \text{NetMF}(\mathbf{A}_1)$, $\mathbf{E}_2 \leftarrow \text{NetMF}(\mathbf{A}_2)$
Normalize the embeddings $\mathbf{E}_1 = \frac{\mathbf{E}_1}{\|\mathbf{E}_1\|_F}$, $\mathbf{E}_2 = \frac{\mathbf{E}_2}{\|\mathbf{E}_2\|_F}$
Approximate, via FRANK-WOLFE, $\mathbf{P}^* = \text{argmin}_{\mathbf{P}\in\mathcal{B}} \|\mathbf{A}_1\mathbf{P} - \mathbf{P}\mathbf{A}_2\|_F^2$
Singular Value Decomposition $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \leftarrow \mathbf{E}_1^T\mathbf{P}^*\mathbf{E}_2$
Compute $\mathbf{Q} = \mathbf{U}\mathbf{V}^T$
for $t = \{1, \ldots, T\}$ **do**
    Approximate $\mathbf{P}_t \leftarrow \text{GREEDYMATCHING}(\mathbf{E}_{1t}\mathbf{Q}\mathbf{E}_{2t}^T)$
    Compute gradient $\mathbf{G}_t = -2\mathbf{E}_{1t}^T\mathbf{P}_t\mathbf{E}_{2t}$
    SVD of descending, with learning rate η, $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \leftarrow \mathbf{Q} - \eta\mathbf{G}_t$
    Update $\mathbf{Q} = \mathbf{U}\mathbf{V}^T$
end for
Estimate $\mathbf{P} \leftarrow \text{GREEDYMATCHING}(\mathbf{E}_1\mathbf{Q}\mathbf{E}_2^T)$

The global minimizer $\mathbf{P}^*$ of (4.40) can be found via the conditional gradient method, also known as FRANK-WOLFE algorithm [54], which is described in Algorithm 6. The authors solve the direction-finding subproblem with the Lightspeed Optimal Transport (LOT) algorithm, which is a regularized variation of the SINKHORN algorithm [55], with regularization parameter λ_0, and described in Algorithm 7. Using $\mathbf{E}_1$ and $\mathbf{P}^*\mathbf{E}_2$, an initial $\mathbf{Q}$ can be generated with orthogonal Procrustes (4.37).

After aligning the embeddings with the final transformation $\mathbf{Q}$, in the last step, the authors proposed the use of a *kd*-tree for fast nearest neighbor search between $\mathbf{E}_1\mathbf{Q}$ and $\mathbf{E}_2$ to match each node in G_1 to its nearest neighbor in G_2 based on Euclidean distance. However, this stage does not find a one-to-one correspondence mapping in all cases. Thus, the GREEDYMATCHING algorithm is applied. CONE is described in Algorithm 8.

Chapter 5

Experimental Evaluation

This chapter adopts a standard experimental setup and compares graph alignment algorithms. The algorithms described in Chapter 4 are evaluated using both real and synthetic graphs, where the edges have been perturbed with a fixed amount of noise and nodes have been permuted.

All algorithms are implemented in MATLAB 2015a. For IsoRank (FINAL-P) [1] and LRA [2] the authors' original MATLAB codes are used. A simple script for Umeyama's method is implemented and for CONE the original Python code [3] is re-implemented in MATLAB. The experiments were carried out on a Windows computer equipped with a AMD Ryzen 5 CPU and 16 GB RAM.

5.1 Datasets

5.1.1 Real-world Networks

Standard network repositories provide plenty of network datasets. The real-world datasets that are used in the experiments can be found in the Koblenz Network Collection (KONECT) [56] and are listed in Table 5.1. These include (i) metabolic networks (C. Elegans and Yeast), where nodes are chemical substances and edges represent chemical interactions, (ii) a communications network at the University Rovira i Virgili (Arenas-Email), where nodes are users and each edge represents that at least one email was sent, (iii) a social network (PolBlog) of hyperlinks between political blogs about politics in the United States of America, and (iv) infrastructure networks: (Airports) flights between US airports in 2010, where each edge represents a connection from one airport to another and E-road the international E-road network, where each edge represents that two cities are connected by an E-road.

5.1.2 Synthetic Networks

The synthetic networks are created following the Erdös-Rényi (ER) model and their statistics are shown in Table 5.2. A random graph $G_{n,p}$ has n nodes and each edge is formed with probability p. Using $p > \frac{(1+\epsilon)\ln n}{n}$ the graph will be connected with high probability [57]. In the experiment, each random graph has 1024 nodes and edge probability in the set $\{0.01, 0.1, 0.25, 0.3, 0.4, 0.5\}$.

[1] https://github.com/sizhang92/FINAL-KDD16
[2] https://github.com/SoheilFeizi/spectral-graph-alignment
[3] https://github.com/GemsLab/CONE-Align

Graph	n	m	$\bar{d}$	c_1	$\left\lvert\frac{\lambda_1}{\lambda_2}\right\rvert$	Type
C. Elegans	453	2,025	8.9	0.124	1.752	Metabolic
E-road	1,039	1,305	2.5	0.035	1.023	Infrastructure
Arenas-Email	1,133	5,451	9.6	0.166	1.223	Communications
PolBlog	1,222	16,714	27.4	0.226	1.236	Hyperlink
Yeast	1,458	1,948	2.7	0.052	1.002	Metabolic
Airports	1,572	17,214	21.9	0.384	3.234	Infrastructure

Table 5.1: Description of the networks used: the number of vertices n, the number of edges m, the average degree $\bar{d}$, the global clustering coefficient c_1, the spectral separation $\left\lvert\frac{\lambda_1}{\lambda_2}\right\rvert$ and the network type.

p	n	m	$\bar{d}$	c_1	$\left\lvert\frac{\lambda_1}{\lambda_2}\right\rvert$
0.01	1,024	5,403	10.6	0.01	1.746
0.1	1,024	52,502	102.5	0.1	5.399
0.25	1,024	131,254	256.4	0.25	9.199
0.3	1,024	157,398	307.4	0.3	10.566
0.4	1,024	210,628	411.4	0.4	12.978
0.5	1,024	261,592	510.9	0.5	15.944

Table 5.2: Description of the synthetic networks: the edge probability p, the number of vertices n, the number of edges m, the average degree $\bar{d}$, the global clustering coefficient c_1 and the spectral separation $\left\lvert\frac{\lambda_1}{\lambda_2}\right\rvert$.

5.2 Experimental Setup

The experimental setup of [58, 30, 59] is adopted. If the original graph is directed, it is converted into undirected by performing a symmetrization step. Additionally, all self-loops and edge weights are removed. Hence, every graph in the experiments is considered simple, undirected, and unweighted. To increase fairness, only the largest connected component of the input graph is considered.

In order to create an instance of the graph alignment problem, for each graph G_1, a "noisy" and permuted version is constructed by randomly adding new edges with probability p_e. The adjacency matrix of G_2 is

$$\mathbf{A}_2 = \mathbf{P}^T[\mathbf{A}_1 + (\mathbf{1} - \mathbf{A}_1) \odot \mathbf{Q}]\mathbf{P}, \tag{5.1}$$

where the operator "$\odot$" denotes the Hadamard (element-wise) product, $\mathbf{Q}$ is the adjacency matrix of a random Erdős-Rényi graph $G_{n,p}$ [57], and $\mathbf{P} \in \mathcal{P}$ denotes a randomly generated permutation matrix. The number of additional edges that appear in $\mathbf{A}_2$ is controlled by varying the noise level p_e, such that the expected percentage of extra edges in G_2 is equal to a fixed number between 0% and 25% of the total edges in G_1. Matrix $(\mathbf{1} - \mathbf{A}_1)$ is the adjacency matrix of the complement of graph G_1 and has $\binom{n}{2} - m$ edges, i.e., the maximum possible edges of an undirected graph with n nodes minus the m edges of graph G_1. The element-wise multiplication with matrix $\mathbf{Q}$ yields a random selection of $\left(\binom{n}{2} - m\right) p$ expected extra edges. Thus, in order to add $p_e m$ edges, p should be chosen such that

equation

$$\left(\binom{n}{2} - m\right) p = p_e m$$

holds. Hence,

$$p = \frac{p_e m}{\left(\binom{n}{2} - m\right)}.$$

For each noise-level, the results are averaged over 20 Monte-Carlo runs.

5.3 Quality Measures

In the graph alignment problem, unfortunately there is no panacea for evaluation, i.e., a best alignment is unknown. The existence of an alignment is guaranteed for the considered problem setup, but the uniqueness of the minimizer is not, in general, given the possible presence of topologically-invariant subgraphs, such as cliques and star graphs. Several widely-used measurements in the field of graph alignment are introduced in this section. In order to explain the definitions of the following metrics, let us recall that each algorithm returns an alignment

$$\hat{\mathbf{P}} = \underset{\mathbf{P}\in\mathcal{P}}{\operatorname{argmin}} \left\|\mathbf{A}_1 - \mathbf{P}\mathbf{A}_2\mathbf{P}^T\right\|_F^2 = \underset{\mathbf{P}\in\mathcal{P}}{\operatorname{argmax}} \operatorname{trace}\left(\mathbf{A}_1\mathbf{P}\mathbf{A}_2\mathbf{P}^T\right), \tag{5.2}$$

and let the ground truth correspondence for the construction of adjacency matrix $\mathbf{A}_2$ in (5.1) be $\mathbf{P}^*$.

5.3.1 Node Correctness (NC)

NC is the percentage of the correctly aligned nodes with respect to the true alignment [60]. In matrix form:

$$\mathrm{NC} = \frac{\mathrm{nnz}\left(\hat{\mathbf{P}} \odot \mathbf{P}^*\right)}{n}, \tag{5.3}$$

where the $\mathrm{nnz}(\cdot)$ function counts the nonzero elements of the argument matrix.

5.3.2 Edge Correctness (EC)

EC is the ratio of the number of conserved edges to the total number of edges in the source network [60]:

$$\mathrm{EC} = \frac{\mathrm{nnz}\Big(\mathbf{A}_1 \odot \left(\hat{\mathbf{P}}\mathbf{A}_2\hat{\mathbf{P}}^T\right)\Big)}{2m}. \tag{5.4}$$

5.3.3 Induced Conserved Structure (ICS)

ICS is the ratio of the number of conserved edges to the number of edges in the subnetwork of G_2 induced on the nodes in G_2 that are aligned to the nodes in G_1 [38]:

$$\mathrm{ICS} = \frac{\mathrm{nnz}\Big(\mathbf{A}_1 \odot \left(\hat{\mathbf{P}}\mathbf{A}_2\hat{\mathbf{P}}^T\right)\Big)}{\mathrm{nnz}\Big(\hat{\mathbf{P}}\mathbf{A}_2\hat{\mathbf{P}}^T\Big)}. \tag{5.5}$$

5.3.4 Symmetric Substructure Score (S^3)

S^3 normalizes the number of conserved edges over both the source and the target graphs [61]:

$$\mathrm{S}^3 = \frac{\mathrm{nnz}\Big(\mathbf{A}_1 \odot \big(\hat{\mathbf{P}}\mathbf{A}_2\hat{\mathbf{P}}^T\big)\Big)}{2m + \mathrm{nnz}\Big(\hat{\mathbf{P}}\mathbf{A}_2\hat{\mathbf{P}}^T\Big) - \mathrm{nnz}\Big(\mathbf{A}_1 \odot \big(\hat{\mathbf{P}}\mathbf{A}_2\hat{\mathbf{P}}^T\big)\Big)}. \tag{5.6}$$

5.3.5 Matched Neighborhood Consistency (MNC)

MNC is the Jaccard similarity of the mapped neighborhood of a node $i \in \mathcal{V}_1$ and a node $j \in \mathcal{V}_2$ [45]. The matrix form of MNC is [62]:

$$\mathbf{MNC} = \mathbf{A}_1\hat{\mathbf{P}}\mathbf{A}_2 \oslash \big(\mathbf{A}_1\hat{\mathbf{P}}\mathbf{1} \otimes \mathbf{1} + \mathbf{1} \otimes \mathbf{A}_2\mathbf{1} - \mathbf{A}_1\hat{\mathbf{P}}\mathbf{A}_2\big), \tag{5.7}$$

where "$\oslash$" denotes the element-wise division. The final score is the average MNC among all nodes

$$\mathrm{MNC} = \mathrm{mean}\Big(\big(\mathbf{MNC} \odot \hat{\mathbf{P}}\big)\mathbf{1}\Big). \tag{5.8}$$

5.3.6 Wall time

Wall time (also called elapsed real time or runtime) is the actual time taken from the start of an algorithm to the end.

5.4 Results and Discussion

5.4.1 Network Statistics

The network statistics of Table 5.1 are used in order to analyze the performance of each algorithm. The number of vertices n and the number of edges m have already been given. The other network statistics are defined in the KONECT Handbook [63].

The average degree $\bar{d}$ is the average number of edges incident to a node in the network

$$\bar{d} = \frac{1}{|\mathcal{V}|}\sum_{u\in\mathcal{V}} d(u) = \frac{2m}{n}. \tag{5.9}$$

The spectral separation $\left|\frac{\lambda_1}{\lambda_2}\right|$ equals the ratio of the largest absolute eigenvalue of the adjacency matrix $\mathbf{A}$ to the second largest absolute eigenvalue.

The global clustering coefficient c_1 (also called transitivity) of a network is the probability that two incident edges are completed by a third edge to form a triangle [64]

$$\begin{aligned} c_1 &= \frac{\big|\{u,v,w \in \mathcal{V} \mid (u,v),(v,w),(w,u) \in \mathcal{E}\}\big|}{\big|\{u,v,w \in \mathcal{V} \mid (u,v),(v,w) \in \mathcal{E}\}\big|} = \frac{3t}{s} = \frac{3\frac{\mathrm{trace}(\mathbf{A}^3)}{3!}}{\sum_{u\in\mathcal{V}}\binom{d(u)}{2}} \\ &= \frac{\mathrm{trace}(\mathbf{A}^3)}{\sum_{u\in\mathcal{V}} d(u)(d(u)-1)}, \end{aligned} \tag{5.10}$$

where t is the number of triangles on the graph, which equals to the number of walks of

length 3 divided by the number of permutations of 3 vertices $3! = 6$, and s is the number of 2-stars (also called wedges), which is a sum of combinations over all vertices. The factor of 3 derives from the fact that the number of triangles ignores multiplicities, in contrast with the number of wedges, and each triangle results in 3 connected triples of vertices, one for each of its 3 vertices. Thus, the clustering coefficient c has values between zero, when it is triangle free, and one, when all possible triangles are formed (i.e., the network consists of disconnected cliques).

The average local clustering coefficient is defined as follows:

$$c_2 = \frac{1}{|\mathcal{V}|} \sum_{u \in \mathcal{V}} c(u), \tag{5.11}$$

where $c(u)$ is the local clustering coefficient of node u. This metric places more weight on the low degree nodes, while the transitivity ratio places more weight on the high degree nodes. The local clustering coefficient is the probability that two randomly chosen and distinct neighbors of u are connected [65]

$$c(u) = \begin{cases} \frac{\left|\{v,w \in \mathcal{V} \mid (u,v),(v,w),(w,u) \in \mathcal{E}\}\right|}{\left|\{v,w \in \mathcal{V} \mid (u,v),(v,w) \in \mathcal{E}\}\right|} = \frac{\mathbf{A}^3[u,u]}{d(u)(d(u)-1)}, & \text{when } d(u) > 1, \\ 0, & \text{when } d(u) \leq 1, \end{cases} \tag{5.12}$$

where $\mathbf{A}^3[u,u]$ is the diagonal element of the third power of adjacency matrix $\mathbf{A}$ corresponding to node u.

Some very clustered parts and some less clustered parts could exist in a network, while another network might have many nodes with a similar clustering coefficient. Thus, the distribution of the clustering coefficient over the nodes in a network raises interest. The cumulative distribution of each real-world and each synthetic network, is illustrated in Figures 5.1 and 5.2, respectively. The distributions are compared with their corresponding normal distributions with mean and standard deviation derived from the true local clustering coefficient of each dataset.

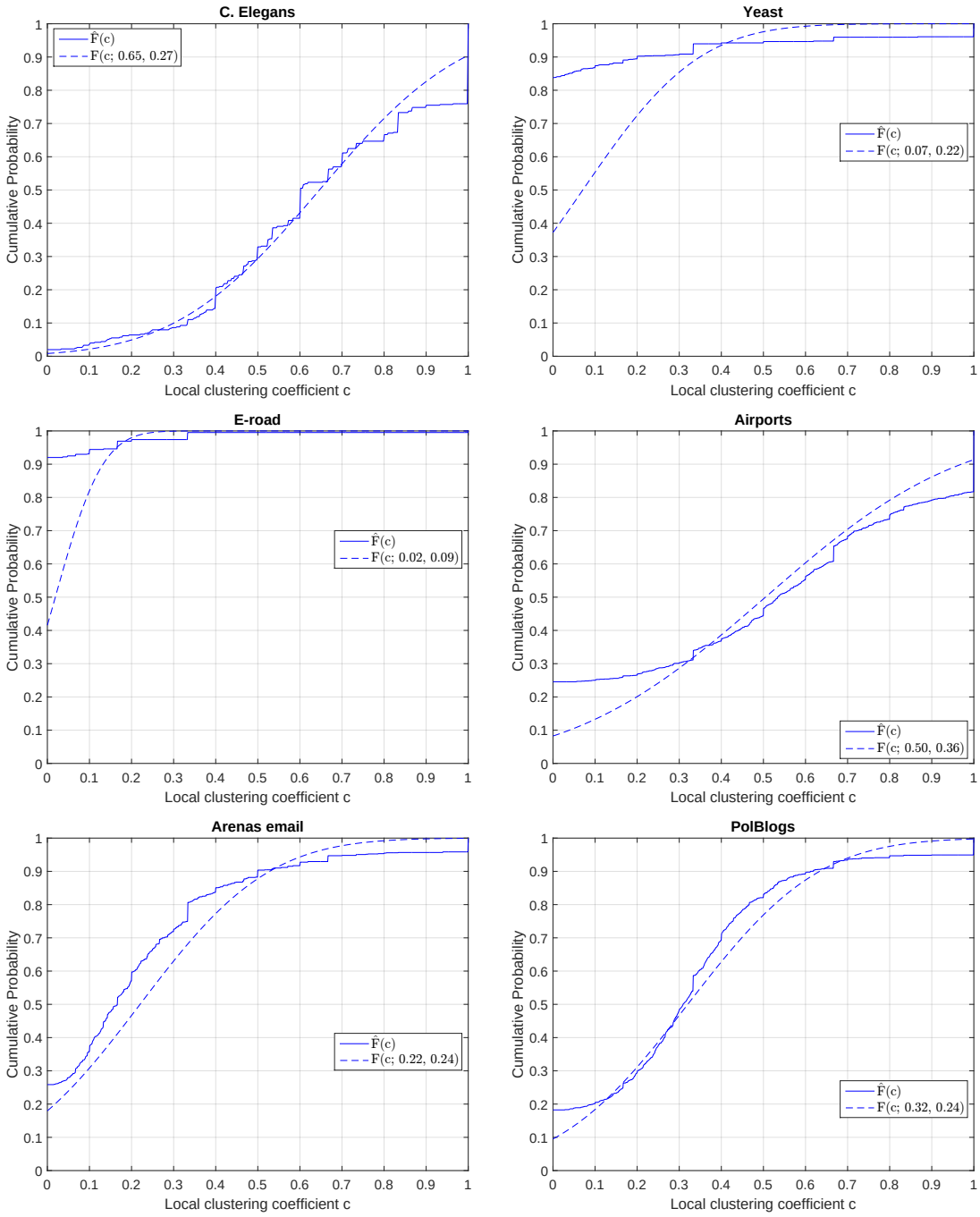


Figure 5.1: Empirical Cumulative Distribution Function $\hat{F}(c)$ of local clustering coefficient $c(u)$ across real-world datasets, compared with the Cumulative Distribution Function of the corresponding normal distribution $F(c; \mu_c, \sigma_c)$, with mean μ_c and standard deviation σ_c.

5.4.2 Real-world networks

Edge Correctness: The EC of each algorithm is shown in Figure 5.3. The performance of UMEYAMA is similar across the datasets and unaffected by their network statistics, except for the YEAST dataset, which is the most challenging one for all algorithms. ISORANK has the worst performance, considering that no prior alignment preferences are given in order to guarantee fairness in the experiment, and improves with spectral separation increasing.

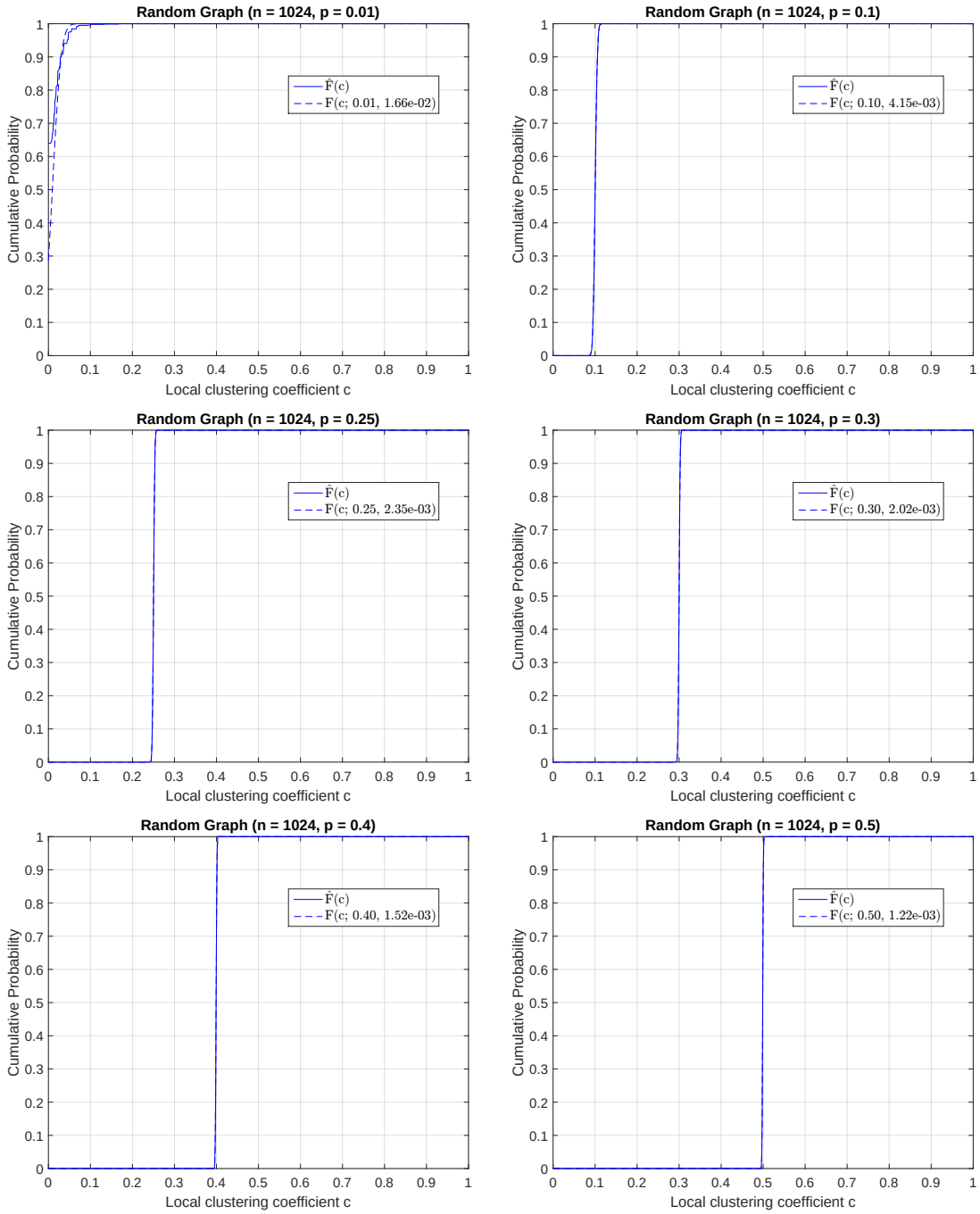


Figure 5.2: Empirical Cumulative Distribution Function $\hat{F}(c)$ of local clustering coefficient $c(u)$ across synthetic datasets, compared with the Cumulative Distribution Function of the corresponding normal distribution $F(c; \mu_c, \sigma_c)$, with mean μ_c and standard deviation σ_c.

The performance of LRA improves naturally with spectral separation increasing, given that the algorithm relies on the k-largest eigenvalues of the adjacency matrix ($k = 3$ in the experiments). CONE has the best overall performance and is affected by how close is the distribution of the clustering coefficient is to a normal distribution in Figure 5.1.

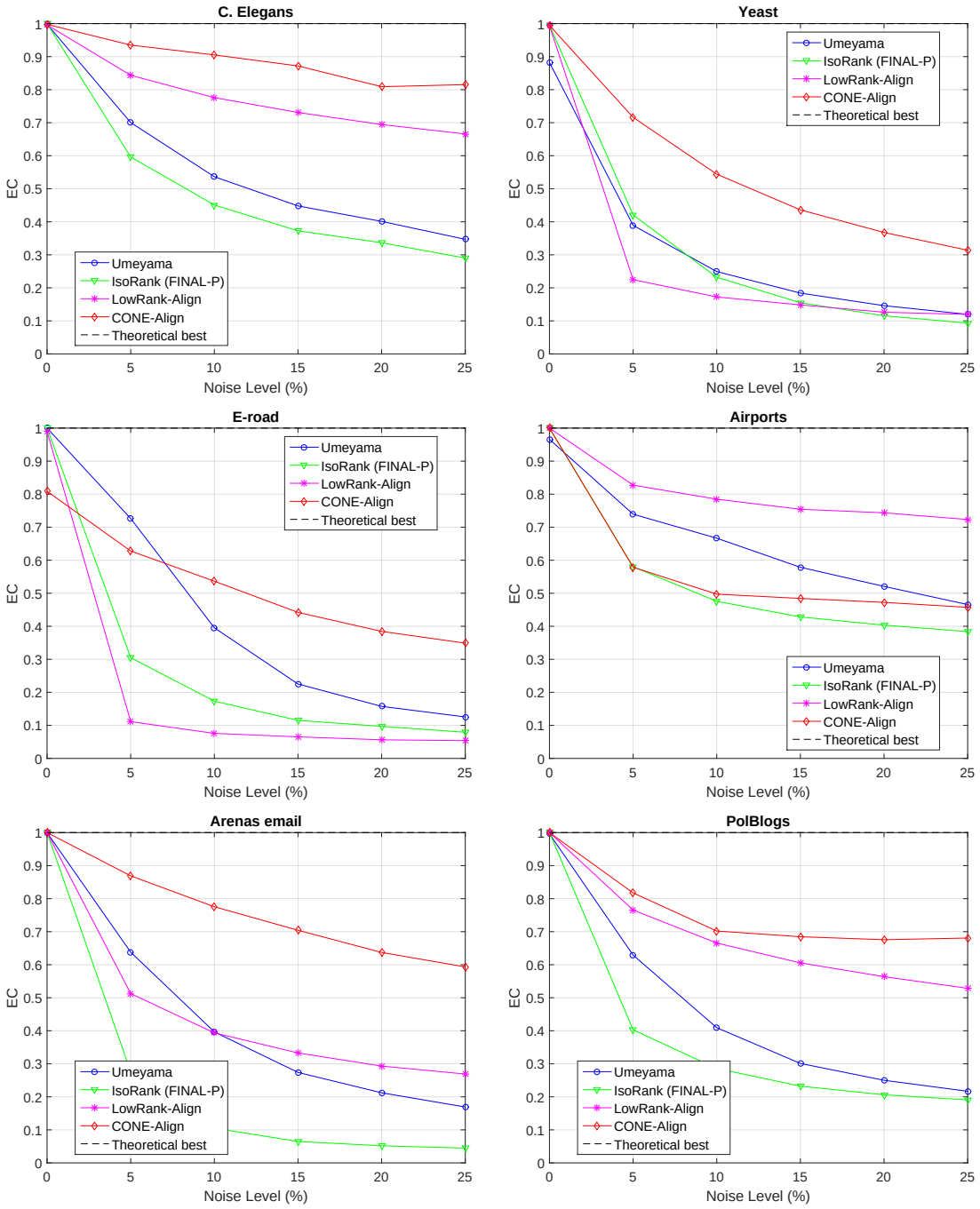


Figure 5.3: Edge Correctness vs. Noise level across real-world datasets. The higher the score, the better.

Node Correctness: The NC of each algorithm is shown in Figure 5.4. Naturally, NC is a more punishing metric than EC, even for 0% noise, considering the symmetries existing in real-world networks [66]. Thus, NC exhibits lower performances, with CONE being the overall best and declining less sharply than the other algorithms as noise increases.

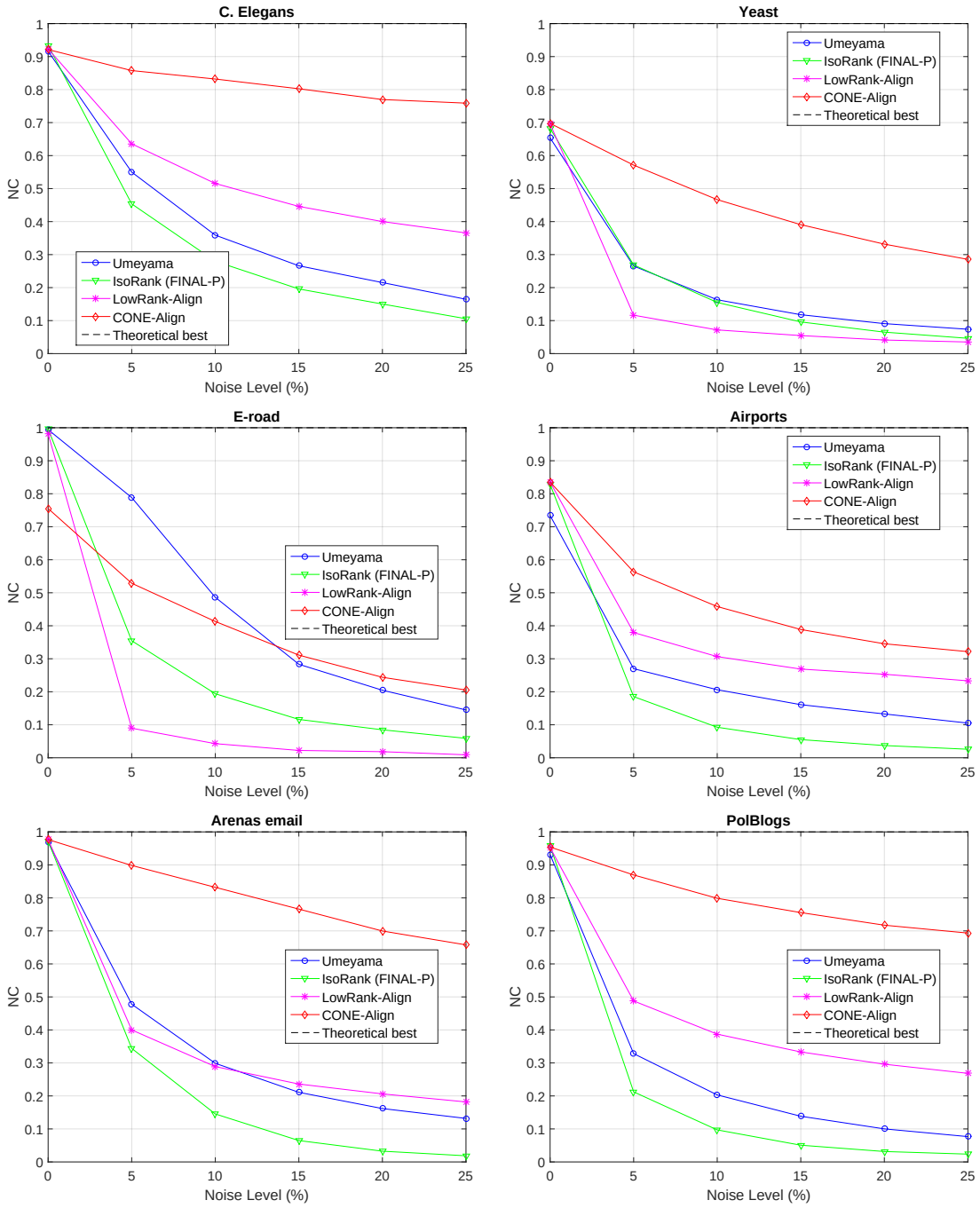


Figure 5.4: Node Correctness vs. Noise level across real-world datasets. The higher the score, the better.

Matched Neighborhood Consistency: MNC, as shown in Figure 5.5, is more strict than NC, except for 0% noise. Note that, in contrast with EC and NC, the theoretical best is affected by the noise level depending on how much the structure of the network changes, which explains the lower performance of the algorithms.

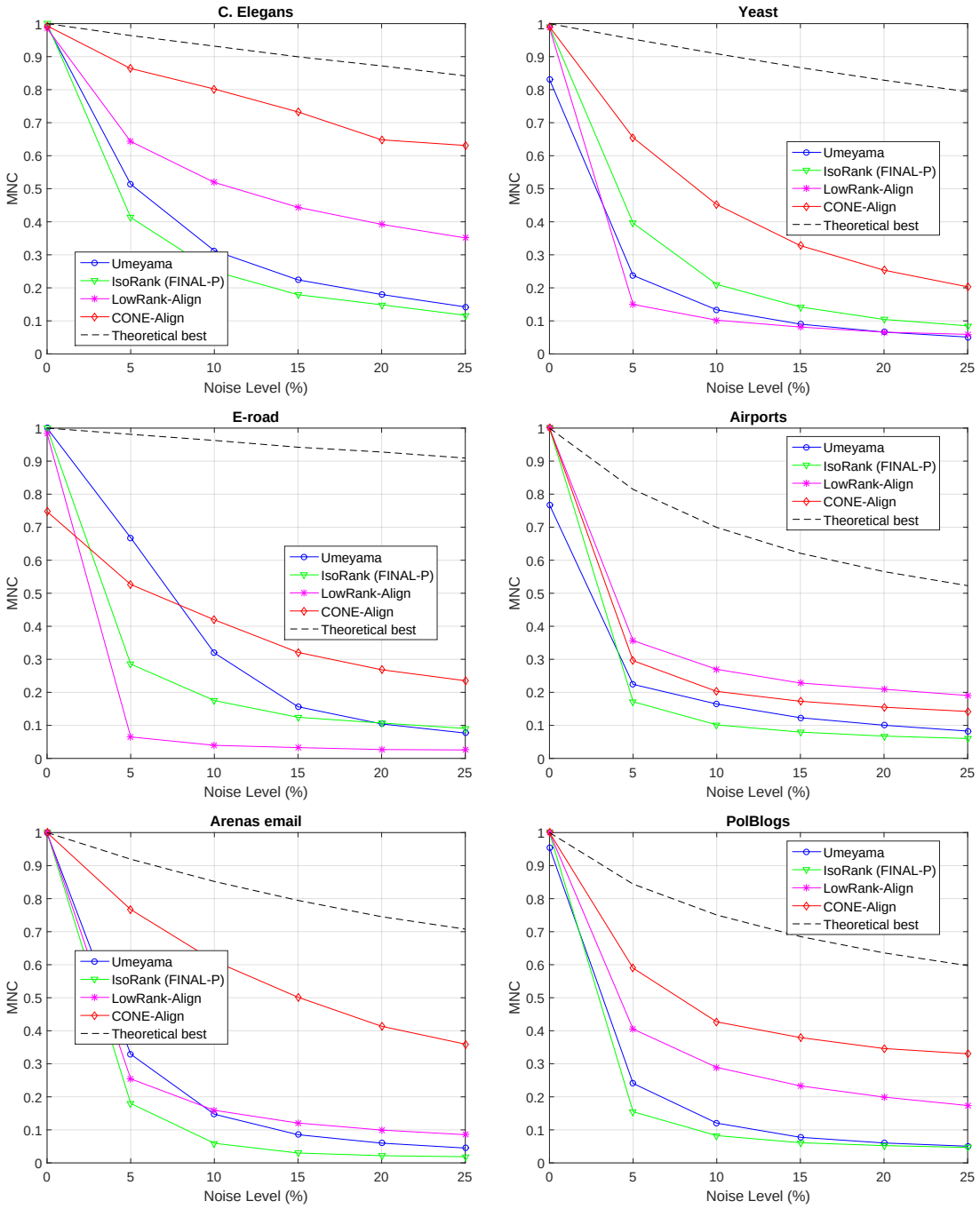


Figure 5.5: Matched Neighborhood Consistency vs. Noise level across real-world datasets. The higher the score, the better.

Induced Conserved Structure: An alignment with a high EC score may not necessarily be the best alignment (Figures 5.4, 5.5). ICS penalizes alignments mapping sparser network regions to denser ones. Thus, ICS scores are slightly lower than EC, following a similar slope, as shown in Figure 5.6.

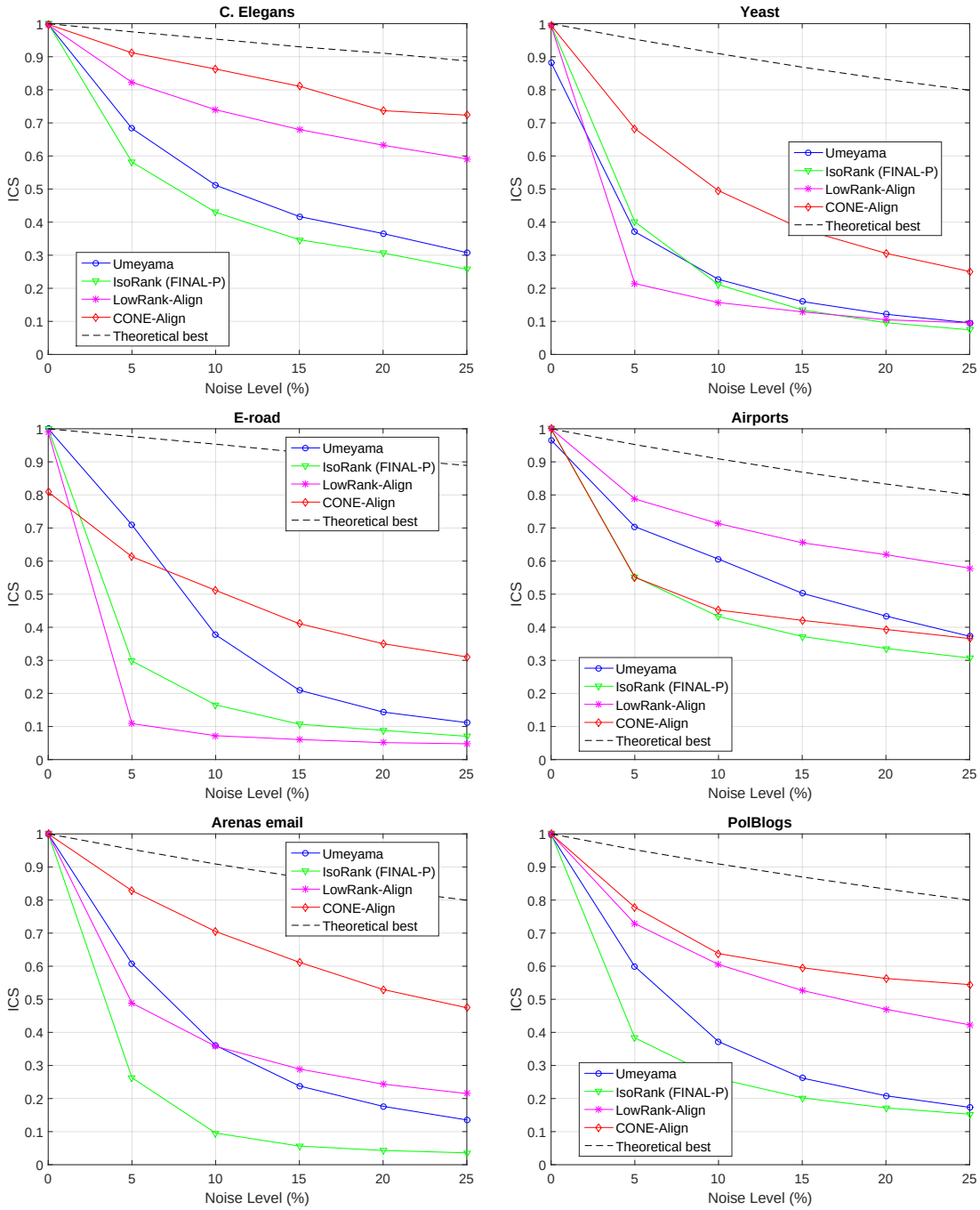


Figure 5.6: Induced Conserved Structure vs. Noise level across real-world datasets. The higher the score, the better.

Symmetric Substructure Score: S^3 penalizes both alignments that map denser network regions to sparser ones and alignments that map sparser network regions to denser ones. Thus, S3 scores are slightly lower than ICS, following a similar slope, as shown in Figure 5.7.

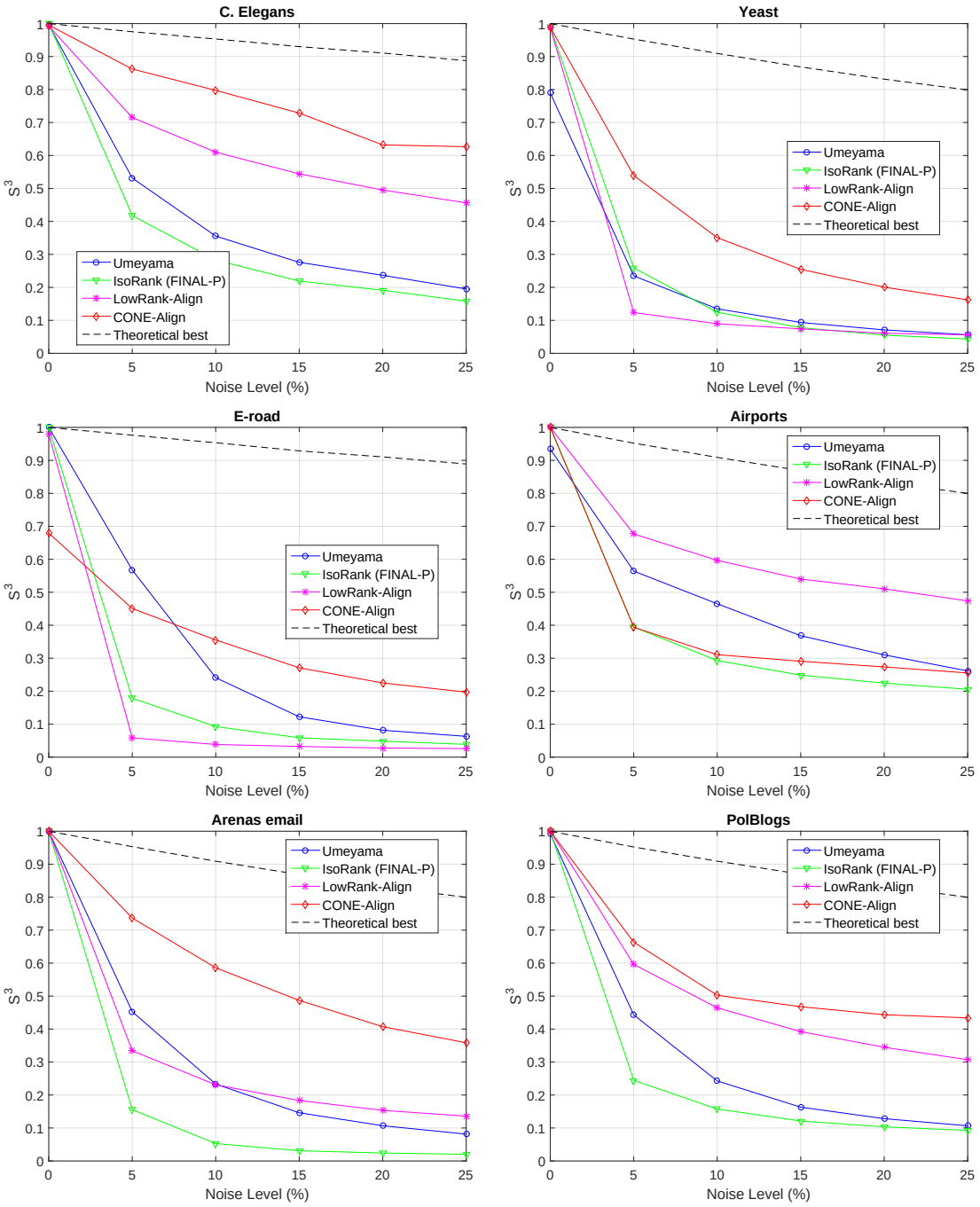


Figure 5.7: Symmetric Substructure Score vs. Noise level across real-world datasets. The higher the score, the better.

Wall time: The wall time of each algorithm, as shown in Figure 5.8, naturally depends on the number of vertices n and the number of edges m and it appears to be unaffected by the noise level. UMEYAMA takes the least time to execute and CONE the most, without being unaffordable.

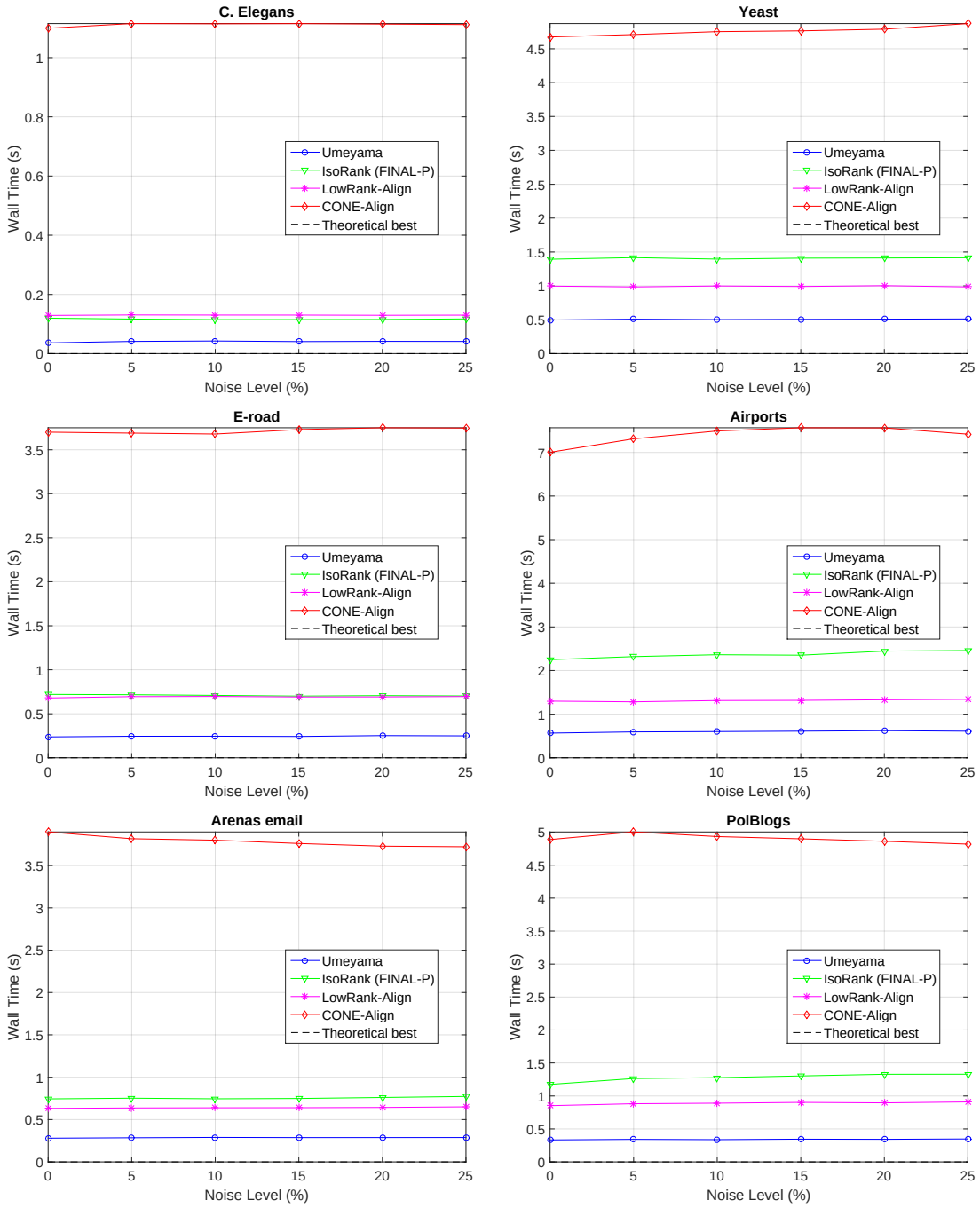


Figure 5.8: Wall time (in seconds) vs. Noise level across real-world datasets. The lower the curve, the better.

5.4.3 Synthetic Networks

Edge Correctness: As shown in Figure 5.9, CONE achieves high EC in synthetic networks, especially in cases where $p \in \{0.1, 0.25, 0.3\}$, considering that, as shown in Figure 5.2, their clustering coefficient distribution is concentrated around its mean, the average local clustering coefficient. UMEYAMA, ISORANK and LRA have commonly low performance, given that the spectrum of Erdös-Rényi graphs yields inadequate content, because it has the same distribution as that of the Gaussian orthogonal ensemble [67]. Their

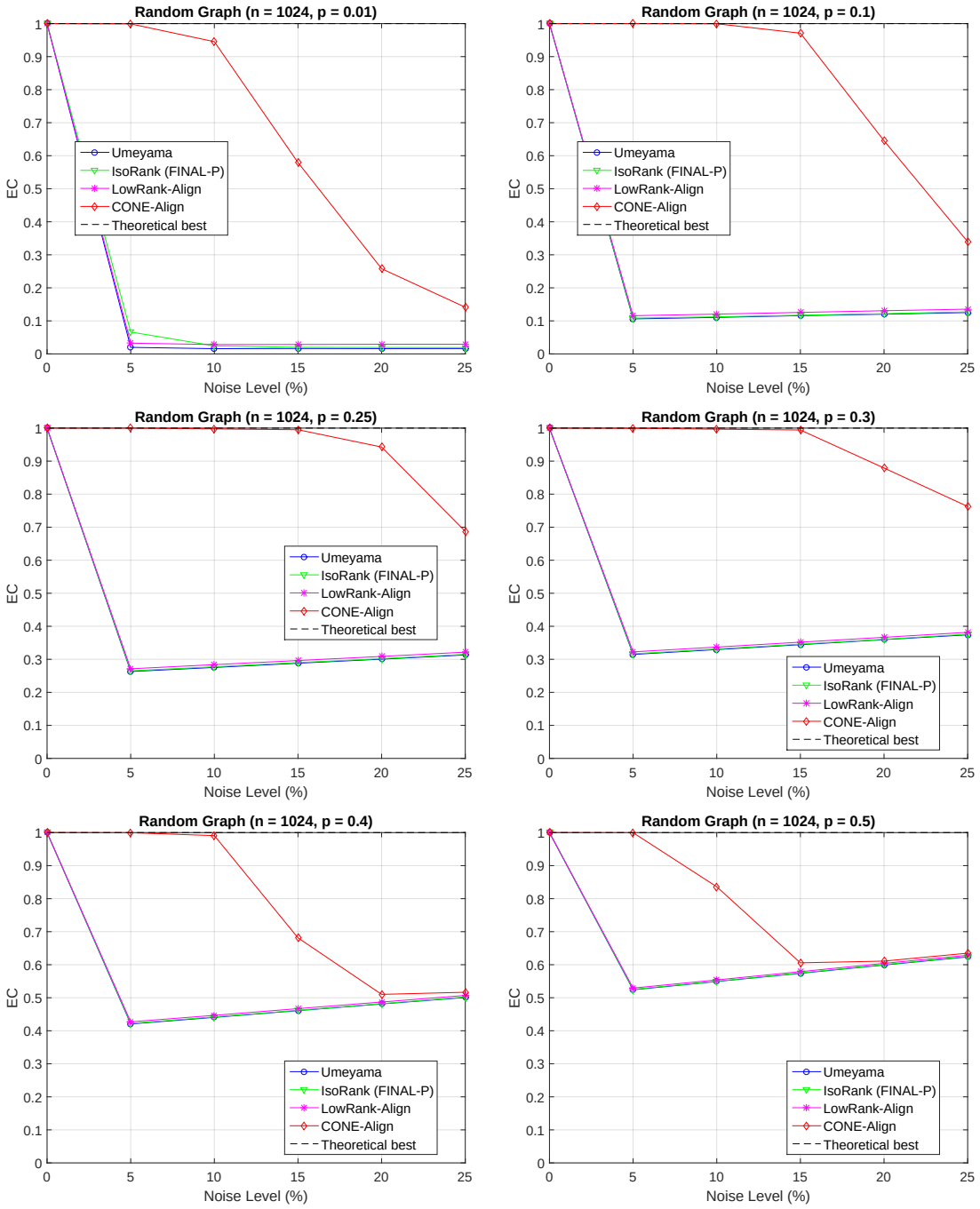


Figure 5.9: Edge Correctness vs. Noise level across synthetic datasets. The higher the score, the better.

performance improves as edge probability p increases.

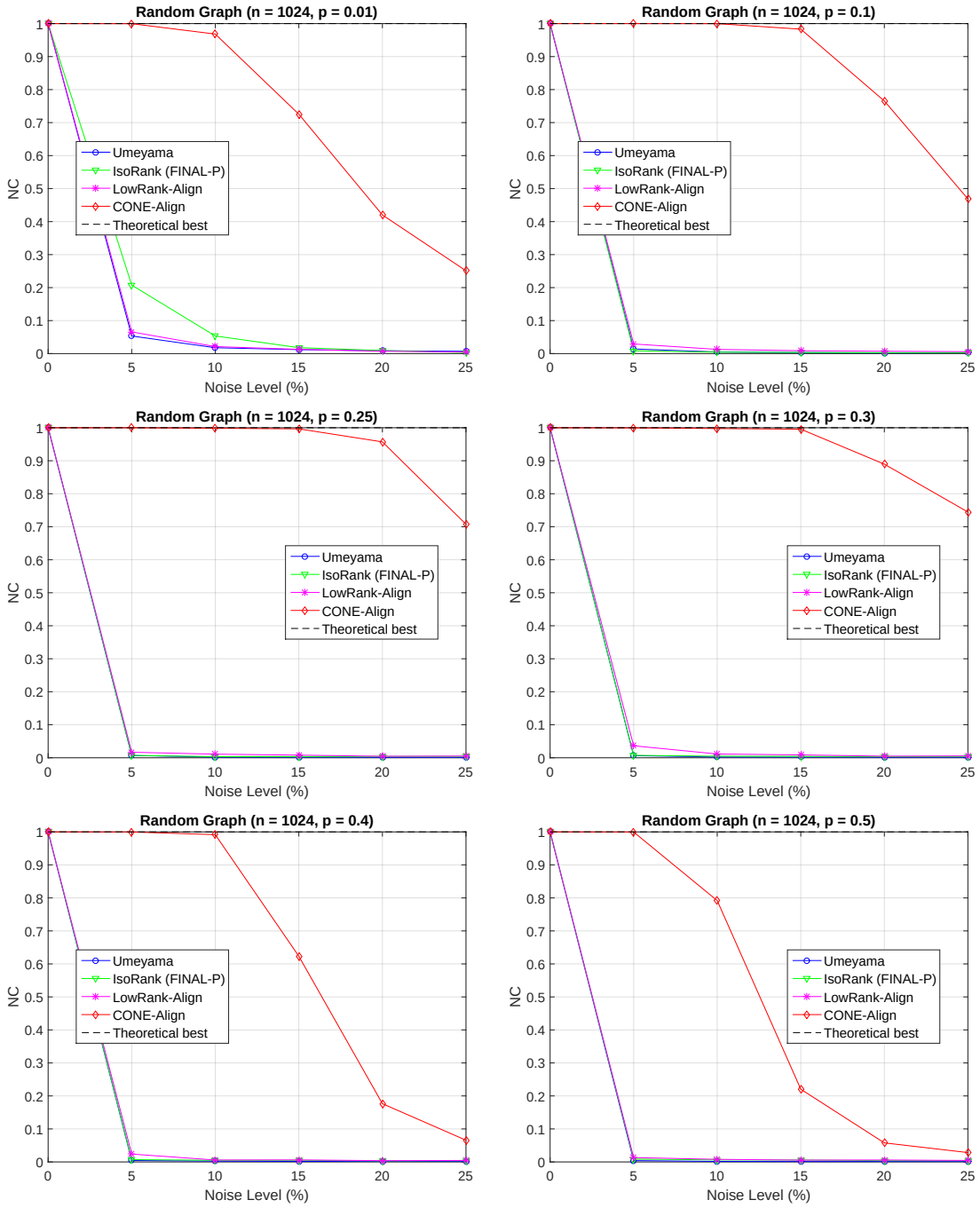


Figure 5.10: Node Correctness vs. Noise level across synthetic datasets. The higher the score, the better.

Node Correctness: As $|p - 0.25|$ increases the NC scores of CONE, as shown in Figure 5.10, differ more from their corresponding EC in Figure 5.9, considering that the asymmetries in a Erdös-Rényi graph are decreasing [68]. The other algorithms achieve zero NC for noise higher than 5%.

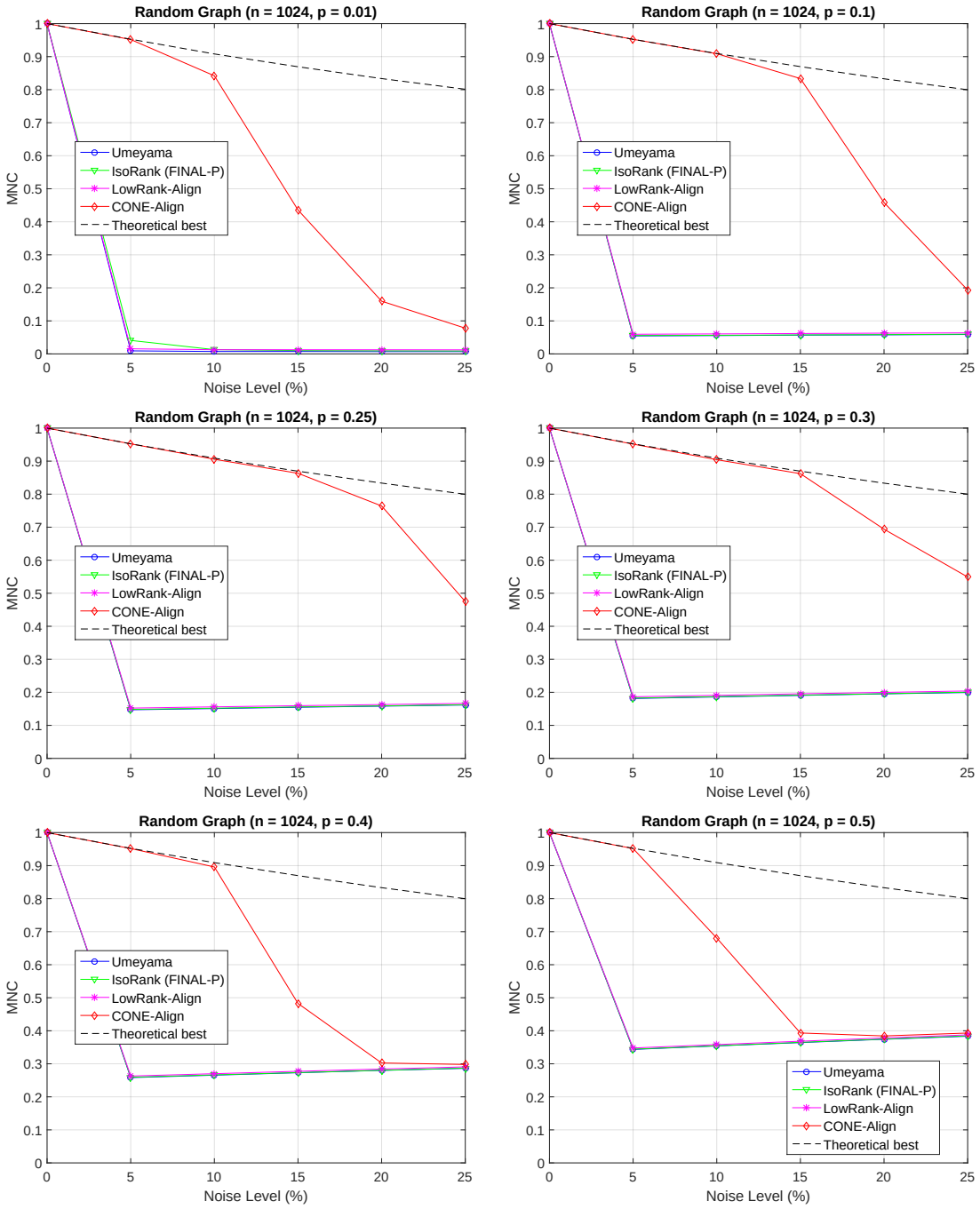


Figure 5.11: Matched Neighborhood Consistency vs. Noise level across synthetic datasets. The higher the score, the better.

Matched Neighborhood Consistency: The theoretical best MNC, as shown in Figure 5.11, is affected by noise level regardless of the edge probability p. This has influence on the performance of the algorithms.

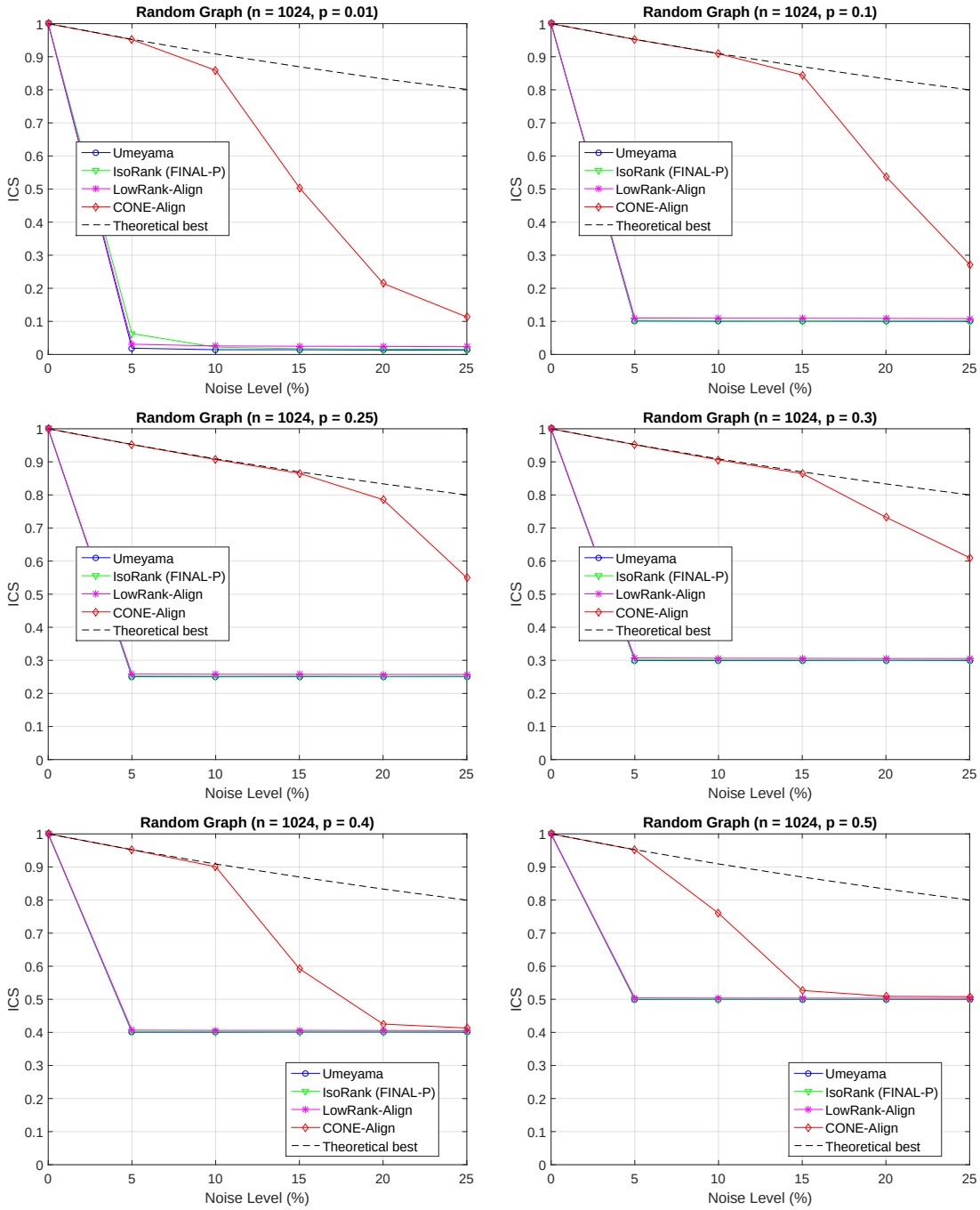


Figure 5.12: Induced Conserved Structure vs. Noise level across synthetic datasets. The higher the score, the better.

Induced Conserved Structure and Symmetric Substructure Score: ICS and S^3, as shown in Figures 5.12 and 5.13, respectively, follow the same pattern as in real-world networks.

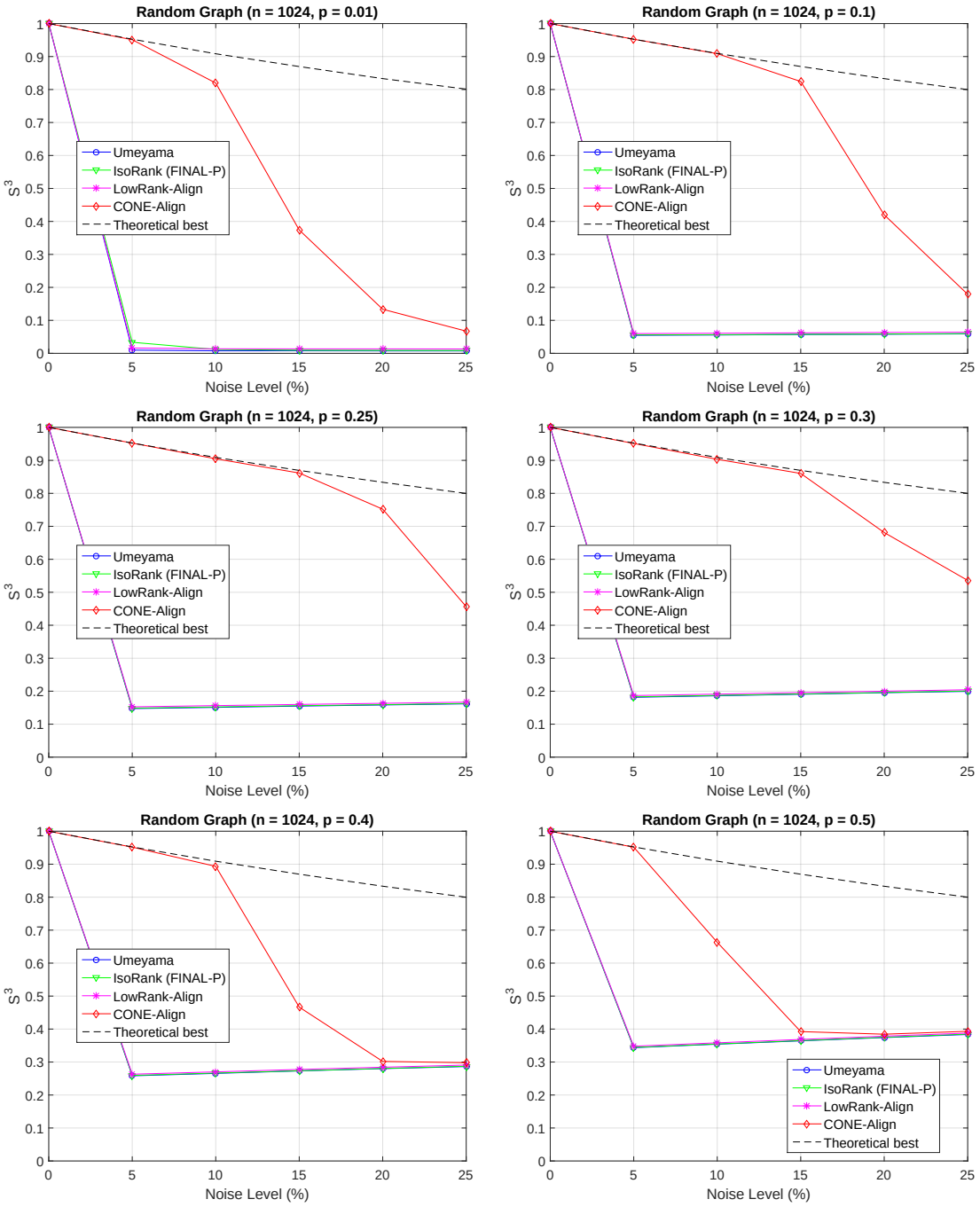


Figure 5.13: Symmetric Substructure Score vs. Noise level across synthetic datasets. The higher the score, the better.

Wall time: As shown in Figure 5.14, the wall time of IsoRank, LRA and CONE is affected by the edge probability p and increases with noise level. The wall time of Umeyama depends only on the number of nodes and is unaffected by the noise level.

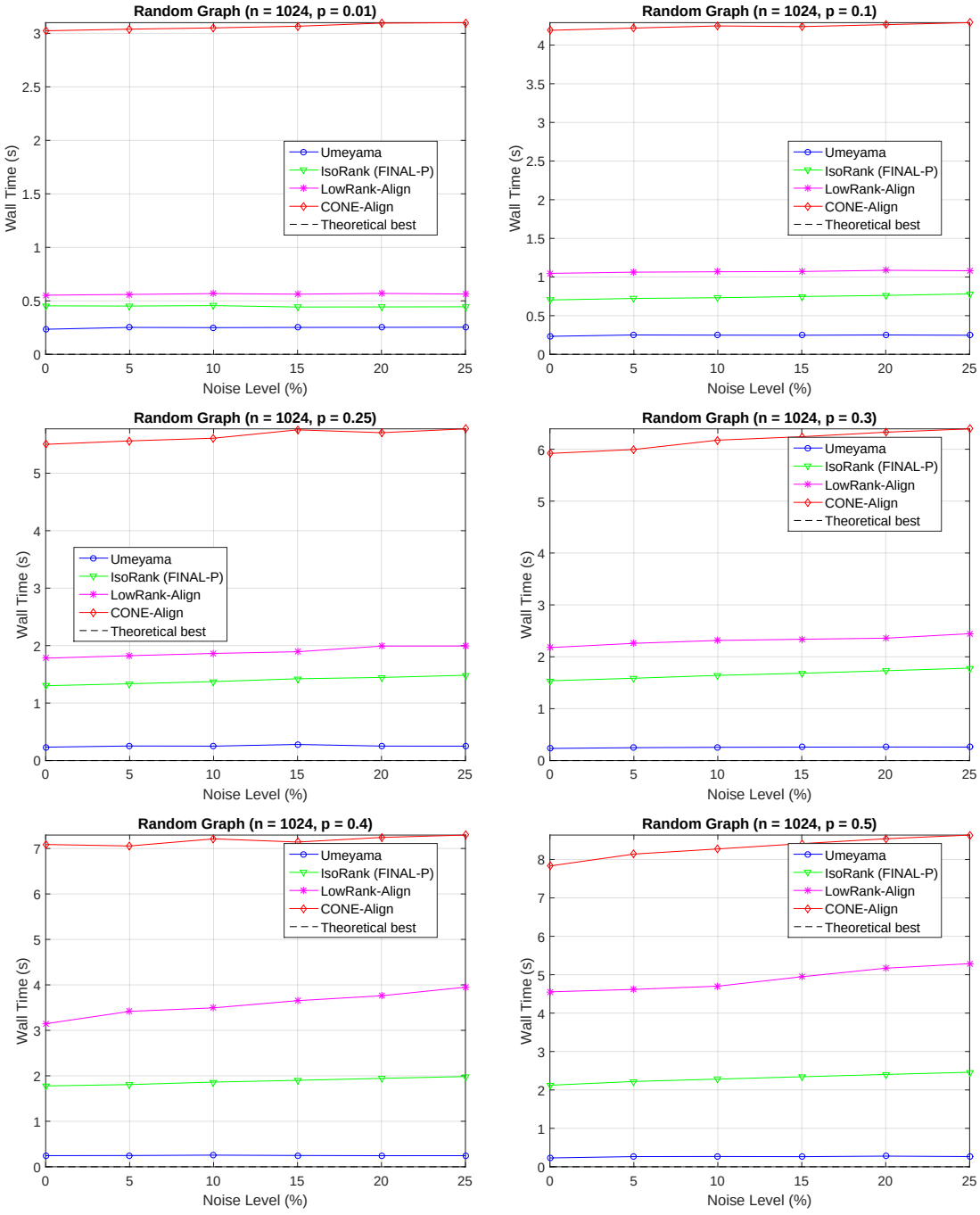


Figure 5.14: Wall time (in seconds) vs. Noise level across synthetic datasets. The lower the curve, the better.

Chapter 6

Conclusion and future work

In this thesis, we provide a comprehensive presentation and comparison of state-of-the-art graph alignment techniques. With generalized mathematical formulations of the graph alignment problem and the Linear Assignment Problem, the algorithms can be compared fairly using several widely used measures in the literature, regarding network topological properties.

Edge Correctness is the most common measure, but it is the most tolerant one. Node Correctness is too strict, because it ignores the possible symmetries in a network. Symmetric Substructure Score and Matched Neighborhood Consistency are the most fair measures.

CONE is affected by the distribution of the local clustering coefficient, performs excellent over synthetic networks and has the best overall performance in real-world networks, but is the most time costly. LRA succeeds when the largest absolute eigenvalues are far apart. IsoRank has the worst overall performance, given that the method relies on prior alignment preference and drops all the constrains of the Quadratic Assignment Problem. Umeyama has steady, but not outstanding, performance over real-world datasets, is the most time efficient and is unaffected by network topological properties.

As future work, one could include exhaustive parameter tuning for the algorithms, instead of using the defaults. Furthermore, directed graphs and attributed networks may need different formulations, leading to different algorithmic approaches. Finally, given that there exist a vast amount of algorithms that solve the graph alignment problem, but are not mentioned in this work, there is always room for further study.

Appendix A

Proofs

A.1 Equivalence of IsoRank and FINAL-P

It is easy to show the equivalence of IsoRank and FINAL-P:

$$\begin{aligned}
\mathbf{s} &= \alpha \mathbf{D}_U^{-\frac{1}{2}}(\mathbf{A}_1 \otimes \mathbf{A}_2)\mathbf{D}_U^{-\frac{1}{2}}\mathbf{s} + (1-\alpha)\mathbf{h} \\
&= \alpha(\mathbf{D}_1 \otimes \mathbf{D}_2)^{-\frac{1}{2}}(\mathbf{A}_1 \otimes \mathbf{A}_2)(\mathbf{D}_1 \otimes \mathbf{D}_2)^{-\frac{1}{2}}\mathbf{s} + (1-\alpha)\mathbf{h} \\
&= \alpha(\mathbf{D}_1^{-\frac{1}{2}} \otimes \mathbf{D}_2^{-\frac{1}{2}})(\mathbf{A}_1 \otimes \mathbf{A}_2)(\mathbf{D}_1^{-\frac{1}{2}} \otimes \mathbf{D}_2^{-\frac{1}{2}})\mathbf{s} + (1-\alpha)\mathbf{h} \\
&= \alpha(\mathbf{D}_1^{-\frac{1}{2}}\mathbf{A}_1 \otimes \mathbf{D}_2^{-\frac{1}{2}}\mathbf{A}_2)(\mathbf{D}_1^{-\frac{1}{2}} \otimes \mathbf{D}_2^{-\frac{1}{2}})\mathbf{s} + (1-\alpha)\mathbf{h} \\
&= \alpha(\mathbf{D}_1^{-\frac{1}{2}}\mathbf{A}_1\mathbf{D}_1^{-\frac{1}{2}} \otimes \mathbf{D}_2^{-\frac{1}{2}}\mathbf{A}_2\mathbf{D}_2^{-\frac{1}{2}})\mathbf{s} + (1-\alpha)\mathbf{h} \\
&= \alpha(\tilde{\mathbf{A}}_1 \otimes \tilde{\mathbf{A}}_2)\mathbf{s} + (1-\alpha)\mathbf{h} \\
&= \alpha\tilde{\mathbf{C}}\mathbf{s} + (1-\alpha)\mathbf{h}
\end{aligned} \tag{A.1}$$

Bibliography

[1] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[2] R. Sharan and T. Ideker, "Modeling cellular machinery through biological network comparison," *Nature biotechnology*, vol. 24, no. 4, pp. 427–433, 2006.

[3] R. J. Trudeau, *Introduction to graph theory*. Courier Corporation, 2013.

[4] S. Umeyama, "An eigendecomposition approach to weighted graph matching problems," *IEEE transactions on pattern analysis and machine intelligence*, vol. 10, no. 5, pp. 695–703, 1988.

[5] F. E. Faisal, L. Meng, J. Crawford, and T. Milenković, "The post-genomic era of biological network alignment," *EURASIP Journal on Bioinformatics and Systems Biology*, vol. 2015, no. 1, pp. 1–19, 2015.

[6] C. Schellewald and C. Schnörr, "Probabilistic subgraph matching based on convex relaxation," in *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*. Springer, 2005, pp. 171–186.

[7] R. Zafarani and H. Liu, "Connecting users across social media sites: a behavioral-modeling approach," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 41–49.

[8] H. Zhu, R. Xie, Z. Liu, and M. Sun, "Iterative entity alignment via knowledge embeddings," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[9] L. Euler, "Solutio problematis ad geometriam situs pertinentis," *Commentarii academiae scientiarum Petropolitanae*, pp. 128–140, 1741.

[10] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artifical Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

[11] L. Stanković, M. Daković, and E. Sejdić, "Introduction to graph signal processing," in *Vertex-Frequency Analysis of Graph Signals*. Springer, 2019, pp. 3–108.

[12] L. Livi and A. Rizzi, "The graph matching problem," *Pattern Analysis and Applications*, vol. 16, no. 3, pp. 253–283, 2013.

[13] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[14] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1105–1114.

[15] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *Proceedings of the eleventh ACM international conference on web search and data mining*, 2018, pp. 459–467.

[16] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[17] C.-Y. Ma and C.-S. Liao, "A review of protein–protein interaction network alignment: From pathway comparison to global alignment," *Computational and Structural Biotechnology Journal*, vol. 18, pp. 2647–2656, 2020.

[18] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty years of graph matching in pattern recognition," *International journal of pattern recognition and artificial intelligence*, vol. 18, no. 03, pp. 265–298, 2004.

[19] P. Foggia, G. Percannella, and M. Vento, "Graph matching and learning in pattern recognition in the last 10 years," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 28, no. 01, p. 1450001, 2014.

[20] S. S. Ray, *Graph theory with algorithms and its applications: in applied science and technology*. Springer, 2013.

[21] J. Kobler, U. Schöning, and J. Torán, *The graph isomorphism problem: its structural complexity*. Springer Science & Business Media, 2012.

[22] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[23] R. E. Burkard and E. Cela, "Linear assignment problems and extensions," in *Handbook of combinatorial optimization*. Springer, 1999, pp. 75–149.

[24] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[25] D. Bertsekas, D. Castañon, J. Eckstein, and S. Zenios, "Parallel computing in network optimization," *Handbooks in Operations Research and Management Science*, vol. 7, pp. 331–399, 1995.

[26] G. Kollias, S. Mohammadi, and A. Grama, "Network similarity decomposition (nsd): A fast and scalable approach to network alignment," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 12, pp. 2232–2243, 2011.

[27] B. Korte and D. Hausmann, "An analysis of the greedy heuristic for independence systems," in *Annals of Discrete Mathematics*. Elsevier, 1978, vol. 2, pp. 65–74.

[28] K. Skitsas, K. Orłowski, J. Hermanns, D. Mottin, and P. Karras, "Comprehensive evaluation of algorithms for unrestricted graph alignment." EDBT, 2023.

[29] B. Jiang, H. Zhao, J. Tang, and B. Luo, "A sparse nonnegative matrix factorization technique for graph matching problems," *Pattern Recognition*, vol. 47, no. 2, pp. 736–747, 2014.

[30] A. Konar and N. D. Sidiropoulos, "Graph matching via the lens of supermodularity," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[31] F. Zhou and F. De la Torre, "Factorized graph matching," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 127–134.

[32] H. D. Sherali and W. P. Adams, *A reformulation-linearization technique for solving discrete and continuous nonconvex problems*. Springer Science & Business Media, 2013, vol. 31.

[33] K. M. Anstreicher and N. W. Brixius, "Solving quadratic assignment problems using convex quadratic programming relaxations," *Optimization Methods and Software*, vol. 16, no. 1-4, pp. 49–68, 2001.

[34] J. T. Vogelstein, J. M. Conroy, V. Lyzinski, L. J. Podrazik, S. G. Kratzer, E. T. Harley, D. E. Fishkind, R. J. Vogelstein, and C. E. Priebe, "Fast approximate quadratic programming for graph matching," *PLOS one*, vol. 10, no. 4, p. e0121002, 2015.

[35] G. W. Klau, "A new graph-based method for pairwise global network alignment," *BMC bioinformatics*, vol. 10, no. 1, pp. 1–9, 2009.

[36] J. Peng, H. Mittelmann, and X. Li, "A new relaxation framework for quadratic assignment problems based on matrix splitting," *Mathematical Programming Computation*, vol. 2, no. 1, pp. 59–77, 2010.

[37] 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*, vol. 105, no. 35, pp. 12 763–12 768, 2008.

[38] R. Patro and C. Kingsford, "Global network alignment using multiscale spectral signatures," *Bioinformatics*, vol. 28, no. 23, pp. 3105–3114, 2012.

[39] S. Feizi, G. Quon, M. Recamonde-Mendoza, M. Medard, M. Kellis, and A. Jadbabaie, "Spectral alignment of graphs," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 3, pp. 1182–1197, 2019.

[40] C. Godsil and G. F. Royle, *Algebraic graph theory*. Springer Science & Business Media, 2001, vol. 207.

[41] N. J. Higham, "Cholesky factorization," *Wiley interdisciplinary reviews: computational statistics*, vol. 1, no. 2, pp. 251–254, 2009.

[42] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[43] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[44] M. Heimann, H. Shen, T. Safavi, and D. Koutra, "Regal: Representation learning-based graph alignment," in *Proceedings of the 27th ACM international conference on information and knowledge management*, 2018, pp. 117–126.

[45] X. Chen, M. Heimann, F. Vahedian, and D. Koutra, "Cone-align: Consistent network alignment with proximity-preserving node embedding," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 1985–1988.

[46] H. Xu, D. Luo, H. Zha, and L. C. Duke, "Gromov-wasserstein learning for graph matching and node embedding," in *International conference on machine learning*. PMLR, 2019, pp. 6932–6941.

[47] A. J. Hoffman and H. W. Wielandt, "The variation of the spectrum of a normal matrix," in *Selected Papers Of Alan J Hoffman: With Commentary*. World Scientific, 2003, pp. 118–120.

[48] R. Singh, J. Xu, and B. Berger, "Pairwise global alignment of protein interaction networks by matching neighborhood topology," in *Annual international conference on research in computational molecular biology*. Springer, 2007, pp. 16–31.

[49] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[50] S. Zhang and H. Tong, "Final: Fast attributed network alignment," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1345–1354.

[51] E. Grave, A. Joulin, and Q. Berthet, "Unsupervised alignment of embeddings with wasserstein procrustes," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1880–1890.

[52] X. Du, J. Yan, and H. Zha, "Joint link prediction and network alignment via cross-graph embedding." in *IJCAI*, 2019, pp. 2251–2257.

[53] P. H. Schönemann, "A generalized solution of the orthogonal procrustes problem," *Psychometrika*, vol. 31, no. 1, pp. 1–10, 1966.

[54] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *Naval research logistics quarterly*, vol. 3, no. 1-2, pp. 95–110, 1956.

[55] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," *Advances in neural information processing systems*, vol. 26, 2013.

[56] J. Kunegis, "Konect: the koblenz network collection," in *Proceedings of the 22nd international conference on world wide web*, 2013, pp. 1343–1350.

[57] P. Erdős, A. Rényi *et al.*, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[58] H. Nassar, N. Veldt, S. Mohammadi, A. Grama, and D. F. Gleich, "Low rank spectral network alignment," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 619–628.

[59] T. Derr, H. Karimi, X. Liu, J. Xu, and J. Tang, "Deep adversarial network alignment," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 352–361.

[60] T. Milenković, W. L. Ng, W. Hayes, and N. Pržulj, "Optimal network alignment with graphlet degree vectors," *Cancer informatics*, vol. 9, pp. CIN–S4744, 2010.

[61] V. Saraph and T. Milenković, "Magna: maximizing accuracy in global network alignment," *Bioinformatics*, vol. 30, no. 20, pp. 2931–2940, 2014.

[62] M. Heimann, X. Chen, F. Vahedian, and D. Koutra, "Refining network alignment to improve matched neighborhood consistency," in *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM, 2021, pp. 172–180.

[63] J. Kunegis, "Handbook of network analysis [KONECT project]," *CoRR*, vol. abs/1402.5500, no. v4, 2018. [Online]. Available: https://github.com/kunegis/konect-handbook/blob/master/konect-handbook.pdf

[64] M. E. Newman, D. J. Watts, and S. H. Strogatz, "Random graph models of social networks," *Proceedings of the national academy of sciences*, vol. 99, no. suppl_1, pp. 2566–2572, 2002.

[65] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[66] D. Smith and B. Webb, "Hidden symmetries in real and theoretical networks," *Physica A: Statistical Mechanics and its Applications*, vol. 514, pp. 855–867, 2019.

[67] L. Erdős, A. Knowles, H.-T. Yau, and J. Yin, "Spectral statistics of erdős-rényi graphs ii: Eigenvalue spacing and the extreme eigenvalues," *Communications in Mathematical Physics*, vol. 314, no. 3, pp. 587–640, 2012.

[68] P. Erdos and A. Rényi, "Asymmetric graphs," *Acta Math. Acad. Sci. Hungar*, vol. 14, no. 295-315, p. 3, 1963.