DEGREE MATRIX COMPARISON FOR GRAPH ALIGNMENT

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Bachelor of Arts

in

Mathematics

by Ashley Wang

Advised by Peter Chin and Peter Mucha

Dartmouth College

Hanover, New Hampshire

June 2025

Abstract

The graph alignment problem, which considers the optimal node correspondence across networks, recently gained significant attention due to its wide applications. There are graph alignment methods suited for various network types, but we focus on the unsupervised geometric alignment algorithms. First and foremost, we propose Degree Matrix Comparison (DMC), a very simple degree-based method that has shown to be effective for heterogeneous networks. Additionally, we propose some variations of this method, including Greedy DMC with lower time complexity, Weighted DMC suitable for aligning weighted graphs, and Ricci Matrix Comparison (RMC) for scenarios where the Forman-Ricci curvature is suitable as a signature of nodes.

We performed detailed theoretical analysis and conducted extensive experiments to demonstrate the potential of this sequence of graph alignment methods. Remarkably, DMC achieves up to 99% correct node alignment for 90%-overlap networks and 100% accuracy for isomorphic graphs. Positive results from applying the variational methods furthermore speak to the validity and potential of the initially proposed DMC. The sequence of methods could significantly impact graph alignment, offering reliable and simple solutions for the task.

Acknowledgements

This undergraduate thesis is a collection of my series of projects since junior year on graph alignment. The two resulting papers are publicly available on arXiv [37, 38]. I would also like to thank a few people who have either academically or emotionally (or both) supported me throughout my undergraduate journey.

I would first like to express my gratitude to Prof. Peter Chin, whom I have worked with for more than a year. He has warmly supported my research journey in my junior and senior years, continuously encouraging me along my way, especially during application season to graduate school when I often doubted my own capabilities. We have bonded well outside of research too.

I would like to thank Prof. Peter Mucha who has given me many pieces of advice throughout my journey to graduate school. We have interacted through a course and my random door knocks which led to conversations prior to working on the thesis. He has also provided me with guidance throughout the thesis with valuable advice.

I would like to thank my family for their unconditional love and support, as always. I am particularly grateful to Dartmouth for giving me the lifelong memory of spending a year in college with my younger brother, Adam Wang '28.

I would also like to thank Guanming Liang '25, an avid supporter of my journey toward becoming a mathematician.

Last but not least, I would like thank my best friend, Ziying Yin, who has known me for ten years and is also a mathematics major at Peking University. I am honored

to have a friend like her who shares similar interests and passions. Her optimism and independence have surely inspired me. It is simply amazing that we are still in close touch, thanks to the invention of video calls, even though we were actually classmates for only three years in middle school.

I sincerely thank all other friends who have heard me out, encouraged me, and inspired me.

Contents

Chapter 1

Introduction

Graph alignment is a critical task with applications across various domains, like social networks, biology, and cybersecurity. In social networks, graph alignment can identify the same user across multiple platforms, providing insights into user behavior and preferences. In biology, it helps in comparing molecular structures and understanding functional similarities between biological networks. In cybersecurity, aligning graphs can detect coordinated attacks and identify malicious entities by correlating data from different sources. Graph alignment is also highly related to the subgraph isomorphism problem in mathematics and computer science, which is an NP-complete problem. Improving the efficiency and simplifying implementation steps of graph alignment algorithms are important in artificial intelligence because they help us learn patterns on networks.

There are supervised graph alignment methods that work well already [23]. This is not surprising as supervised and semi-supervised methods exploit node attributes, such as semantic features of users [34, 42]. However, they break down when users disguise their identities and node attributes are no longer reflective of local properties [13]. Moreover, they require some knowledge of the node labeling on a network prior to the alignment. There are also unsupervised methods that rely on the power of node

attributes, like WAlign [11]. We, however, focus on the unsupervised graph alignment problem for unattributed plain graphs, mainly leveraging geometric properties.

Degree Matrix Comparison (DMC), which is introduced in Section 3.1, is a method that fits the task. We list some previous unsupervised methods that exploit graph geometries: REGAL, which is a representation learning-based method [14] ; FINAL, which mainly utilizes graph topology but enhances it with node attributes [43] ; Klau, which utilizes a Lagrangian relaxation approach in an optimization problem [16] ; and IsoRank, which constructs an eigenvalue problem for every pair of input graphs and then uses k-partite matching techniques [31]. For a more comprehensive view of past unsupervised alignment methods, see [32].

DMC was constructed with the following two core ideas in mind. First of all, degree is the most accurate and direct description of local structure. Moreover, in representing geometric structure, it makes sense to include both local and global information, like in [18, 25], especially in compact matrix form (like in adjacency matrices and graph Laplacian matrices). This is not just because "the more the better", rather it is a consequence of the Friendship Paradox; this will be more thoroughly discussed in Chapter 4. More specifically, we propose foregoing the need to know exact connections between nodes, like in an adjacency matrix, and record accurately two-layer exact information—degree of a node and its neighbors' degrees in a matrix—in a clever way, for later use of the Hungarian algorithm, which will be explained in Section 3.1.3. Since we are aligning unattributed graphs, a method based on information about links between labeled nodes is not rigorous, because the assignment of node IDs could be random on the pair of graphs to be aligned. Although the Hungarian algorithm is widely known and has been integrated into other graph alignment methods [5], our approach merits further attention and research due to its exceptional applicability to real heterogeneous networks—rather than restricted ideal graphs like

k-partite graphs or complete graphs—with very simple implementation steps.

This method is somewhat intuitive, since we are attempting to describe a local neighborhood using degrees of neighbors. On top of intuition, we also attempted to provide theoretical explanations that could support and provide insight into this method (and its variations). The few things we try to highlight in the theoretical sections are: a simplified model of analyzing alignment behavior that shows high-degree nodes are more likely aligned with high-degree nodes in heterogeneous networks; signature vectors for nodes in our proposed degree matrix can somewhat approximate a description of local graph Ricci curvature; a derived Curvature-Laplacian equation kindles another type of graph alignment method based on curvatures, namely Ricci Matrix Comparison.

Chapter 2

Problem Set Up: Graph Alignment

In this section, we introduce the *Graph Alignment* problem, addressing possible vagueness in the understanding of the problem as meticulously as possible.

Definition 2.1 (Graph)**.** A *graph* is a pair $G = (V, E)$, where elements $v \in V$ are called vertices (or nodes), and E is a set consisting of unordered (for our purposes only) pairs $\{v_1, v_2\}$ for some but not necessarily all of $v_1, v_2 \in V$.

Geometrically, G could represent an object with vertices and edges that connect some of the vertices. The presence of $\{v_1, v_2\} \in E$ indicates there is an edge between the vertices v_1, v_2. We talk about graphs in a manner that assumes a geometric interpretation. But there are other ways to interpret what we call a "graph"; a most direct example is the adjacency matrix, which is a binary matrix representation of graphs.

Definition 2.2 (Graph Alignment)**.** If we have $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ for graph alignment, we try to find the optimal one-to-one mapping $f : V_1 \to V_2$ to maximize the number of $v_1 \in C \subseteq V_1$ such that $v_1 = v_2 = f(v_1) \in V_2$. This process is called *Graph Alignment*.

To perform graph alignment, we need to generate a pair of graphs with common

structure for comparison. We introduce two graph sampling methods that help us first sample size-wise manageable smaller graphs from larger complex graphs, then further sample subgraphs from a smaller graph that retain common structure (but not isomorphic most of the times).

Section 2.1

Random Walk Graph Sampling

One of our methods utilizes random walk on graphs to extract subgraphs [13, 21].

Definition 2.3 (Random Walk (on graphs))**.** Suppose we start random walk at $v \in V$. We first find the set of neighbors of v, which are vertices connected to v by at least one edge, that we denote as S_{nei}. Then we randomly pick $v_n \in S_{nei}$, and move from v to v_n. Then we do the same thing for v_n, and all following nodes iteratively, until called to stop. This process is called random walk.

We take subgraph $G_s = (V_s, E_s)$, with $n = |V_s|$ distinct nodes, of a real world network $G_r = (V_r, E_r)$ through random walk. This is achieved through the following procedures: we pick random $v_0 \in V_r$, and initiate random walk on G_r until we either get one connected component (a graph that cannot be represented as the disjoint union of two graphs) with n distinct nodes, or exhaust a connected component in G_r with insufficient nodes, whichever comes first. If the latter is true, we randomly pick a new (different from all previous nodes that we walked on) node $v_n \in V_r$ and continue random walk until we obtain a sampled graph with the desired n distinct nodes and as few connected components as possible. V_s is the set that contains the n distinct nodes from the random walk process just described. $E_s \subseteq E_r$ inherits edges between sampled nodes in V_s from G_r.

To construct $G_1 = (V_1, E_1) \subseteq G_s$ and $G_2 = (V_2, E_2) \subseteq G_s$ for graph alignment, we perform random walk on G_s to first obtain a set of nodes that serves as the common

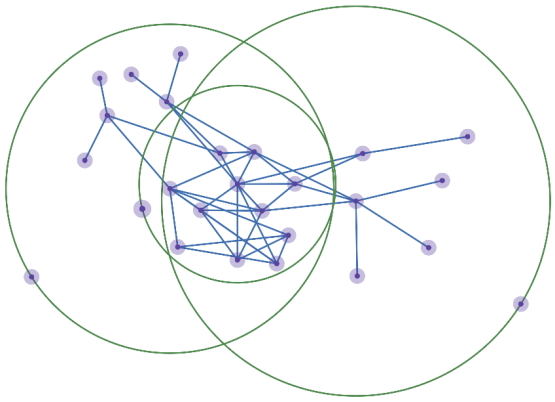


Figure 2.1: Graph Sampling with Random Walk

set of nodes $C = V_1 \bigcap V_2 \subset G_s$. The number of vertices we want to stop at, for the random walk, is $|C| = np$, where n is like before the number of distinct nodes in G_s and p is what we call the "overlap percentage" (with respect to G_s). We then take arbitrary $H_1, H_2 \subseteq (V_s \setminus C)$ such that $|H_1| = |H_2|$ and $V_s \setminus C = H_1 \sqcup H_2$ (disjoint union). Let $V_1 = C \sqcup H_1$ and $V_2 = C \sqcup H_2$, then $|V_1| = |V_2|$. $E_1 \subseteq E_s$ is the inherited set of edges between nodes in V_1 from E_s, likewise for E_2.

Figure 2.1 is a demonstration for what we are trying to take off the entire graph. There are three green circles containing different regions of the graph; the two larger ones contain the same number of nodes, and the smallest circle contains the common region between the two possible sampled graphs through random walk. This is not the only sampling possible.

Section 2.2

Edge Deletion Graph Sampling

The second graph sampling method is from [14]. Suppose G_s is the same graph as in Section 2.1, then let $G_1 = G_s = (V_1, E_1)$, and we delete edges of G_1 with probability p_d to obtain another graph, which is G_2. If $G_2 = (V_2, E_2)$, then $V_2 = V_1$, and $E_2 \subseteq E_1$.

Figure 2.2 is a simple demonstration of the edge deletion graph sampling method,

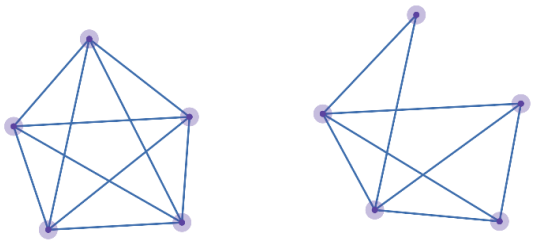


Figure 2.2: Graph Sampling with Edge Deletion

where two edges from the left graph were randomly deleted to obtain the right graph.

Chapter 3

Proposed Alignment Methods

In this chapter, we introduce several graph alignment methods, each suited for a specific purpose. Section 3.1 introduces the core method, Degree Matrix Comparison, which is the basis of deriving other variations in other sections of Chapter 3.

Section 3.1

Degree Matrix Comparison

3.1.1. Overview of Method (DMC)

Our main proposed method requires creating two *degree matrices* and comparing them. We denote $\deg(v)$ as the degree of a node v, where degree is simply the number of edges connected to a vertex, or more formally, the number of unordered pairs in the edge set E that contains the vertex.

Definition 3.1 (Degree Matrices)**.** Consider a pair of graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. By our problem set up in Chapter 2, $N = |V_1| = |V_2|$. Let $V_1 = \{v_i\}_{i=1}^N$ and $V_2 = \{w_j\}_{j=1}^N$. Let $m_1 = \max(\{\deg(v_i)\}_{i=1}^N)$ and $m_2 = \max(\{\deg(w_j)\}_{j=1}^N)$, and $m = \max(m_1, m_2)$. Then the *degree matrix* M_1 for G_1 is a matrix of dimension (N, m), where each row left-aligns, for a unique $v_1 \in V_1$, its neighbors' degrees in

ascending order, with zero-padding to the right. We define M_2 similarly.

The order of rows in M_1 and M_2 is arbitrary. After obtaining M_1 and M_2, we use the Hungarian algorithm (see Section 3.1.3) to minimize cost between rows of the matrices, producing a map between rows of M_1 and M_2. We migrate this assignment directly to the assignment $f : V_1 \to V_2$ between nodes. We call this entire process the *Degree Matrix Comparison (DMC)*.

3.1.2. Examples for Degree Matrix Comparison

We provide examples here to demonstrate the concept of a degree matrix and the DMC process. For the degree matrix formulation, let us examine the graph (assumed to be a local picture taken from a much larger graph) in Figure 3.1 with circled node as the origin. Then the boxed nodes are the first neighbors of the origin. Starting from the first neighbor in the upper left corner and going counter clock-wise, the degrees of the five first neighbors are $3, 4, 3, 5, 3$. Suppose (arbitrarily) m in this case is 10, then the row vector representation for the circled node would be $[3, 3, 3, 4, 5, 0, 0, 0, 0, 0]$ in the degree matrix for the larger graph. We do this for every node in the larger graph which this smaller piece is taken from to build a degree matrix.

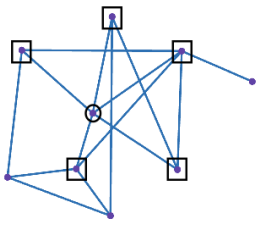


Figure 3.1: Example Graph to Demonstrate Degree Matrix Formulation

Now we demonstrate the DMC process. Consider the two graphs in Figures 3.2 and 3.3 as a pair that we are to align.

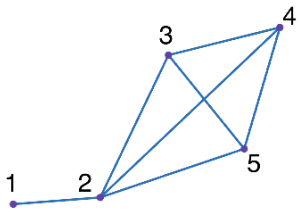


Figure 3.2: F_1 to demonstrate DMC

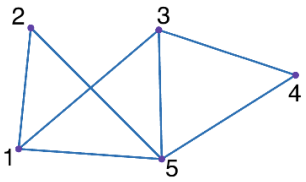


Figure 3.3: F_2 to demonstrate DMC

Let the graph in Figure 3.2 be $F_1 = (V_{F_1}, E_{F_1})$ and the graph in Figure 3.3 be $F_2 = (V_{F_2}, E_{F_2})$. The node set V_l of the largest overlapping subgraph consists of all five nodes, or: $V_l = V_{F_1} = V_{F_2}$. However, the graphs are not isomorphic, so the edge set E_l of the overlapping subgraph is a strict subset of E_{F_1}, E_{F_2}, which in mathematical notation is: $E_l \subsetneq E_{F_1}, E_{F_2}$. Let the edge between node 3 and node 4 in F_2 be w. Then the overlapping subgraph takes the shape of $F_2 \backslash \{w\}$. The degree matrices for F_1 and F_2 are

$$\begin{pmatrix} 4 & 0 & 0 & 0 \\ 1 & 3 & 3 & 3 \\ 3 & 3 & 4 & 0 \\ 3 & 3 & 4 & 0 \\ 3 & 3 & 4 & 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 2 & 3 & 4 & 0 \\ 3 & 4 & 0 & 0 \\ 2 & 3 & 4 & 0 \\ 3 & 4 & 0 & 0 \\ 2 & 2 & 3 & 3 \end{pmatrix},$$

Then we use the Hungarian algorithm (see Section 3.1.3) to minimize the cost of row alignment between the two matrices above. The algorithm aligns nodes $1, 2, 3, 4, 5$

of F_1 to nodes $2, 5, 1, 3, 4$ of F_2 (one could check understanding of the concepts with this result). This is not an optimal result, but it is expected since the graphs are not heterogeneous (the relationship between performance of DMC and heterogeneity of graph will be discussed in depth in Chapters 4 and 5). This example is purely for demonstrating DMC.

In practice (implementing in code), the Hungarian algorithm (see Section 3.1.3) can be replaced by a computationally more efficient `linear_sum_assignment` function from `scipy.optimize` in Python which yields the same assignment [7]. However, in terms of comprehension, the Hungarian algorithm is more straightforward. The complexity of DMC is at most $O(N^3)$ for a pair of graphs that each has N nodes [9, 36]. When $m << N$, which is typical for large sparse graphs, the complexity is $O(N^2 \cdot m) << O(N^3)$.

3.1.3. Hungarian Algorithm

In this section, we introduce the Hungarian algorithm, which was named after two Hungarian mathematicians Dénes König and Jenö Egerváry. It is essentially an optimization algorithm that minimizes the cost of an assignment between two sets of objects. In our case, we minimize the cost of assigning rows between two degree matrices. One would first calculate a cost matrix between two degree matrices. Since the graph sampling methods in Chapter 2 produce graph pairs with exactly the same number of nodes, say n, we would have a cost matrix C of dimension $n \times n$ with each entry c_{ij} being the cost (Euclidean distance) between the i^{th} row in the first degree matrix and the j^{th} row in the second degree matrix. With the cost matrix in hand, we perform steps following Algorithm 1.

Algorithm 1 Hungarian Algorithm

Input: A cost matrix C of size $n \times n$.
Step 1: Subtract the row minimum.
for each row i in C **do**
  Subtract the minimum value of row i from all elements in row i.
end for
Step 2: Subtract the column minimum.
for each column j in C **do**
  Subtract the minimum value of column j from all elements in column j.
end for
Step 3: Cover all zeros with a minimum number of lines.
while not all zeros are covered **do**
  Identify rows and columns containing uncovered zeros.
  Cover all zeros using the minimum number of horizontal and vertical lines.
end while
Step 4: Check the number of lines.
if the number of lines is equal to n **then**
  An optimal assignment is possible.
  Go to Step 6.
else
  Go to Step 5.
end if
Step 5: Adjust the matrix.
Find the smallest uncovered value.
Subtract this value from all uncovered elements.
Add this value to elements at intersections of covering lines.
Return to Step 3.
Step 6: Make the assignments.
Select zeros such that no two are in the same row or column.
Return the assignment and compute the total cost.
Output: An optimal assignment and the total cost.

Example. We provide a simple example to demonstrate the Hungarian algorithm.

Suppose we have a cost matrix

$$\begin{pmatrix} 1 & 2 \\ 3 & 5 \end{pmatrix}$$

which is of size 2×2. Performing Step 1 in Algorithm 1, we get

$$\begin{pmatrix} 0 & 1 \\ 0 & 2 \end{pmatrix}$$

which leads us to Step 2, through which we obtain

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

which leads us to Step 3. A good way to cover zeros is through drawing a horizontal line in row 1 and a vertical line in column 1. In Step 4, we find that the number of lines is 2, equal here to n since the cost matrix is 2×2. Therefore, an optimal assignment is possible, and we move on to Step 6. In row 2, the only zero is the entry c_{21}, which leaves us to choose c_{12} as the other zero for our assignment. This indicates that our eventual total cost is minimized to $2+3=5$, which is indeed the lowest cost possible since the other assignment leads to a total cost of $1+5=6>5$.

Section 3.2

Greedy DMC

3.2.1. Overview of Method (Greedy)

The complexity of DMC can be reduced through replacing the Hungarian algorithm with a combination of a greedy heuristic and the Hungarian algorithm (see Algorithm 2), but accuracy is compromised. The greedy heuristic, with $O(N^2) \sim O(N^3)$ complexity and assignment threshold ϵ, serves as a preparation step to reduce reliance on the Hungarian algorithm. We call this reduced version the *Greedy DMC*.

We bring to attention the case where $\epsilon = 0$. While this is a special case of the

Greedy DMC algorithms with different ϵ thresholds, it is not "Greedy" in any sense. In fact, this is an algorithm that first finds *exact* matches between rows in M_1 and M_2, then performs the Hungarian algorithm on remaining rows. As can be seen in Chapter 5, Greedy DMC's $\epsilon = 0, 10$ cases are very effective algorithms like DMC, and are worth studying in the future.

3.2.2. Hungarian Algorithm Prepared by Greedy Heuristic

The greedy heuristic is another optimization algorithm. Different from the Hungarian algorithm which is a global optimization algorithm, it considers only local cost minimization. Again, since we are aligning two graphs with the same number of nodes, our cost matrix C resulting from them is a square matrix of dimension $n \times n$.

Algorithm 2 Hungarian Algorithm Prepared by Greedy Heuristic

Input: Cost matrix C of size $n \times n$, threshold $\epsilon > 0$.
Initialize $nodes[i] \leftarrow$ unassigned, $\forall i \in \{1, \ldots, n\}$.
Initialize $unused[j]$, $\forall j \in \{1, \ldots, n\}$.
while any node i is unassigned **do**
  for each unassigned node i **do**
    Identify the first entry j such that $unused[j]$ still exists and $c_{ij} < \epsilon$.
    if an entry j was found **then**
      Assign node j on second graph to node i on first graph.
      Remove $unused[j]$.
    else
      Leave $node[i]$ unassigned.
    end if
  end for
end while
if any tasks remain unassigned **then**
  Assign remaining tasks using the Hungarian algorithm (see Algorithm 1).
end if
Return: Final assignment of agents to tasks.

Example. Let us use the following cost matrix:

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{3} \\ 3 & 5 \end{pmatrix},$$

and let the tolerance be $\epsilon = 1$. We first examine row 1, and notice that when $j = 1$, we have $c_{11} = \frac{1}{2} < 1 = \epsilon$, therefore we directly align the 1^{st} row of the first degree matrix to the 1^{st} row of the second degree matrix. Then we are left with aligning another row vector to row $i = 2$. Notice that $j = 1$ is taken, so we can only pick the alignment $i = 2$ to $j = 2$. If there were more rows, we would continue finding the first-possible-alignment for all rows before performing the Hungarian algorithm on "leftovers" (when there are no entries in a row in the cost matrix that is within the tolerance level, we leave row i unassigned as a "leftover"). In this case, the alignment cost is not minimized as $\frac{1}{2} + 5 > \frac{1}{3} + 3$, but computational cost is lower than iterating through all possible cases.

Section 3.3

Weighted DMC

While the previous methods are meant for unweighted plain graphs, the Weighted DMC is designed for weighted graphs (with weights on edges instead of nodes). These graphs can no longer be fully represented by just an adjacency matrix, which counts number of edges but makes no note of varying weights. Therefore, we propose a variation of the DMC that incorporates this extra piece of information, keeping in mind that we do not want the degree matrix size to change substantially (to save computational power). But first we introduce the idea of a Weighted Degree Matrix.

Definition 3.2 (Weighted Degree Matrix)**.** Let M be a degree matrix for a graph G

(as defined in Definition 3.1). Then the *Weighted Degree Matrix* for the same graph G is formed by multiplying each entry $m_{ij} \in M$ by the weight of the edge between the node corresponding to row i and the node corresponding to row j on a second graph, and then rearranging the resulting entries within each row.

The Weighted DMC is the process of performing the degree matrix comparison, except we replace previous degree matrices with weighted degree matrices. Weighted DMC performed well on multiple datasets (see Chapter 5), suggesting it deserves further research—much like Greedy DMC. In particular, its high performance on protein-protein interaction (PPI) networks (often used to test graph alignment) indicates that DMC's success is not coincidental.

3.3.1. Example for Weighted Degree Matrix

We provide a quick example for constructing a weighted degree matrix. We examine the weighted graph provided in Figure 3.4. We write out an initial matrix containing

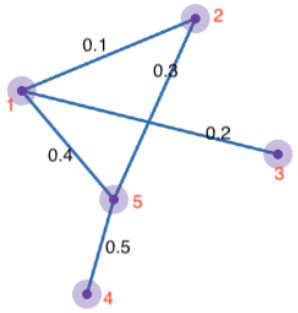


Figure 3.4: Graph for Example Weighted Degree Matrix

degrees of neighbors without rearrangement. Let row i correspond to node i as labeled in red, and we record degrees of neighbors by ascending order of node labels (initial

node labeling is arbitrary). Then we obtain (assuming largest possible degree is 4)

$$\begin{pmatrix} 2 & 1 & 3 & 0 \\ 3 & 3 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}.$$

As usual, we record non-zero entries first, and then fill in missing entries with zeros. For example, row 1 is for node 1, and since the nodes connecting to node 1 are nodes 2, 3, and 5, we record the degrees of these three nodes by their label order: 2, 1, 3. There is one empty position left in the first row, so we fill it in with 0. The other rows are constructed with the same logic.

Section 3.4

Ricci Matrix Comparison

Recall that in Section 3.1, we introduced the Degree Matrix Comparison (DMC) method for graph alignment. The corresponding degree matrices record for each node in a row in the matrices the neighbors' degrees. As will be motivated in Sections 4.5 and 4.6, another possible alternative or variation to the DMC is using the discrete graph Ricci curvature at each node as signature values. Instead of having degrees of a node's neighbors in each row, we replace these values with the Forman-Ricci curvatures of these same neighbors. We reorder curvatures within each row to ascending order and pad zeros to the right again. Then we apply the Hungarian algorithm. Directly, we call this variation Ricci Matrix Comparison (RMC). A formal definition of the Forman-Ricci Curvature is provided in Definition 4.4.

We provide an explicit example for constructing a row vector in a Ricci matrix.

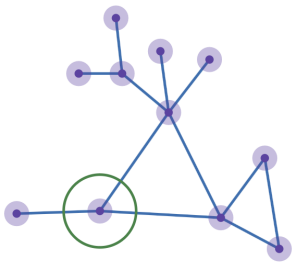


Figure 3.5: Example graph for demonstrating Ricci row vector construction.

Suppose we try to look for the row vector in a Ricci matrix (analogue of degree matrix) for the circled node in Figure 3.5. We first observe that the neighbors of this origin O have degrees of 1, 4, and 5. Let us call these nodes A, B, and C. We try to calculate the Forman-Ricci curvature at each of these neighbors (the formula is simple for unweighted graphs). For node A, the curvature is $2 - \deg(A) - \deg(O) = -2$ and the summation sign is dropped since it only has degree one. For nodes B and C, we calculate their curvatures in the same way, summing up the curvatures on all adjacent edges. For node B, its adjacent edges have curvatures -5, -4, -4, and -7. Therefore, the Forman-Ricci curvature of node B is their sum, -20. Following a same procedure, we get that the node curvature for C is -27. So in a row vector, we have [-27, -20, -2, 0, 0, ..., 0]. The number of zeros we pad is dependent on the magnitude of m, the maximum degree on both graphs. The dimensions of the matrices remain the same as degree matrices since the number of nonzero entries of a row is still reflective of the degrees of the nodes. Note here that we will not encounter a complex connected graph with a node with curvature 0, since each node has at least a degree of 1.

Chapter 4

Theoretical Examination

Section 4.1

Motivations for Constructing DMC

One theoretical motivation for the method is that degree matrices remain *invariant* in response to arbitrary node labeling. By invariant, we mean that degree matrices are in the same equivalence class defined by the following equivalence relation.

Theorem 4.1 (Equivalence Relation on Degree Matrices.)**.** *We define a binary relation between two degree matrices: degree matrices A, B are said to be related (i.e. $A \sim B$) if we can transform A to become B only through row swap operations. Moreover, we claim that this is an equivalence relation.*

Proof. A binary relation is an equivalence relation if and only if the relation is reflexive, symmetric, and transitive. For any matrix A, $A \sim A$ is true since we can obtain A from performing no row swap operations on A, hence the relation is reflexive.

$A \sim B$ implies there is a series of row swaps that can transform A to B. Therefore, we can also obtain A from B through reversing the row swap process and so $B \sim A$. Hence, the relation is symmetric.

$A \sim B$ and $B \sim C$ imply we can transform A into B through row swaps, and also transform B into C through row swaps. This implies we can obtain C from A through first performing row swaps that transform A into B and then applying more row swaps that transform B into C. Hence we can obtain C from A through row swaps and $A \sim C$, making the binary relation transitive. □

The invariant property is crucial for graph alignment for unattributed graphs (or on attributed graphs but not using attributes) because this means we are only leveraging geometric properties that are inherent of graphs and not relying on other information affiliated to them.

Apart from the invariant property, we also point out that degree matrices are efficient in storing information. A degree matrix ($N \times m$) encodes information more compactly than an adjacency matrix ($N \times N$), since it not only contains degrees of neighbors for each node, but also implicitly records degrees of each node through the number of non-zero entries in each row, unlike the adjacency matrix which only records the degree distribution (one would need to incorporate node labeling to get an exact picture of the graph).

Additionally, combining local (degrees) and global (assignment) perspectives is important because the of the Friendship Paradox. In general terms, it is the observation that, on average, friends of a person typically have more friends than that person. The Friendship Paradox shows how local observations can be completely shifted when we examine the same neighborhood globally [2, 20].

To show that this phenomena is real, we provide an explicit example in Figure 4.1, which was taken from "The Network Pages" [27]. We observe that this graph is one edge away from being a complete graph (an undirected graph where all pairs of nodes have an edge in between), so it is complex. In other words, all "are friends" with each other except between two of all nodes. If we examine any one of the "smiley faces"

which is connected to all other nodes except for itself locally, we would come up with the conclusion that it has more friends than its friends' average. Moreover, this is true for most of the nodes ("smiley faces") on the graph. Therefore, if we examine all nodes locally, we would conclude that "most of the nodes are more popular than its friends."

While this is true, we could come up with a different conclusion if we take a global perspective. The article in [27] introduced the concept of "Friendship Bias" as the value of subtracting the degree of a node from the average degree of its neighbors. When there is positive bias, it somewhat indicates a node is "less popular" (less connected to other nodes compared to its neighbors), and vice versa for negative bias. For the graph in Figure 4.1, we have that the bias for any of the "smiley faces" is $\frac{10\times 8+9\times 2}{10} - 10 = -0.2$, since there are 8 nodes of degree 10 among neighbors of a "smiley face", and 2 nodes of degree 9 among the same set of neighbors. Similarly, we can calculate for any of the "sad faces" that the bias is $\frac{10\times 9}{9} - 9 = 1$. If we pursue a global perspective, it makes sense to take the mean of the biases of all nodes, which leads us to an average bias of $\frac{1\times 2 - 0.2\times 9}{11} = \frac{0.2}{11} > 0$. Globally, we arrive at the conclusion that, *on average*, the average degree of neighbors is larger than degrees of nodes (i.e. your friends have more friends than you do on average).

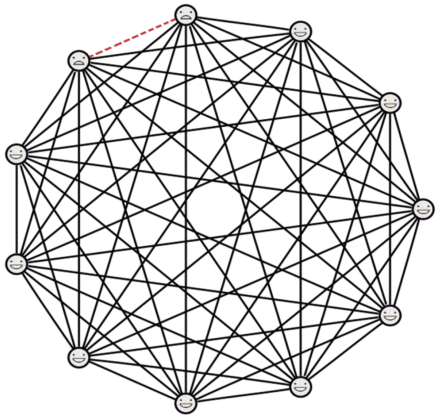


Figure 4.1: Example Graph to Demonstrate Friendship Paradox

Heterogeneous graphs tend to be more easily distinguishable, since they have more variations across the entire graph. By heterogeneous, we do not necessarily mean complex: a complete graph with many edges is complex, but not heterogeneous, since the degrees follow a uniform distribution. Since our method is simple, it intuitively makes sense that it suits well for heterogeneous graphs that are inherently rich in information.

Suppose there is a graph that has nodes with degrees that are almost all unique. Note here that the phrase "almost all unique" is used to carefully address the possible scenario as follows: if we allow at most one edge between every pair of vertices on a graph $G_{example}$ with $n_{example}$ vertices and require at least degree one for all nodes, then the maximum degree is at most $n_{example} - 1$ for every node, and in this case it is impossible to create a bijection between possible degrees and the nodes. With a heterogeneous graph with most degrees different, most nodes have row vectors that are relatively unique and distinguishable.

Section 4.2

Spectrum of Graph Models

In this section, we introduce a range of graph models, on which we apply DMC for comparison in Section 4.3.

4.2.1. Erdős-Rényi Graphs

We will start from the most basic Erdős-Rényi random graph generation model. One common way to define this is to fix the total number of vertices, say n_{er}, then attach edges between pairs of vertices with some probability p_{er}. The resulting graph is typically denoted as $G(n_{er}, p_{er})$ [8]. One interesting property of these graphs is that their degree distributions are binomial, as formulated in Equation 4.1. Each edge

is present IID (independent and identically distributed) with probability p, so the number of edges connected to a selected node is a binomial distribution of $(n-1)$ trials of probability p:

$$\mathbf{P}(\deg(v) = k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}. \tag{4.1}$$

The probability $\mathbf{P}$ considers the possibility of having degree of k (being connected to k edges) when the largest possible degree is $n-1$, given the total number of nodes is n. p in (4.1) is the same p_{er} for generating the random graph.

4.2.2. Barabási-Albert Graphs

In our range of models, Erdős-Rényi is one of the two extremes (completely random attachment of edges). The other "extreme" graph generation model that we use is the Barabási-Albert model. It considers the preferential attachment phenomenon which is prevalent in real world networks [3]. Instead of attaching new edges with uniform probability, it is possible to attach new edges with weighted probabilities. Suppose we have m_0 nodes with some connections initially, then for the $(m_0+1)^{\text{th}}$ node, the probability that it will be attached to a previous node v_i is

$$\mathbf{P}(v_i) = \frac{\deg(v_i)}{\Sigma_{i=1}^{m_0} \deg(v_i)} \tag{4.2}$$

We can easily check that the probabilities of attaching the new node to any previous node sums up to one; moreover, higher degree nodes attract more new nodes. This usually leads to what we call *hubs*, which are local centers that are attached to many low degree nodes. It is common for hubs to exist in biological and social networks, and hubs add to the heterogeneity of graphs. However, having hubs with too much concentration could lead to confusion when differentiating low-degree nodes.

Therefore, it is always good to strike a balance between concentrating and diffusing nodes, but how this is achieved in the real world is yet unknown.

Another consequence of the preferential attachment is a near *power law* degree distribution. We follow a proof in [1]. Suppose we observe the discrete process of adding edges from a continuous perspective for any node v_i starting from when the node first appears in the graph. Moreover, assume we connect m edges between every fresh node and previous nodes (therefore from a continuous perspective edges are added at constant rate) and suppose the sum of degrees of all nodes at time t is N_t. Then, using (4.2), we have the following differential equation to represent the change of edges attached to a randomly picked node if $k_i(t)$ is a function that records the degree of node v_i with respect to time:

$$\frac{dk_i(t)}{dt} = m\frac{k_i(t)}{N_t}. \tag{4.3}$$

Since we start with a small number of nodes and edges but grow a large graph, N_t can be approximated as $2mt$ (initial part will be negligible in size). Then we arrive at the equation

$$\frac{dk_i(t)}{dt} = \frac{k_i(t)}{2t}. \tag{4.4}$$

We endow this differential equation with an initial condition $k_i(t_i) = m$ if t_i is the time when v_i was just added to the graph. Our time frame for observing (4.4) is $[t_i, +\infty)$. Then we solve this equation as usual. Through

$$\int \frac{1}{k_i} dk_i = \frac{1}{2}\int \frac{1}{t} dt, \tag{4.5}$$

we obtain

$$\ln(k_i) = \frac{1}{2}\ln(t) + C, \tag{4.6}$$

and plugging in the initial condition we get

$$C = \ln(m) - \frac{1}{2}\ln(t_i) \tag{4.7}$$

and hence

$$\ln(k_i) = \frac{1}{2}(\ln(t) - \ln(t_i)) + \ln(m). \tag{4.8}$$

Then

$$k_i(t) = (\frac{t}{t_i})^{\frac{1}{2}} \cdot m. \tag{4.9}$$

We now find an expression for the probability

$$\mathrm{P}(k_i(t) < k)$$

where k is an arbitrary positive degree. Then it follows that

$$\mathrm{P}(k_i(t) < k) = \mathrm{P}((\frac{t}{t_i})^{\frac{1}{2}} \cdot m < k) = \mathrm{P}(t_i > \frac{m^2 t}{k^2}), \tag{4.10}$$

which we can rewrite as

$$\mathrm{P}(k_i(t) < k) = 1 - \mathrm{P}(t_i \leq \frac{m^2 t}{k^2}). \tag{4.11}$$

Since the rate of growth of total number of edges is constant, we can assume that the growth rate is 1. Moreover, we start with m_0 nodes before adding nodes with preference. Time follows a uniform distribution, hence

$$\mathrm{P}(t_i \leq \frac{m^2 t}{k^2}) = \frac{1}{m_0 + t} \cdot \frac{m^2 t}{k^2}. \tag{4.12}$$

If we take the partial derivative of (4.11) with respect to k after plugging in (4.12),

we get that the probability density function (pdf) $f(k)$ satisfies

$$f(k) \propto k^{-3}, \tag{4.13}$$

which shows that the degrees follow a power law distribution.

4.2.3. Poisson Distribution: Bridging Erdős-Rényi and Barabási-Albert

We then move along the spectrum of Poisson distributions (see Figure 4.2) with different parameter values, which can approximate distributions from binomial (black) to power law (yellow). A formal representation of the Poisson distribution for each node v is as follows:

$$\mathrm{P}(\deg(v) = k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{4.14}$$

where λ is a constant parameter.

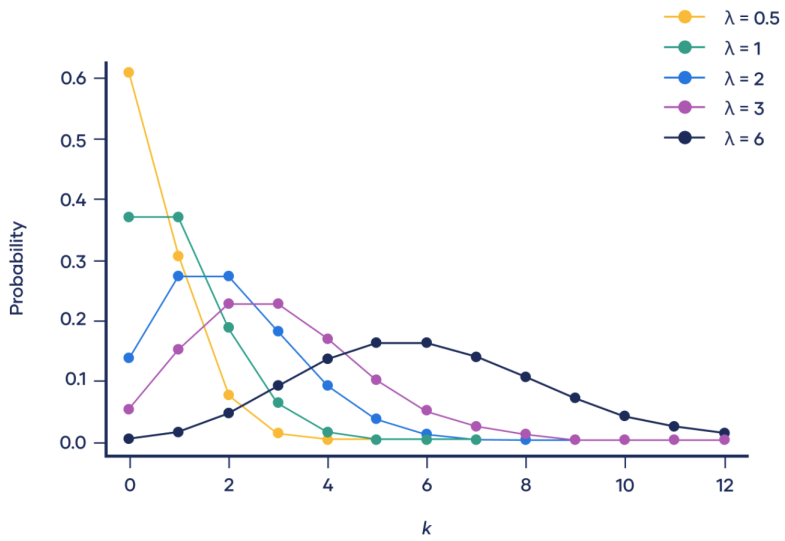


Figure 4.2: Evolution of Poisson distribution with decreasing λ. Taken from Scribbr.

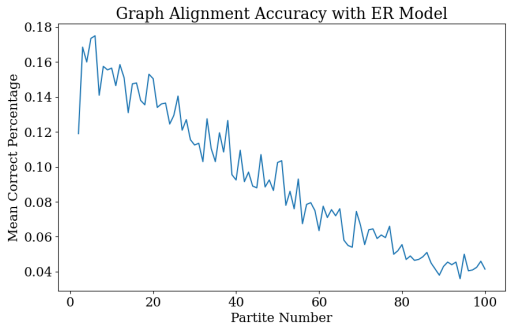


Figure 4.3: Performance of DMC applied to Erdős-Rényi generated graphs with respect to different partite numbers.

Section 4.3

DMC Experiments on Synthetic Graphs

4.3.1. The Extreme Cases

Definition 4.2 (k-partite Graph)**.** A *k-partite Graph* is a graph with nodes that can be split into k independent sets, where nodes within each set are never connected.

We applied DMC to graphs generated through the Erdős-Rényi, Barabási-Albert, and Chung-Lu (to generate graph distributions that are Poisson) models. We examine the performance of DMC on the graphs with respect to different partite numbers k. For our Erdős-Rényi experiments, we fix total number of nodes to be $n_{er} = 100$, then split the nodes into k partite groups; the partition of nodes is arbitrary. After that, we connect edges between different partite groups with probability of $p_{er} = 0.5$. Then we use the random walk graph sampling method to obtain two graphs with 90% overlap, and perform DMC. We run each experiment for 20 times, then take the mean performance. In Figure 4.3, we note that there seems to be a negative linear relationship between the performance of DMC with respect to partite numbers, which is interesting despite the overall low performance. This decay as the graph becomes complete (n-partite when there are n nodes) is expected as the degree distribution

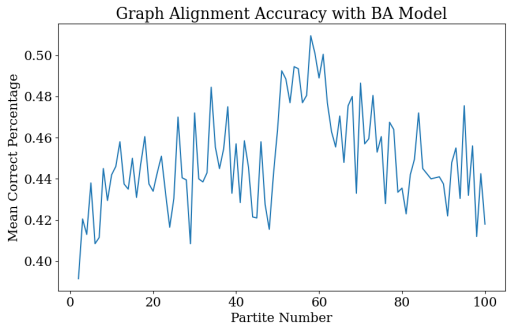


Figure 4.4: Performance of DMC applied to Barabási-Albert generated graphs with respect to different partite numbers.

becomes uniform.

For our Barabási-Albert experiments, we do a similar thing. To create "k-partite Barabási-Albert graphs", we start each experiment by partitioning nodes into k groups. Every time we add a new node, we connect it to five different previous nodes (add 5 edges) that are not in the same partition group. If there are less than five nodes available, we simply attach the maximum number of edges possible. We also run each experiment for 20 times in this case. Examining Figure 4.4, we see that applying DMC to Barabási-Albert modeled graphs yields relatively uniform and better performance compared to Erdős-Rényi. It is worth pointing out performance peak in the "middle interval" of partite numbers. As will be seen in following discussions, this is a recurrent feature that points to a significant component of heterogeneity: moderate balance between expanding hubs and disconnecting nodes.

4.3.2. The Intermediate Cases

For our Chung-Lu experiments, we use Poisson distributions with different λ parameter values as our initial degree distributions (see Figures 4.5, 4.6, 4.7, and 4.8), and we set a maximum degree limit of 20. In this case, we also start with a k-partition before proceeding to the Chung-Lu steps using a prior Poisson distribution. We also run each experiment for 20 times in this case.

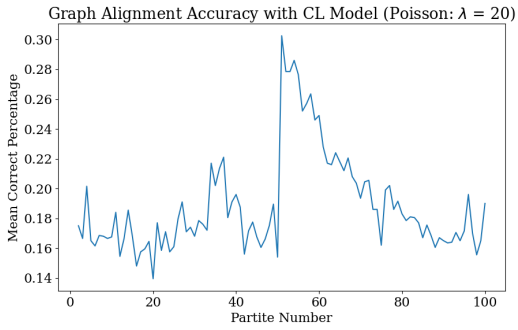


Figure 4.5: Performance of DMC applied to Chung-Lu generated graphs with respect to different partite numbers when parameter $\lambda = 20$.

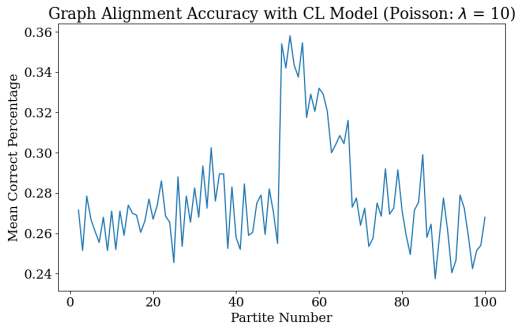


Figure 4.6: Performance of DMC applied to Chung-Lu generated graphs with respect to different partite numbers when parameter $\lambda = 10$.

We present a spectrum of examples with $\lambda = 20, 10, 5, 1$. Since a large λ value causes a Poisson distribution to become close to binomial (assuming a large enough number of nodes), we observe a somewhat similar negative linear relationship to that in Figure 4.3 in the latter half of the plot in Figure 4.5. Moreover, the performance peaks near the median of partite numbers, as hinted in the previous section.

Through comparing Figures 4.5, 4.6, 4.7, and 4.8, we see that as λ decreases, the plot looks more and more uniform and closer to Figure 4.4. Additionally, as we move from the "Barabási-Albert end" (i.e. $\lambda = 1$) to the "Erdős-Rényi end" (i.e. $\lambda = 20$), the mean performance over all partite numbers decreases (see Figure 4.9) as one would expect since the distribution is moving from power law to binomial, losing hubs and along with it heterogeneity.

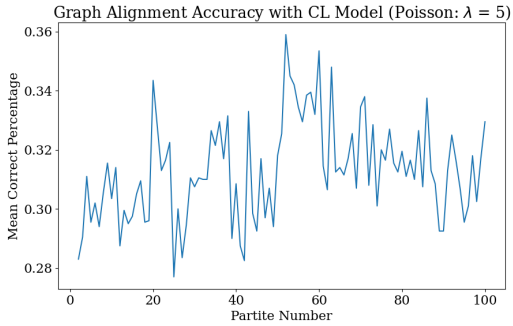


Figure 4.7: Performance of DMC applied to Chung-Lu generated graphs with respect to different partite numbers when parameter $\lambda = 5$.

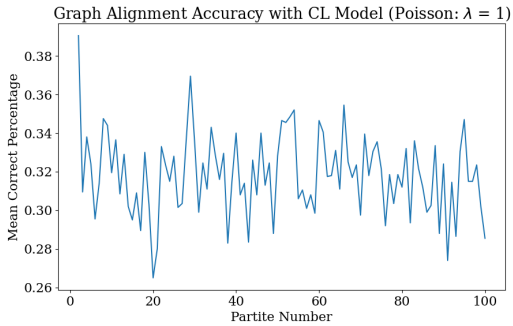


Figure 4.8: Performance of DMC applied to Chung-Lu generated graphs with respect to different partite numbers when parameter $\lambda = 1$.

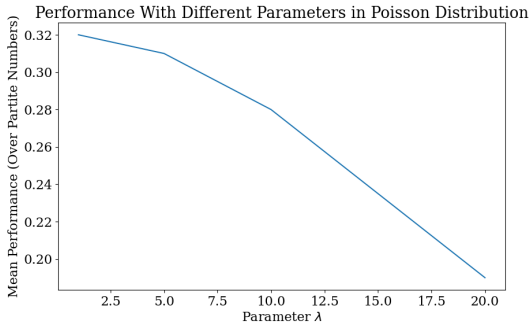


Figure 4.9: Mean performance of DMC applied to Chung-Lu generated graphs with respect to different parameter values.

Section 4.4

Performance Analysis for DMC

We provide a simplified analysis of performance of the DMC here, aiming to show that the DMC is at least effective in differentiating high- and low-degree nodes. In a Barabási-Albert graph, due to the formation of hubs, a significant amount of nodes are either a concentrated hub center or close to a leaf. Therefore, we can assume the row vector in a degree matrix for a high degree node (maybe a hub) looks something like

$$[1, 1, 1, 1, 1, 1, 1, 2, 3, 0, 0], \tag{4.15}$$

just as an example. The main properties of this row vector are that entries are small numbers (low degree nodes) and there are many non-zero entries, due to its own large degree. For a leaf, the row vector will look like

$$[100, 0, 0, 0, 0, 0, 0], \tag{4.16}$$

for instance. In these low-degree types of row vectors, we observe that most entries are zero, and the few non-zero entries tend to be large. To perform a cleaner analysis, let us directly examine the following row vectors:

$$\mathbf{v}_1 = [k, 0, 0, ..., 0] \tag{4.17}$$

for a leaf connected to a hub with $\deg(v) = k$, and

$$\mathbf{v}_2 = [1, 1, 1, ..., 1] \tag{4.18}$$

for the hub, assumed to be connecting to k leaves for simplicity. Both of these row vectors have a length of k, since extra 0's hanging at the end does not affect cost of alignment. Moreover, suppose there is another high-degree node row vector with the same length which takes the form

$$\mathbf{v}_3 = [1+\epsilon, 1+\epsilon, ..., 1+\epsilon]. \tag{4.19}$$

One might wonder whether the accumulation of small differences between each entry in the row vectors lead to a large enough difference that confuse the alignment of a high-degree node with another high-degree node versus with a low-degree node.

Lemma 4.3. *In a cost-minimizing Hungarian algorithm, the high-degree node vectors* $\mathbf{v}_2$ *and* $\mathbf{v}_3$ *will be aligned out of the three vectors in Equations 4.17, 4.18, and 4.19.*

Proof. We assume $k \to \infty$, and $\epsilon \to 0$. Then the cost of aligning $\mathbf{v}_1$ and $\mathbf{v}_2$ is $(k-1)^2 + (k-1)\cdot(0-1)^2 = k(k-1) \approx k^2$. This is the cost of aligning a low-degree row vector with a high-degree row vector. We also compute the cost of aligning $\mathbf{v}_2$ with $\mathbf{v}_3$, which is $k \cdot (1+\epsilon-1)^2 = k\epsilon^2 \approx 0$. Even if we relax our assumption so that $\epsilon \to 0 < C < \sqrt{k}$ rather than having $\epsilon \to 0$, the squared effect will make the alignment of high to high more reasonable. □

Section 4.5

Motivation for Constructing RMC

We introduce graph curvatures as discrete analogues of curvatures for smooth surfaces. The most common types of graph curvatures are the Forman-Ricci curvature and Ollivier-Ricci curvature. The Ollivier-Ricci curvature has been used in [30] for community detection in complex networks, which shows that curvature can be utilized as a tool for alignment. However, we choose to use the Forman-Ricci curvature for a

first attempt to construct a graph alignment algorithm that is both curvature-based and matrix-based because it is computationally cheap, even though it sacrifices some accuracy.

In this section, we record an equation that we discovered for unweighted graphs that relates the Forman-Ricci curvature [15, 19, 33, 39], graph Laplacian, and a slight variation of signature vectors in the degree matrix. We introduce these concepts one by one before deriving what we name the "Curvature-Laplacian" equation in Section 4.5.4.

4.5.1. Forman-Ricci Curvature

Definition 4.4 (Forman-Ricci curvature for graphs)**.** Forman-Ricci curvature is defined on edges. In other words, we assign a value to each of the edges on a graph which we call curvature (discretized Ricci curvature). For a weighted graph, the formula is

$$\mathbf{Ric}(e) = w_e \left(\frac{w_{v_1}}{w_e} + \frac{w_{v_2}}{w_e} - \sum_{e_l \sim v_1} \frac{w_{v_1}}{\sqrt{w_e w_{e_l}}} - \sum_{e_l \sim v_2} \frac{w_{v_2}}{\sqrt{w_e w_{e_l}}} \right) \tag{4.20}$$

where e is an edge between nodes v_1 and v_2, and $w_e, w_{v_1}, w_{v_2}, w_{e_l}$ are weights on edges and strengths on nodes [39]. The notation $e_l \sim v_i$ means the edge e_l is attached to the node v_i. It follows that for unweighted graphs, the curvature for each edge is

$$\mathbf{Ric}(e) = 2 - \deg(v_1) - \deg(v_2) \tag{4.21}$$

if we take all weights to be uniformly 1.

According to [39], we can define a node-based version of curvature using Definition 4.4 simply by taking the sum of all edge curvatures of edges connected to the node, or:

$$\mathbf{Ric}(v) = \sum_{e_l \sim v} \mathbf{Ric}(e_l). \tag{4.22}$$

Whether the curvature function is for a node or an edge is evident from the variable.

4.5.2. Graph Laplacian

Now we move on to introducing the graph Laplacian by deriving it as a discrete analog of the continuous Laplacian. We will first derive the discrete Laplace operator from the continuous Laplacian, then obtain the graph Laplacian from the discrete Laplace operator [6]. The idea that the graph Laplacian serves as a numerical approximation of the continuous Laplacian is well established [4, 24]. That being said, we offer a (hopefully) friendly derivation to highlight the deep and well-founded connection between the graph Laplacian and the continuous Laplacian. A continuous Laplacian on function $f : \mathbb{R}^n \to \mathbb{R}$, can be written as

$$\Delta f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2} \tag{4.23}$$

if x_i's are Cartesian coordinates. In order to move from the continuous end to a discrete graph version, we incorporate numerical approximations. Using the Finite Difference Method, we can define the first derivative of f with respect to $\mathbf{x} = (x_1, ..., x_n)$ as

$$f'(\mathbf{x}) = \lim_{\epsilon \to 0} \frac{f(\mathbf{x} + \epsilon) - f(\mathbf{x})}{\epsilon} \tag{4.24}$$

where ϵ is a small n-dimensional perturbation vector. With Equation 4.24, we can derive a Laplacian with the central approximation in a one-dimensional setting:

$$\Delta f = f''(\mathbf{x}) = \lim_{\epsilon \to 0} \frac{\frac{f(\mathbf{x}+\epsilon) - f(\mathbf{x})}{\epsilon} - \frac{f(\mathbf{x}) - f(\mathbf{x}-\epsilon)}{\epsilon}}{\epsilon} = \lim_{\epsilon \to 0} \frac{f(\mathbf{x} + \epsilon) + f(\mathbf{x} - \epsilon) - 2f(\mathbf{x})}{\epsilon^2}. \tag{4.25}$$

In the discrete case for graphs, we take $\epsilon = 1$ (assuming edge lengths are uniformly 1) and let $\mathbf{x}$ be a node. Since this is a one-dimensional setting, $\mathbf{x} + \epsilon = \mathbf{x} + 1$ is the node to the right of $\mathbf{x}$ and similar definition holds for $\mathbf{x} - 1$. This is illustrated in

Figure 4.10. Therefore, Equation 4.25 can be reformed as

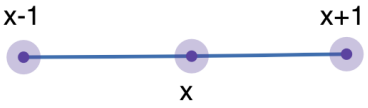


Figure 4.10: Illustration of one-dimensional graph.

$$-\Delta f = [f(\mathbf{x}) - f(\mathbf{x}+\epsilon)] + [f(\mathbf{x}) - f(\mathbf{x}-\epsilon)]. \tag{4.26}$$

Suppose we extend this to a two-dimensional setting. Then

$$\Delta f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}, \tag{4.27}$$

which is, by slight variation of the one-dimensional case in Equation 4.25,

$$\Delta f(x,y) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon,y) + f(x-\epsilon,y) - 2f(x,y)}{\epsilon^2} + \frac{f(x,y+\epsilon) + f(x,y-\epsilon) - 2f(x,y)}{\epsilon^2}, \tag{4.28}$$

which can be reduced to

$$\Delta f(x,y) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon,y) + f(x-\epsilon,y) + f(x,y+\epsilon) + f(x,y-\epsilon) - 4f(x,y)}{\epsilon^2}. \tag{4.29}$$

Using again the idea that $\epsilon = 1$ and (x, y) coordinates are nodes, we get that

$$-\Delta f(x,y) = \sum_{(x_l,y_l)\in V} f(x,y) - f(x_l,y_l) \tag{4.30}$$

where $V = \{(a,b)|(a,b) = (x,y)\pm(1,0) \text{ or } (x,y)\pm(0,1)\}$. The two-dimensional graph is illustrated in Figure 4.11.

Following the idea above for the one-dimensional and two-dimensional cases, we

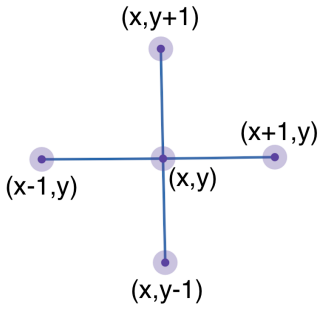


Figure 4.11: Illustration of two-dimensional graph.

can write

$$-\Delta f(v) = \sum_{v_i \sim v} f(v) - f(v_i) \tag{4.31}$$

in general where $v_i \sim v$ denotes the nodes incident to v (connected by an edge). By deriving a discrete version of the continuous Laplace operator, we have a basis to move on to obtain the graph Laplacian.

Since $f : \mathbb{R}^n \to \mathbb{R}$ produces a real number, the vector $\mathbf{f} = [f(v_1), f(v_2), ..., f(v_n)]^T$ is an n-dimensional column vector with respect to a graph $G = (V, E)$ with the node set $V = \{v_i\}_{i=1}^n$. Then

$$\Delta \mathbf{f} = [\Delta f(v_1), \Delta f(v_2), ..., \Delta f(v_n)]^T \tag{4.32}$$

and examining a single row (one entry) we have that

$$-\Delta f(v_i) = \sum_{v_l \sim v_i} f(v_i) - f(v_l) = \deg(v_i) f(v_i) - \sum_{v_l \sim v_i} f(v_l). \tag{4.33}$$

Let us define an $n \times n$ matrix (which we will later call the graph Laplacian)

$$L = D - A \tag{4.34}$$

where D is a diagonal matrix with $\deg(v_i)$ as the only non-zero entry in the i-th row,

and A is the adjacency matrix (an $n \times n$ matrix where $a_{ij} = 1$ if edge $\{i, j\} \in E$ and $a_{ij} = 0$ otherwise). Then it is no coincidence that

$$L\mathbf{f} = -\Delta\mathbf{f} \tag{4.35}$$

by Equation 4.33. As hinted above, L in Equation 4.34 is the *graph Laplacian* (let us assume we are examining finite graphs). This definition of the graph Laplacian aligns with previous work [17]. Note that we are referring to what is also called the combinatorial graph Laplacian. Other forms of graph Laplacians include the normalized and symmetrized graph Laplacian [28, 41].

4.5.3. Variation of Signature Row Vector from a Degree Matrix

In Definition 3.1, we recorded degrees of neighbors of a node in ascending order in each row. Each of these row vectors can be viewed as a signature vector for a node. In this section, we propose a possible variation of this signature vector, allowing node labeling. In particular, we propose that the signature vector for a node v_i can be modified to an $1 \times n$ row vector $\mathbf{s}$ such that

$$s_l(v_i) = \begin{cases} \deg(v_l) & \text{if} \{v_l, v_i\} \in E \\ \deg(v_i) & \text{otherwise.} \end{cases} \tag{4.36}$$

Let us call $\mathbf{s}$ the *labeled signature vector* of v_i.

4.5.4. The "Curvature-Laplacian" Equation

Finally, we have come to the section where we can introduce an interesting equation. The fact that the Ricci curvature at a node can be represented with the graph Laplacian (which is fixed for the same graph), the signature vector, and the degree of the node itself, motivates us to derive a curvature-directed graph alignment algorithm

similar to the DMC.

Theorem 4.5 (Curvature-Laplacian Equation)**.** *Let Ric(v) be defined as in Equation 4.22. Let L be the graph Laplacian. Let* $\mathbf{s}$ *be the labeled signature vector of a node* v_i*, as introduced in the previous section. Then*

$$Ric(v_i) - (L\mathbf{s}^T)_i = 2deg(v_i)(1 - deg(v_i)) \tag{4.37}$$

is true for an unweighted graph. $(L\mathbf{s}^T)_i$ *is the i-th entry of the resulting vector.*

Proof. By Equations 4.21 and 4.22, we know

$$\mathrm{Ric}(v_i) = \sum_{v_l \sim v_i} 2 - \deg(v_i) - \deg(v_l). \tag{4.38}$$

Since $Lf = -\Delta f$, we have that

$$L\mathbf{s}^T = -\Delta \mathbf{s}^T, \tag{4.39}$$

and hence

$$(L\mathbf{s}^T)_i = -(\Delta \mathbf{s}^T)_i = \sum_{v_l \sim v_i} s_i - s_l = \sum_{v_l \sim v_i} \deg(v_i) - \deg(v_l). \tag{4.40}$$

It follows naturally that

$$\mathrm{Ric}(v_i) - (L\mathbf{s}^T)_i = 2\deg(v_i) - \deg(v_i)^2 - \sum_{v_l \sim v_i} \deg(v_l) - \deg(v_i)^2 + \sum_{v_l \sim v_i} \deg(v_l), \tag{4.41}$$

which reduces to

$$\mathrm{Ric}(v_i) - (L\mathbf{s}^T)_i = 2\deg(v_i)(1 - \deg(v_i)). \tag{4.42}$$

This completes the proof. □

We then present to the reader another interesting result, where the mean of the difference $\mathrm{Ric}(v_i) - (L\mathbf{s}^T)_i$ holds constant over the probability distribution of degrees of a graph with the preferential attachment phenomenon, as discussed in Section 4.2.2.

Corollary 4.6 (Curvature-Laplacian Constant)**.** *Suppose we have a model graph developed with preferential attachment. Then the mean of the difference* $Ric(v_i) - (L\mathbf{s}^T)_i$ *with respect to degrees is a constant determined by the graph degree distribution.*

Proof. According to Equation 4.13, we can write

$$\mathrm{P}(\deg(v) = x) = Cx^{-3}. \tag{4.43}$$

Then the mean (expected value) of the specified difference for a graph with all nodes attached to some edge is

$$\int_1^\infty Cx^{-3} \cdot 2x(1-x)dx \tag{4.44}$$

by substituting $\mathrm{Ric}(v_i) - (L\mathbf{s}^T)_i$ through Equation 4.37. We calculate the integral

$$C\int_1^\infty \frac{2(1-x)}{x^2}dx = C\int_1^\infty \frac{2}{x^2}dx - C\int_1^\infty \frac{2}{x}dx = 2C \tag{4.45}$$

which is constant. We discard an infinity remainder term since, in reality, we would be examining finite graphs. □

4.5.5. Possible Implications of the Equation

Our first observation is not necessarily based on the equation, but inspired by it. That is, the graph curvature at each node has a positive upper bound, but not necessarily on the negative end.

To show this, note that the Ricci curvature at some node v_i is represented by

$$\text{Ric}(v_i) = 2\text{deg}(v_i) - \text{deg}(v_i)^2 - \sum_{v_l \sim v_i} \text{deg}(v_l) \tag{4.46}$$

and it has an upper bound

$$\text{Ric}(v_i) \leq 1 \tag{4.47}$$

since $f(x) = 2x - x^2$ (x is $\text{deg}(v_i)$) has 1 as upper bound and the term $-\sum_{v_l \sim v_i} \text{deg}(v_l)$ must be nonpositive. Moreover, we show that there will not be a lower bound on a connected graph with n nodes. Still examining Equation 4.46, we observe that $f(x) = 2x - x^2, x \in \mathbb{Z}^+$ has smallest value when x is maximized. In the connected graph's case (not allowing multiple edges between two nodes), a node can at most be connected to $n-1$ other nodes, so $x_{max} = n-1$. Therefore, $f(x)_{min} = f(x_{max}) = 2(n-1) - (n-1)^2$. By quadratic functions' properties, we know $f(x)_{min} \to -\infty$ as $n \to \infty$. Since, again, the remainder term is nonpositive, we arrive at the conclusion that $\text{Ric}(v_i)_{min} \to -\infty$ as $n \to \infty$.

Section 4.6

More Theory for Applying RMC

If we scrutinize Equation 4.37 more closely, we will notice that when node degree is low, the left hand side is small. This implies that $(L\mathbf{s}^T)_i$ can be used to approximate the Forman-Ricci curvature. More generally, we can think of $\mathbf{s}^T$ as a kind of representation of local curvature, since the graph Laplacian is fixed for the same graph.

One could apply the RMC directly to graphs as we did before. But this does not work as well as the DMC as one would expect. Degree directly and accurately describes a node, but the discrete graph curvature is a derived analogue of the con-

tinuous curvature. However, we thought it would be interesting to explore RMC on graphs that are supposed to "look more like" continuous objects. When approximating a smooth surface with a discrete grid, it is always better to use more nodes. This is just like how we desire smaller (and hence more) time steps when applying numerical methods to plotting functions.

Definition 4.7 (Line Graph)**.** A *line graph* $L(G)$ of an undirected graph $G = (V, E)$ is a graph that takes the edge set E of G to be the new node set so that $L(G) = (E, E')$, where E' is the set of all connections between new vertices that were originally adjacent edges in G.

We introduce the line graph in Definition 4.7 since it is one of the types of graphs that is more like an approximation of a continuous surface. Every local node, along with the edges connected to it, in G becomes a locally complete component in the new line graph $L(G)$. This clustering helps form compact local structures that will help with approximating a smooth surface with filled volume.

We can use the line graphs of graphs to help with graph alignment mainly due to the following result, proved in [40]. It essentially states that, in most cases, we can correspond each connected graph with a unique line graph.

Theorem 4.8 (Whitney Isomorphism Theorem)**.** *If the line graphs of two connected graphs are isomorphic, then the underlying graphs are isomorphic, except in the case of the triangle graph K_3 and the claw $K_{1,3}$, which have isomorphic line graphs but are not themselves isomorphic.*

For those who are unfamiliar with specific graph notations, K_n is a complete graph with n nodes. So K_3 is just a triangle. $K_{m,n}$ is the notation for a complete bipartite graph that connects all possible pairs of nodes across two different groups, with m and n nodes in them, while connecting no edges at all between nodes in the same

group. Therefore, $K_{1,3}$ is just a star with 3 edges. This graph has one center and it connects to three other nodes, with no other connections.

Section 4.7

RMC on Standard Tori

4.7.1. Tiling Methods to Construct a Torus

We will highlight several attempts to tile both a two-dimensional ring and a three-dimensional torus uniformly with regular shapes. Our goal is to have the ability to approximate the standard one-hole torus with regular tiling methods, and this can be generalized to higher-genus objects through simply connecting one-hole components. A regular tiling of a smooth torus with polygons is not possible, and approximations with polygons are important because this will be helpful in extracting graphs from the tiling. Our focus is not on finding new tiling methods, but rather to make clear the type of tiling we need to make the extracted graph to be useful. One could look into [12, 26] and Penrose tiling, for example, to appreciate the many types of geometric tiling, which is a fascinating topic that has been studied for a long time.

Two-Dimensional Tiling. To examine two-dimensional tiling for an approximated ring, we point out that there are only three edge-to-edge regular tiling in a plane (tile with only the same type of standard polygons), and the shapes that can do this are: equilateral triangles, squares, and regular hexagons. This type of tiling is also called regular tessellation. We discuss the monohedral tilings using equilateral triangles and squares, and also mention dihedral tiling with the two shapes. While hexagons embody very interesting mathematical properties, we will not dive deep into it since it is not as basic a unit as a triangle or a square. An equilateral triangle tiling is given in Figure 4.12. Possible two-dimensional rings are formed by triangles

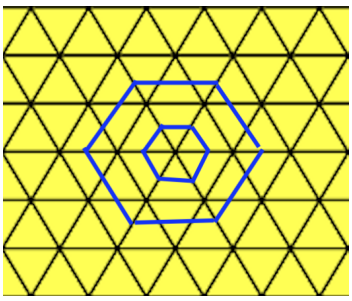


Figure 4.12: Equilateral triangular tiling. Blue hexagons indicate triangles that form a hexagonal ring.

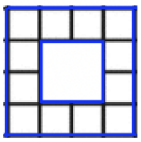


Figure 4.13: Square tiling for an approximate ring.

traced by the blue hexagons. This case is simple to implement and is a good basic model to start with. As we explore a curvature-based graph alignment method, this type of tiling gives us different curvatures for edges and nodes at different locations of the torus. Another simple approach is to use squares to build a square frame (see Figure 4.13), which is topologically a one-hole object. However, the triangle approach fits our needs better, because for squares, being a node near the hole or farther away does not necessarily change the curvature.

As mentioned above, we can also create a mixed tiling of squares and triangles (see Figure 4.14). This can be achieved through tiling six squares that each share one edge with a hexagon, then fill the gaps in the ring with equilateral triangles. Again, this tiling is valuable because we have variations in degrees of nodes (and hence the curvatures).

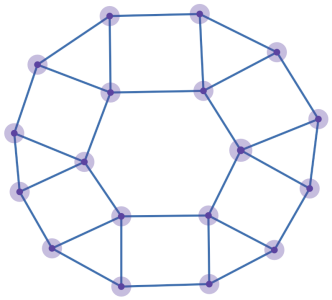


Figure 4.14: Equilateral triangular tiling. Blue hexagons indicate triangles that form a hexagonal ring.

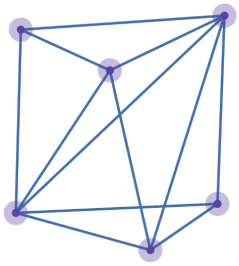


Figure 4.15: Triangulating a prism within three-dimensional torus.

Three-Dimensional Tiling. We first create a two-dimensional ring, then lift a parallel layer and connect all corresponding nodes in the two copies. This already produces a torus-like shape. If one were looking for further triangulation, one could do so easily in the triangular case, where we obtain prisms that replace the original flat triangles in the plane. Within these prisms, we can add three edges to divide them into tetrahedrons (see Figure 4.15).

Essentially, we are looking for the simplest tiling method that produces a non-uniform degree (curvature) distribution over different locations of the torus. For example, we would want the curvature of a node near the hole to be different from that of a node on the outer brink.

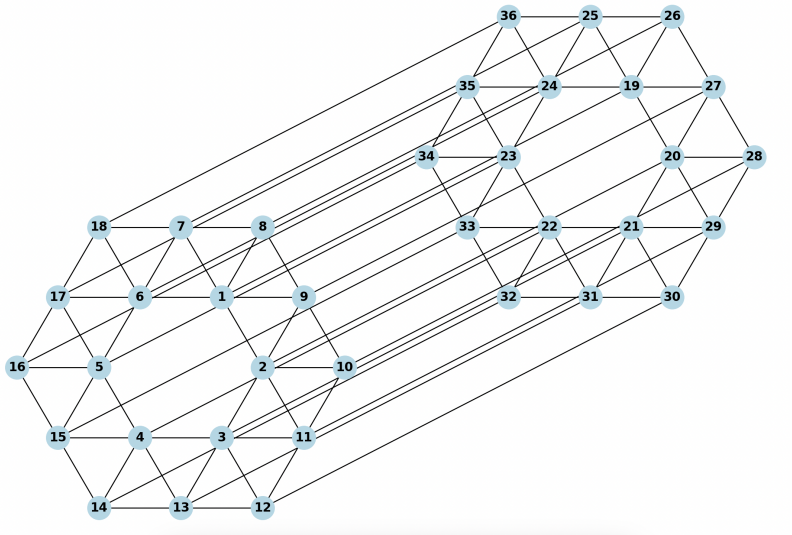


Figure 4.16: 3D triangular tiling of torus.

4.7.2. Thought Experiment: Applying RMC to Tori

We examine the effects of applying RMC to a pair of identically tiled tori created by first creating a two-dimensional triangular tiling as described in Section 4.7.1 and then lifting it (see Figure 4.16). If we examine the node curvatures of this graph, we find the following spectrum of distribution (see Figure 4.17). There are three types of nodes in the torus, and we find that each of their Ricci matrix row vector will be different, which will help us differentiate the three types of nodes at different locations of the graph. Let us call the nodes with smallest curvature A, medium curvature nodes B, and the rest C. A is closest to the center (bordering the hole of the torus), B and C are on the outer brink of the "doughnut" shape. Because of the discretized implementation, B and C types become different, even though in the continuous case they should be the same. But we see that $|\mathbf{Ric}(B) - \mathbf{Ric}(C)| < |\mathbf{Ric}(A) - \mathbf{Ric}(B)|$, so at least the nodes on the outer brink are more similar to each other. The Ricci matrix row vectors for nodes A, B, C are $[\mathbf{Ric}(A), \mathbf{Ric}(A), \mathbf{Ric}(A), \mathbf{Ric}(B), \mathbf{Ric}(B), \mathbf{Ric}(C)]$, $[\mathbf{Ric}(A), \mathbf{Ric}(A), \mathbf{Ric}(B), \mathbf{Ric}(C), \mathbf{Ric}(C), 0]$, and $[\mathbf{Ric}(A), \mathbf{Ric}(B), \mathbf{Ric}(B), \mathbf{Ric}(C), 0, 0]$, respectively. If we apply RMC to two tori of this type, we will be able to completely

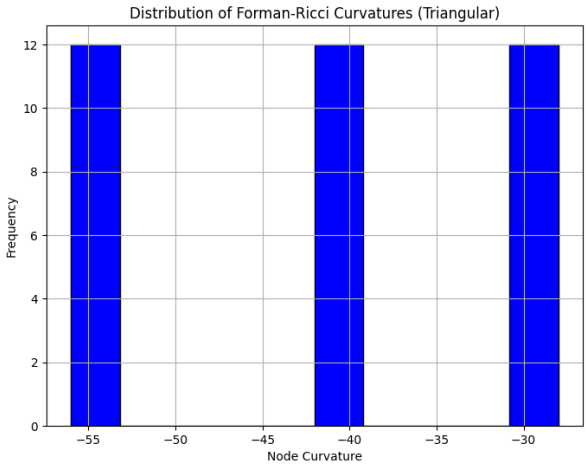


Figure 4.17: Distribution of curvature in triangle-tiled torus.

align the hole of the torus to the hole on another torus with the same shape, since the A nodes are located at and only located at the "boundaries" of the hole (see for example, nodes labeled 1 to 6 in Figure 4.16). We do not elaborate on the square tiling and mixed tiling cases as the methodology to approach them is the same. We will get distinct row vectors for nodes at different locations (by "different" we mean geometrically non-isomorphic).

Chapter 5

Experimental Results on Real World Networks

We first list here the sources of downloaded datasets. From COSNET [44], we downloaded the Flickr, Last.fm, and Myspace networks; from SNAP [22], we downloaded the Facebook and YouTube networks, along with the CA and PA roadmaps, and a multi-layer tissue PPI network (unweighted); from STRING [35], we downloaded five weighted PPI networks for different species that are typical subjects in biological studies; all other networks are taken from Network Repository [29]. We created a combined PPI network incorporating all tissues; this file is included in the Github repository affiliated with [37]. Before we dive into results for specific graphs, we note here that all isomorphic graph alignment yielded 100% correct results for DMC, Greedy DMC, and Weighted DMC.

Method	Score (%)	Complexity
DMC	96.09	$\sim O(n^2) - O(n^3)$
Greedy DMC	50.46	$\sim O(n^2) - O(n^3)$
REGAL	$80 < p < 90$	$\sim O(n) - O(n^2)$
FINAL	$35 < p < 45$	$\sim O(n^2)$
Klau	$20 < p < 30$	$\sim O(n^5)$
IsoRank	$5 < p < 15$	$\sim O(n^4)$
Two-Step	4.42	$\sim O(n^2) - O(n^3)$
Euc Dist.	0.15 (near zero)	$\sim O(n^2) - O(n^3)$

Table 5.1: Comparison with baselines on PPI network.

Section 5.1

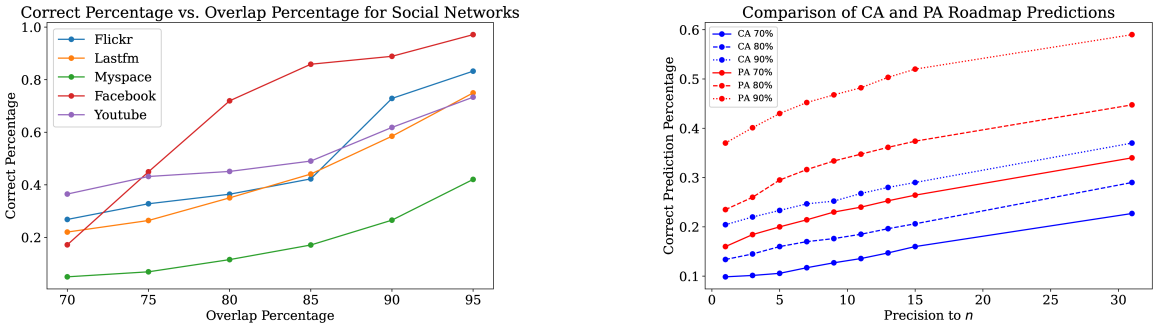
Unweighted (DMC and Greedy DMC)

In Table 5.1, we display statistics that compare DMC and Greedy DMC with baselines after applying all methods to the unweighted multi-layer PPI network [11, 14, 31, 43]. Using the edge deletion sampling method, we set deletion probability $p_d = 0.01$. Regarding the different graph alignment methods, we note that the Greedy DMC uses a threshold of $\epsilon = 100$. "Two-Step" is, as its name suggests, a two-step correction method that uses degrees for alignment, and refines that first-step alignment through average degrees of neighbors. "Euc Dist." aligns two graphs by minimizing Euclidean distances between embeddings. We can observe through direct comparison that DMC has highest accuracy, but larger complexity compared to REGAL and FINAL [14, 43]. The Greedy DMC strikes a similar balance compared to FINAL: while it is more accurate, it could also be higher in time complexity.

After showing the potential of DMC and Greedy DMC by comparing to baselines on a classic network (PPI), we turn to demonstrating that DMC works better when networks are more heterogeneous. We first draw an observed correlation between variance in degree distribution and performance of the DMC; the correlation is not strict, but implies from an empirical perspective that the DMC works well for hetero-

Networks	Mean	Variance	Total Nodes (N)
Flickr	52.36	5330.29	5000
Last.fm	26.03	2164.02	5000
Myspace	7.59	113.32	5000
Facebook	85.06	4040.27	700
YouTube	14.52	960.27	5000
CA Roadmap	2.55	0.96	1000
PA Roadmap	2.88	0.78	1000

Table 5.2: Means and variances of degree distributions of G_s.



(a) Social Networks (b) Roadmaps

Figure 5.1: We use the random walk graph sampling method for these experiments.

geneous networks. Table 5.2 provides a summary of statistics for social networks and roadmaps, and Figure 5.1 provides results applying DMC to the networks. The best performing Facebook and YouTube have large variances in their degree distributions. Comparing Flickr, Last.fm, and Myspace, Flickr and Last.fm are more suitable for DMC, which is most likely due to their much larger variance in degree distribution. However, the correlation between variance and performance can be false at times, as can be seen from the California and Pennsylvania roadmaps. The two roadmaps have close degree distribution means and variances, with the Pennsylvania one having lower degree variance, but it outperforms the California roadmap significantly.

Moreover, we show that DMC works better on graphs with high densities and clusterings in Table 5.3. We provide results of applying DMC to more biological networks randomly chosen from the Brain Network and Biological Network sections

Networks	Score	Density	Clustering
grid-fission-yeast	.9932	**.0123**	**.1874**
mouse-retina-1	.9900	**.9983**	**1.5539**
ce-gn	.9894	**.0218**	**.1839**
fly-drosophila-medulla-1	.9378	**.0211**	**3.8729**
grid-human	.8440	.0014	.0800
mouse-kasthuri	.6156	.0032	0
yeast	.5822	.0018	.0708

Table 5.3: Table of experiments on biological networks, ordered by DMC's performance on $p = 90\%$ overlap subgraphs.

in the Network Repository database. In Table 5.3, "grid-fission-yeast", "ce-gn","grid-human", and "yeast" are from the Biological category, and "mouse-retina-1", "fly-drosophila-medulla-1", and "mouse-kasthuri" are from the Brain category. We note that the highlighted densities are those above the 0.01 threshold, and for the average clustering coefficient the threshold is 0.1 (density and clustering taken from Network Repository). The graphs with both a high density and a high clustering coefficient outperformed the other graphs significantly.

In this paragraph, we examine the behavior of DMC and Greedy DMC applied to the unweighted multi-layer PPI network under different conditions. For DMC, we see in Figure 5.2 that DMC's performance decays as deletion probability (noise level) increases from 0.01 to 0.1. This is expected and no surprise for a heuristic algorithm, but it is worth pointing out that the performance does not rapidly drop to near zero values even with ten times the experimental noise level, implying that the method has potential to withstand noise. In Table 5.4, we see that when $\epsilon = 0, 10$, the Greedy DMC algorithm maintains high performance. We will study these special cases in the future.

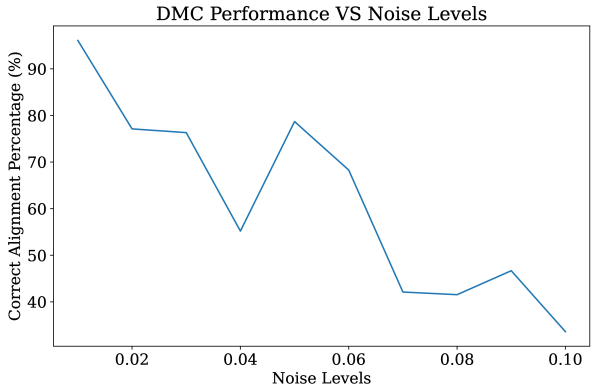


Figure 5.2: Performance Decay Plot

ϵ	Score (%)
0	96.58%
10	96.32%
100	50.46%
1000	0.18%

Table 5.4: Greedy DMC Performance

Section 5.2

Weighted DMC

As previously mentioned, we also dedicate a brief section to the Weighted DMC to demonstrate its potential, which in turn will also support that DMC, albeit its simplicity, is a valid approach. Table 5.5 provides the results of applying the Weighted DMC to five weighted PPI networks for different species that are typical subjects of biological studies, all taken from the STRING dataset [35]. Among the five species, E. Coli is the only prokaryotic organism, meaning that it has simpler cell structure lacking a sophisticated nucleus. We can reasonably speculate that the PPI for E. Coli is less heterogeneous, leading to a less accurate Weighted DMC result. We have also discussed in Section 5.1 that heterogeneous networks tend to be more suitable for the simple DMC. The other four PPI networks all yielded performance above 95%, which

Species	Score (%)
Human	99.19
Yeast	98.06
Mouse	97.27
Fruit Fly	96.26
E. Coli	91.46

Table 5.5: Weighted PPI Networks

is high.

We run the entire process, from random walk to graph alignment, for ten times and take their mean performance. All experiments sampled 5000-node graphs except for E. Coli, where we sampled 3000-node graphs, due to its limited size. We used $p_d = 0.01$ deletion probability for all experiments.

We have also run experiments on social networks, where the results are reported in Table 5.6. "soc-sign-bitcoin-otc" and "soc-sign-bitcoin-alpha" are taken from the SNAP database [22], and "music-cotagging", "ai-cotagging", "apple-cotagging", "travel-cotagging", and "economics-cotagging" are taken from the Computer Science Department webpage of Cornell University [10]. The "cotagging" networks are networks by keywords on Stack Exchange. We used the edge deletion sampling method, setting deletion probability $p_d = 0.01$, and run all experiments ten times to take the mean performance. We can speculate here that including weight information of edges has helped us gain more insight into local graph structure, since the overall performance of the Weighted DMC on the social networks is higher than that of DMC applied to unweighted social networks.

Network	Score (%)	Sampled Nodes
soc-sign-bitcoin-otc	77.44	5000
soc-sign-bitcoin-alpha	82.08	3000
music-cotagging	98.91	467
ai-cotagging	96.95	285
apple-cotagging	98.65	1063
travel-cotagging	97.35	1779
economics-cotagging	98.27	369

Table 5.6: Other Weighted Networks

Section 5.3

Applying RMC to Line Graph of PPI

5.3.1. Experiments on Line Graphs of Complex Networks

As mentioned earlier, the protein-protein interaction (PPI) network is a common type of network used to test alignment methods. We used the same combined PPI network as before in [37]. We transform this PPI network to its line graph form using the `line_graph` function from `networkx`. Since the line graph transformation is expensive, we first take a 1000-node graph from the original PPI network through random walk, then transform that sampled graph into a line graph. Afterwards, we sample two 500-node subgraphs for alignment using RMC. This is done by doing random walk on the 1000-node graph just obtained, and then we create a second graph for alignment through random deletion of edges of the first 500-node graph at a probability of $p = 0.01$. We use the same random walk method and alignment set up as before. We run the same experiment for ten times (see Table 5.7). Furthermore, we note that the CPU model used was Apple M1 with 16GB of memory. In particular, we implemented the experiments using Python 3.9.6 on Pycharm. Running the entire process, from random walk, to creating the line graph, and the graph alignment took around twenty hours. Graph alignment for ten times takes less than an hour. If we

Round	Absolute Node Count	Score (%)
1	416	83.2%
2	445	89.0%
3	464	92.8%
4	448	89.6%
5	452	90.4%
6	476	95.2%
7	438	87.6%
8	462	92.4%
9	432	86.4%
10	460	92.0%

Table 5.7: RMC on Line Graph of PPI Network

refer to Table 5.1 for baseline results, we see that RMC is more accurate than most methods except for DMC and is at a similar level as REGAL. While the line graph generation is lengthy, the time complexity of RMC itself is on the same level as DMC, at around $O(n^2) - O(n^3)$.

Chapter 6

Conclusion and Future Work

We proposed several graph alignment methods. For unweighted networks, we proposed DMC and Greedy DMC (a computationally cheap variation). We also proposed the Weighted DMC for weighted graphs. Moreover, we explored the use of the Forman-Ricci curvature as a main feature of nodes in matrix comparison.

We showed that the methods are theoretically motivated, mainly through performance analysis and deriving the Curvature-Laplacian equation, and analyzed experimental results. DMC's accuracy on classic PPI networks proves to be higher than carefully chosen baselines, and it works best for heterogeneous graphs, which has been demonstrated from multiple perspectives. This process included providing an overview of synthetic graph generation models and a short discussion of the difficulty of modeling complex networks. Along with the proposal of RMC, we discussed possible smooth surface discretization methods that utilize regular shapes as units. In the search for graphs that are suitably described by curvatures, we not only examined the discretization of a torus, but also proposed using line graphs of complex networks for RMC alignment.

Future works could examine special cases of Greedy DMC (tolerance $\epsilon = 0, 10$). One could also test this sequence of methods with more datasets to clearly define

the optimal set of networks for these algorithms which can be easily replicated. One could also attempt to derive more theoretical guarantees for performance, which is possible since there are initial theoretical guarantees for DMC. Last but not least, we suspect the Curvature-Laplacian equation to be related to the Euler characteristic.

Bibliography

[1] Réka Albert and Albert-Laśzló Barabási, *Statistical mechanics of complex networks*, Reviews of Modern Physics **74** (2002), 47–97.

[2] Nazanin Alipourfard, Buddhika Nettasinghe, Andrés Abeliuk, Vikram Krishnamurthy, and Kristina Lerman, *Friendship paradox biases perceptions in directed networks*, Nature Communications **11** (2020), no. 707.

[3] Albert-László Barabási, *Network science*, Cambridge University Press, 2016.

[4] Mikhail Belkin and Partha Niyogi, *Towards a theoretical foundation for laplacian-based manifold methods*, Journal of Computer and System Sciences (2008).

[5] Sébastien Bougleux, Benoit Gaüzère, and Luc Brun, *A hungarian algorithm for error-correcting graph matching*, International Workshop on Graph-Based Representations in Pattern Recognition (Cham), Springer, 2017, pp. 118–127.

[6] Fan Chung, *Spectral graph theory*, CBMS Regional Conference Series in Mathematics (1997).

[7] David F. Crouse, *On implementing 2d rectangular assignment algorithms*, IEEE **52** (2016), no. 4, 1679–1696.

[8] Rick Durrett, *Random graph dynamics*, Cambridge University Press, New York, NY, USA, 2009.

[9] Jack Edmonds and Richard M. Karp, *Theoretical improvements in algorithmic efficiency for network flow problems*, Journal of the ACM **19** (1972), no. 2, 248–264.

[10] Xiang Fu, Shangdi Yu, and Austin R. Benson, *Modelling and analysis of tagging networks in stack exchange communities*, Journal of Complex Networks **8** (2020), no. 5.

[11] Ji Gao, Xiao Huang, and Jundong Li, *Unsupervised graph alignment with wasserstein distance discriminator*, Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD '21) (New York, NY, USA), Association for Computing Machinery, 2021, pp. 426–435.

[12] Branko Grünbaum and Geoffrey C. Shephard, *Tilings by regular polygons*, Mathematics Magazine **50** (1977), no. 5, 227–247.

[13] Allison Gunby-Mann, *Machine learning for graph algorithms and representations*, Ph.D. dissertation, Thayer School of Engineering, Dartmouth College (2024).

[14] Mark Heimann, Haoming Shen, Tara Safavi, and Danai Koutra, *Regal: Representation learning-based graph alignment*, 27th ACM International Conference on Information and Knowledge Management (CIKM '18) (Torino, Italy), Association for Computing Machinery, 2018, pp. 117–126.

[15] Jürgen Jost, *Curvature of graphs*, 2013.

[16] Gunnar W. Klau, *A new graph-based method for pairwise global network alignment*, BMC Bioinformatics **10** (2009), no. S59.

[17] Aleksey Kostenko and Noema Nicolussi, *Laplacians on infinite graphs: Discrete vs continuous*, arXiv Preprint arXiv:2105.09931 (2021).

[18] Oleksii Kuchaiev, Tijana Milenković, Vesna Memišević, Wayne Hayes, and Nataša Pržulj, *Topological network alignment uncovers biological function and phylogeny*, Journal of the Royal Society Interface **7** (2010), no. 50, 1341–1354.

[19] Wilmer Leal, Guillermo Restrepo, Peter F. Stadler, and Jürgen Jost, *Forman-ricci curvature for hypergraphs*, Advances in Complex Systems **24** (2021), no. 1.

[20] Kristina Lerman, Xiaoran Yan, and Xin-Zeng Wu, *The "majority illusion" in social networks*, PLoS One **11** (2016), no. 2.

[21] Jure Leskovec and Christos Faloutsos, *Sampling from large graphs*, Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '06) (New York, NY, USA), Association for Computing Machinery, 2006, pp. 631–636.

[22] Jure Leskovec and Andrej Krevl, *SNAP Datasets: Stanford large network dataset collection*, `http://snap.stanford.edu/data`, 2014.

[23] Chengjiang Li, Yixin Cao, Lei Hou, Jiaxin Shi, Juanzi Li, and Tat-Seng Chua, *Semi-supervised entity alignment via joint knowledge embedding model and cross-graph model*, Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (Hong Kong, China), Association for Computational Linguistics, 2019, pp. 2723–2732.

[24] Ulrike Von Luxburg, Mikhail Belkin, and Olivier Bousquet, *Consistency of spectral clustering*, The Annals of Statistics (2008).

[25] Marianna Milano, Mario Cannataro, and Pietro H. Guzzi, *Glalign: Using global graph alignment to improve local graph alignment*, IEEE International Conference on Bioinformatics and Biomedicine (BIBM '16) (Shenzhen, China), IEEE, 2016, pp. 1695–1702.

[26] Kinga Nagy and Viktor Vígh, *Monohedral tilings of a convex disc with a smooth boundary*, Discrete Mathematics **346** (2023), no. 1, 1–14.

[27] Azadeh Parvaneh, *The friendship paradox–and how it might produce a biased world*, 2002.

[28] J. Wilson Peoples and John Harlim, *Spectral convergence of symmetrized graph laplacian on manifolds with boundary*, arXiv Preprint arXiv:2110.06988 (2025).

[29] Ryan A. Rossi and Nesreen K. Ahmed, *The network data repository with interactive graph analytics and visualization*, `https://networkrepository.com`, 2015.

[30] Jayson Sia, Edmond Jonckheere, and Paul Bogdan, *Ollivier-ricci curvature-based method to community detection in complex networks*, Nature: Scienctific Reports **9** (2019), no. 9800.

[31] Rohit Singh, Jinbo Xu, and Bonnie Berger, *Global alignment of multiple protein interaction networks with application to functional orthology detection*, PNAS **105** (2008), no. 35, 12763–12768.

[32] Konstantinos Skitsas, Karol Orłowski, Judith Hermanns, Davide Mottin, and Panagiotis Karras, *Comprehensive evaluation of algorithms for unrestricted graph alignment*, Proceedings of the 26th International Conference on Extending Database Technology (EDBT '23), OpenProceedings, 2023, pp. 260–272.

[33] Remanan P. Sreejith, Karthikeyan Mohanraj, Ju rgen Jost, Emil Saucan, and Areejit Samal, *Forman curvature for complex networks*, arXiv Preprint arXiv:1603.00386 (2016).

[34] Steffen Staab and Rudi Studer, *Handbook on ontologies*, Springer, Berlin, Heidelberg, 2009.

[35] Damian Szklarczyk, Rebecca Kirsch, Mikaela Koutrouli, Katerina Nastou, Farrokh Mehryary, Radja Hachilif, Annika L. Gable, Tao Fang, Nadezhda T. Doncheva, Sampo Pyysalo, Peer Bork, Lars J. Jensen, and Christian von Mering, *The string database in 2023: protein-protein association networks and functional enrichment analyses for any sequenced genome of interest*, PubMed **51** (2023), no. D1, D638–D646.

[36] Nobuaki Tomizawa, *On some techniques useful for solution of transportation network problems*, Networks **1** (1971), no. 2, 173–194.

[37] Ashley Wang and Peter Chin, *Degree matrix comparison for graph alignment*, arXiv Preprint arXiv:2411.07475 (2025).

[38] ———, *Ricci matrix comparison for graph alignment: A dmc variation*, arXiv Preprint arXiv:2505.15831 (2025).

[39] Melanie Weber, Emil Saucan, and Jürgen Jost, *Characterizing complex networks with forman-ricci curvature and associated geometric flows*, Journal of Complex Networks **5** (2017), no. 4, 527–550.

[40] Hassler Whitney, *Congruent graphs and the connectivity of graphs*, American Journal of Mathematics **54** (1932), no. 1, 150–168.

[41] David P. Williamson, *Spectral graph theory (lecture notes)*, 2016.

[42] Xu Yang, Cheng Deng, Zhiyuan Dang, Kun Wei, and Junchi Yan, *Self-sagcn: Self-supervised semantic alignment for graph convolution network*, 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (Nashville, TN, USA), IEEE, 2021, pp. 16770–16779.

[43] Si Zhang and Hanghang Tong, *Final: Fast attributed network alignment*, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16) (San Francisco, CA, USA), Association for Computing Machinery, 2016, pp. 1345–1354.

[44] Yutao Zhang, Jie Tang, Zhilin Yang, Jian Pei, and Philip S. Yu, *Cosnet: Connecting heterogeneous social networks with local and global consistency*, Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15) (New York, NY, USA), Association for Computing Machinery, 2015, pp. 1485–1494.